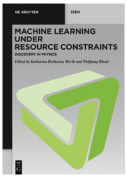
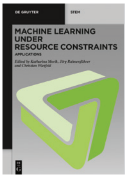
Katharina Morik, Peter Marwedel (Eds.)

Machine Learning under Resource Constraints · Fundamentals

Also of interest



Volume 2
Machine Learning under Resource Constraints.
Discovery in Physics
Morik, Rhode (Eds.), 2023
ISBN 978-3-11-078595-1, e-ISBN 978-3-11-078596-8



Volume 3
Machine Learning under Resource Constraints.
Applications
Morik, Rahnenführer, Wietfeld (Eds.), 2023
ISBN 978-3-11-078597-5, e-ISBN 978-3-11-078598-2

Machine Learning under Resource Constraints

Final Report of CRC 876

Editor in Chief
Katharina Morik

Volume 1/3

DE GRUYTER

Machine Learning under Resource Constraints

Fundamentals

Edited by
Katharina Morik and Peter Marwedel

DE GRUYTER

Editors
Prof. Dr. Katharina Morik
TU Dortmund University
Department of Computer Sciences
Chair for Artificial Intelligence
Computer Science 8
Otto-Hahn-Str. 12
44221 Dortmund
Germany

Prof. Dr. Peter Marwedel
TU Dortmund University
Computer Science 12
Otto-Hahn-Str. 16
44227 Dortmund
Germany

ISBN 978-3-11-078593-7
e-ISBN (PDF) 978-3-11-078594-4
e-ISBN (EPUB) 978-3-11-078612-5
DOI https://doi.org/10.1515/9783110785944

Library of Congress Control Number: 2022949268

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available on the Internet at http://dnb.dnb.de.

Cover image: Collaborative Research Center 876
Printing and binding: CPI books GmbH, Leck

www.degruyter.com

Contents

Preface —— XI

1 Introduction
Katharina Morik, Jian-Jia Chen —— 1

1.1 Embedded Systems and Sustainability —— 2

1.2 The Energy Consumption of Machine Learning —— 4

1.3 Memory Demands of Machine Learning —— 9

1.4 Structure of this Book —— 12

2 Data Gathering and Resource Measuring —— 15

2.1 Declarative Stream-Based Acquisition and Processing of OS Data with kCQL
Christoph Borchert, Jochen Streicher, Alexander Lochmann, Olaf Spinczyk —— 16

2.2 PhyNetLab Test Bed
Mojtaba Masoudinejad, Markus Buschhoff —— 34

2.3 Zero-Power/Low-Power Sensing
Andres Gomez, Lars Suter, Simon Mayer —— 46

3 Streaming Data, Small Devices —— 71

3.1 Summary Extraction from Streams
Sebastian Buschjäger, Katharina Morik —— 73

3.2 Coresets and Sketches for Regression Problems on Data Streams and Distributed Data
Alexander Munteanu —— 85

4 Structured Data —— 99

4.1 Spatio-Temporal Random Fields
Nico Piatkowski, Katharina Morik —— 100

4.2 The Weisfeiler-Leman Method for Machine Learning with Graphs
Nils Kriege, Christopher Morris —— 116

4.3 Deep Graph Representation Learning
Matthias Fey, Frank Weichert —— 129

4.4 **High-Quality Parallel Max-Cut Approximation Algorithms for Shared Memory**
Nico Bertram, Jonas Ellert, Johannes Fischer — **144**

4.5 **Millions of Formulas**
Lukas Pfahler — **160**

5 **Cluster Analysis — 179**

5.1 **Sparse Partitioning Around Medoids**
Lars Lenssen, Erich Schubert — **182**

5.2 **Clustering of Polygonal Curves and Time Series**
Amer Krivošija — **197**

5.3 **Data Aggregation for Hierarchical Clustering**
Erich Schubert, Andreas Lang — **215**

5.4 **Matrix Factorization with Binary Constraints**
Sibylle Hess — **227**

6 **Hardware-Aware Execution — 249**

6.1 **FPGA-Based Backpropagation Engine for Feed-Forward Neural Networks**
Wayne Luk, Ce Guo — **250**

6.2 **Processor-Specific Code Transformation**
Henning Funke, Jens Teubner — **263**

6.3 **Extreme Multicore Classification**
Erik Schultheis, Rohit Babbar — **272**

6.4 **Optimization of ML on Modern Multicore Systems**
Helena Kotthaus, Peter Marwedel — **285**

7 **Memory Awareness — 305**

7.1 **Efficient Memory Footprint Reduction**
Helena Kotthaus, Peter Marwedel — **306**

7.2 **Machine Learning Based on Emerging Memories**
Mikail Yayla, Sebastian Buschjäger, Hussam Amrouch — **325**

7.3 **Cache-Friendly Execution of Tree Ensembles**
Sebastian Buschjäger, Kuan-Hsun Chen — **338**

8 **Communication Awareness — 359**

8.1 **Timing-Predictable Learning and Multiprocessor Synchronization**
Kuan-Hsun Chen, Junjie Shi — **360**

8.2 **Communication Architecture for Heterogeneous Hardware**
Henning Funke, Jens Teubner — **379**

9 Energy Awareness — 405

9.1 **Integer Exponential Families**
Nico Piatkowski — **406**

9.2 **Power Consumption Analysis and Uplink Transmission Power**
Robert Falkenberg — **423**

Bibliography — 437

Index — 485

List of Contributors — 489

Preface

Machine learning has been part of Artificial Intelligence since its inception. Only a perfect being need not learn; all others, be they humans or machines, need to learn in order to enhance their capabilities. In the 1980s, learning from examples and modeling human learning strategies have been investigated in concert [490]. The formal statistical basis of many learning methods was put forward later and is still an integral part of machine learning [298]. Neural networks have always been in the toolbox of methods. Integrating all the pre-processing, exploitation of kernel functions, and transformation steps of a machine-learning process into the architecture of a deep neural network increased the performance of this model type considerably [265]. Modern machine learning is challenged by the amount of data and by the demand of real-time inference. This has led recently to an interest in computing architectures and modern processors. For many years, the machine-learning research could take the von Neumann architecture for granted. All algorithms were designed for the classical CPU. Issues of implementation on a particular architecture were ignored. This is no longer possible. The time for independently investigating machine learning and computational architecture is over.

Computing architecture has experienced a similarly rampant development from mainframe or personal computers in the last century to very large compute clusters and ubiquitous computing of embedded systems in the Internet of Things. Cyber-physical systems' sensors produce a huge amount of streaming data that need to be stored and analyzed. Their actuators need to react in real-time. This establishes a close connection with machine learning. Cyber-physical systems and systems in the Internet of Things consist of diverse components, heterogeneous both in hard- and software [470]. Modern multi-core systems, graphic processors, memory technologies, and hardware-software codesign offer opportunities for better implementations of machine-learning models.

Machine learning and embedded systems together now form a field of research that tackles leading edge problems in machine learning, algorithm engineering, and embedded systems. Machine learning today needs to make the resource demands of learning and inference meet the resource constraints of used computer architecture and platforms. A large variety of algorithms for the same learning method and diverse implementations of an algorithm for particular computing architectures optimize learning with respect to resource efficiency while keeping some guarantees of accuracy. To give just one example: the trade-off between a decreased energy consumption and an increased error rate needs to be theoretically shown for training a model and for model inference. Pruning and quantization are ways of reducing the resource requirements by either compressing or approximating the model. In addition to memory and energy consumption, timeliness is an important issue, since many embedded systems are integrated into large products that interact with the physical world. If the results are delivered too late, they may be useless. As a result, real-time guarantees are needed for

https://doi.org/10.1515/9783110785944-203

such systems. To efficiently utilize the available resources, e.g., processing power, memory, and accelerators, with respect to response time, energy consumption, and power dissipation, different scheduling algorithms and resource management strategies need to be developed.

We have dedicated three books to this emerging field of research. They present the results of 12 years of research in 12 projects that were pursued at the TU Dortmund University in the collaborative research center CRC 876 ("Providing Information by Resource Constrained Data Analysis"), funded by the Deutsche Forschungsgemeinschaft (DFG). A collaborative research center is the most selective type of DFG funding. Proposals are submitted in a two-step procedure. The proposals outline a perspective of 12 years in a composition of projects that together shape a research field with a large impact. If this first step is accepted, a detailed proposal for the first phase is submitted and carefully reviewed. After the first phase, its results together with a detailed proposal for the second phase are reviewed and may result in ending the CRC. Otherwise, the second phase starts and at its end, the results and the proposal for the third phase are submitted. At most, three phases are funded. A CRC is a strategic measure of German research funding. The CRC 876 boosted the careers of project leaders. Overall, CRC 876 had 36 project leaders, only 8 of them have been members from the beginning to the end. Hence, career opportunities could be offered to additional colleagues. The CRC 876 with its graduate school boosted the career of Ph.D. students: until 2021, more than 80 dissertations were successfully completed. Uncounted Bachelor and Master theses have been supervised. From this wealth we draw the content of the three books. In addition, guest authors contribute invited chapters.

- The first book establishes the foundations of this new field. It goes through all the steps from the acquisition of data, their summary, and clustering to the different aspects of resource-aware learning.

  Several learning methods are inspected with respect to their resource requirements and how to enhance their scalability on diverse computing architectures: deep neural networks, graph neural networks, tree ensembles, matrix factorization, and probabilistic graphical models.
- The second book is about machine learning for astroparticle and particle physics. Instruments such as the Large Hadron Collider or Cherenkov telescopes or the IceCube gather petabytes of data within which the relevant ones need to be detected, often in real-time, and be stored for further analysis. This builds upon the fundamental issues of the first book and moves into the pipeline of data acquisition, storage, and access, feature extraction, and learning. Here, machine learning is part of the probabilistic rationalism of epistemology. The physical knowledge is encoded in the Monte Carlo simulation and annotates the observations recorded by the instruments. The interpretation of learned models is to enhance physical knowledge. This yields a circle of theory development that is supported by machine learning.

- The third book describes how resource-aware machine-learning methods solve real-world problems in the areas of medicine, industry 4.0, traffic and smart cities, and mission-critical communication.

Each book is self-contained. Together they offer a comprehensive study of machine learning and embedded systems becoming real-time systems, saving energy and offering solutions to other fields. They represent an overview of the state of the art in studying the mutual dependence of machine learning and embedded system design. The presentation of this overview has been made feasible by an early vision of the importance of linking the two domains. We are enthusiastic about the fact that the vision underlying the creation of CRC 876 has become a main line of research worldwide. An early start has allowed us to study the links intensively. Now we would like to entrust to novices and masters alike what we have learned along the long journey of CRC 876, hoping that they might be inspired to work implementations of machine learning and embedded systems.

Enjoy!
Katharina Morik
Peter Marwedel

1 Introduction

Katharina Morik
Jian-Jia Chen

Abstract: An enormous amount of data is constantly being produced around the world, both in the form of large volume as in that of large velocity. Turning the data into information requires many steps of data analysis: methods for filtering and cleaning the data, joining heterogeneous sources, extracting and selecting features, summarizing and aggregating the data, learning predictions, estimating the uncertainties of the learned model and monitoring the model fitness in its deployment. All these processes need to scale up, whether the long analysis workflows are integrated into a deep learning architecture, or not. The data ecosystems no longer allow us to take von Neumann architecture for granted where only compilers or application systems address hardware issues. Specialized architectures for accelerating machine learning have been developed, and machine learning algorithms have been tailored to novel computer architectures. Both trends are aiming at efficiency, in particular the efficient use of given resources: the real time of execution, the amount of energy, memory and communication. In the struggle for sustainability resource restrictions are of utmost importance. Energy consumption in particular receives considerable attention. We believe that resource efficiency cannot be achieved by better machine learning algorithms or by better hardware architectures alone. It demands the smart combination of hardware and algorithms.

This chapter introduces the fundamentals of machine learning under resource constraints. Resource-aware machine learning is a new and important research field. It is motivated by the following three issues.

– The *resource constraints*, regarding energy consumption, memory requirements, real-time processing, and communication, are to be inspected and investigated under a large diversity of scientific viewpoints.
– The trend towards the *Internet of Things (IoT)* and the many data producing devices like cyber-physical systems or *embedded systems* pose a challenge to data processing. It has led to the programming paradigms of distributed analysis and federated analysis, with data summaries or compression as hot topics.
– The integration of *machine learning and modern hardware* has started to raise international awareness and research efforts.

We want to describe the new field and highlight the contributions of the Collaborative Research Center (CRC) 876 to creating it. The topical overview of machine learning

https://doi.org/10.1515/9783110785944-001

under resource constraints is divided into three sections. First, we discuss research on embedded systems and sustainability in Section 1.1. Then, we focus on machine learning and its energy consumption (Section 1.2). Section 1.3 considers approaches to reducing another important resource: the memory requirements of machine learning. Finally, in Section 1.4, we give an overview of the chapters of this book, which follow the steps of the data analysis process (Section 1.4).

1.1 Embedded Systems and Sustainability

Efficient and high-speed computing has always played a central role in the innovation of information and communication technology (ICT). It is rooted in the widely circulated document “First Draft of a Report on EDVAC” (Electronic Discrete Variable Automatic Computer) by John von Neumann [527]. Over decades, the von Neumann architecture, which consists of a processor unit, a control unit, a memory unit, and input/output peripherals, has been used to efficiently execute programs.

Although ICT has enabled many applications with a high impact on human society, more and more electricity is consumed worldwide. The growth of ICT also has a global impact on the sustainability of the electricity and CO2 footprint worldwide. It is projected that by 2030 ICT will account for 7 % and 20 % of global demand under the optimistic and expected estimated scenarios, respectively [18]. Hence, hardware and software researchers and engineers cannot simply ignore energy efficiency when evaluating their systems and workloads. With Moore’s law and Dennard scaling, the improvement of clock frequency of central processing units (CPUs) continued over decades until roughly 2005–2007. Nowadays, the transistor counts in integrated circuits are still growing, but the frequency improvement has ceased as power consumption and thermal dissipation have become the scaling bottleneck.

The discontinuation of Dennard scaling has resulted in the boosting of application-specific hardware accelerators in modern computers to perform efficient and high-speed computing. When Graphics Processing Units (GPUs) were introduced (in 1999 by Nvidia), they were only designed to accelerate the rendering of graphics. Today, application-specific GPUs have become general-purpose vector processors. For machine learning algorithms, specific accelerators include Google’s Tensor Processing Units (TPUs) and Apple’s Neural Engines. Until the late 1980s, information processing could only be performed on large mainframe computers. Later, the innovation of system integration and technology miniaturization enabled *embedded systems*, i.e., information processing embedded in enclosing products.

Nowadays, embedded systems are pervasive in human society and are widely used in cars, trains, planes, telecommunication, fabrication, ambient intelligence, and decision making. Such embedded systems typically interact with the physical environment to collect information and/or control/influence the physical environment. They share certain common characteristics and have to adhere to certain resource constraints, independent of the application area. Embedded systems are the core of many innovations, such as cyber-physical systems (CPS), Internet of Things (IoT), and Industry 4.0.

The pervasiveness of embedded systems and sensors contributes to the *big data* computing paradigm, in which data is collected and processed in the cloud. However, transferring data to the cloud consumes time and energy and may not be feasible due to privacy concerns. To address such issues, edge computing, in which embedded edge-nodes process their data locally and potentially share an abstracted model among each other, is an emerging computing paradigm. Such a paradigm shift is also motivated by privacy and security concerns and pushed by governmental policies, such as the California Consumer Privacy Act and the European Union's General Data Protection Regulation, GDPR, which disallow sending/storing sensitive user data to central servers. For example, Gaia-X is an initiative to establish an ecosystem for the next generation of data infrastructure complied with GDPR.

Such a paradigm shift is also driven by the advances of IoT and embedded devices. The annual DataSphere and StorageSphere forecasts published by International Data Corporation (IDC) in 2021 show that "IoT data (not including video surveillance cameras) is the fastest-growing data segment, followed by social media."[1] According to Statista,[2] the number of IoT devices will reach 25.44 billion in 2030.

Embedded systems and IoT devices do not just imply that the computation power is insufficient. Furthermore, they are typically subject to stringent resource constraints due to the design optimization for *resource efficiency* without sacrificing dependability. Specifically, their energy consumption needs to be particularly small. One study [282] analyzes the tradeoff between performance, measured as MobileNet v1 throughput, and the carbon footprint of mobile devices from Google, Huawei and Apple. They concluded that "from 2017 to 2019, software and hardware optimizations primarily focused on maximizing performance, overlooking the growth trend of carbon footprint." [282]

Under resource constraints, memory can be critical both for the code size and the run-time stack size since larger on-chip memory capacity generally leads to higher cost and higher energy/power consumption. Nowadays, the speed of off-chip memories is much slower than that of processors, resulting in the *memory wall* problem. In response, memory hierarchy has been developed in the last decades to enable the illusion that a large memory capacity can be created without significantly losing efficiency. Under such a scheme, modern embedded processors may have either a hardware-managed cache or a software-managed scratchpad memory (SPM), which can be utilized for performance and energy improvement by exploiting temporal and spatial locality.

Under von Neumann architecture, data movement between the physically-separating processing and memory units can be a performance bottleneck for both

[1] https://www.idc.com/getdoc.jsp?containerId=prUS47560321.

[2] https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

energy consumption and performance. Such a bottleneck can be avoided by considering hardware, that offers processing capabilities so that the data resides without the need to move it. This includes Logic-in-Memory (LiM) and Processing-in-Memory (PiM). LiM can be achieved by realizing Boolean logic (e.g., XNOR, NAND, etc.) using both conventional CMOS [8] and emerging beyond-CMOS [535] technologies. PiM can be achieved by exploiting the memory as a crossbar array for efficient vector-matrix operations. For example, TSMC has recently demonstrated an application-specific integrated circuit (ASIC) chip at the 22 nm node, which offers an SRAM-based full-precision PiM macro [138]. In January 2022, Samsung published a crossbar array of spin-transfer-torque magnetoresistive random-access memory (MRAM) for in-memory computing [352].[3]

In summary, edge computing is considered a key pillar to support artificial intelligence and machine learning in pushing our societies to an unprecedented technological revolution. In view of the above discussions, embedded system designers must understand how machine learning algorithms work and machine learning algorithm designers must understand how the underlying hardware can be efficiently utilized for executing machine learning algorithms.

In this book, we inspect cooperative work that aims at resource efficiency for a sustainable future. Focusing on embedded systems, the contributions in Chapter 6 discuss hardware-aware machine learning, including learning on FPGAs in Section 6.1, optimizing the learning on multicore systems in Sections 6.4 and 6.3 and processor-specific transformations in Section 6.2. Furthermore, memory awareness is investigated in Chapter 7, covering memory footprint reduction in Section 7.1, machine learning based on emerging memories (potentially beyond the classic von Neumann architecture) in Section 7.2, and cache-friendly machine learning in Section 7.3.

When devices are connected, communication, synchronization, and offloading are essential. With this in mind, effective synchronization with resource sharing, communication with potential failures, and probabilistic timing information are investigated in Section 8.1. Section 8.2 considers bandwidth limitations of different execution models and coprocessor-accelerated optimization.

The next sections focus on machine learning and introduce the Chapters on energy- and memory-saving machine learning methods.

1.2 The Energy Consumption of Machine Learning

Machine learning has always been a central part of Artificial Intelligence (AI). Already Allen Turing argued that programming a computer cannot scale up to the performance

[3] https://news.samsung.com/global/samsung-demonstrates-the-worlds-first-mram-based-in-memory-computing.

that a learning machine can achieve [673]. According to the AI index of Stanford University in 2022, publications in pattern recognition and machine learning have more than doubled since 2015. Other areas strongly influenced by deep learning, such as computer vision, data mining, and natural language processing have seen smaller increases.[4]

The classes of algorithms in machine learning are too many to be characterized, here. The field of machine learning covers a wide range. A bird's-eye view sees different *approaches*: geometric (e.g., decision trees, support vector machines), probabilistic (e.g., probabilistic graphical models, Bayesian models), combinatoric (k-means, frequent sets), logic (e.g., inductive logic programming), reinforcement models (e.g., bandit models), and neural networks (deep learning).

At a more technical level, we see *learning tasks* that specify the formal basis of machine learning methods, defining what is learned (classification, regression, probability density, cluster model), from what it is learned (real-valued vectors, time series, categorical data, count data), under which constraints (quality criteria, streaming/online, distributed). As is common in statistics, the term "model" is used not only for the class of possible learning results given the types of input, output and quality criteria, but also for a particular instance, the learning result.

Combining approaches and learning tasks, we see the areas of machine learning. All of them are growing. Several algorithms have been developed within these areas. Many of them use algorithms for underlying inner procedures or compose learning methods using *building blocks* such as kernel functions, matrix factorization, optimization, regularization, or sampling. Investigating machine learning at all levels, from the models to hardware architectures, is the particular profile of the research that has been undertaken by the Collaborative Research Center 876 (CRC 876).

Today, resource restrictions are of utmost importance. Energy consumption in particular receives considerable attention. Machine learning is put to good use in order to save energy for sustainability. Google considers the application of DeepMind's machine learning to its data centers to be its most important application. The energy used for cooling could be reduced by up to 40 % through machine learning.[5] Machine learning algorithms themselves are enhanced for low energy demands. One of the invited talks at the International Conference on Machine learning (ICML) 2018—Max Welling's "Intelligence per Kilowatt-hour" supports our approach to joining embedded systems and machine learning research. Its author said, "The next battleground in AI might well be a race for the most energy efficient combination of hardware and algorithms." CRC 876 has contributed in exactly to this race. The results of its work are reported here. In the following, we refer to the approaches described in this book concerning machine learning and the resources of energy and memory.

[4] aiindex.stanford.edu.

[5] https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/.

In general, *green computing* and *sustainable computing* have received considerable interest. On the one hand, machine learning decreases the ecological footprint of processes in many applications (see, e.g., [418]). On the other hand, machine learning itself might use tremendous amounts of energy. This is particularly true for big language models such as GPT-3. A careful analysis by Patterson et al. [556] compares the CO_2 footprint of different natural language learners. The relation of the run-time of training, the number of processors and their average power consumption together with the power usage efficiency of the particular computing center where the machine learning algorithm is executed gives an estimated kWh. This, in turn, is used to calculate tons of CO_2 equivalents: $tCO_2e = kWh \times kgCO_2e$ per kWh : 1000, where CO_2e accounts for carbon dioxide and other greenhouse gases, as opposed to CO_2 which only covers dioxide. GPT-3 uses 552.1 t of CO_2 for training its 175 billion parameters, when using the V100 processor. The mere energy consumption is 1287 mWh for its 14.8 days of training. In general, the ecological footprint of machine learning should be reported [305]. Tools for estimating the energy consumption of particular machine learning algorithms have been implemented for regular computing clusters [650].

In contrast to the research efforts regarding deep learning on large data-centers, investigating the energy consumption of small devices has not yet received enough attention. However, as we have shown above, the Internet of Things (IoT) connects billions of small devices and produces extremely large amounts of data. Their energy consumption needs to be particularly small. For an embedded system that is not plugged into the grid, the availability of energy is a critical constraint for its lifetime. Even for an embedded system plugged into the grid, the cost of energy due to increased computing performance can be critical. Unnecessary energy consumption should also be avoided to extend the lifetime of the embedded system. The power awareness and energy efficiency of information processing on devices of the IoT are important for sustainable computing. In this book, we focus primarily on small devices.

1.2.1 Measuring Energy Consumption

Investigating energy efficiency requires measuring energy consumption. Measuring the true energy consumption directly is difficult because of noise sampling and the need to use minimal resources for the sensing itself. The hardware-integrated sensing instruments are often not precise enough for determining the energy consumed by software running on an embedded system. An easy-to-use system for direct energy sensing and an energy model for ARM processors based on linear regression have been developed [105, 106]. Extremely restricted are the ultra-low-power devices that are used in logistics, e.g., devices attached to a container. Based on reliable measurements, the energy harvesting of devices with photovoltaic elements can be realized such that the operating time is enhanced. Section 2.2 of this book shows the PhyNet testbed for energy neutral sensor networks. A batteryless system, its indoor solar harvesting, and energy measurements are presented in Section 2.3. There, even the implementation of a lightweight deep learning algorithm is included. Predicting the power consumption in a communication network is particularly challenging. Section 9.2 presents methods for modeling power consumption of embedded devices for different wireless communication technologies including a machine learning-based method for estimating the transmit power from the available performance indicators, like strength and quality of the received signal.

1.2.2 Different Processors

The energy consumption of different processors varies greatly. Using the example of Quadratic Unconstrained Binary Optimization (QUBO), implemented by an evolutionary algorithm (EA), showed a stable order of magnitude of energy consumption over diverse data-sets and parameter settings [510]. We indicate the numbers here because, in general, such information needs to be given in scientific papers on machine learning. Moreover, the Watt figures found in the QUBO experiments show the typical pattern of magnitudes:

- Field Programmable Gate Arrays (FPGAs), 10^0W,
- CPU (Intel Core i7-9700K)10^1W,
- GPU (Nvidia GEFORCE RTX 2080 Ti)10^2W,
- QA (Quantum Annealer, IBM)10^4W.

Our compute cluster consumes on average 6.25 kWh, or in the order of magnitude of 10^3W. Of course, the particular energy consumption depends also on the number of variables (here: 1024) and the parameters of the EA, e.g. the number of children in each generation. However, the advantage of the FPGA can be estimated already on the basis of these numbers. Since the compute cluster consumes 10^3 times as much energy as does the FPGA, it would need to solve the learning problem in less time—here in the order of 10^{-9} seconds—to use about the same energy, and this is not realistic. Another example studies different implementations of applying a learned Decision Tree (DT) model [110]. The classification is implemented in two different ways. One implements the algorithm as is, the other unfolds the tree into if-else structures, aka compilation. Energy consumption is then measured for FPGA (Xilinx Artix-7 Z-7020 FPGA with 53 200 lookup tables, 106 400 Flip-Flops (FF) in total combined with 4.9 MB block ram and 220 DSP units), and an ARM processor (Cortex-A9 with 666 MHz, 512 MB DDR RAM and 512 kB cache). Each learned tree contains an average of 1349 nodes and roughly 675 different paths from the root node to a leaf node. Throughput is measured as elements per millisecond, energy consumption as nanoJoule per element. The native implementation on FPGA uses 0.008 W or 6.84 nanoJoule per element to classify, on the ARM processor 1.53 W or 105.5 nanoJoule per element to classify. The unfolded tree uses on FPGA 0.068 W or 45.95 nanoJoule per element to classify, and on the ARM processor 1.53 W or 52.76 nanoJoule per element to classify.

This general ranking of the order of energy consumption makes FPGAs attractive for machine learning. In this book, Section 6.1 investigates reconfigurable multilayer perceptron training on FPGAs and compares it with the PyTorch implementation. Furthermore, Section 6.1 explores FPGA implementation of a Multilayer Perceptron (MLP), a fundamental neural network structure for machine learning. FPGAs are especially advantageous for deep learning because they support customized data types, where GPUs support only a limited number of data types.

1.2.3 Reduced Run-Time and Real-time Processing

Many approaches to reducing the energy consumption of machine learning reduce the run-time of a learning method. We take into account the execution of learning programs and learned models not only regarding time complexity, but also regarding real-time ability. Since many embedded systems are integrated into large products that interact with the physical world, timeliness is an important issue. If the results are delivered too late, they may have become useless. The computing of summaries “on the fly”, as presented in Section 3.1, is designed to save memory and energy. In general, algorithms for data streams require fast computing and memory reduction, as we discuss below.

The link between energy and memory reduction also becomes clear in the approach to graph deep learning in Section 4.3, where a general message passing is scaled up for arbitrarily large graphs. The remarkable speed-up of clustering run-time as demonstrated in Section 5.1 certainly saves energy as well. Exploiting parallelism, even for non-uniform workloads, is the key in Section 6.2 to reducing the run-time and increasing the data throughput for database query execution. Section 6.3 describes extreme multicore computation, which exploits the independence of training for several thousand labels. It trains each class versus all others using thousands of cores, each one learning to predict one of the many classes. Along with the number of cores, the hardware-aware parallel training solvers speed up until a saturation is reached and the speedup scales only sublinearly.

Reducing run-time through an adaptive scheduling brings together the multi-core computing architecture and machine learning. Section 6.4 examines the optimization of the execution of diverse machine learning algorithms for parallel execution on a multi-core architecture. The optimization itself also uses machine learning, namely, the Bayesian model-based optimization . The Resource-aware Model Based Optimization (RAMBO) framework saves energy through the run-time reduction.

1.2.4 Minimizing Energy Consumption of Machine Learning Processes

If minimizing the energy of machine learning processes builds upon the analysis of the algorithms, statistical guarantees can be given. Exponential families are a model of learning that covers many learning tasks, e.g., the estimation of probability density as it is used by, say, topic models, or the prediction of the maximally likely state as it is used by naive Bayes or conditional random fields. A careful analysis of learning models may lead to running of very complex machine learning tasks on very limited and even ultra-low energy devices. This book offers such an approach in Section 9.1, which describes the Integer Markov Random Fields (IntMRF) along with their theoretical foundations. Note that it is the underlying model class that is restricted to the integers; it is not just a restriction of the state space to integers. Here, the state space may be a random discrete space without any additional constraints. The reduced run-time and energy savings are due to the cheaper operations. The novel bit-length propagation algorithm (BL-Prop) allows computing using integers only, i.e. real numbers are not quantized afterwards, but all the learning processing uses only integers. In addition to previous work ([567]), Section 9.1 introduces the novel numerical optimization method IntGD for convex objective functions. It is based on an accelerated proximal algorithm for non-smooth and non-convex penalty terms. For integer gradients computed via BL-Prop, IntGD is guaranteed to deliver a pure integer learning procedure in which the final parameter vector as well as all intermediate results are integers. Integer Markov random fields are almost as expressible as real-valued ones are, but can be executed on an ultra-low-power device that does not offer floating-point operations [570].

As we have seen, there are multiple ways to reduce the energy consumption of machine learning: developing algorithms for more energy efficient processors (FPGAs), tailoring machine learning algorithms, optimizing their execution for a reduced run-time, and even developing novel learning algorithms designed to save energy.

1.3 Memory Demands of Machine Learning

1.3.1 Deep Learning

Deep learning challenges the GPU memory due to its many hyperparameters, tensor alignment, particular convolution algorithms, and operator scheduling. In a detailed analysis of 4960 failed deep learning runs, Yanjie Gao and colleagues found 8.8 % of them were caused by the exhaustion of GPU memory [242]. They then developed an estimate for the GPU memory needs of deep learning models. In this book, the memory demands of Graph Neural Networks (GNNs) are part of the work that is presented in Sections 4.2 and 4.3. The usual mini-batch training becomes difficult in GNNs because of the interdependency of neighboring nodes. The exponential growth of the graphs has been shown in [455], which proposes sampling of edges. A more general solution for diverse GNN architecture is presented in Section 4.3. The novel GNN AutoScale framework of message passing succeeds in making GNN applicable even in a streaming setting, since for a single epoch and layer, each edge is processed just once.

The quantization of deep learning results in binary values of weights and activations, reducing the memory consumption drastically [325]. Binarized Neural Networks (BNN) offer more lightweight processing. Combining machine learning and computer architecture work has led to BNN on FPGAs for fast inference on very large streaming data from astroparticle physics [112]. A further step towards the close interplay of algorithms and hardware is to take into account modern memory technologies. Again, we see the close relationship between energy consumption and memory architecture in the case of approximate or non-volatile memories that reduce the energy consumption but increase the bit error rate. For BNNs, bit flips in the weights or the activation values of the network decrease the accuracy of the model. How many bit errors can be tolerated at the hidden layers? The idea of max margin optimization, developed for Support Vector Machines (SVM) [680], inspired a formulation of a bit error tolerance metric that could be inserted into the BNN training [113]. Machine learning anticipates hardware errors and thus produces a robust learned model for the energy-saving computing architecture. Section 7.2 explains this approach of reducing the bit error rate within the training of a BNN in more detail.

1.3.2 Summaries and Clustering

Data summary or aggregation is necessary in order to learn from distributed sensor streams. Sketching or sampling has been theoretically investigated for clustering data streams [70, 103]. Coresets and sketches summarize data such that they can be analyzed by any learning algorithm and they can deliver approximately the same result as would result from training on the full dataset [516]. Section 3.2 analyzes coresets and sketches for distributed and streaming data. The analysis covers approaches to Bayesian and generalized linear regression. A sparse subspace of the original high-dimensional data space is proven to be sample-efficient. The data reduction saves not only memory but also run-time and energy demand.

Summaries with a fixed memory size are often developed using submodular functions. For video summarization, a submodular set function could be optimized subject to privacy constraints [495]. In 3.1, sieve streaming with fixed-size memory is enhanced for sampling the most informative observations “on the fly”. In addition to saving resources, the novel ThreeSieves algorithm offers summaries for human interactive data exploration.

Unsupervised learning partitions data in many different ways. This book presents the clustering of graph data in Section 5.1 and of curves in Section 5.2. The scalability of hierarchical agglomerative clustering is considerably enhanced by the BETULA algorithm in Section 5.3.

Some problems occur as building blocks of learning algorithms. Matrix factorization is one of them. An approach of Binary and Boolean matrix factorization that is robust with respect to noise is presented in Section 5.4. It uses proximal gradient descent optimization and allows overlapping clusters.

Another one is the max dicut problem: partitioning of a directed graph into two subsets such that the sum of the edge-weights between the two subsets is maximized. Section 4.4 investigates this problem for parallel algorithms, that scale for very large graphs.

1.3.3 Executing Machine Learning

On the level of programming languages and operating systems, smart resource utilization reduces the memory footprint [388]. Moreover, the dynamic sharing of memory can be optimized [596]. Section 7.1 presents a memory management layer between the R interpreter and the operating system that reduces the memory footprint by allocating memory only to pages in the memory that are required.

Decision trees (DTs), although one of the earliest machine learning algorithms, still pose research challenges. Training several thousand DTs leads to millions of decision nodes that must be stored in memory and processed in order to apply the learned model to new data. Hence, inferences using DT ensembles demand a smart memory layout. Cache memory moderates between the main memory and the processor. Preventing cache misses requires a well-designed memory layout. Section 7.3 offers an implementation that optimizes the memory layout while preserving the original ensembles' accuracy. A code generator automatically adapts to underlying architectures.

1.3.4 Regularization and Reparametrization

Regarding models of learning, the reduction of memory demand has been investigated for the exponential families. The memory consumption of Markov Random Fields (MRFs) is dominated by the size of its parameter vector. Since each parameter is usually accessed multiple times during inference, they should be stored in a cache memory. The key to compression is regularization and reparametrization, which exploit redundancies in the true parameters. The general idea can be applied to discrete Markov random fields and to multivariate Gaussian models [575]. Section 4.1 presents spatio-temporal random fields. They model spatial networks as graphs and connected layers of these graphs as temporal relations. A piecewise linear reparametrization of the parameters of a clique (a part of the graph) is weighted by a decay vector, and the full model is weighted by a corresponding decay matrix. In spatio-temporal random fields, it is assumed that value changes at nodes do not change in sudden jumps over time. The reparametrization of spatio-temporal random fields based in this assumption is proven to be universal, i.e. it is a bijection.

1.4 Structure of this Book

The book covers contributions from machine learning and embedded systems and includes, in addition, algorithmic and database research that supports the overall goal of resource-constrained data analysis. Its structure follows that of the workflow. It starts with data of different kinds. Then it moves to executing machine learning and the particular resource constraints, namely memory, communication, and energy. Each chapter offers an introductory summary of its sections.
The book is organized as follows:

- This book starts in Chapter 2 with the *data collection* of embedded system deployments. Section 2.1 presents the system kCQL for collecting complex operating system data. kCQL acquires and combines event streams and system states of operating systems while maintaining low overheads. A physical sensor network testbed is presented in Section 2.2 that can be used for large-scale energy accounting, position tracking, application testing, and system data collections. Modeling, analysis, calibration, and evaluation of batteryless in-door energy harvesting systems are presented in Section 2.3.
- Chapter 3 considers *data streams* in resource-constrained embedded systems. Regarding summary extraction from streams, Section 3.1 presents an insertion-only data stream learning algorithm based on maximizing submodular functions. Section 3.2 covers a brief technical introduction to coresets and sketches and highlights their importance for the design of data stream algorithms.
- Chapter 4 presents methods and techniques to learning *models for structured data* with resource-awareness. In Section 4.1, a probabilistic learning model of spatio-temporal random fields is introduced, which reduces memory consumption without loss of the accuracy through universal reparameterization. In Section 4.2, the Weisfeiler-Leman algorithm is connected to learning methods such as graph kernels and Graph Neural Networks (GNNs). In Section 4.3, a unified and scalable framework for message passing in GNNs is proposed. Section 4.4 presents algorithms to compute cuts in directed graphs with high quality, which scales well in shared memory. Section 4.5 presents a new technique based on GNNs to search for scientific papers, utilizing key mathematical formulas instead of key words.
- Chapter 5 considers *clustering* in four complementary sections. Section 5.1 exploits the sparseness of data for reducing memory requirements and run-time. Section 5.2 handles sequences of points using the Fréchet distance and details an approximation for their clustering. Section 5.3 characterizes methods for hierarchical clustering that are well suited for streaming data processing on edge devices with limited

resources. Section 5.4 offers a novel optimization subject to binary constraints for matrix factorization, a method that is entailed in many learning algorithms.

- Chapter 6 deals with the heterogeneity of *execution platforms* of embedded systems. Section 6.1 presents the acceleration of learning neural networks on Field-Programmable Gate Arrays (FPGAs). Parallel executions utilizing graphics processors (GPU) for efficient database query processing and multicore systems for accelerating extreme multi-label classification are presented in Section 6.2 and Section 6.3, respectively. The RAMBO framework that can efficiently optimize machine learning models on heterogeneous distributed systems is discussed in Section 6.4.
- Chapter 7 presents optimizations of machine learning algorithms with respect to *memory*. Section 7.1 demonstrates that the memory footprint can be effectively reduced by leveraging application-specific knowledge. Section 7.3 proactively optimizes the memory layout in the implementation of the machine learning model to favor the underlying cache memories with a probabilistic perspective. Furthermore, Section 7.2 presents how learning models can accurately process in environments with unreliable memories if we take bit errors into account during machine learning training.
- Chapter 8 considers embedded systems under *communication* constraints. It covers synchronization with resource sharing, communication with potential failures, and probabilistic timing information in Section 8.1, and discusses bandwidth limitations of different execution models and coprocessor-accelerated optimization in Section 8.2.
- Chapter 9 is about *energy efficiency*. Section 9.1 shows how to reduce the power consumption of complex learning models such as Markov random fields through integer-only operations. The novel Bit-Length Propagation (BL-Prop) and integer gradient descent (IntGD) algorithms can be executed even on ultra-low-power (ULP) micro-controllers. Section 9.2 uses machine learning in order to estimate the power consumption of diverse communication technologies in wireless systems unleashed from the power grid and light-weighted small wearables.

Each chapter and section is self-contained. You may select the chapter or section you want to read by topic or by data flow of data analysis processes. You may want to read an overall chapter or just some sections. Because we have written the book with teaching in mind you can select a number of sections for specialized courses. Of course, we also encourage readers seeking an in-depth understanding of the resource-efficient combination of hardware and machine learning algorithms to read the entire book!

2 Data Gathering and Resource Measuring

This book starts with chapters ordered in analogy to the data analysis workflow before it investigates particular resources. Data is the raw material for all machine learning applications. Hence, gathering data is the first step. Where in the beginning of machine learning, tables and later on databases were the only source of data, nowadays petabytes of data is produced by a huge variety of embedded systems. However, collecting the data of embedded system deployments such as *wireless sensor networks*, *Industry 4.0* and *Internet of Things* environments has several constraints due to their strict resource limitations. This chapter discusses approaches and tools to handle data collecting in embedded systems.

First, a framework for collection of complex operating system data is presented with kCQL. Based on an extensible data model, kCQL's declarative database-like queries can acquire and combine event streams and system states while maintaining low overheads. This simplifies the development of complex analyses.

Second, PhyNetLab, a large-scale physical sensor network testbed, is presented. PhyNetLab supports the acquisition of data from real-world embedded system deployments. Aiming at mobile Industry 4.0 applications, it enables energy consumption accounting, position tracking, application testing, and system data collections on a large scale.

Third, batteryless systems are investigated in the guest contribution by Andres Gomez. He presents an indoor solar harvesting dataset that supports the modeling, analysis, calibration, and evaluation of energy harvesting systems. Moreover, hand gesture detection using a SmartCard exploits a lightweight Deep Neural Network (DNN).

https://doi.org/10.1515/9783110785944-002

2.1 Declarative Stream-based Acquisition and Processing of OS Data with kCQL

Christoph Borchert
Jochen Streicher
Alexander Lochmann
Olaf Spinczyk

Abstract: Logging and debugging facilities of computer operating systems as well as subsystem-specific tools do not provide sufficient information and cannot cope with the volume and frequency required for data acquisition within the operating system. This has led to several highly versatile dynamic operation system kernel instrumentation frameworks, such as SystemTap and DTrace. These frameworks minimize the performance impact on normal operation and allow complex analyses. However, such event-based analyses need to be programmed in a complicated imperative manner at a rather low level of abstraction. Conversely, a more recent framework, PiCO QL, offers a declarative, and thus more powerful, database-like interface to the kernel state. However, it is not able to trace events. We present kCQL, an approach that aims at providing the best of both worlds. Based on an extendable data model, declarative database-like queries can acquire and combine event streams and system states. This simplifies the development of complex data analyses. At the same time, a common data model and architecture provide the optimization of query execution and the reuse of common subexpressions of different queries. The approach has numerous practical applications, which are discussed at the end of the section.

2.1.1 Introduction

Computer operating systems readily expose a vast array of information, internal state, event logs, and even basic statistics on events and resource utilization to be inspected by developers and system administrators. Usually, this goes along with a set of tools to further process and interpret the data. A well-known example for such a system data interface is *procfs*, which can be found in many Unix-like operating systems.

The utilization of those interfaces certainly imposes some impact on the normal operation. However, while access to state has an effect only when it actually takes place, continuously tracing events or function calls causes a constant overhead, even if the generated data goes unused. Thus, it is restricted and not all interesting data is accessible that way.

As a remedy, there are dynamic instrumentation or tracing frameworks and *Operating-Systems Data Acquisition Frameworks* (OSDAFs) such as *SystemTap* [196] and

DTrace [119] that can retrieve more information at a higher level of abstraction, or even, partially, in a declarative way.

The more recent *PiCO QL [233]* enables (non-modifying) SQL queries over a relational representation of the kernel state, something which is not possible using existing event-based data acquisition tools. However, PiCO QL does not allow the tracing of events.

2.1.2 Operating-System Data Acquisition Frameworks

A multitude of methods and tools was devised to extract data from the operating system without repeated manual instrumentation and recompilation. For example, LTTng [172] and ftrace enable the static activation and deactivation of performance-critical instrumentation. However, these frameworks are inflexible with regard to the data they acquire. For example, the function tracer ftrace covers only function calls, the respective function arguments, and context information such as the process identifier.

Generic instrumentation frameworks such as kprobes [400] allow on-demand tracing of almost any instruction in the Linux kernel, but are tedious to use and potentially dangerous, because they allow arbitrary modification of the data structures. Unlike kprobes, *ExtOS* [42] and *AnyCall* [248] focus on the safe execution of user-level code within the OS kernel and thereby facilitate *Near Data Processing* [41]. However, both approaches use imperatively written code.

Ideally, OSDAFs provide a high-level view and languages to define instrumentation, data collection and possible on-site processing. For simplicity, we refer to all these definitions as “queries”, even if they are written imperatively. Generic OSDAFs have to provide access to potentially any part of the operating-system (OS) without the need for recompilation, and should impose only a minimal overhead at runtime. The following concepts and processing steps are common in existing OSDAFs:

- *Events* are the primary source of information. The starting point is usually low-level events, i.e. points in the operating system’s control flow. It is possible to trace function calls and some frameworks also allow the tracing of single instructions. High-level events abstract from the low-level control-flow events. For example, they may indicate the reception of a network packet or the invocation of a system service. Events can have associated context data. For function boundaries, those are the calling parameters and the return value of the respective function. Accordingly, high-level events offer higher-level context data, such as the destination address of a network packet, which may not be readily available as a function parameter. The provision of events that can be used in a data acquisition task is part of the framework or an extendable library that belongs to it. Some frameworks also allow for adding new high-level events based on other high-level events.
- *OS State* is any part of the operating system’s state. Low-level state directly refers to the contents of variables and data structures, whereas high-level state provides

some interpretation, for example, direct access to the name of a process issuing an operating-system call. The provision of state that can be used in a data acquisition task is analogous to the provision of events.

- *Streams* consist of the data flow that is generated from events and their associated context data. For example, tracing all system calls regarding the I/O system may already suffice for some tasks. For other tasks, further processing or combination with state is necessary, such as when supplying the name of the calling process to a trace of system calls.

2.1.2.1 Analysis of Existing Frameworks

In the first three event-driven OSDAFs listed below, queries usually consist of a set of probes. A probe handles the data generation from events (or probe points). It consists of a declarative specification of the events to probe and a probe body that generates the data.

SystemTap A SystemTap [196] probe body is an imperatively written piece of code that processes the event data, combines it with OS state, and generates output via printf statements. It is written in a C-like language and is compiled to actual C code with additional safety checks, which uses kprobes [400] to instrument the kernel code. Besides probes, SystemTap also allows user-defined functions and global state. It can also contain plain but unsafe C code. The set of traceable events and accessible states is extendable. Most of it is part of a library (the tapsets) that is written in the same language as the scripts.

DTrace DTrace [119] queries are written in the C-like and restricted (no cycles in the CFG) D language. D programs solely consist of a set of probes that are compiled to bytecode for execution in a virtual machine. Access to kernel state is possible via built-in variables. Event and state provision is the responsibility of providers, which decouple the details of data provision from the queries. While D is an imperative language, associative arrays and aggregation functions allow for semi-declarative one-line queries.

Fay There are also frameworks that allow for a completely declarative specification of data retrieval. A declarative specification allows the transformation and optimization of queries for performance. Fay [202] enables tracing for clusters of Windows machines. The *Hotpatching* mechanism serves as a hook for probes, which are responsible for data collection. Fay can either be scripted from the Windows *PowerShell* or via a declarative interface that allows writing SQL queries on possible trace points, which are then translated to a set of probes and distributed processing across multiple hosts. The queries are formulated as relational operations on the already existing probe output. Fay does not allow the combination of event-based data with context information as a language mechanism. Rather, the probes alone are responsible for collecting all relevant state information in the instrumentation code and providing it as event context data.

PiCO QL PiCO QL [233] does not deal with events at all. Instead, it provides a relational interface to the kernel data structures that can be inspected via SQL queries. Although there is a wide range of applications for this kind of interface, queries that are based on events such as incoming network packets cannot be answered this way. Thus, timely data acquisition requires high-frequency polling. Its authors call it a complementary approach to existing event-based systems that do not allow model-based declarative access to internal data structures.

2.1.2.2 Comparison

DTrace offers providers that implicitly extend the model of available data and are not bounded to any specific way of low-level data provision. SystemsTap is also extendable in that it provides high-level state and events based on existing high-level abstractions. By contrast, Fay offers declarative queries on event streams that can be optimized automatically. PiCO QL is complementary as it represents state in a relational data model that can be queried via SQL.

Our goal is to integrate event streams and state into a common model and provide language mechanisms that support queries that have the combined expressiveness of PiCO QL and Fay and the extensibility of DTrace and SystemTap without performance loss. This would offer the best of all models in one framework.

2.1.3 kCQL: A Relational Streaming Interface for OS Data

A relational interface offers an expressive and powerful way to access kernel data as described by the work on PiCO QL. Using that as a basis, we show how streams are integrated into a relational data model and our query language *kCQL*.

2.1.3.1 Data Model

A high-level language that acquires and accesses OS data works on some kind of model of the available events, their context data, and the available kernel state. This model can be implicitly given, such as by the probe definitions in the tapsets of SystemTap. DTrace even allows the definition of complex data types for probe arguments (i.e., context data for events). The entirety of traceable functions and their arguments as well as raw kernel data structures also contribute to the model.

In a relational model, both relations and streams contain tuples with a fixed set of attributes. Figure 2.1 shows a possible relational representation of a subset of the kernel data structures and event stream. The process and socket IDs act as primary keys in the

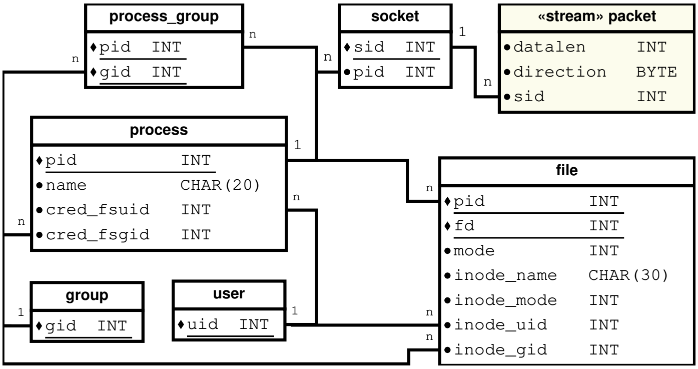


Fig. 2.1: (Non-exhaustive) relational representation of kernel state and event streams.

process and socket relations, and are used as foreign keys in the socket relation and the packet stream.

Thus, the packet stream looks like a relational database table definition. Its columns represent the event's context data. The difference between relations and streams is that tuples can be removed from the former, but not from the latter. Streams are *monotonous* and *infinite*. Conceptually, they can be regarded as an ever-growing relation.

2.1.3.2 Relational Stream Query Languages

For data acquisition, we do not consider queries that modify relations or streams. Thus, the incorporation of streams into our relational model as seen above allows us to treat them almost like a database table. For example, the following operations from the relational algebra could be executed on every tuple of one stream to produce another stream:

- projection (e.g., `SELECT datalen FROM packet`)
- filtering (e.g., `SELECT * FROM packet WHERE datalen > 100`)
- joining to relations (e.g., `SELECT pid, datalen FROM packet`
  `JOIN socket ON packet.sid = socket.sid`)
- union of streams (with the same schema)

Joining two streams S1 and S2 (as opposed to joining a stream to a relation) can be defined based on their conceptual view as a relation without deletions. Each tuple from S1 is joined to each tuple that S2 contains (or "has produced") so far and vice versa [712]. An application scenario is a trace of read system calls, extended by the information whether they actually triggered disk I/O, which is also an event stream. However, there is no need to permanently store all read system calls that ever happened, but only a limited *window* of these. Such a window (see next below) is a time-varying relation that can be joined with the stream of I/O events.

2.1.3.3 CQL

Our work builds upon the *Continuous Query Language* (CQL) [20] from Stanford. CQL closely resembles standard SQL. In contrast to other stream query languages, such as Aurora [1], it does not contain direct stream-to-stream operators (e.g., filtering or projecting a stream). Streams have to be converted to a relation before they can be processed by relational operators. Consequently, CQL augments SQL by four operators to convert between streams and relations:

- *Windows* generate a time-varying relation that contains all stream tuples that belong to the current window. The window size can be specified by the number of tuples it contains (**row** window) or by the maximum timestamp difference between the oldest and the newest tuple in it (**range** window). A **slide** parameter (tuples or duration, respectively) determines the size of the steps to move the window down the stream.
- *ISTREAM* generates a stream from a relation that contains every tuple (or, in other words, rows) *inserted* into that relation.
- *DSTREAM* generates a stream for *deleted* tuples.
- *RSTREAM* generates a stream from a relation. Every time the relation changes, all of its tuples are inserted into the stream.

The restriction of all other operations to relations seems like a disadvantage that makes an efficient implementation of a CQL-based *data stream management system* (DSMS) impossible. However, these restrictions only apply to the language level, whereas the internal query processing may look completely different.

2.1.3.4 Examples

This section presents a few queries and their output.

Stream Queries Packet logging tools, such as *tcpdump*, usually trace packets without assigning them to the involved process. Nevertheless, this is possible by looking at the socket numbers of the transport protocol and by using other tools, such as *lsof*, to find the processes using these sockets. However, it is more convenient to do this in one step. The query is shown in Listing 2.1. As we cannot directly operate on the packet stream, we use a window to transform it into a time-varying relation. The relation generated by a special form of a window, the now window, is rather peculiar, as the total time it contains anything is zero. The tuples from the packet stream are inserted into the window, and then deleted from it directly thereafter. The relation resulting from joining the window on the *socket* and *process* relations also behaves in the same manner. Using the *RSTREAM* operator, a stream is then generated from these tuples. The result of this query is a continuous flow of tuples as shown in Table 2.1.

Listing 2.1: *Packets*: Assigns network packets to processes.

```
Packets: RSTREAM (
 SELECT packet.datalen AS len,
  packet.direction AS dir,
  process.pid AS pid,
  process.name AS pname
 FROM packet [now], socket,
  process
 WHERE packet.sid = socket.sid
  AND socket.pid = process.pid
);
```

Tab. 2.1: Example output of the *Packets* query (Listing 2.1).

Timestamp	len	dir	pid	pname
14...546486	1474	>	3728	nc
14...551490	32	<	3728	nc
14...556010	114	>	3378	sshd
14...559973	70	>	3736	wget
...	...	...	...	...

Listing 2.2: PacketAggr: Sums up outbound network traffic in packets and bytes for each process in 5-minute intervals.

```
PacketAggr: RSTREAM (
  SELECT pname, COUNT(*), SUM(len)
  FROM Packets [RANGE 5 minutes SLIDE 5 minutes]
  WHERE dir = '>' GROUP BY pid
);
```

We can use that as a basis for aggregations and summaries. For example, Listing 2.2 shows a query that gathers accumulated outbound network traffic (bytes and packets) in 5 minute intervals, using the Packets query as a data stream source.

Continuous Queries In some cases, we do not want a stream but actually a relation as an output where tuples can also be deleted. For example,

```
SELECT pid, name FROM process;
```

looks like a relational snapshot of the process list, but it is a continuous query in CQL. Consequently, its output contains insertions and deletions as shown in Table 2.2. In addition, the initial state of the query is captured (tuples with fictional timestamp 1, as we cannot know the real time).

Access to Other Address Spaces To analyze performance issues of specific applications, kernel data alone is not sufficient. If applications provide respective data sources, they can be combined with kernel data. For example, the query *ApacheIO* in Listing 2.3

Tab. 2.2: Example output of "`SELECT pid, name FROM process;`".

Timestamp	Insertion/Deletion	pid	name	(Comment)
1	+	1	init	initial relation
1	+	2	kthreadd	
...	...	...	...	
1	+	4260	man	
1	+	4281	kcql.elf	
1439915918621953832	-	4260	man	exiting processes
1439915923580930234	-	3679	bash	
1439915928436850678	+	4285	screen	starting process
1439915928439601848	-	4285	screen	screen invokes exec syscall
1439915928439601848	+	4285	bash	
...	...	...	...	

Listing 2.3: *ApacheIO*: Output I/O load summary per file served by Apache.

```
ApacheIO:
SELECT Apache_HttpReqs.file, Apache_HttpReqs.ipSrc, COUNT(IO.duration),
 AVG(IO.duration) AS avgduration
FROM Apache_HttpReqs, IO
WHERE Apache_HttpReqs.pid = IO.pid GROUP BY Apache_HttpReqs.file;
```

summarizes the number and average duration of disk operations per file served by the Apache web server.

Multiple Instances of Data Sources The relational model is easily extendable to multiple instances of a data source, such as multiple network interfaces. From the modeling perspective, this is just an additional identifying column in the respective relation or stream. Thus, the Packets query (Listing 2.1) and the subsequent aggregation (Listing 2.2) still work and could be even extended by aggregation per network interface.

2.1.4 Implementation

An overview of kCQL's architecture is given in Figure 2.2. We differentiate between the clients and the kCQL core. Clients submit queries to the core and receive data continuously until they explicitly revoke the query. The core generates a query execution plan and processes the data according to the running queries. The necessary data sources, however, are also provided by (other) clients. In this respect, clients are comparable to the providers of DTrace. As clients can run in the kernel space and in any user process, data has to be transported across address spaces. Instead of moving data to one central

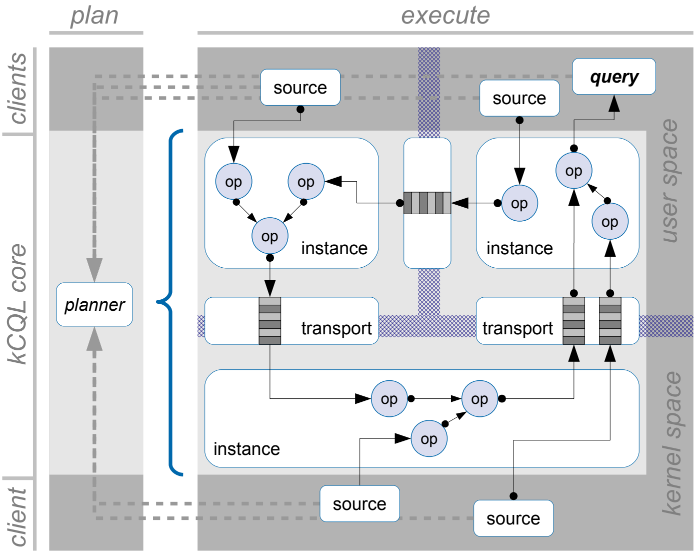


Fig. 2.2: Architecture of kCQL

location for processing and then distributing the query results to the clients, the query plan is partitioned in a way that tries to minimize data flow between address spaces. Thus, each address space has its own instance of the DSMS engine, each processing a different part of the query.

2.1.4.1 DSMS Engine

The heart of kCQL, its DSMS engine, is based on Stanford's data stream management system STREAM [21]. It is responsible for query plan generation and for query execution. STREAM was built for pulling data from synchronous data sources that generate new tuples on demand. In contrast, kCQL works with asynchronous OS events. The necessary modifications are described in Section 2.1.4.2.

Time-varying relations are represented as update streams, containing tuples annotated with a timestamp and a tag for insertion or deletion. This does not only apply to the query output (as shown in Table 2.2), but also to the query input (data sources) and intermediate relations generated by relational operators (e.g., joins). As a consequence, relations based on actual operating system data, such as the process table, also have to be transformed into such an update stream.

Query Plan Generation After parsing the queries and the data source descriptions, STREAM generates a directed acyclic data-flow graph of relational operators, from the data sources to the query outputs. At this point, we introduce a step that partitions the graph into the participating address spaces. After that, auxiliary structures are added to the query plan, which is then instantiated for execution. The engine instances contain the following elements:

- *Synopses* provide a view on all tuples that are currently relevant for an operator or contained in a relation. They also maintain indices to efficiently scan them using predicates (e.g., to find join partners).
- *Operators* perform relational operations or conversions between streams and relations. For some operators, such as joins, the streaming representation of relations is not sufficient, as they need a view on all tuples that are contained in a relation at the time of the currently processed input tuple. They do so by means of a *synopsis*. *Source operators* and output operators interface between the client and the engine instance. For kCQL, we add senders and receivers to transport streams and relations across address spaces.
- *Queues* buffer the tuple streams between operators. The queue elements contain the tuple's timestamp and, for relations, the type (insertion or deletion). To avoid copying the tuple contents, the queue elements do not contain, but rather point to, their location in a shared store.
- *Stores* contain the actual tuple contents and are shared between communicating operators. They are used by operators and their synopses. The tuple contents are extended by meta-information such as reference counting.

Besides basic optimizations of the query plan, such as merging, filtering, and projection into other operators, STREAM also replaces relational operators by pure stream operators where appropriate. For the *Packets* query in Listing 2.1, STREAM does not produce an actual window from the packets stream rather, it directly joins the stream tuples on the relations. The same applies to filtering and projection.

Consistency by Temporal Monotonicity If we join a tuple from a stream (e.g., the packets) on a relation (e.g., the process list), the join partner (e.g., the process) might not exist anymore in the actual OS data structure (e.g., because the process might have already been terminated). Using STREAM, that does not lead to inconsistencies, because operators dequeue and process element by element in timestamp order. That means that a join always dequeues the element from the stream and joins it on the tuples in its synopsis. Only after that does it process the deletion in the relation and updates its synopsis. This way, consistency is ensured also for all other operators, including pure relational joins and windows.

However, that requires all queues to be ordered with respect to timestamps: an operator must not enqueue a tuple with timestamp t_0 after it enqueued a tuple with a timestamp $t_1 > t_0$. For processing operators, that means that the timestamp of an output tuple is always the maximum of the timestamps of the tuples it is based on. For a full join of relations R and S this means: if a tuple x with timestamp t is inserted into

R, then the generated output tuples, namely the results of joining $\{x\}$ with S, have the timestamp t .

Query Execution In STREAM, a single thread schedules operators in a round-robin fashion, which is, in our adaptation, one thread per address space. The time slices are given as a total maximum number of elements that can be processed from the input queues.
An operator is blocked if its input queues are empty, when its output queue is full, or when it encounters a temporal monotonicity stall. The latter condition can only occur with multiple inputs. For operators with one input (e.g., filtering or windowing), preserving temporal monotonicity is straightforward: in each step, take the next element from the queue, remember its timestamp, and process it. If that leads to the production of output elements, they all get the memorized timestamp.

Operators with multiple inputs have to determine the queue whose head element has the oldest timestamp, and then proceed with that element like a single-input operator. If one of the queues is empty, the operator cannot take the oldest element from the non-empty queues, because the respective predecessor upstream still might enqueue an element with an even older timestamp into the empty queue.

2.1.4.2 Enabling Asynchronous Events

Besides distributing operators across address spaces, we adapt the DSMS engine to asynchronous OS event streams.

Asynchronous Sources Usually, the source operators pull a tuple from their associated sources in each execution step. As this does not make sense with events that occur asynchronously, the data sources in kCQL write the tuples into a buffer, and the respective source operators read from that. At the start of query execution, all sources that deliver relations have to dump the complete relation. For a relation on all OS processes, this means iterating the whole process list and delivering every process as an insertion element, as seen in Table 2.2. Hereafter, it has to generate new tuples when the relation has to be updated. We use kprobes for the instrumentation of events that generate streams and updates to relations.

Scheduling The scheduling thread in STREAM runs continuously until a given number of tuples is processed or it is stopped explicitly. This works fine as long as data sources deliver a new element every time they are asked for it. The buffers of our asynchronous data sources, however, can run empty. In that case, the continuously running scheduler would waste CPU time.

Thus, our modified scheduler only runs operators that actually have work to do (when they are not blocked, as explained in Section 2.1.4.1). If there is no such operator, the scheduler is suspended. It can be resumed by asynchronous sources after they have produced a tuple, and by the transport when it receives an announcement from another address space.

Breaking Temporal Monotonicity Stalls Temporal monotonicity stalls do not prevent other upstream operators from further filling the non-empty queues. As the queues are bounded, stalls propagate upstream to the data sources. In contrast to synchronous operation, we cannot stop pulling data from these sources until the stall is cleared, as we would miss events.

Thus, if we have at least one non-empty input queue, but are required to take the next element from an empty queue, we try to find the oldest possible timestamp an element enqueued into this empty queue could have. After that, we re-evaluate the monotonicity condition with that timestamp.

To find the oldest possible timestamp, the operator asks the respective upstream input operator. Every operator implements that method, and it slightly differs depending on the operator type:

- *Sources* return the timestamp from the head element in the ring buffer. If that buffer is empty, they return the value of a synchronized variable (per address space) that always contains the most recent timestamp of any newly produced data.
- *Single-input operators* return the timestamp of the input queue head. If the queue is empty, they ask their upstream input operator.
- *Multiple-input operators* return the oldest timestamp of all queue heads. If *all* queues are empty, they ask all their upstream input operators, and return the minimum value.
- *Receivers* return the timestamp of the head element in the transport. If there is none, the sender cannot directly return anything that would break the stall. However, it prompts the respective sender in the peer address space to send an empty element that just contains the required timestamp. Thus, the stall can be broken in the next round.

2.1.4.3 Cross-Address-Space Transport

Queue elements need ordered stream-based, but not necessarily synchronous interprocess communication. Using one of the existing synchronous inter-process communication mechanisms would either require a system call for every tuple, or manually implemented buffers on each side. Thus, we decided to use ring buffers in shared memory segments. We use synchronous communication (*procfs* for kernel–user, message queues for user–user) only to wake up sleeping senders and receivers: receivers sleep when the ring buffer is empty, and the scheduling thread sleeps if no other operator has work to do. When the corresponding sender writes new elements into the ring buffer, it signals the scheduler to wake up the receiver. The same works vice versa for a sender that goes to sleep because of a full transport ring buffer. To avoid shared and synchronized stores between different address spaces, we also write the tuples (additionally to timestamp and type) directly into the ring buffer.

2.1.5 Evaluation

To evaluate kCQL, we examine both the overall runtime overhead and the synchronous delay that is imposed by diverting the kernel's control flow to event processing before it can resume normal operation. We also measure both quantities for SystemTap and PiCO QL for comparison.

Our evaluation platform is a desktop computer with an Intel Core i5-3570 processor and Ubuntu Server 14.04. The clock frequencies of the four cores are fixed at 3.4 GHz each. We use the Vanilla Linux kernel 3.14.17 for our implementation, configured with Ubuntu's generic configuration. We perform the evaluation under the following loads: SysBench's [382] prime number calculation (CPU and user mode only) and a full Linux kernel build (*x86_64-defconfig*, CPU and I/O activity).

2.1.5.1 Queries

We use two queries for a quantitative evaluation of our approach. Both were implemented as SystemTap scripts, which resulted in considerably more lines of code. For the Packets query, we implemented two versions: one that closely resembles kCQL's mode of operation, using an incrementally updated copy of the process list (182 lines of code), and one that directly accesses the process list whenever a packet is received (159 lines of code). The SystemTap implementation of the *Files* query consists of 46 lines of code.

Assigning Network Packets to Processes ("Packets") The first query has already been shown in Listing 2.1 in Section 2.1.3.4. For the quantitative evaluation, a second machine was connected to the machine under test with a direct gigabit Ethernet connection, transmitting TCP packets at full speed.

Finding Files Currently Opened with Insufficient Permissions ("Files") The second query gathers files opened for reading by processes that do not have the necessary access permissions. This query solely works on relations, which are, however, processed continuously, immediately delivering a tuple as soon as the aforementioned case occurs. The query is shown in Listing 2.4.

2.1.5.2 Overall Runtime Overhead

To quantify the overall runtime overhead imposed by data acquisition, we measured the time SysBench and the kernel build took without data acquisition, as well as with different queries using kCQL, SystemTap, and PiCO QL.

Listing 2.4: *Files*: A continuously updated relation containing files opened with currently insufficient permissions.

```
Files:
SELECT DISTINCT P.name, F.inode_name, F.inode_mode & 0400,
  F.inode_mode & 040, F.inode_mode & 4
FROM process AS P, file AS F, process_group AS PG
WHERE P.pid = F.pid AND P.pid = PG.pid AND F.mode & 1 = 1
  AND (F.inode_uid != P.cred_fsuid OR F.inode_mode & 0400 = 0)
  AND (P.cred_fsgid != PG.gid OR F.inode_mode & 040 = 0)
  AND F.inode_mode & 4 = 0;
```

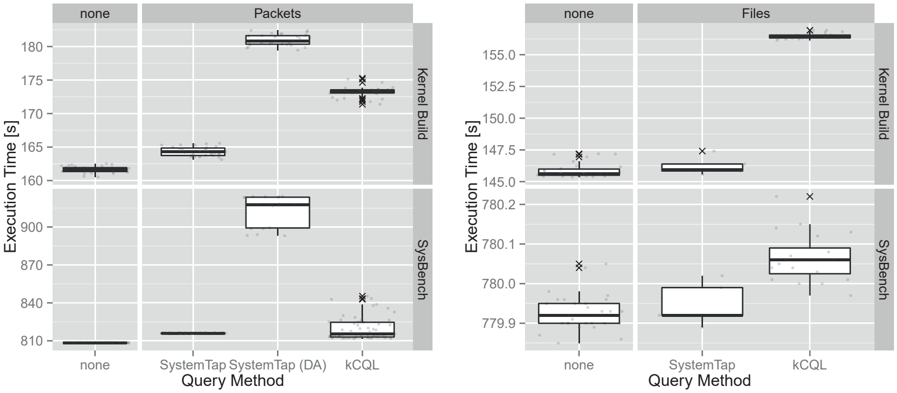


Fig. 2.3: Clean runs of kernel build and SysBench compared with runs that simultaneously execute SystemTap or kCQL with the queries *Packets* (left) and *Files* (right). The different baselines are due to the presence (left) or absence (right) of incoming TCP packets.

Comparison with SystemTap Figure 2.3 contains box plots of the execution times of SysBench and the kernel build for both queries, using kCQL and SystemTap. We also sent network packets to the machine under test while generating the baseline for the *Packets* query. Figure 2.4 is based on the same numbers, and shows the relative runtime and overhead compared with the baseline.

It is apparent that direct access to the kernel state (DA) is unfavorable for the *Packets* query. The tailored SystemTap scripts generate less overhead than kCQL by using incrementally updated copies of the kernel state. While kCQL does not excel here, the overhead is still reasonable.

The SystemTap implementation of the *Files* query can make use of an invariant: the situation we track in the query can only occur directly after a call to *setuid*. We will discuss this further in Section 2.1.6.1.

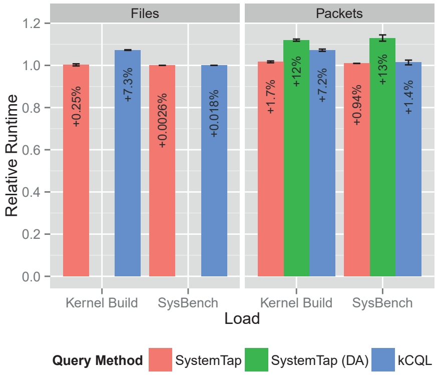


Fig. 2.4: Comparison of average relative runtime and overhead in percent compared with clean runs of kernel build and SysBench.

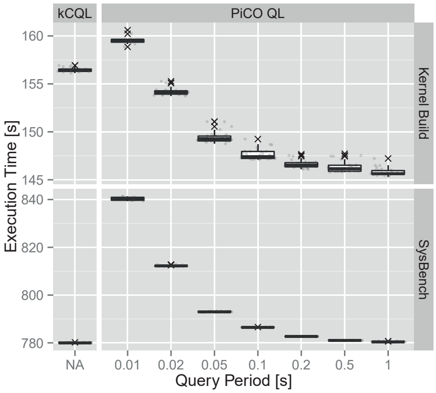


Fig. 2.5: Runtime overhead for the *Files* Query compared with PiCO QL with different invocation periods.

Comparison with PiCO QL PiCO QL is based on a traditional relational interface without a notion for data streams. Thus, we try to achieve the same functionality with frequent polling. The polling frequency plus the query execution time of PiCO QL corresponds to the latency. In addition, the polling frequency governs the trade-off between latency and performance overhead, which is why we evaluate PiCO QL[1] with multiple polling frequencies and find the value that leads to the same overhead as kCQL.

Figure 2.5 shows boxplots for the kernel build and SysBench runtimes without the query and with the query using kCQL and PiCO QL with different invocation periods (x-axis). Invoking PiCQ QL with the *Files* query every 10 to 20 milliseconds imposes roughly the same overhead for the kernel build as running the continuous query with kCQL.

2.1.5.3 Synchronous Overhead

Figure 2.6 shows histograms of the synchronous overhead (based on 100 000 samples of one run each), which is the partial time consumed by query processing directly after and in the same context of an event (e.g., the according interrupt context). Not surprisingly, the direct-access-variant of *Packets* performs worst, because every network packet leads to a complete iteration over the process list. For the other query implementations, it is

[1] This is based on git commit hash 7b2ad66e4a89229f6be392e73c1bda21e2b01434.

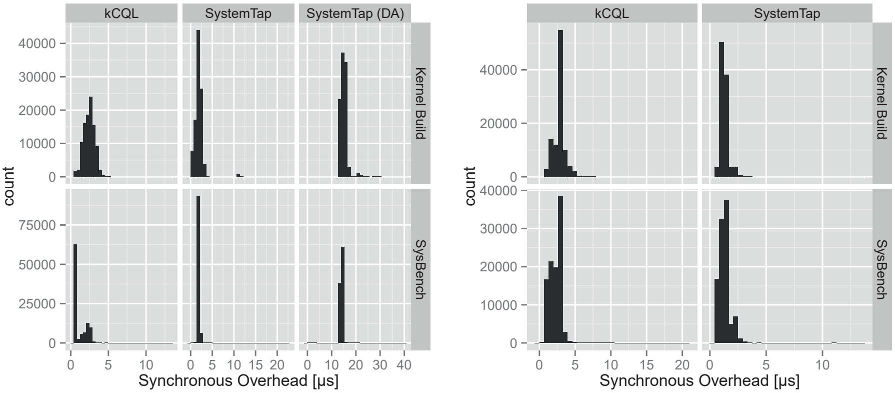


Fig. 2.6: Synchronous Overhead for *Packets* (left) and *Files* (right) with SystemTap and kCQL.

generally lower than that of kCQL, probably due to the fact that kCQL does not yet use per-CPU buffers for the synchronous part of query processing.

Since the query execution of PiCO QL is triggered asynchronously and not by an event, there is no “synchronous latency”. However, PiCO QL invokes `stop_machine()` before executing the query, thus delaying normal operation on all CPUs for the whole query execution time. PiCO QL takes about 730 microseconds to answer the file query on our machine.

2.1.6 Discussion

This section elaborates on the interface of the query language kCQL, security aspects, and practical applications of our approach.

2.1.6.1 Query Interface

The main point of declarative languages is the ability to formally specify “what” rather than “how”. The idea is that the best “how” can be derived automatically. In our case, this comprises the derivation of a query plan, the placement of the operators, and the way of accessing data. This gets particularly interesting when we have multiple simultaneous queries. Due to the strict semantics of the data model and query, common subexpressions of queries can be reused [620]. Considering non-experts, directly specifying “how” also gives more room for inefficient implementations and even fatal errors. The latter point is usually avoided by a combination of restricted languages and dynamic checks as discussed in Section 2. Nevertheless, for a specific query, experts achieve better results by directly specifying “how”.

The hand-crafted SystemTap variant of the *Files* query made use of the expert knowledge that the condition of interest (a file opened by a process that does not have reading permission for it) can only occur directly after a certain event. Certainly, the kCQL query could use a stream of calls to *setuid*, enabling the same trick. The fact that we can achieve the same output with two differently performing declarative queries seems to defy the point of automatically finding the best "how". However, that is not an issue of declarativeness but rather due to inherent relationships between different OS data sources that are not yet visible to kCQL and subject to future work.

2.1.6.2 Security

Currently, only the root user is allowed to use kCQL. To allow ordinary users to use kCQL, we could check if the user has the permission or capability to read from the parts of the kernel state that are required for the query. We could also enforce predicates that filter tuples. For example, a user might only be allowed to receive tuples from the process relation that contain the respective user ID attribute. However, it is not obvious what to do when tuples or attributes that the user cannot access are present in predicate evaluation (e.g., in joins deep in the query plan), but not in the actual output.

2.1.6.3 Practical Applications

This section presents three case studies from our project that show how data collected in the context of the Linux kernel can be used to better understand and improve systems. In two of the case studies we have tapped the Linux kernel of Android-based smartphones. The third aims at improving the Linux kernel in general.

Open Smartphone Data Collection Data gathered on smartphones reveals a lot about users, but also about the behavior and efficiency of the underlying system software. During a 4-month study, we collected an anonymized open dataset on Android smartphones, which is now freely available for research.[2] The data was collected from various sources in the Linux kernel and Android's application framework with MobiDAC [576], which is the predecessor of kCQL. The dataset has a formal meta model [649] and was used, for instance, to predict the next mobile network cell in which a moving user is likely to show up and to learn a smartphone energy model that can predict the energy consumption of a future time window based on past history with a small error.

Reproducible Load Tests Based on another data collection in Android's Linux kernel and user-level services, we created representative resource usage profiles for arbitrary Android apps [445]. This was done by recording traces of low-level events that are signaled to the app, such as the arrival of a GPS fix or network packet, actions executed

[2] http://sfb876.tu-dortmund.de/mobidata.

by the app as a reaction, such as file I/O or display activity, and pauses. The traces can be mixed to create arbitrary app profiles. Hereby the challenge is to adapt pause durations and simulated event timestamps so that the mixed profile is realistic. System software developers can playback the mixed traces as reproducable load tests without the need for any real apps and, thus, can avoid any requirements on existing server connections, user activity, and so on.

Learning OS Locking Rules With the proliferation of multicore and manycore systems the Linux kernel has tremendously grown in complexity, because concurrent controls have to be coordinated in a fine-grained manner. Various kinds of locks are being used for this purpose. Even for experienced system software developers, it has become very difficult to determine the correct sequence of locks that have to be acquired before a particular member of an in-kernel data structure may be accessed safely. Few kernel components have a specification of their locking rules and most of them are outdated. Based on kernel-level data acquisation, namely lock creation and usage events, we developed LockDoc [446] as a possible solution to the kernel developers' dilemma. LockDoc learns locking rules for data structures. It can thus (1) generate documentation, (2) identify outdated rules in existing documentation, and (3) find bugs in Linux by identifying rare event sequences that violate the learned rules. A number of documentation and code improvements have already been contributed and integrated into the Linux kernel.

2.1.7 Conclusion

Tapping the control and data flow of an operating system has its risks, as does any way of tampering with a complex system. It can however provide us with vital information that could hardly be obtained otherwise. The proper tools and abstractions help to mitigate the risk.

Over a period of several years we have therefore developed kCQL, which combines the best ideas of existing frameworks to a unique tool. It has a highly expressive query language, a resource-efficient implementation, and supports data aggregation very close to the data source.

Different application areas have been explored. It turned out that data gathered in the system software context can be used to learn much about user and application behavior, that we could precisely mimic application profiles, and that we could even improve the Linux kernel and its documentation.

2.2 PhyNetLab Test Bed

Mojtaba Masoudinejad
Markus Buschhoff

Abstract: Wireless sensor networks have matured to a point that they are ready for their integration into industrial applications. However, before performing any real-world roll-out, some aspects need detailed analysis. In addition to checks for the application performance and durability, system modularity and energy neutrality are two important concerns requiring accurate analysis. These requirements led to the development of PhyNetLab, a test bed for material flow and warehousing applications on wireless sensor networks.

Entities in industrial systems should be highly modular to enable flexible and reusable systems, and to ease the process of updating or upgrading system components after deployment. This provides easy-to-setup systems that are dynamically improvable while minimizing post-deployment modification effort and costs. Hence, required design principles for both hardware and software is explained by the case study of the PhyNetLab test bed.

Energy neutrality is a fundamental requirement for wireless sensor networks in logistics and production, because the infeasability of battery management of several thousand network nodes will contracept any endeavor to become wireless here. This section shows several means to achieve energy neutrality by using energy harvesting, automatically generated energy models, and online energy accounting.

In addition to the hardware requirements, an industrial scale wireless sensor network also has several software requirements, and these are narrowed down even further when implementing a test bed for such a use case. Most importantly, PhyNetLab uses Kratos, a real-time operating system based on C++ and AspectC++ that allows modular, maintainable, and highly configurable code. Next to the language and framework properties, Kratos employs energy consumption accounting for peripheral devices while still running under heavy resource constraints.

The effectiveness and usability of the PhyNetLab test bed is further showcased by presenting a material handling process of a production system that was entirely built using PhyNetLab. Not only does it serve as a proof of concept for such a test bed, it also provides insights for possible future works discussed at the end of this section.

2.2.1 Introduction

Research in Wireless Senor Networks (WSN) has been continuously advanced in the last years [209]. These networks are from diverse fields of applications with different aims, specifications, and limitations. In addition to the singular WSNs developed separately, federations of WSN have been built as well, with MoteLab [702], FIT/IoT-Lab [231], Indriya [180], and WISEBED [304] as some famous implementations of them. While these platforms show proof of concept and enable testing and development of WSNs, some aspects are still open. Among them are the development of communication protocols for specific applications and energy-aware system design and operation in a large-scale deployment. Moreover, industrial roll-out requires more intensive system analysis to assure reliability in long run continuous operation under full load in industrial environments.

Among the different fields of application, in-house logistics or indoor materials flow and warehousing are perfect candidates. On one hand, decentralization and modularization are considered key elements for future materials handling and warehousing applications [599]. On the other hand, multiple electronic entities have been developed to enable smart operation and communication of objects in this field [472]. Consequently, fundamentals of an industrial use case are available and accessible scientific concepts can be evaluated in this field.

Energy constraints due to the impossibility of recharging a large number of devices and the size limitations of such mobile entities make them hardware with extreme constraints. Meanwhile, the high dynamics of such processes with very fast paces makes hard-coded algorithms inefficient. Consequently, these applications and the PhyNetLab are perfect candidates for development, optimization and evaluation of machine learning algorithms on hardware with extreme resource constraints.

Parts of this section are taken from [104] and [471] with the consent of the authors.

A Wireless Sensor Network Test Bed for Warehousing An experimental materials handling and warehousing platform is designed as a test bed for development, testing, and optimization of industrial WSN case studies. A research hall with more than 600 m^2 in TU Dortmund University is used here. Due to the flexibility requirements of future in-house logistics, no component is stationed permanently in this area. For enabling the transport of objects, five mobile robots are provided. In addition to the typical transport boxes in two standard sizes, there are mobile workstations that can be positioned dynamically according to the production process demands. These elements provide a base for replicating different in-house logistics scenarios including: non-stationary materials flow, different dynamic warehousing, and dynamic production planning and process design.

The missing element for connecting these entities and for establishing a dynamic smart system is the addition of smart electronic components with communication functionality. These solutions should be small in size, light-weight, maintenance-free (or low

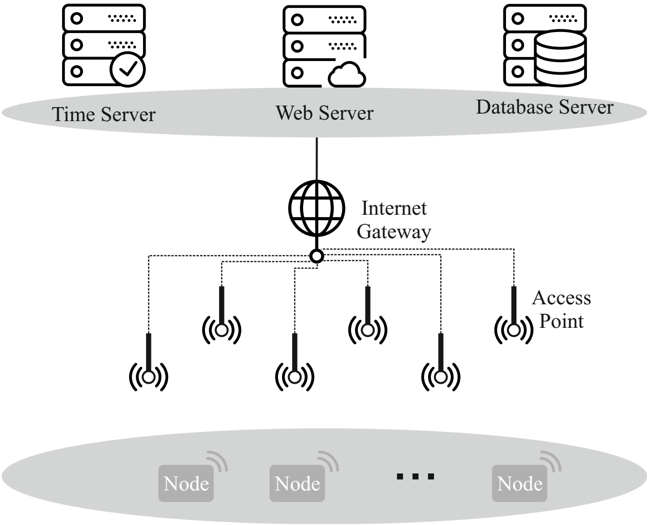


Fig. 2.7: Schematic structure of the PhyNetLab test bed.

maintenance), and autonomous. Moreover, they have to be energy-neutral to eliminate the need for periodic recharges or battery exchanges. The process of developing such a system will be discussed in the rest of this section. However, though we discuss the design process, the main goal of such a system is the development, optimization, and evaluation of materials flow systems that employs a WSN.

2.2.2 General Structure of the Test Bed

This experimental test bed is composed of three main layers. While a large number of entities reside in the physical layer mounted on objects including transportation boxes and workstations, a middle layer is made up of six access points (AP) that enable communication using radio interfaces at 866 MHz. Access to the outer world (internet) is provided via a gateway that mainly accesses servers for time and data, in addition to a web server that serves as a user interface. An abstract overview of the structure of PhyNetLab is presented in Figure 2.7.

In addition to the overall structure of the WSN, PhyNetLab includes some extra infrastructure which makes it an ideal experimental test bed. First, there is a motion capturing system for tracking objects within the environment with sub-millimeter accuracy and a frequency of up to 300 Hz. This system includes a large number of cameras emitting infra-red light. Objects to be tracked are marked with specific reflectors that can be observed by the cameras. Each object gets a set of reflectors with a unique physical distribution. The formation of these reflectors is stored in the software of motion capturing system. Combining views from different cameras provides accurate positioning of each object within this system. This position data can be accessed inside PhyNetLab by all nodes in the two lower levels.

A fundamental necessity for evaluating the energy neutrality of nodes (empowered by photovoltaic (PV) and of energy harvesting) is a controlled lighting. This is possible inside PhyNetLab via a smart lighting management that provides a controlled light intensity to replicate diverse work and warehousing light scenarios.

2.2.3 Hardware

While the upper-tier hardware of PhyNetLab uses commercial servers with specific software developed for this purpose, the middle layer APs are based on modified Raspberry Pi boards. These are supplied with two 868 MHz transceivers in addition to two WiFi modules. In parallel to WiFi communication with the top tier and internet, dedicated transceivers enable communication with the field-level nodes. However, all used hardware are off-the-shelf components specifically programmed for the PhyNetLab test bed application.

The key hardware aspect of PhyNetLab is the heterogeneity of the field level nodes in a large scale. Meanwhile, these nodes should enable fast modification while preferably using the same interface. Hence, nodes in the field level (called PhyNode) that are battery-powered have a modular design with a main board (MNB) and a swappable board (SB). All nodes have a similar MNB, enabling interaction for a basic communication via ZigBee. Furthermore, the MNB is used for system flashing and power supply while each SB has a specific hardware configuration that can be changed over time. A general overview of a PhyNode board front view is presented in Figure 2.8, clearly showing the separation between two modules with a single connection using an 8-pin port.

The MNB is the fix compartment of PhyNode and provides fundamental functionality via air software flashing. Therefore, it has a simple construction with a schematic structure shown in Figure 2.9.

The MNB has a power module made of a Li-ion polymer battery with 1 250 mA h and a typical voltage range of 3 V to 4.2 V. Its voltage controller keeps the system in a safe operational range, while protecting the battery. It also includes a RF transceiver for communication using ZigBee, chosen mainly due to its low energy demand and generality of use. This construction improves the process of flashing new software.

For its core functionality, an SB can have a large set of components with different configurations. A schematic structure of the most advanced version of PhyNode components is presented in Figure 2.10. Some modules such as sensors, user interfaces and energy harvesting devices are optional and are not available in some batches. However, all nodes use a MSP430FR5969 MCU from Texas Instruments, which provides 64 KiB of FRAM as well.

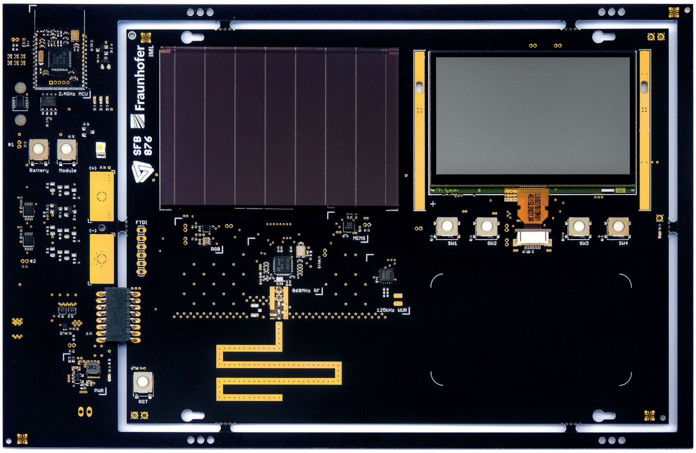


Fig. 2.8: A PhyNode board's front view. SB in the inner part is physically separated from the MNB while having a single electrical connection through an 8-pin port. SB can be simply changed with different versions to enable hardware diversity and as new components evolve over time.

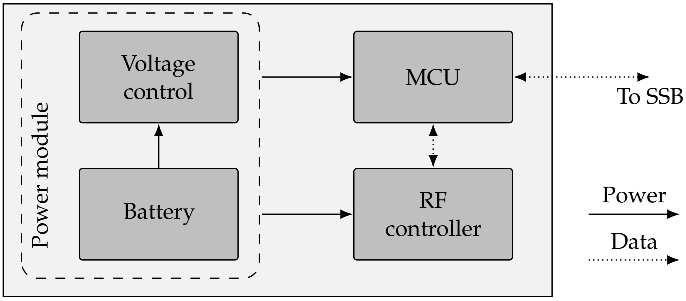


Fig. 2.9: Schematic structure of PhyNode's main board.

To build a heterogeneous network, five different configurations of the SB module are used to create 350 PhyNodes within the PhyNetLab. This diversity provides the possibility of checking scenarios and solutions made of nodes with dissimilar hardware specifications. This heterogeneity is essential for real-world industrial applications, because mixing different solutions and versions of devices is a very common practice in the industry.

2.2.4 Software

2.2.4.1 Test Bed Requirements

An energy-neutral embedded system as a test bed for IoT applications has several requirements that go beyond the scope of typical embedded-systems engineering. Thus,

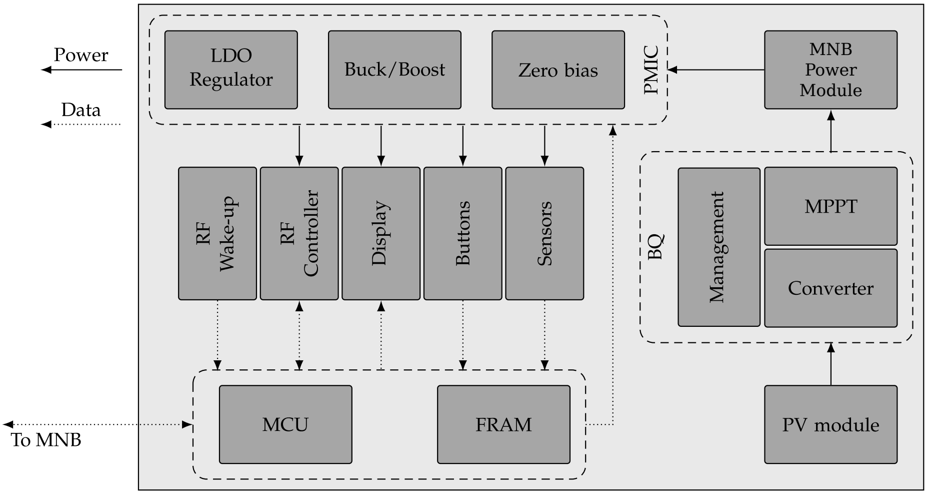


Fig. 2.10: Schematic structure of PhyNode's swappable board.

to deliver a stable and re-usable code base for hardware access and scheduling, an embedded real-time operation system called *Kratos* was developed for PhyNetLab. Kratos fulfills the following requirements of PhyNetLab applications:

- *Tailoring*: The plethora of features and variants in Kratos, combined with the high resource constraints in an embedded low-power system, require an easy way to select and configure components for an application.
- *Separation of concerns (SoC)*: The operating system code base shall be stable and maintainable. This requires a separation of the concerns of Kratos developers, driver/component developers, and application developers.
- *Inversion of control (IoC)*: As part of the SoC strategy, IoC enables calls from the operating system to the application code without altering the operating system code base. In combination with tailoring, this allows for (configurable) calls to drivers, e.g. for their initialization at system startup, and to application code.
- *Crosscutting concerns*: In contradiction to the SoC paradigm, there are also *crosscutting concerns* in a complex system. Here, common functionality (like the logging of function calls) shall be developed as an own component on top of the existing code base. As exemplified with the logging functionality, such a functionality must not alter the source code in order to maintain the SoC. Another example shown later in more detail is an energy consumption accounting during driver calls that does not alter the original driver code. This is barely possible in purely imperative or object-oriented languages.
- *Energy analysis*: To collect PhyNode performance data, a detailed energy consumption analysis is necessary. Online energy measurement cannot fulfill this

requirement, as it cannot be done for single system components, e.g. peripheral hardware components, on the same chip as the microcontroller.

As a result of these requirements, Kratos was developed in AspectC++ [644, 645], an aspect-oriented programming (AOP) language extension for C++. The AspectC++ compiler works as a pre-compiler for the target platform toolchain. AspectC++ can inject or exchange source code into an existing code base during compile time in a process called "code weaving". To do so, AspectC++ supplies its own language to identify code locations ("joinpoints"), and a well-structured C++ alike language for the definition of code fragments to weave, known as "advices". A combination of joinpoints and advices is called an *aspect*.

An example use of AspectC++ in PhyNetLab was the utilization of the PhyNode display, which requires a higher voltage for bus signals than delivered by the processor in low energy mode. Instead of altering the display drivers, and thus making them dependent on the processor type and rendering them unusable for other hardware platforms, PhyNode's display code weaves low-high power mode switching aspects into the drivers, leaving their original code untouched. Furthermore, this behavior enables *tailoring*: simply by choosing whether to compile this aspect or not is enough to enable or disable the power mode switching without ever touching the code base. By that, code developers are not required to anticipate possible future extensions by using "#ifdef" statements for their configuration.

The AOP also helps to enable the IoC paradigm. Coming back to the earlier example of driver initialization, drivers can now employ an aspect to insert a call to their initialization code into Kratos. For this purpose, Kratos has a set of empty "hook" functions where aspects are supposed to weave their code. Again, deactivating a driver simply requires not compiling its code and this aspect.

In conclusion, the use of AspectC++ enables SoC, IoC, crosscutting concerns, and supports tailoring at a very detailed level of granularity without the hard-to-maintain syntactic overhead of "#ifdef" directives [63].

2.2.4.2 Energy Accounting (Energy Models)

An important parameter for the analysis and evaluation of test bed experiments is their energy consumption. In systems with a highly constrained supply of energy, it is important to understand what energy is used for, and which component is responsible for its consumption. However, this is hard to answer by simple measurement. Moreover, an "online" measurement, i.e., having measurement equipment on board and performing measurements during runtime, has several disadvantages, including:

1. It consumes power by itself.
2. It cannot distinguish between different consumers.
3. It requires CPU time for storing data.
4. It needs a communication channel to the CPU, acquiring an important resource.

As an alternative, software energy models can help to estimate the power consumption per hardware component. Since software models require a computational Overhead, there are several design alternatives. *Offline* models calculate the energy consumption at design time, thus enabling highly detailed and precise models. However, they can only statistically anticipate external events, such as incoming communication requests and user interaction. By contrast, *online* models can dynamically adapt to the situation, but seemingly have a trade-off between computational demands and accuracy. Nevertheless, it was shown in [105] that highly accurate results can be achieved in many real-world scenarios with low computational effort.

To achieve this, an energy model for a component has to follow a certain methodology. In the used modeling scheme, each component is modeled as a cost-annotated Finite State Machine (FSM) consisting of energy states. The FSM has two types of cost annotations: a state is annotated by average *power* costs, so a state's energy consumption can be calculated by multiplying the costs by the time the machine resides in the respective state; transitions, however, are annotated by average *energy* costs, so that switching between states can potentially have an energy impact.

In the given modeling scheme, driver function calls and interrupt service routines cause state changes. So, there is a mapping between driver functionality and energy model, and the energy model has to be constructed accordingly. In driver implementations on other operating systems, this might not be feasible, as there is a *semantic gap* between the driver function call interface and the energy model (see Figure 2.11). Kratos allows us to close the semantic gap by using a mapping scheme for an energy model. However, the driver interface implementation still has to be mappable to transitions of the model in general.

It is not obvious that this form of modeling is always feasible while maintaining sufficient accuracy, since it might become a problem for arbitrarily complex hardware. However, the complexity of hardware in energy neutral systems is limited, and a practical survey of models for typical components like radio transceivers, sensors, CPUs, serial bus drivers, different MCU platforms, etc., guided by the automated energy modeling system shown in Section 2.2.4.3, yields accurate results in practice.

The energy model of a component can be used for online energy accounting within the component drivers and other operating system modules. A low overhead implementation simply counts the numbers of transitions, i.e., driver calls, and maintains a time lapse for each state of the FSM. When energy values are requested from the accounting system, calculating the energy consumption of a component is as complex as multiplying the count of each driver called by the respective transition energy, and multiplying the time lapse of each state by the respective state power.

CPU energy consumption is a special case, because a CPU usually has no driver implementation. However, CPU energy was accounted using the same modeling scheme in Kratos. A CPU model segregated different power modes or sleep modes as FSM states. Instead of instrumenting a driver for accounting, the Kratos scheduler is used that controls all CPU energy state transitions.

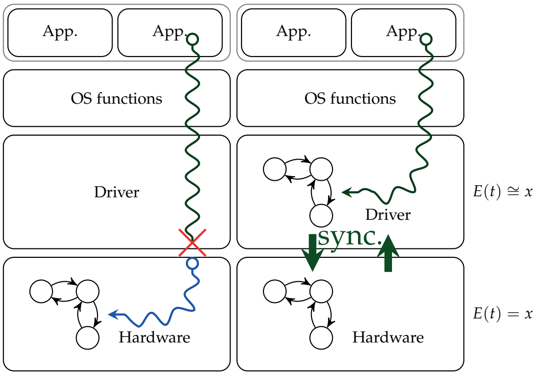


Fig. 2.11: Coherent and incoherent driver. The incoherent driver on the left allows no code-to-model mapping (semantic gap). On the right, this is achieved by respecting the model structure within the code. Source: [104]

The energy accounting code is injected into the original driver and scheduler code by using the AOP. To work towards an automation of this process, which is called *instrumentation*, a set of tools were developed that are able to import energy model files and mappings between drivers and transitions, and to generate AspectC++ aspects for the respective driver. Again, this enables application developers to quickly decide whether to use instrumented or original drivers without altering the driver code base.

2.2.4.3 Energy Model in the Loop

The instrumentation system described before allows for quickly altering the cost annotations of a model. It also allows for automated re-deployment of code to the actual hardware. This can be used to determine the costs of states and transitions in a process of supervised learning.

Thus, comparing actual power measurements to accounted energy can sharpen the energy model. This can be achieved by using a measurement loop as depicted in Figure 2.12. A device under test is programmed with an instrumented firmware. The used model is a preliminary state machine with transition mapping, yet without cost annotations. The operating system is additionally instrumented by a test pattern generator that calls driver functions in a pre-configured order. The device running this firmware is externally equipped with a power measurement unit. Both accounting and measurement values are delivered to an external analysis engine, which determines the quality of the cost solution and automatically creates a new model to re-iterate the process. The loop either ends if a desired accuracy is reached or no significant quality increase can be achieved throughout multiple runs. Also, the analysis engine can identify structural problems within the FSM, e.g., states that show a great variance in their power consumption.

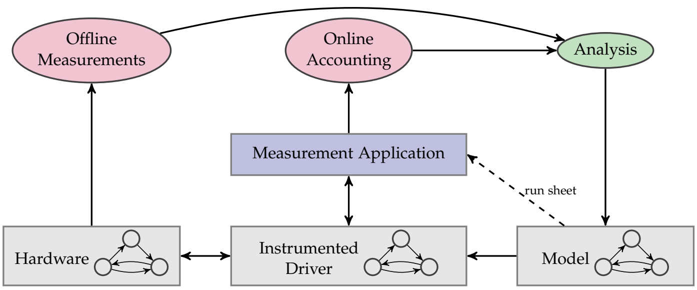


Fig. 2.12: Measurement loop. Solid lines resemble automatic steps; dashed lines allow for manual intervention. Source: [104]

This automated model annotation process opens the door for more complex models. Until now, the model was limited to accounting driver calls without respecting function call arguments. Introducing call argument costs (as a mathematical function of the argument set) can greatly increase the accuracy of a model [106]. However, this comes at a cost: the online evaluation of a driver's energy consumption can now become as complex as the argument cost function.

The measurement application that is generated for the target hardware is driven by a *run sheet*, as shown in Figure 2.12. The run sheet defines validity ranges for all driver function arguments. The synthesized measurement application iterates through the power set of these ranges. The resulting data is analyzed in three steps:

1. **Identification:** In this phase, all arguments influencing the energy consumption are selected. For this purpose, reference measurements are taken while keeping the observed argument constant. Afterwards, the observed argument is varied throughout several iterations. If the varied argument leads to an energy consumption that is out of the noise bounds of the reference measurements, the observed argument is considered influential.
2. **Single-argument regression:** The cost function for a single argument is identified by using eight regression methods (linear, logarithmic, shifted logarithmic, exponential, square, fraction, root, one-bit count). The function $f(\vec{p})$ with the lowest *sum of squared residuals* (SSR) is chosen for the next step.
3. **Multi-argument regression:** All identified functions $f(\vec{p}) \in F$ are combined within a compound function shown in Equation 2.1. The coefficients for this functions are determined using regression while minimizing the error. The result is the assumed cost function.

$$g(\vec{p}) = \sum_{F^{'} \in \mathcal{P}(F)} \left(a_{F^{'}} \cdot \prod_{f \in F^{'}} f(\vec{p}) \right) \tag{2.1}$$

Tab. 2.3: Symmetric model error of static and parameter-aware (right) model attributes for CC1200 and nRF24 transceivers in Monte-Carlo cross validation. Parameter influence is shown in the middle.

Model attribute	Influencing			Error	
	Data rate	**Payload length**	**tx power**	**Static**	**Model**
CC1200 TX power	✓	✓	✓	12 %	0.7 %
CC1200 TX time	✓	✓	–	87 %	0.1 %
CC1200 RX power	✓	✓	–	0.2 %	<0.1 %
nRF24 TX power	✓	–	✓	34 %	1.2 %
nRF24 TX time	✓	–	–	16 %	<0.1 %
nRF24 RX power	✓	–	–	2.2 %	<0.1 %

To verify this approach's validity, TI CC1200 and nRF24L01+ radio transceivers, a low-power I^2C temperature sensor (LM75B), and a synthetic peripheral with programmable power consumption behavior were modeled and tested [106]. Both transceivers contain IDLE, RX, TX, and SLEEP states. For the evaluation of the dynamic model approach, three adjustable parameters were present: transmission power, bit rate, and transmitted data length.

Table 2.3 shows the determined influence of parameters on model attributes and the model error both for static and dynamic model attributes. The model error was assessed using 200 Monte-Carlo cross-validation runs. Data was split into 2/3 for training and 1/3 for validation.

The results show that function arguments significantly influence energy consumption, occasionally in unexpected ways. For example, it was expected that the power consumption during TX is constant for the CC1200, and payload length would only influence the energy consumption through TX duration. In reality, even the actual power depends on the payload length. It turned out that the CC1200 has a fixed preamble with separately set transmission power, so the preamble/payload duration ratio (and hence TX power) also depends on the payload size. By contrast, the nRF24 transmissions use a fixed packet length by default, so the energy consumption of a packet transmission is completely independent of the payload length.

With the synthetic peripheral, several functions for state power consumption were tested. The used function was reliably detected within less than 0.7 % model error. Even in a pessimistic parameter-aware cross-validation setting (i.e., the parameter combinations of training and validation set are mutually exclusive, which is rarely the case in real-world usage), correct functions were determined in at least 90 % of cases, and model error did not exceed 1.4 %.

For most parameter-independent transceiver states and the temperature sensor, model errors below 0.8 % were observed for state power consumption. The only exceptions were the CC1200 SLEEP state, showing random deviations of 7 % independent of the parameter settings, and few ultra-low-power states, which suffered from the limited accuracy of the available measurement equipment. Absolute errors were below 1.2 µW here.

Modeled transition energy showed errors of 1 to 10 % (5 µJ) and transition duration up to 2.5 %. Only errors for transitions longer than 100 µs were correctly measured in keeping with the horizontal accuracy of the measurement equipment. Assuming an average of two transitions per second, overall model error was below 1.5 %.

2.2.5 Experiment Examples

After developing, building, and evaluating both the hardware and software of PhyNetLab, we conducted a simple case study by using measurement information for rough indoor localization [473]. In this evaluation experiment, light intensity, temperature, accelerations, and passive metrics such as Received Signal Strength Indicator (RSSI) were measured in addition to the exact position of each node from the motion capturing indoor localization system. Using a large collection of this information, different machine learning applications were then utilized. These methods not only include decision trees and random forests, but also k-nearest neighbours, Support Vector Machines (SVM), and Naive Bayes classifiers. Although they performed well in predicting the position, only a small selection of them can be implemented on a PhyNode due to the extremely constrained memory, computation, and energy resources. This shows the necessity of further research on development and optimization of ML models for devices with extreme resource constraints. After providing proof of concept for usage and implementation possibilities of PhyNetLab, it was opened for real-world case studies.

In the first industrial evaluation application of PhyNetLab, the effect of integrating cyber-physical systems (PhyNodes) within a dynamic materials flow system for production lines was tested. The results of these experiments, reported in [730], have enabled an initial analysis of decentral production planning using cyber physical devices. Furthermore, these results help to establish AuDePrOC as a tool for a systematic decentral strategy analysis [730].

In the next step, PhyNodes are integrated into mobile workstations to enable not only a flexible materials flow, but also a dynamic factory and production system [285]. Such a plant can reorganize its overall structure to adapt itself according to the current production necessities in an optimal manner. Hence, such a production system can remove or add extra entities to the overall system dynamically by making use of the infrastructure provided by PhyNetLab. These two initial experiments show both the potentials of decentral and modular systems and the possible challenges ahead of systems designed for their use. All in all, PhyNetLab and PhyNode provide a base test bed for real-world evaluation (such as the application of communication aspects in Section 9.2.2) and roll-out. Moreover, the tools and experiences collected during its development will pave the road for more futuristic test beds to collect data and adapt designs to them.

2.3 Zero-Power/Low-Power Sensing

Andres Gomez
Lars Suter
Simon Mayer

Abstract: Over the past few decades, batteries have played a central role in the design of wireless sensing systems. Large storage devices provide a stable energy supply, ensuring long system lifetimes even when energy consumption is highly variable. This storage capacity is a central tenet in the design of time-based sensing applications, which can gather information about the system's surroundings periodically. While a large energy storage capacity has certain benefits, it also has several drawbacks. They have limited recharge cycles, are costly to manufacture, and possibly include harmful, poisonous materials. They can also increase the form factor significantly, and impose restrictions on the temperature range of operation. Current trends point toward the deployment of billions of interconnected sensing devices gathering information from their surroundings, also known as the Internet of Things (IoT)). For this vision to become a reality, power systems will need to be small, cheap, low-maintenance, reliable, efficient, and scalable. While energy flow is absolutely necessary for IoT devices to function, large energy storage capacity is not. Minimized energy provisioning will make the IoT more economically viable and environmentally friendly. It also restricts the use of high-power peripherals and introduces intermittence, raising new challenges in application development. This contribution presents an overview of the main challenges for low-power sensing with limited energy storage. Starting from hardware considerations for high-efficiency energy harvesting, the benefits and limitations of batteryless sensors are investigated. New software techniques are deemed necessary to address these limitations, requiring close synergies between low-power software and hardware components.

2.3.1 Introduction

Information has become one of the most important factors in modern economies, playing a key role in sectors such as healthcare, infrastructure, and supply chains, among many others. Whenever information from the physical domain is required, sensing systems must be employed to gather the relevant data in a scalable and affordable manner. Designing these systems for long-term deployments is a difficult challenge since traditional battery-powered devices would be restrictive in terms of size, cost, reliability, and maintenance. Large energy storage elements can provide a stable power supply, leading to potentially long system lifetimes even when the system's power demands are highly variable. However, current trends point toward the deployment of billions of interconnected embedded systems sensing data from their surroundings that will be integrated into the IoT. In many IoT use cases, it is desired that these sensing devices disappear physically as well as psychologically and that they require little maintenance, motivating the use of mobile, wireless sensor nodes. Energy harvesting is widely regarded to play an increasingly important role in supplying enough energy to this new type of resource-constrained devices. However, even though costs have fallen, few IoT products have embraced solutions based on energy harvesting. This is partly due to a mismatch in both the power density and the timeliness of energy production with respect to consumer requirements.

A new class of batteryless sensing systems has recently emerged that provides a sustainable, long-term solution to supplying the expected large numbers of IoT devices with sufficient operating power. These energy-opportunistic systems are functional only when the environment provides enough energy for their operation; otherwise, they consume zero energy. This contribution focuses on the design challenges for the efficient execution of batteryless sensing applications, specifically those powered by light. The work presented here perfectly complements the detailed mathematical models for indoor photovoltaics discussed in Section 3.6 in Volume 3.

Starting from a study of the energy constraints imposed by indoor environments, the main optimization criteria for batteryless platforms are introduced. Based on this formalism, different application scenarios for wearable and statically placed sensors are presented. These applications and their hardware considerations are discussed in detail, as well as the software optimizations necessary to execute them reliably and efficiently in a batteryless system.

If system operation depends on the energy extracted from its surroundings, it becomes of fundamental importance to understand the dynamics of the environmental conditions. In the worst-case scenario, environmental conditions are non-deterministic and highly variable. For robust operation, systems relying on energy harvesting, therefore, need to tolerate [261] or adapt [11] to variable harvesting conditions. Data from the spatially and temporally variable environment and the energy that can be extracted through harvesting are highly valuable for dimensioning, calibrating, and testing such systems. Extensive irradiation data for outdoor solar harvesting is available from weather service stations around the world, typically reaching back many decades. By contrast, indoor solar harvesting data is only sparsely available, but becoming increasingly critical for many IoT applications that target deployment in this environment such as building automation and assisted living.

We discuss an extensive indoor energy harvesting dataset, first presented in [632], that addresses the lack of long-term indoor solar harvesting traces. While other works have performed illuminance measurements in indoor environments [267], this work jointly monitors the extracted energy from the solar panel, the energy stored in the battery, and the ambient conditions. The combination of power measurements using a real harvesting system implementation and rich ambient sensor data enables diverse opportunities for analysis and evaluation, including power estimation, energy harvesting source modeling, and harvesting system efficiency analysis, to mention a few. One of the key insights from this dataset is understanding quantitatively the energy variability of indoor photovoltaic harvesters over a two-plus year period. The same indoor solar cell can produce anywhere between 0 and potentially hundreds of joules during one day, depending on geographical location, architectural design, and many other variables.

When following the batteryless sensing paradigm, this potentially large energy flow should be consumed as soon as the energy is harvested. The main reason for this energy-opportunistic operation is that the alternative, to store energy for future use, would require an expensive, rechargeable energy storage device. As an example, a 500 mAh lithium ion polymer battery can easily cost several USD, even at volume, while a 47 μF ceramic surface mount capacitor can cost 0.1 USD. Minimized energy provisioning will make the IoT more economically viable, environmentally friendly, and potentially more energy-efficient. When a battery-powered device maintains an application circuit energized for many years, a considerable amount of that stored energy will be spent in *sleep* mode. Even if this is a highly optimized low-power mode, most commercially available microcontrollers can reduce their power consumption to a few μWs in the best-case scenario, assuming all other peripherals can be turned off. Consequently, these few μWs can, over many years, turn into thousands of joules that were not spent on the actual application, but just on keeping the system energized. Keeping the system on is essential for many sensing applications, but not all. Energy-driven sensors will spend harvested energy as soon as possible by running the application. During night periods when there is no energy, the batteryless sensor will not be able to support the energy-efficient *sleep* mode, and will turn off completely, consuming zero watts until the environment can supply energy again. Batteryless systems thus spend a comparably tiny amount of energy to keep the system in *sleep*; most of the energy will be spent actually executing the application.

Batteryless sensing systems perform efficient data sampling when their surrounding environment provides enough energy. However, even when a transducer is large enough to directly power an application circuit in the correct voltage and current range, there is no guarantee that it will harvest at its maximum power point [631]. If the application circuit adjusts its operating point to extract the maximum power from the transducer, it will most likely not be the application circuit's optimal operating point. The most energy-efficient operation for the application circuit depends only on the application and the peripherals it uses, not the environmental conditions. To maximize the harvested energy and minimize the application's energy cost, the system architecture needs to have separate voltage domains for the harvesting and application circuits. The power conditioning circuitry can thus become transducer-independent and allow impedance matching for maximum power transfer without affecting the application circuitry. The application circuit can also dynamically adjust its own operating voltage according to the application requirements. These architectural requirements will be discussed in detail, along with the Energy Management Unit (EMU) solution, first proposed in [259]. In the years since this architecture has been experimentally demonstrated to be both energy-efficient and robust in a wide range of operating conditions.

EMU-based sensing systems leverage voltage and current decoupling to efficiently execute task-based applications. As such, an environment that supplies only 10 µW at 1 V can still supply application circuits running at 3 V with tasks consuming up to 100s of mW. Batteryless applications can thus execute reliably, even with unfavorable environmental conditions. We will present two batteryless application scenarios based on the EMU architecture. One details statically positioned ambient sensors that can transmit data asynchronously using a Bluetooth Low Energy (BLE) radio. Another focuses on a wearable sensing application running on a batteryless system for accelerometer-based gesture detection. We present the entire flow in this embedded learning application, from data acquisition to model training and system performance optimization.

The remainder of this contribution is structured as follows: the indoor photovoltaic dataset is discussed in Section 2.3.2, the Energy Management Unit (EMU) architecture is presented in Section 2.3.5, the first batteryless sensing system for ambient monitoring is introduced in Section 2.3.8, the second batteryless system for gesture detection is discussed in Section 2.3.9, the analysis of both systems is presented in Section 2.3.14, and we conclude in Section 2.3.17.

2.3.2 Energy Availability with Indoor Photovoltaics

Large datasets can assist the design and evaluation process of energy harvesting IoT systems for outdoor scenarios. Since the harvesting characteristics in indoor environments vary significantly, these datasets are unsuitable for designing and evaluating harvesting-based systems intended for indoor applications. In indoor environments, the harvestable energy is severely limited, thus imposing strict energy constraints for harvesting-based systems. Furthermore, the harvesting characteristic can change drastically even for systems deployed close to one another. To understand the energy indoors and appropriately design harvesting-based IoT systems, long-term indoor harvesting data from various indoor locations is needed. Such data also enables extensive evaluations of energy harvesting systems indoors. We discuss the collection of an indoor harvesting dataset that consists of measurements of the power harvested by a solar panel, the energy buffered in energy storage, and the sensor data describing the system's ambient condition.

2.3.3 Measurement Setup and Deployment

The indoor harvesting dataset, first presented in [632], is collected with a custom-designed monitoring platform shown in Figure 2.13. The platform contains a solar panel

(AM-5412, 50 mm x 33 mm) whose output is measured. The bq25505 harvesting chip includes a boost converter with maximum power point tracking (MPPT) that ensures the solar panel operates efficiently and also stores the harvested energy in a virtual battery circuit. Since the measuring platform does not contain an application to consume the harvested energy, an energy storage component would overflow and measurements associated with it would not be consistent with the typical behavior of an energy harvesting IoT system. Instead, the virtual battery circuit emulates a system's energy storage continuously operating at a typical voltage of 4.2 V. It maintains a consistent operating point of the harvester management component and provides harvesting measurements that align with the behavior of a harvesting-based system. To record the ambient conditions, the platform contains two TSL45315 light sensors and a BME280 humidity sensor. The light sensors are located on two opposite sides of the solar panel and provide illuminance lighting conditions that the solar panel is exposed to while harvesting energy. The humidity sensor measures the ambient relative humidity, air pressure, and temperature.

The custom platform is designed to be used in conjunction with the Rocketlogger platform [631]. The Rocketlogger is a measurement device with a small form factor that can seamlessly provide high-accuracy measurements for an extensive range of currents. These features enable the RocketLogger to be used for long-term deploying logging highly variable conditions found in indoor environments. The Rocketlogger can thus measure the energy harvested by the solar panel and flowing into the harvesting chip and the output of the bq25505 component as well as the ambient sensor data. The sensors, energy extraction, and circuitry required for the virtual battery are powered by the Rocketlogger, ensuring that the harvesting system is not affected. As an independent observer, the RocketLogger has minimal impact on the system being measured and provides the necessary data to relate environmental conditions to electrical power signals.

Five measurement platforms consisting of the custom monitoring platform and Rocketlogger as shown in Figure 2.13 are deployed throughout a floor in an office building at ETH Zurich, Switzerland. Figure 2.14 depicts the locations of the measurement platforms. Due to construction, one platform was moved and the figure shows both its initial and subsequent location. The deployments cover diverse environments. As such the platforms are exposed to different mixtures of artificial and natural light and direct and indirect sunlight during various times of the day. Additionally, the platform's orientations within rooms are varied and the occupancy patterns of the rooms range from regularly and permanently occupied offices to only sporadic occupancy. All deployment locations are described in Table 2.4.

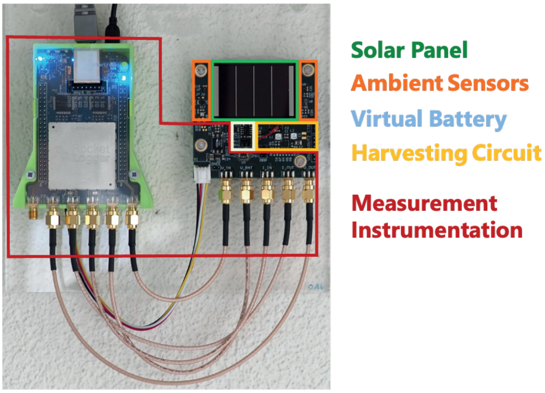
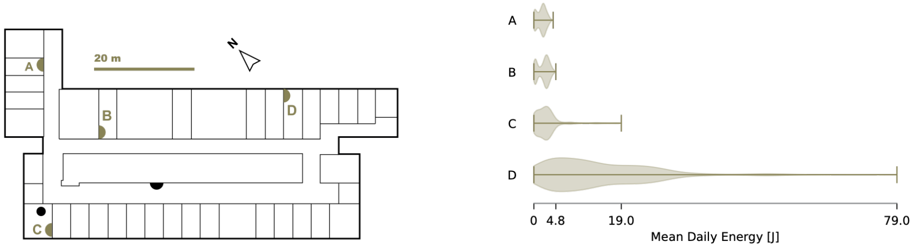


Fig. 2.13: Measurement setup includes RocketLogger, solar panel, harvesting circuitry, and a virtual battery. Sensors measure the ambient condition.

2.3.4 Energy Harvesting Dataset

The collected long-term indoor harvesting dataset is publicly available. The extensive indoor energy harvesting dataset [632] contains power trances and ambient sensor data covering more than two years starting in July 2017. The time span for which the data from each location is available is listed in Table 2.4.

The energy harvested on average during a day is determined for each location and summarized in Table 2.4. The table also shows the 75 % percentile of the absolute deviation from the mean. The energy yield varies drastically between locations despite their proximity, highlighting the strong spatial variability characteristic of indoor harvesting. Furthermore, the temporal variability of the energy availability in indoor environments is visible with the wide range that the percentile spans. Station A primarily harvests energy from artificial light. The illuminance levels there have certain patterns, like



(a) Floor plan

(b) Diverse harvesting characteristics

Fig. 2.14: The measurement platforms were wall mounted on an office floor. The installations experienced diverse conditions affecting the daily harvested energy. Some locations (e.g. Station D) receive much natural light and thus have a large energy budget. Others (e.g. Station A) mainly harvest from artificial light and harvest significantly lower energy per day.

Tab. 2.4: Summary of four measurement platform deployments: environment characteristics, measurement timespan, and mean daily energy yield.

Station	Location	Natural light	Timespan	Mean daily energy
A	Office	Little	22 months	2.02 ± 1.64 J
B	Office	Little	24 months	1.57 ± 1.28 J
C	Office	Medium	24 months	2.87 ± 2.36 J
D	Laboratory	High	24 months	14.18 ± 11.67 J

reduced harvested energy during non-working days. Station D received direct sun light for limited periods, resulting in a maximum daily harvested energy over 20× more than the other stations dominated by artificial light.

2.3.5 Batteryless System Design

In the previous section, we have seen how energy harvesting can have high spatial and temporal variability. Even when the same solar cell is deployed on a single floor, they will have very different energy budgets, which can also be difficult to predict. If the embedded systems are not designed to handle this variability, the environmental conditions can have a catastrophic impact on the overall system performance. Battery-powered devices could absorb this variability, but current trends in energy harvesting systems point towards a significant reduction in storage capacity due to cost, size, and environmental considerations. The trade-off in doing so is that a minimal service cannot be guaranteed for long periods of time. This is because as storage capability decreases, the behavior of these systems becomes more immediately influenced by the environment.

Duty-cycling is a common dynamic power management technique that allows a system to adjust its average energy consumption by introducing Low Power Mode (LPM). However, in order to perform single tasks such as reading a sensor value or transmitting a data packet, these systems need to be able to buffer the required energy. Otherwise, environmental conditions can rapidly change and turn off the load before it completes its task. Consequently, we argue that a novel energy management unit (EMU) is needed to provide energy guarantees for such disadvantageous scenarios in an efficient, transducer-agnostic manner. Due to the limited energy intake in batteryless systems, the unit should self-start requiring as little time and energy consumption as possible. During those short periods of limited energy intake, it maximizes the energy build-up by harvesting at the source's optimal power point. When powering the load with short energy bursts, it provides a control interface to the load so its optimal power point can be tracked. In this section, we present an EMU that satisfies these requirements, as shown in Figure 2.15.

2.3.6 Batteryless System Architectures

In recent years, the research community has focused on systems with very limited energy storage capacity. In the most extreme case, energy storage is so limited that guaranteed application progress occurs at a very fine granularity, possibly down to a few instructions per activation cycle. Depending on the environment, transducer, and application, different types of circuits might be needed to supply the system with the energy necessary for program progress at a supported voltage range. Generally speaking, there are three types of architectures for batteryless sensing systems:

Directly-Coupled When the transducer has an I-V curve compatible with the application circuit, it can be directly connected. These systems typically use a small decoupling capacitance (<20 μF) to buffer small amounts of energy. If the energy storage is too small, atomic tasks such as sensor measurements and radio transmission are not supported, since their energy requirements are too large for a small transducer with limited energy storage. In [39, 341], the authors propose a combined HW/SW approach to perform computation when the source can directly sustain a computational load during short periods of time. These works use volatile logic that requires state-retention mechanisms. In [424, 698], the authors present storage-less and converter-less harvesting systems in which the load uses frequency scaling to track the maximum power point of the source. While frequency scaling can maximize the energy input in CPU-based applications, it does not minimize the load's energy consumption and is limited to a narrow active power range. Even though directly-coupled systems avoid converter losses, if the power input is below this narrow active range, the load cannot be powered and the system's efficiency immediately drops to 0 %. Unfortunately, this is often the case in batteryless systems. When the energy source and load have incompatible operating points, decoupling them with converters becomes a necessity. As opposed to traditional, battery-based systems, decoupled batteryless systems have a limited energy buffer between the source and load.

Boost Converter Only The authors of [158, 159] propose a low-power management system that requires very low input voltage and current. These works are able to start the energy conversion at very low input power levels, but require a large buffer capacitor at the converter input. Consequently, both approaches suffer from very long start-up times of at least 18 minutes due to charging a large input capacitance of 140 mF at a constant input power of 2.5 µW. As will be explained, our capacitance is chosen to minimize the cold-start energy and time.

Boost Buck Converter Combination The authors of [152, 524] use a boost converter for optimal power point tracking. However, the first proposed system utilizes RF harvesting to accumulate charge in a supercapacitor and then power a camera application

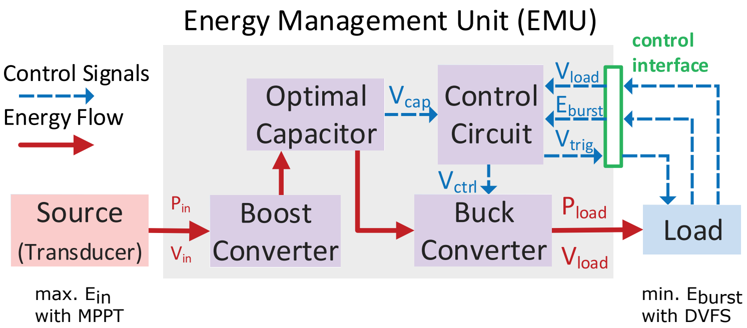


Fig. 2.15: Dynamic Energy Burst Scaling simultaneously optimizes both the energy input and output, even when the transducer and application circuit operate at different voltage and current.

with a buck converter. The second uses a reconfigurable energy architecture that can adapt the energy capacity depending on the application's energy mode. The boost/buck converter topology with an energy buffer serves as a basis for the approach presented in this contribution and has been successfully demonstrated in many other works such as [262, 630].

2.3.7 Energy Management Unit (EMU)

EMU-based systems decouple the load from the source and efficiently build up charge regardless of the load's operating point. We now describe our model of EMU-based systems, which captures the time evolution of the energy storage device as a function of both the environment and application circuit. One of the main goals is to derive equations that can apply to a wide variety of energy sources and loads. This model can be used to optimize important system parameters, namely the EMU's start-up costs and the energy burst size.

EMU Performance The amount of energy buffered in the EMU depends on several parameters including the input and load power, and the system's non-idealities. The equation governing the time-dependent energy level in a capacitor is as follows:

$$\begin{aligned} E'_{cap}(t) = \frac{d}{dt}E_{cap}(t) &= \eta_{boost}\left(V_{in}(t), I_{in}(t)\right) \times P_{in}(t) \\ &\quad - P_{load}(S_i)/\eta_{buck} - P_{leak}(t) \end{aligned} \tag{2.2}$$

In this equation, the positive term represents the energy intake, while the negative ones represent the energy consumption.

Input Power The system has only one power input, $P_{in}(t)$, supplied by the transducer. We focus on the adverse scenario where $P_{in} < P_{load}$ and $V_{in} < V_{load,min}$. This means that directly coupling the transducer to the load is not possible since it would not meet voltage requirements. Furthermore, the batteryless sensing system can be placed in dynamic environments. In these cases, maximizing the system's overall energy flow demands that the source's maximum power point be tracked.

Load Power In the proposed model, the load can have two states (S_i): active or inactive. When active, the load is characterized by three quantities: $E_{burst,i}$, $V_{load,i}$, $P_{load,i}$; where $E_{burst,i}$ defines the energy burst size required for one execution of task i, $V_{load,i}$ its supply voltage, and $P_{load,i}$ the power consumption during the execution of task i. These parameters were characterized experimentally. In the inactive state, the load is in deep sleep and awaits the trigger from the energy management unit. Though the actual power consumption during deep sleep depends on the hardware, complex sensing systems typically consume a few µWs. If the deep sleep power is higher, possibly due to additional enabled peripherals, it will simply take longer for the EMU to accumulate the energy necessary for the next burst.

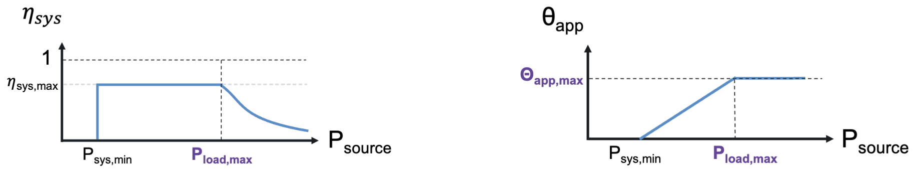
Converter Efficiencies Since decoupled systems can have the source and load operating at different voltages, converters are needed. This step, while necessary, introduces non-negligible losses, which are represented by boost and buck converter efficiencies $\eta_{boost}(V, I)$ and η_{buck}. The boost converter's efficiency is particularly sensitive to the operating voltage and current, meaning it must be parameterized. These efficiencies were also characterized experimentally, and a simple look-up table is used for simulations.

Other Energy Losses Unfortunately, converter inefficiencies are not the only sources of energy losses. The maximum power point tracking unit and the control circuit also consume energy. The consumption of the control circuit I_{ctrl} and buck converter I_{buck} consists of a constant current and resistive component and hence depends on V_{cap}. For the energy buffer, a capacitor of size C_{cap} and resistive leakage R_{cap} is assumed. Considering these components, the system leakage is summarized as:

$$\begin{aligned} P_{leak}(t) =& V_{cap}(t) \times \left(I_{ctrl}\left(V_{cap}(t)\right) + I_{buck}\left(V_{cap}(t)\right)\right) \\ &+ V_{cap}(t)^2 / R_{cap}. \end{aligned} \tag{2.3}$$

Equations 2.2 and 2.3 can accurately describe the time evolution of the system's energy levels. They will be used in the remainder of this section to estimate how different parameters impact the system's losses, and to then calculate the optimal parameters that minimize the losses.

Given the system model presented above, we can start optimizing the cold-start energy and start-up time. By definition, this is the fixed start-up cost to turn a batteryless system on. In order to minimize these fixed costs for a given input power, we need to minimize the start-up time defined as:



(a) Maximum efficiency is limited by the boost and buck converter.

(b) Application's execution rate has a linear dependency with input power.

Fig. 2.16: EMU-based systems are most efficient within a specific input power range. Optimized EMU implementations can have a $P_{\text{sys,min}}$ of almost 10 µW at 380 mV, and a $P_{\text{load,max}}$ of almost 500 mW.

$$t_{start\text{-}up} = \left\{ t \mid V_{cap}(t) = \sqrt{\frac{2\int_0^t E'_{cap}(\tau)\,d\tau}{C_{cap}}} = V_{load} \right\} \tag{2.4}$$

However, the minimum capacitance is limited by the EMU's maximum supported voltage swing, as shown in the following equation:

$$C_{min,i} = \frac{2E_{load,i}}{\eta_{buck}(V_{max}^2 - V_{load,i}^2)}, \tag{2.5}$$

where $E_{load,i}$ and $V_{load,i}$ are the energy and voltage required to execute task i, and V_{max} is the EMU's maximum supported voltage. The optimal capacitor value is then selected as the highest $C_{min,i}$ among all tasks i. An optimized energy storage can both guarantee the atomicity of task execution and also minimize the start-up time. This forms the basis for the reliable execution of batteryless applications, even under variable and unpredictable energy harvesting conditions.

Once the capacitor size has been tuned for any specific application, the EMU can "abstract away" the environment and absorb its power variability. By decoupling the application from the environment, the overall system energy efficiency and application execution rate can be viewed as a function of the input source power (P_{source}), as shown in Figure 2.16. When the input power P_{source} is below the activation threshold $P_{\text{sys,min}}$, EMU-based systems will remain fully powered down and have zero-percent energy efficiency. Satisfying the input power condition is necessary but not sufficient, as there is also a minimum voltage requirement, typically $V_{\text{source}} > 380\,\text{mV}$, for the harvesting subsystems to self-start. After the system can turn on, the overall energy efficiency jumps and remains relatively constant until the load has reached its maximum power consumption, $P_{\text{load,max}}$. After this threshold is surpassed, the energy efficiency decreases since there is an energy surplus being wasted due to the impossibility to consume or store the energy for later use. This can be seen in Figure 2.16b, where the application's execution rate, and its duty-cycle, increase linearly from 0 % at $P_{\text{sys,min}}$ to 100 % at $P_{\text{load,max}}$. The main difference between different applications will be the slope, which depends on the energy consumed by a single activation. Power-hungry applications will have a lower slope, covering a larger input power range between $P_{\text{sys,min}}$ and $P_{\text{load,max}}$.

2.3.8 Ambient Sensing Using Batteryless Sensors

Batteryless systems with passive elements such as Radio Frequency Identification (RFID) cards have been in wide circulation for decades, but they perform only simple computation, have a small memory capacity, and require specialized readers to energize and communicate with them. More recently, researchers have studied batteryless systems with active components to harvest more abundant primary (naturally occurring) energy to perform complex sensing, processing, and broadcasting. Batteryless systems with active elements such as photovoltaic cells [152], thermoelectric generators, [593] or kinetic energy harvesters [313] can have high power densities.

2.3.8.1 The MiroCard Platform

The MiroCard, first presented in [260], is a batteryless smart-card powered by light. Since MiroCards covered by any light-blocking material cannot be remotely energized, their activation is exclusively on-demand: when a user *chooses* to expose them to light. The MiroCard is less than 2 mm thick, and has a surface area of only 45 mm × 60 mm, as shown in Figure 2.17. The top side is covered by an organic solar panel with an active area of 35 mm × 53 mm, and all electronics are placed on the bottom. Thanks to its optimized hardware and software, the MiroCard is able to harvest enough energy to communicate wirelessly, even in low indoor lighting conditions down to 170 lx. While its component costs are low, several Swiss Francs at high volume, it is indeed more expensive than passive Automatic identification and data capture (AIDC) technologies such as RFID. However, the active batteryless technology behind the MiroCard offers key advantages in addition to higher power densities. The MiroCard's Cortex M3 provides high processing capabilities for advanced applications with secure communication protocols and also features enough memory for Internet application protocols such as Constrained Application Protocol (CoAP).

2.3.8.2 Overview

The MiroCard project is an evolution of the Transient BLE Sensor Node project [630]. The hardware design is based on the EMU first introduced in [259], which proposed current and voltage decoupling between a transducer and the application circuit through *energy bursts*. In doing so, simultaneous optimization of energy harvesting , through MPPT, and application energy, through Dynamic Voltage and Frequency Scaling (DVFS), become possible.

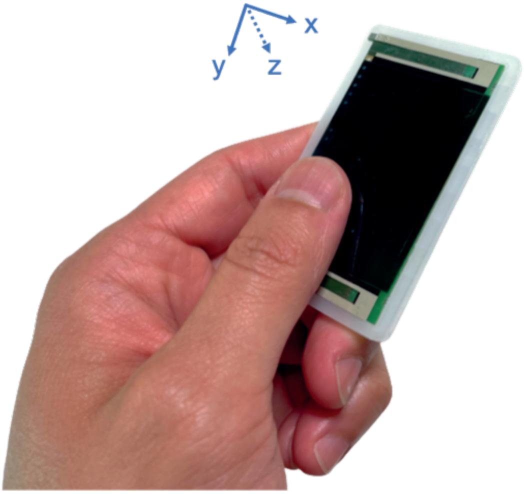


Fig. 2.17: The batteryless MiroCard hosts multiple ambient sensors, including an accelerometer, but can operate only when exposed to light. Since any non-transparent material covering the solar cell prevents the device from sensing, processing, and transmitting, it is immune to RF skimming.

2.3.8.3 Energy Characterization

The MiroCard's power consumption was recorded using a RocketLogger measurement device [631]. A DC source was connected at the V_{cap} point, meaning it supplies the entire card, including the harvester chip, and the application circuit (including the down conversion). This measurement thus encapsulates all of the leakage sources and converter inefficiencies present during batteryless operation. The measured power trace of a single activation can be seen in Figure 2.18, with annotations indicating the system state. Using an external trigger, many activations are recorded, and the average energy consumption of the base BLE application is measured to be 175.31 µJ, including converter inefficiencies. Adding temperature and humidity increases the application energy consumption by around 30 µJ [631]. Two 47 µF ceramic capacitors are enough to guarantee energy bursts of these sizes when $V_{cap} \in [2.8\,\text{V}, 4.37\,\text{V}]$, even if it is less than the chip specification of 150 µF.

2.3.8.4 Start-Up Time Measurements

As discussed previously, one benefit of storing energy in small capacitors is that the RC charging constant is very low, so the system can charge up quickly. Effectively, this means that when a system is completely energy-depleted, it can behave in an energy-opportunistic manner even if the environment sporadically generates small amounts of energy. To fairly measure how fast an EMU-based device wakes up, the system must first be completely depleted, since any leftover charge would artificially decrease the start-up time. We thus define the start-up as the amount of time after light exposure that a fully depleted MiroCard takes to go from fully depleted until it transmits the first BLE packet. To ensure reproducibility and fairness, the MiroCards have V_{cap} and ground shorted before being exposed to different illuminance conditions. They are then placed in a solar testbed [629], which offers a controlled illuminance environment. The start-up time for five different illuminance conditions is measured and recorded.

Figure 2.19 shows one sample measurement. When a fully-off MiroCard is first exposed to light, it enters a startup phase where the solar panel voltage V_{src} is first clamped to 330 mV as it charges its internal capacitors. In this phase, the AEM10941 harvester chip optimizes the charge transfer to its small storage capacitor and quickly stabilizes the regulated V_{cc} voltage, as shown in Figure 2.19. Afterward, the MiroCard enters energy-driven execution where it stays in LPM, consuming only 2.47 µA, as it waits for an EMU trigger. The EMU triggers the application once the maximum capacitor voltage of 4.37 V is reached, and three identification beacons are transmitted. In this experiment, the raw BLE packet size is 42 bytes, containing 25 bytes of advertisement data. The MiroCard can integrate current and historical sensor data (e.g. temperature and humidity) at the cost of slightly larger energy consumption, as presented in [630]. As the environment provides more light, the MiroCard's execution rate increases automatically, thanks to the EMU's energy proportionality and the stateless nature of the application, where each activation is independent. In dynamic environments, MPPT plays an important role in optimizing the energy input, especially if the MiroCard is only exposed to light for short periods of time. The measurements at different luminosity levels are summarized in Table 2.5.

2.3.9 Gesture Detection on the Batteryless MiroCard

This section covers the documentation of implementing gesture recognition on a batteryless smart card. The following discussion provides a brief overview of different

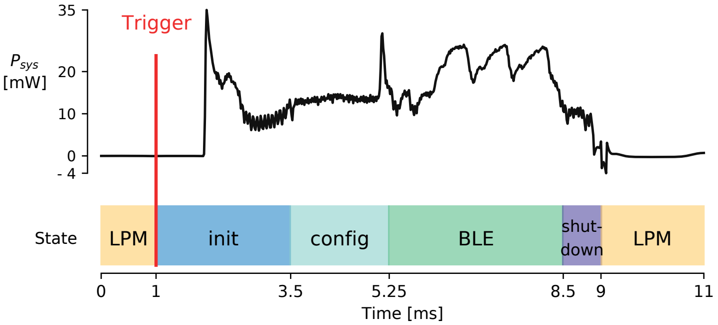


Fig. 2.18: In LPM, the MiroCard's average system current is only 2.47 µA. When triggered, a single activation broadcasts 3 BLE packets and lasts less than 8 ms.

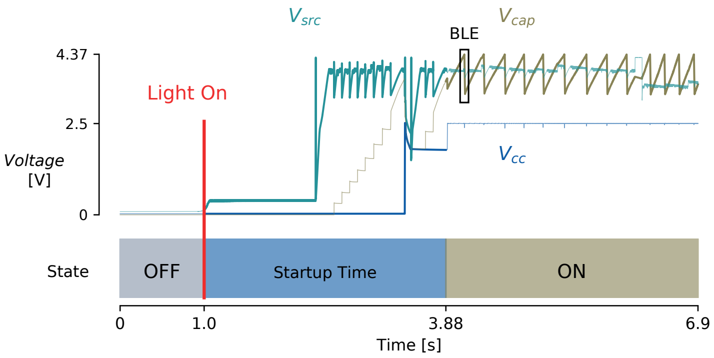


Fig. 2.19: Power-on trace of a MiroCard, indoors with natural and artificial light (2 600 lx). It starts up within 2.9 s and transmits BLE beacons at an average rate of 16.25 pkt/s. BLE transmission is triggered when V_{cap} = 4.37 V (black box).

Tab. 2.5: Performance of ambient sensing application in indoor-light conditions.

Luminosity	Startup time	Average input power	Average comm. rate
2 600 lx	2.88 s	978 μW	16.25 pkt/s
1 000 lx	7.17 s	372 μW	6.17 pkt/s
500 lx	13.62 s	181 μW	2.92 pkt/s
250 lx	33.49 s	85 μW	1.9 pkt/s
170 lx	60.04 s	60 μW	0.75 pkt/s

approaches for gesture detection. Afterward, the methodology to develop a gesture detection model for batteryless embedded systems is discussed in detail.

2.3.10 Approaches to Gesture Detection

The term “gesture recognition” must be narrowed because there are different types of gestures. [498] differentiate diverse forms of human gesture detection. Full-body motion, for example, analyzes people’s body movements in sports or rehabilitation. Facial expressions, i.e. head gestures, allow the tracking of eye movements or the estimating of a person’s mood. The third gesture type refers to hand or arm gestures. This work focuses on the latter since people will use their hands to handle a smart card the size of a credit card. Therefore, the term gesture recognition is used as a synonym for hand gesture recognition in the further course of this work. There exist different approaches for detecting hand gestures explained in the following parts.

Camera One method is gesture detection using cameras. Several works [125, 641, 667] have demonstrated a gesture detection system that processes a video stream from a camera. Once a specific gesture has been detected, the system can trigger different actions.

Acceleration As discussed earlier, the light-powered MiroCard contains an accelerometer, which can measure forces induced by gesture movements. A competing technology known as Electromyography (EMG) technology can measure the electrical activity produced by muscles to detect certain gestures, though some argue is still too expensive from a financial, computation- and power-consuming perspective [380]. While we focus on detecting gestures performed on a handheld device, other works have studied gesture detection with one or more accelerometers at different locations. In [181], the authors used accelerometer data recorded from the wrist for gesture detection. [677] propose equipping each finger with an accelerometer to feed the trained support vector classifier model with acceleration data. This approach allows recognizing each finger's position to detect deaf language letters to simplify translation and communication. All the approaches presented above work on batteries or with power cables. Some related works suggest batteryless gesture recognition. [653] introduced a SmartWheelTag to recognize hand gestures based on changes in RFID patterns. [670] present the CapBand, a wristband with an ultra-low power design, to detect gestures using environmentally harvested energy and a capacitance sensing system. The prototype for demonstration purposes harvests energy using a solar panel. [360] show a wristband architecture with flexible solar panels covering the whole band. Gesture recognition relies on EMG technology in this case. [436] use photodiodes for energy harvesting and gesture detection. All photodiodes harvest enough energy to process an algorithm that predicts gestures based on data from fluctuations in ambient light supply. This approach allows the detection of finger motions. The question of which approach best suits gesture recognition on a smartcard arises since most presented prototypes are worn on the wrist or attached to fingers. One related work is from [551], who propose an RFID tag combined with an accelerometer for gesture recognition for access permission checking. A trained k-nearest neighbor model recognizes the external data from the RFID tag after transmission. Nevertheless, this contribution utilizes a different approach, which will be discussed in the following part.

2.3.11 Batteryless Machine Learning

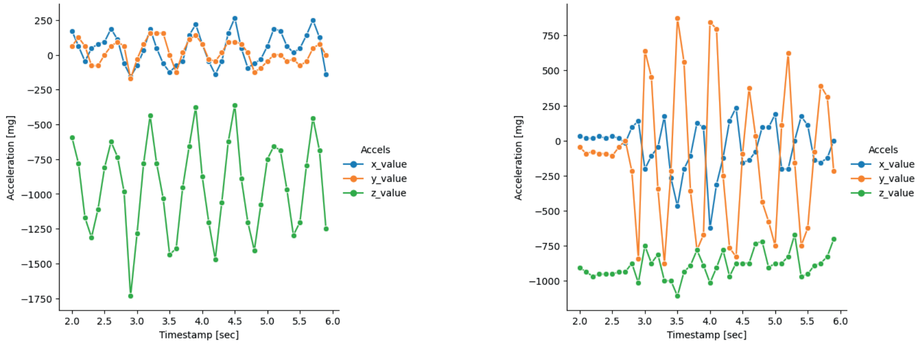
Lightweight Neural Nets Running a machine learning model in a batteryless system imposes stringent restrictions on the model. The model must execute quickly, and have low memory and energy consumption requirements for light-powered, real-time gesture recognition. infXL offers a Deep Neural Network (DNN) called lightweight (Lt-Wt) net introduced in [370]. Lt-Wt net is specially developed for resource-constrained applications.

Convolutional Neural Networks (CNN) generally require many more operations and memory. However, the proprietary Lt-Wt model reduces operations, memory footprint, and logic to improve the network's speed and lower costs for storage and processing. These adaptions result in higher energy efficiency and allow local processing on a low-power device. Its lightweight architecture simplifies porting the model's code to different platforms. It basically contains four elements: one RAM array for inputs and outputs, a lookup table as ROM, network definitions as ROM, and control rules.

When designing a new machine learning application, the first step is typically data acquisition. In our scenario, where accelerometer data will be classified into different gestures, gathering high-quality data using the MiroCard is very important. Existing datasets for different gestures would have different signatures since even the weight distribution on the card can change the way the user movement is registered. We thus recruited multiple users to record a new dataset with three classes of gestures using the MiroCard. The recorded raw data needs preprocessing before using the infXL toolchain, which automatically trains, tests, and validates the model. We will now discuss in detail these steps and their outcomes. An accurate gesture classification model can be integrated into the MiroCard as a simple human-machine interface to trigger different actions.

Data Acquisition We implemented the following data acquisition process. The MiroCard is powered by a cable connected to a Raspberry Pi. This also allows the transmission of the accelerometer's data from the MiroCard to the RPi, where it will be stored in separate files. Data recording is performed with multiple users, which physically interact with the MiroCard. First, a quick explanation session aims to introduce the tasks to the participant. Then, a participant must record data for each gesture. The user shakes the card sideways, then up/down, and finally, they move it randomly in a way that is distinctly not sideways or up/down. It must be stated that the participant must not stick to one position but has to change the card orientation. Therefore, shaking the MiroCard upwards and downwards can also happen along different axes. A recording session for one gesture lasts around three minutes. The participant makes a break after each 3-minute session to restart data logging. This way of splitting the gesture recordings simplifies data labeling. Therefore, the total time for data recording is between ten and fifteen minutes per participant. Finally, each file contains a time series of approximately 1800 data samples, given that the accelerometer operates at 10 Hz for 3x60 seconds. One data sample consists of an x-, a y-, and a z-value. A total of 12 users participated in the data acquisition process.

Figures 2.20b and 2.20a show windows of 4 seconds for each gesture using the same user as the above images. These plots provide a better view of the data behavior of each gesture. The first figure, 2.20a, shows a large difference between the peaks and bottoms of the z-values while the x- and y-values remain almost stable and accordingly represent shaking the card upwards and downwards. Furthermore, the observation



(a) Up/Down gesture close-up.

(b) Sideway gesture close-up.

Fig. 2.20: Four-second close-ups of the XYZ data values for two gestures. Oscillations are noticeable in all data channels, with different amplitudes depending on the direction of movement.

that the z-values mean is around 1g allows deducing that this plot represents a shaking upwards and downwards. The graphs in Figure 2.20b indicate sideways shaking since the z-values' changes are relatively small compared with the values of the y-axis. A *random* gesture provides a reference movement different from the previous patterns. These include both random movements in 3D space, as well as standing still in different orientations.

Model Training and Validation To train the Lt-Wt model, the entire labeled dataset must be partitioned for training and testing. The training subset is split into features and labels and then used to train multiple Lt-Wt models via TensorFlow and supplementary algorithms. The models are evaluated with the testing dataset to estimate precision, recall, F1-score, and accuracy measures. A model picker finds the final network that meets the application's requirements. Here, we present the evaluation of the final Lt-Wt network.

2.6 shows the original dataset containing over 68 000 data samples from recordings of 12 participants (2 female, 10 male). The infXL toolchain splits the balanced subset into training and testing data with a ratio of 69:31. 30 % of the training dataset is used for validation. Therefore, 48.3 % of the total balanced subset, or 32,865 data samples, is used for training. 2.6 also shows that the distribution of the three classes is even. The reason for the difference is that some recordings are not terminated exactly after three minutes. The overflow is not cut out to maximize the ultimately usable dataset while balancing.

Classification Model Accuracy The model's performance is evaluated with the testing dataset to estimate precision, recall, F1-score, and accuracy measures. The evaluation results for the model accuracy can be seen in Table 2.7. The F1-score for all

Tab. 2.6: Distribution of labels and data within the dataset

Label distribution		
	Absolute size	Relative size
Total exemplars	68 036	100 %
Class 0	22 583	33.2 %
Class 1	22 357	32.9 %
Class 2	23 096	33.9 %
Data distribution		
	Absolute size	Relative size
Training exemplars	32 865	48.3 %
Validation exemplars	14 085	20.7 %
Testing exemplars	21 086	31 %
Total	68 036	100 %

three classes is constantly over the 90 % mark. Overall, the classification model has an accuracy of 94 %. The model performs best in differentiating between sideways and up/down, whereas the biggest weakness is distinguishing between sideways and random since the model is wrong in 581 cases out of 14 379. It is important to remember that the model is capable of generalizing the device motion, regardless of the orientation. This implies that the classification is quite robust and will be able to recognize the gestures among different users.

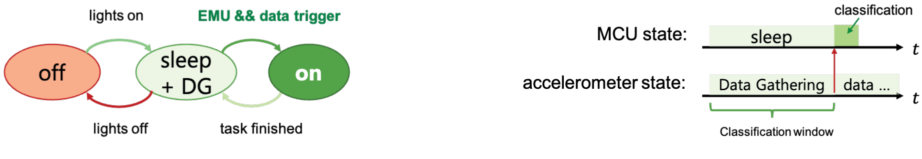
Tab. 2.7: Evaluation of model performance

Class	Precision	Recall	F1-Score	Sample size
Class 0	0.92	0.93	0.92	6933
Class 1	0.95	0.95	0.95	6707
Class 2	0.95	0.94	0.95	7446

2.3.12 Batteryless Classification of Time Series Data

The gesture detection models all require *time series data* comprising 20 data samples, equaling 60 input values (1 sample contains one X, Y, and Z point each). 20 data samples equal a time window of 2 seconds since the accelerometer operates at 10 Hz. These numbers have been chosen to accommodate human-made gestures of different lengths. Though accelerometers have a wide range of possible sampling frequencies, 10 Hz was chosen as a good trade-off between the power consumption and the required sampling frequency for human movement).

This data dependency for the classification task fundamentally changes the energy-driven behavior of traditional EMU-based batteryless sensing systems. As shown in Figure 2.21, it is no longer enough to have buffered the energy required by the classification task, since 20 data samples are now required. Effectively, the behavior with gesture classification is time-triggered, and can only occur when the environmental energy is enough to support it. If this is the case, then the 20 samples are copied from the accelerometer's FIFO buffer to the microcontrollers as soon as they are ready. In its simplest form, batteryless gesture detection does not allow an overlapping windowing approach, where the stride length can be smaller than the window size. The reason is that the data classification would no longer be stateless, and would demand a higher frequency of processing the Lt-Wt network and, therefore, increased energy consumption.



(a) Finite State Machine.

(b) Gesture Classification Timeline.

Fig. 2.21: Batteryless systems processing time series data require not just the *energy ready* signal from the EMU. Gesture classification imposes a data requirement that must be met at the same time as the energy requirement.

2.3.13 Experimental Evaluation

To characterize the active energy requirements of the gesture detection application, a set-up similar to the presented in Section 2.3.8.3 was used. A DC source provided a stable voltage to the MiroCard, whose current and voltage are recorded by the RocketLogger. Additional GPIOs are used by the application to signal its internal state. This information is then used to annotate the power trace. Figure 2.22 shows the energy consumption for gesture detection and BLE transmission. The EMU triggers the MCU after 2 seconds if there is enough harvested energy to process the Lt-Wt and send the result as a BLE beacon. The time window of 2 seconds relies on the accelerometer's sensing frequency of 10 Hz and ensures the availability of 20 data samples. This trigger is marked by the red line. The classification process takes around 27 ms after the initial boot-up and configuration. The Lt-wt net's power consumption is relatively stable, between 20 and 35 mW. The three peaks between 10 mW and 20 mW at the very end represent the individual BLE transmissions. The entire activation consumed 723 µJ, and is dominated by the classification task. After the energy consumption burst finishes, the processor shuts down to minimize the consumption. The accelerometer, however, remains continuously enabled to gather data, which brings the sleep current consumption to 26 µA between energy bursts.

After this energy characterization, it is determined that the system requires an equivalent capacity of 1 mF, which is enough to guarantee energy bursts of the required size when $V_{cap} \in [2.8\,\mathrm{V}, 4.37\,\mathrm{V}]$. To understand how this storage element affects reactivity, the start-up time was measured. Comparable to the methodology described in Section 2.3.8.4, the capacitors were shorted before exposing them to light. This measurement was made in an indoor environment, combining natural and artificial light, with a combined illuminance of 900 lx measured with an illuminance meter. The solar panel current and voltage, and the capacitor voltage were measured using the RocketLogger. Figure 2.23 shows the measurement trace, which indicates that the gesture detection application can start up within 11 s, and sustain data acquisition, classification, and transmission under that illuminance condition.

2.3.14 Analysis

In previous sections, we have introduced two batteryless applications running on the light-powered MiroCard platform: ambient sensing and gesture detection. The first is a purely energy-driven application, where a wireless sensor transmits information about the environment. This first application is energy-driven because each sensor activation is stateless and thus independent from the other. The sensor's activation frequency automatically adjusts to balance the energy flow as the environmental energy changes throughout the day. This energy proportionality has been a key characteristic in most batteryless sensing systems, including the MiroCard. Thanks to this principle, MiroCard users have full agency over the device's operation. If a user stores the MiroCard in a non-

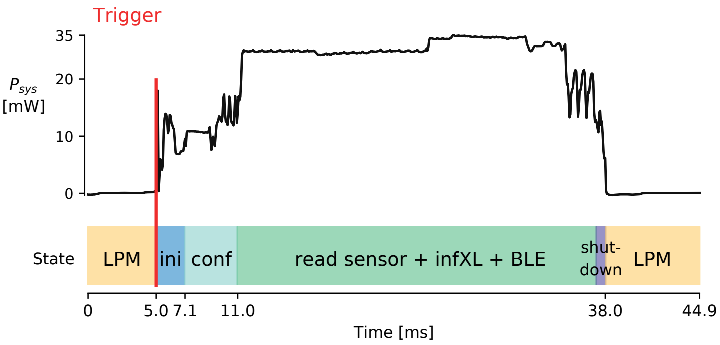


Fig. 2.22: The gesture application keeps the accelorometer enabled between activations of the classification process, bringing the sleep current to 26 µA. When enough data has been buffered, a single activation classifies the movement and broadcasts 3 BLE packets, lasting less than 33 ms.

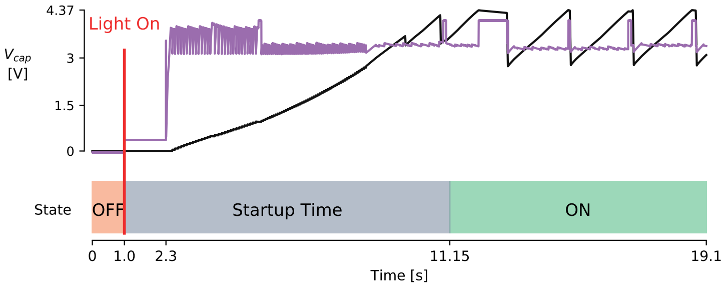


Fig. 2.23: Start-up time for the gesture detection application, measured indoors with natural and artificial light (900 lx). It starts up within 10.15 s. After this, the movement classification and BLE beacon transmission occur every two seconds.

transparent location, it is physically impossible to energize the batteryless MiroCard. However, as soon as the user decides to expose it to light, the transducer is able to produce energy, and only then will the MiroCard start sensing and transmitting data.

2.3.15 Energy Proportionality vs Time Series Processing

The second application we have discussed is gesture detection on the batteryless MiroCard. As opposed to the ambient sensing scenario, each sensor activation has strict timing and data dependency on the previous one. The classification model requires 20 data samples. At the specified sampling frequency, this requires 2 s. If the MiroCard is unable to harvest enough energy for classification during those two seconds, either data will be lost or the processing will fail. This is a fundamental limitation of using batteryless sensors: there is no guarantee that energy can be harvested fast enough to sustain a minimum service level. We have shown, however, that during those time periods where the environmental conditions can sustain the MiroCard, it is able to properly classify gestures in non-overlapping data windows. The non-overlapping part is a limitation that arises from the lack of data retention in the chosen LPM for the microcontroller. In practice, an SRAM block can be kept on with the accelerometer data buffer. However, this increases the sleep current further and can decrease the overall energy efficiency of the system. A side effect of the non-overlapping detection windows is that if a gesture happens to be split by this artificial division, it is possible that the classifier will not detect it. Consequently, the model's high accuracy will not be directly visible if the gesture is too short. Luckily, users gesturing with the MiroCard typically do so for several seconds, so this issue is mitigated.

2.3.16 Contextualizing Indoor Energy Harvesting

At the beginning of this contribution, we have argued that indoor photovoltaics allows sensing systems to tap into a vast source of primary energy. This primary energy comes in two flavors: natural sunlight shining into indoor spaces and artificial lighting installed for human use. Though the average energy harvested in one day can vary greatly depending on the location, time of year, and human presence, it can be as high as tens of joules per day. For statically deployed MiroCards doing ambient sensing, this can translate to over 28 000 activations per day, or roughly 1 activation (and three packets) per second during working hours, even under the conservative estimates from artificial lighting only. In essence, the MiroCard acts as a batteryless information fountain providing long-term, maintenance-free data, even in dimly lit environments. Increased energy efficiency is just one of the benefits of batteryless sensors. For wearable applications, the fast start-up time is another key differentiating feature. Users can, for privacy and security concerns, ensure that MiroCards are completely powered off by just covering the photovoltaic panel. When a user chooses to utilize a MiroCard, it must get exposed to light, and it should self-power as fast as possible. This naturally depends on the illuminance conditions. We have shown how a gesture-detecting MiroCard can turn on within 11 s when the environment provides 900 lx, which is easily achieved with the combination of natural and artificial light indoors.

2.3.17 Conclusions

This contribution presents a new class of batteryless sensing systems, capable of gathering data in an energy-efficient, scalable, and environmentally friendly manner. Due to their high power density and low cost, we focus primarily on photovoltaic energy harvesting, which can utilize widely available indoor lighting in human-occupied areas. We also present a multi-year dataset of indoor energy harvesting measurements in an office building, demonstrating the potentially large, but highly variable daily energy budgets. Even if the instantaneous harvested power can vary significantly in short time periods, Energy Management Unit (EMU)-based designs can reliably execute batteryless sensing applications. Though the designer cannot directly control how much energy is available, it is possible to control under what conditions the application executes. By buffering small amounts of energy, designers can control the operating voltage and maximize the amount of work done per unit of energy. These small amounts of energy are enough to execute a wide variety of sensing applications. We demonstrate a batteryless ambient sensor that can be deployed in a fixed position, and generate thousands of packets per day. These batteryless sensors are more efficient than their battery-based counterparts because, during the night period, they are completely powered off, consuming zero power. This contribution also introduces a gesture detection application on a batteryless smartcard. Using manually gathered data, a machine learning model was trained and deployed to the smartcard. The extensive experimental evaluation validates the reliability and energy efficiency of EMU-based designs. This new class of batteryless sensing systems is capable of executing complex sensing applications and promises a more scalable and cost-effective approach to distributed data gathering for IoT systems.

3 Streaming Data, Small Devices

Big data often stem from sensors that stream their measurements continuously. Imagine for instance an embedded system that acts upon certain sensor inputs. *Data streams* naturally occur in the Internet of Things, embedded systems and cyber-physical systems. In particular, we encounter them in all kinds of *small devices* that have strictly limited resources like limited energy, communication bandwidth, memory, and computational power.

To model those scenarios from an algorithmic perspective and to quantify the trade-off between the required resources and the accuracy achievable by a learning algorithm, the algorithmic community has introduced the streaming model [523]. A data stream algorithm makes one pass[1] over the data, presented in N items one by one. Hereby, it maintains a summary of the stream whose size is limited to a *sublinear* amount, often polylogarithmic in N or even constant. We distinguish between different streaming models with increasingly dynamic updates:

Insertion-only data stream New items are only *added* to the data stream. The algorithm processes one item after another and views each item as a new instance.

Dynamic data stream Items can be *added* or *deleted* from the data stream. The algorithm processes the insertion or deletion of an item one after another. New items can be added to the stream or already processed items can be removed from the stream.

Turnstile data stream Every single feature or coordinate of data items can be modified by *adding* or *subtracting* values to update the current state. This is the most general case in which an algorithm needs to process the insertion of new items and updates to old items within the stream including their removal from the stream when subsequent updates add up to zero.

This chapter shows general algorithmic approaches to process and summarize streaming data and surveys recent research in this area, including several contributions of the CRC 876. It also highlights the importance of these topics for teaching so that the next generation of researchers and practitioners may tackle future challenges in this area.

Section 3.1 on summary extraction from streams presents an insertion-only data stream algorithm to maximize submodular functions, which are very important and have many applications. Prominent examples include maximizing entropy, and mutual information of selected subsets of data. The section surveys several state-of-the-art algorithms for the problem and gives an own technical contribution. It covers algorithmic and analytical methods for data streams and relaxations of worst-case conditions to

[1] The number of passes is often relaxed to a small constant or logarithmic amount for problems where single pass algorithms are impossible to obtain or where a multi pass algorithm allows significantly improved results over what is possible in a single pass.

https://doi.org/10.1515/9783110785944-003

model *typical* behavior via probabilistic assumptions. The section may serve as a basis for one lecture.

Section 3.2, on coresets and sketches, introduces general concepts for summarizing data streams with respect to specific computational problems such as regression, classification, and clustering. It covers a brief technical introduction to coresets and sketches and highlights their importance for the design of data stream algorithms. It surveys the state of the art with a focus on contributions within the CRC 876. Each subsection introduces one of the main research directions and provides briefly the central ideas behind the results. The section may serve as a basis for a seminar or short lecture series on the topic.

3.1 Summary Extraction from Streams

Sebastian Buschjäger
Katharina Morik

Abstract: As processing capabilities increase, more and more data is gathered every day everywhere on earth. While machines are becoming more and more capable of dealing with these large amounts of data, humans cannot keep up with the amount of data generated every day. They need small and comprehensive representative samples of data, which capture all the informative parts of the data, in other words: a data summary. Formally, we formulate the data summarization problem as a function maximization problem with a cardinality constraint in which we seek to maximize a utility function f while selecting up to K elements in total.

Due to their compelling theoretical properties, submodular functions have been widely adopted as a utility function for data summarization. Submodular functions are set functions that reward adding a new element to a smaller set more than adding the same element to a larger set and thereby naturally lead to small and comprehensive summaries. This fits the restricted resources of small devices. We want to do a step further and model the summarization as a streaming algorithm. Streaming algorithms evaluate each data item once and decide immediately, on-the-fly with a limited memory budget, if an item should be added to the summary or not. These algorithms can be run on small, embedded devices *while* data is generated and thereby provide a data summary *anytime* with minimal computational costs.

In this contribution, we discuss the framework of submodular functions in more detail and survey the current state of the art for streaming submodular function maximization. We analyze each algorithm for performance guarantees as well as runtime and memory consumption. We end the contribution with a comprehensive comparison between algorithms for real-world summarization tasks over data streams with and without concept drift.

3.1.1 Introduction

While computers can process terabytes of data within seconds, humans are often overwhelmed with the sheer amount of information available. Humans can inspect and interact well with small, representative samples of data. Such a *data summary* must capture all the informative parts of the data while being small and comprehensive.

In recent years, submodular optimization has found its way into the toolbox of machine learning and data mining. It offers a well-established mathematical framework to select small and comprehensible summaries for a variety of different tasks. The field of online submodular optimization studies algorithms that view each item only once and then either add it to the summary or discard it.

Exploiting submodular optimization for summarization faces algorithmic challenges right where it is needed most, namely, in the context of the Internet of Things (IoT), particularly regarding sensor networks and distributed processing, that needs to be communication-aware and energy saving. Most of the data is produced by small embedded electronics with limited processing and limited storage capabilities. Thus, a data summary should be captured *on-the-fly* while the data is being generated and before storing it. Currently, the best performing online algorithms offer an $\mathcal{O}(\frac{1}{2} - \varepsilon)$ approximation ratio where ε also influences the memory consumption of the algorithm. Even moderate choices for ε quickly result in an unmanageable resource consumption. Feldman et al. [220] showed that this approximation ratio is the best possible for data stream algorithms and that any algorithm with a better worst-case approximation guarantee essentially stores all the elements of the stream (up to a polynomial factor in K, where K is the summary size).

Existing algorithms are designed for the mathematical *worst case* and thereby have a worst-case approximation guarantee. We argue, that most practical applications are much more well-behaved. This insight allows us to move beyond the worst case and design an algorithm that delivers a good data summary under moderate assumptions. The resulting algorithm offers a probabilistic approximation ratio of $(1-\varepsilon)(1-1/\exp(1))$ with high probability $(1-\alpha)^K$, where α is the desired user certainty and K is the summary size. It performs $\mathcal{O}(1)$ function queries per item and requires $\mathcal{O}(K)$ memory. Note, that this result does not contradict the upper bound of $\mathcal{O}(\frac{1}{2}-\varepsilon)$ from [220] since our algorithm offers a better approximation quality with high probability, but not for the worst case.

In the next section we will discuss the framework of submodular function maximization. After that, we discuss existing algorithms, whereas Section 3.1.4 details the novel `ThreeSieves` algorithm. Section 3.1.5 presents practical experiments and Section 3.1.6 concludes the contribution. Parts of this text were previously published as a conference paper in [109].

3.1.2 Submodular Function Maximization over Streams

In this contribution, we consider the problem of maximizing a submodular function over a data stream and focus on the task of data summarization. More formally, we consider the problem of selecting K representative elements from a ground set D into a summary set $S \subseteq D$. To do so, we maximize a non-negative, monotone submodular set function $f\colon 2^D \to \mathbb{R}_+$ which assigns a utility score to each subset:

$$S^\star = \underset{S \subseteq D, |S|=K}{\arg\max} f(S) \tag{3.1}$$

For the empty set, we assume zero utility $f(\emptyset) = 0$. We denote the maximum of f with $OPT = f(S^\star)$. A set function can be associated with a marginal gain which represents the increase of $f(S)$ when adding an element $e \in D$ to S:

$$\Delta_f(e|S) = f(S \cup \{e\}) - f(S)$$

We call f *submodular* iff for all $A \subseteq B \subseteq D$ and $e \in D \setminus B$ it holds that

$$\Delta_f(e|A) \geq \Delta_f(e|B)$$

The function f is called *monotone*, iff for all $e \in D$ and for all $S \subseteq D$ it holds that $\Delta_f(e|S) \geq 0$.

In general, the maximization of a submodular set function is NP-hard [214], which makes solving Equation 3.1 difficult. Therefore, a natural approach is to find an approximate solution. Nemhauser et al. [230] presented a simple $(1 - (1/\exp(1))) \approx 63\,\%$ greedy approximation algorithm (denoted as `Greedy` in this contribution) for solving Equation (3.1) which runs in linear time and requires a fixed memory budget. Greedy offers a constant approximation guarantee and only requires $\mathcal{O}(K)$ memory. The disadvantage is that it requires K iterations over the entire ground set, which is costly if the ground set is very large. Moreover, multiple iterations are impossible for streaming data. Several streaming algorithms have been proposed that read each item exactly once (when D is stored on disk) or process it once on arrival (a 'true' streaming setting). An overview of these algorithms and their theoretical properties can be found in Table 3.1. It is noteworthy that the majority of these algorithms achieve a $1/2 - \varepsilon$ approximation guarantee, where ε is the desired approximation quality. A recent analysis by Feldman et al. in [220] implies that this approximation ratio is the best possible in a streaming setting and any algorithm with a better *worst-case* approximation guarantee essentially stores all the elements of the stream (up to a polynomial factor in K). Unfortunately, for all these algorithms the memory budget and the number of function evaluations per item depend on ε. Even a moderate choice of ε turns memory and runtime requirements unmanageable for small devices.

We recognize, that the worst case is often a pathological case whereas practical applications are usually much more well-behaved. Therefore, some papers recently proposed to *ignore* these pathological cases and develop algorithms with a *better* approximation guarantee in *most* cases, while using fewer function queries and less memory [109, 485, 517]. The first algorithm for monotone submodular function maximization with cardinality constraints that ignores edge cases is the `Three Sieves` algorithm proposed in [109]. It estimates the probability of finding a more informative data item on-the-fly and only adds those items to the solution that are unlikely to be 'out-valued' in the future. The resulting algorithm offers a *probabilistic* approximation ratio of $(1 - \varepsilon)(1 - 1/\exp(1)) > 1/2 - \varepsilon$ with probability $(1 - \alpha)^K$, where α is the desired user certainty. It performs $\mathcal{O}(1)$ function queries per item and requires $\mathcal{O}(K)$ memory.

Tab. 3.1: Algorithms for non-negative, monotone submodular function maximization with cardinality constraint K. `ThreeSieves` offers the smallest memory consumption and the smallest number of queries per element in a streaming-setting. Adapted from [109].

Algorithm	**Approx. Ratio**	**Memory**	**Queries**	**Stream**	**Ref.**
Greedy	$1 - 1/\exp(1)$	$\mathcal{O}(K)$	$\mathcal{O}(1)$	✗	[230]
StreamGreedy	$1/2 - \varepsilon$	$\mathcal{O}(K)$	$\mathcal{O}(K)$	✗	[258]
Random	$1/4$	$\mathcal{O}(K)$	$\mathcal{O}(1)$	✓	[215]
PreemptionStreaming	$1/4$	$\mathcal{O}(K)$	$\mathcal{O}(K)$	✓	[91]
IndependentSetImprovement	$1/4$	$\mathcal{O}(K)$	$\mathcal{O}(1)$	✓	[121]
SieveStreaming	$1/2 - \varepsilon$	$\mathcal{O}(K\log(K)/\varepsilon)$	$\mathcal{O}(\log(K)/\varepsilon)$	✓	[32]
SieveStreaming++	$1/2 - \varepsilon$	$\mathcal{O}(K/\varepsilon)$	$\mathcal{O}(\log(K)/\varepsilon)$	✓	[367]
Salsa	$1/2 - \varepsilon$	$\mathcal{O}(K\log(K)/\varepsilon)$	$\mathcal{O}(\log(K)/\varepsilon)$	(✓)	[540]
ThreeSieves	**$(1 - \varepsilon)(1 - 1/\exp(1))$ with prob. $(1 - \alpha)^K$**	**$\mathcal{O}(K)$**	**$\mathcal{O}(1)$**	✓	**[109]**

3.1.3 Related Work

For a general introduction to submodular function maximization, we refer interested readers to [393] and for a more thorough introduction into the topic of streaming submodular function maximization to [124]. Most relevant to this contribution are non-negative, monotone submodular streaming algorithms with cardinality constraints. To the best of our knowledge, there exist six different algorithms which we survey here. The theoretical properties of each algorithm are summarized in Table 3.1.

While not a streaming algorithm, the `Greedy` algorithm [230] forms the basis of many algorithms. It iterates K times over the entire dataset and greedily selects the element with the largest marginal gain $\Delta_f(e|S)$ in each iteration. It offers a $(1-(1/\exp(1))) \approx 63\,\%$ approximation and stores K elements. `StreamGreedy` [258] is its adaption to streaming data. It replaces an element in the current summary if it improves the current solution by at least v. It offers an $1/2 - \varepsilon$ approximation with $\mathcal{O}(K)$ memory, where ε depends on the submodular function and some user-specified parameters. The optimal approximation factor is only achieved if multiple passes over the data are allowed. Otherwise, the performance of `StreamGreedy` degrades arbitrarily with K (see the Appendix of [32] for an example). We therefore consider `StreamGreedy` not to be a real streaming algorithm. Similar to `StreamGreedy`, `PreemptionStreaming` [91] compares each marginal gain against a threshold $v(\mathcal{S})$. Here, the threshold dynamically changes depending on the current summary $\mathcal{S}$, which improves the overall performance. It uses constant memory and offers an approximation guarantee of $1/4$. Feige et al. show in [215] that for any non-negative submodular function a uniformly chosen random set is a $1/4$ approximation. A uniform random set over a data stream can be obtained via reservoir sampling [688]. Also Chakrabarti and Kale proposed in [121] a streaming algorithm with approximation guarantee of $1/4$. Their algorithm stores the marginal gain of each element upon its arrival and uses this ‘weight’ to measure the importance of each item. We call this algorithm `IndependentSetImprovement`. Norouzi-Fard et al. [540] propose a meta-algorithm for submodular function maximization called `Salsa`, which uses different algorithms for maximization as sub-procedures. The authors argue, that there are different types of data streams and for each stream type, a different thresholding rule is appropriate. Their algorithm offers a $1/2 - \varepsilon$ approximation, but some of the thresholding rules require additional information about the data stream such as its length or density. Since this is unknown in a true streaming setting, this algorithm is not completely streaming-capable.

The first real streaming algorithm with $1/2 - \varepsilon$ approximation guarantee was proposed by Badanidiyuru et al. [32] and is called `SieveStreaming`. `SieveStreaming` tries to estimate the potential gain of a data item before observing it. Assuming one knows the maximum function value OPT beforehand and let $|S| < K$, then an element e is added to the summary S if the following holds:

$$\Delta_f(e|S) \geq \frac{OPT/2 - f(S)}{K - |S|} \tag{3.2}$$

Since OPT is unknown beforehand one has to estimate it before running the algorithm. Assuming one knows the maximum function value of a singleton set $m = \max_{e \in D} f(\{e\})$ beforehand, then the optimal function value for a set with K items can be estimated by submodularity as $m \leq OPT \leq K \cdot m$. The authors propose the management of different summaries in parallel, each using one threshold from the set $O = \{(1+\varepsilon)^i \mid i \in \mathbb{Z}, m \leq (1+\varepsilon)^i \leq K \cdot m\}$, so that for at least one $v \in O$ it holds: $(1-\varepsilon)OPT \leq v \leq OPT$. In a sense, this approach sieves out elements with marginal gains below the given threshold – hence the authors name their approach `SieveStreaming`. Note, that this algorithm requires the knowledge of $m = \max_{e \in D} f(\{e\})$ before running the algorithm. The authors also present an algorithm to estimate m on-the-fly which does not alter the theoretical performance of `SieveStreaming`. Recently, Kazemi et al. proposed in [367] an extension of the `SieveStreaming` called `SieveStreaming++`. The authors point out, that the currently best performing sieve $S_v = \arg\max_v \{f(S_v)\}$ offers a better lower bound for the function value and they propose to use $[\max_v \{f(S_v)\}, K \cdot m]$ as the interval for sampling thresholds. This leads to an algorithm in which sieves are removed once they are outperformed by other sieves and new sieves are introduced to leverage the better estimation of OPT. `SieveStreaming++` does not improve the approximation guarantee of `SieveStreaming`, but only requires $\mathcal{O}(K/\varepsilon)$ memory instead of $\mathcal{O}(K \log K/\varepsilon)$.

3.1.4 Getting More by Doing Less

`SieveStreaming` and its extension offer a worst-case guarantee on their performance and indeed they can be considered optimal, providing that there is an approximation guarantee of $1/2 - \varepsilon$ under polynomial memory constraints in ε and K [220]. However, we also note that this worst case often includes pathological cases, whereas practical applications are usually much more well-behaved. One common practical assumption is, that the data is generated by the same source and thus it follows the same distribution, e.g. for a certain time frame. In this contribution, we want to investigate these better behaving cases carefully. This allows us to present an algorithm that improves the approximation guarantee, while reducing memory and runtime costs in these cases. More formally, we will now assume that the items in the given sample (batch processing) or in the data stream (stream processing) are independent and identically distributed (iid). Note, that we do *not* assume any specific distribution. From a data streams perspective this assumption means, that we ignore concept drifts and assume that an appropriate *concept drift detection* mechanism is in place, so that summaries are, e.g., re-selected periodically. For batch processing this means, that all items in the batch should come from the same (yet unknown) distribution. Please note, that in this case we do *not* assume that all possible samples come from the same distribution, but we merely assume that the sample we are given is consistent in the sense that all items come from *the same* distribution. This is true for *all* data samples, where data items are independent from each other, as we could simply define the overall distribution as a mixture of simpler distributions. We now use this assumption to derive an algorithm with $(1 - \varepsilon)(1 - 1/\exp(1))$ approximation guarantee of high probability.

`SieveStreaming` and its extension maintain $\mathcal{O}(\log\left(K\right)/\varepsilon)$ sieves in parallel, which quickly becomes unmanageable even for moderate choices of K and ε. Both algorithms show the following behavior: many sieves in `SieveStreaming` quickly fill-up with uninteresting events if their novelty threshold is *too small*. `SieveStreaming++` exploits this observation by removing small thresholds early on and focuses on the most promising sieves in the stream. If the novelty threshold is *too large*, both algorithms deliver sieves that never include any item. Actually, there are only a few thresholds that produce small and comprehensive summaries.

The management of many sieves, each with its own threshold might be unnecessary. Instead of using many sieves with different thresholds we use only a single summary and carefully calibrate the threshold: we start with a large threshold that rejects most items, and then we gradually reduce this threshold until it accepts some — hopefully the most informative — items.

As discussed, the set $O = \{(1+\varepsilon)^i \mid i \in \mathbb{Z}, m \le (1+\varepsilon)^i \le K \cdot m\}$ offers a sufficient approximation of OPT. We start with the largest threshold in O and decide for each item whether we want to add it to the summary or not. If we do not add any of these items (the exact threshold T for this will be discussed later) to S we may lower the threshold to the next smaller value in O and repeat the process until S is full.

The key question now becomes: How to choose the threshold T appropriately? If T is too small, we will quickly fill up the summary before any interesting items arrive that would have exceeded the original threshold. If T is too large, we may reject interesting items. Certainly, we cannot determine with absolute certainty when to lower a threshold without knowing the rest of the data stream or knowing the ground set entirely, but we can do so with a bounded probability. More formally, we aim at estimating the probability $p(e|f, S, v)$ of finding an item e which exceeds the novelty threshold v for a given summary S and function f. Once p drops below a user-defined certainty margin τ, i.e.,

$$p(e|f, S, v) \leq \tau$$

we can safely lower the threshold. Now, we have transformed the original problem of choosing the right threshold of utility to that of choosing the right length of T and arrive at the problem of estimating the probability of making the right choice. Moreover, this probability must be estimated on-the-fly. Most of the time, we reject e so that S and $f(S)$ are unchanged and we keep estimating $p(e|f, S, v)$ based on the negative outcome. If, however, e exceeds the current novelty threshold we add it to S and $f(S)$ changes. In this case, we do not have any estimates for the new summary and must start the estimation of $p(e|f, S, v)$ from scratch. Thus, with a growing number of rejected items $p(e|f, S, v)$ tends to become close to 0 and the key question is how many observations do we need to determine—with sufficient evidence—that $p(e|f, S, v)$ will be 0.

The computation of *confidence intervals* for estimated probabilities is a well-known problem in statistics. For example, the confidence interval of binominal distributions can be approximated with normal distributions, Wilson score intervals, or Jeffreys interval. Unfortunately, these methods usually fail for probabilities near 0 [81]. However, there exists a more direct way of computing a confidence interval for heavily one-sided binominal distribution with probabilities near zero [351] when the novelty of items is independent and identically distributed (iid). Then, the probability of not adding one item in T trials is:

$$\alpha = \left(1 - p(e|f, S, v)\right)^T \Leftrightarrow \ln(\alpha) = T \ln\left(1 - p(e|f, S, v)\right)$$

A first order Taylor approximation of $\ln(1 - p(e|f, S, v))$ reveals that

$$\ln\left(1 - p(e|f, S, v)\right) \approx -p(e|f, S, v)$$

and therefore $\ln(\alpha) \approx T(-p(e|f, S, v))$ leading to:

$$\frac{-\ln(\alpha)}{T} \approx p(e|f, S, v) \leq \tau$$

Hence, the confidence interval of $p(e|f, S, v)$ after observing T events is $\left[0, \frac{-\ln(\alpha)}{T}\right]$. For example, with 95 % certainty the confidence interval of $p(e|f, S, v)$ is $\left[0, -\ln(0.05)/T\right]$ which is approximately $[0, 3/T]$ leading to the term *Rule of Three* for this estimate [351]. We can use the Rule of Three to quantify the certainty that with high probability there will not be a novel item in the data stream after observing T items.

Note, that we can set α and the user-defined threshold τ and then compute the minimum of the required number of observations T with the above relationship. Alternatively, we may directly specify the maximum number of observations T as a user parameter instead of α and τ, thus removing one hyperparameter. We call our algorithm

ThreeSieves due to its usage of the Rule of Three. It is depicted in Algorithm 1 and analyzed theoretically in Theorem 1.

Algorithm 1: ThreeSieves algorithm.

```
$O \leftarrow \{(1+\varepsilon)^i \mid i \in \mathbb{Z}, m \le (1+\varepsilon)^i \le K \cdot m\}$
$v \leftarrow \max(O);\ O \leftarrow O \setminus \{\max(O)\}$
$S \leftarrow \emptyset;\ t \leftarrow 0$
for next item $e$ do
    if $\Delta_f(e|S) \ge \frac{v/2-f(S)}{K-|S|}$ and $|S| < K$ then
        $S \leftarrow S \cup \{e\}$
        $t \leftarrow 0$
    else
        $t \leftarrow t+1$
        if $t \ge T$ then
            $v \leftarrow \max(O)$
            $O \leftarrow O \setminus \{\max(O)\}$
            $t \leftarrow 0$
```

Theorem 1. ThreeSieves *has the following properties [109]:*

- *Given a fixed groundset D or an infinite data stream in which each item is independent and identically distributed (iid),*
- ThreeSieves *outputs a set S such that $|S| \le K$ and with probability $(1-\alpha)^K$ it holds for a non-negative, monotone submodular function f: $f(S) \ge (1-\varepsilon)(1-1/\exp(1))OPT$.*
- *It does one pass over the data (truly streaming) and stores at most $\mathcal{O}(K)$ elements.*

3.1.5 Experiments

We now experimentally evaluate the following four questions:

Q1 Is ThreeSieves' performance competitive against the other algorithms in its practical performance given its probabilistic guarantee?

Q2 If ThreeSieves is competitive, how does it relate to a uniform random selection of summaries?

Q3 How does ThreeSieves behave for different T and different ε?

Q4 How large is the resource consumption of ThreeSieves in comparison with the other algorithms?

We ask each algorithm to select a summary with exactly K elements. Since most algorithms can reject items, they may select a summary with fewer than K elements. This makes a comparison between different algorithms difficult, because it favors algorithms with larger summaries (f is monotone and hence adding items to the summary *always* increases the function value), but not necessarily better summaries. For a fair comparison we ensure that all algorithms select a summary of size K by re-iterating over the entire dataset as often as required until K elements have been selected, but at most K times. We compare the relative maximization performance of all algorithms to the solution of Greedy. We also measure the runtime and memory consumption of each algorithm. The runtime measurements include all re-runs, so that many re-runs over the data-set result in larger runtimes.

We will focus on two real-world data-sets. First, the ForestCover [157] data-sets contains 286 048 examples of different forest cover types. Forest cover is the amount of land area that is covered by forest. This proportion is structured into classes. The learning task for this data-set is to predict the class of each cover by using the 10 provided cartographic variables that are obtained via remote sensing. Second, the Creditfraud [443] data-set contains 284 807 fraudulent and legal bank transactions. The learning task for this data-set is to classify each transaction using their 29 features. However, we are interested to see a data summary for a user' manual inspection of the data. Hence, in our experiments we ignore the class information and aim at selecting a diverse set of examples based on the features. More experiments using the novel `ThreeSieves` algorithm can be found in [109].

We extract summaries of varying sizes $K \in \{5, 10, \dots, 100\}$ maximizing the log-determinant

$$f(S) = \frac{1}{2} \log \det(\mathcal{I} + a\Sigma_S). \tag{3.3}$$

$\Sigma_S = [k(e_i, e_j)]_{ij}$ is a kernel matrix containing all similarity pairs of all points in S, $a \in \mathbb{R}_+$ is a scaling parameter, and $\mathcal{I}$ is the identity matrix. In [619], this function is shown to be submodular. Its function value does not depend on the ground-set D, but only on the summary S, which makes it an ideal candidate for summarizing data in a streaming setting. In [111], it is proven that $m = \max_{e \in D} f(\{e\}) = 1 + aK$ and that $OPT \le K \log(1+a)$ for kernels with $k(\cdot, \cdot) \le 1$. This property can be enforced for every positive definite kernel with normalization [268]. In our experiments we set $a = 1$ and use the RBF kernel $k(e_i, e_j) = \exp\left(-\frac{1}{2l^2} \cdot \|e_i - e_j\|_2^2\right)$ with $l = \frac{1}{2\sqrt{d}}$ where d is the dimensionality of the data. We vary $\varepsilon \in \{0.001, 0.005, 0.01, 0.05, 0.1\}$ and $T \in \{500, 1000, 2500, 5000\}$. [2]

We present two different sets of plots. Figure 3.1 contains plots for varying K with a fixed $\varepsilon = 0.001$ (top figure) and plots for varying ε with fixed $K = 50$ (bottom plot). Both figures show the relative performance, the runtime and the memory consumption for different algorithms. Note, that we excluded `Random`, `IndependentSetImprovement`, and `Greedy` for varying ε as their performance is independent of it.

[2] The code for these experiments is available under https://github.com/sbuschjaeger/SubmodularStreamingMaximization/.

Performance over Different K ThreeSieves with $T = 5000$ and Salsa generally perform best with a very close performance to Greedy for $K \geq 20$. For smaller summaries with $K < 20$ all algorithms seem to underperform, yet Salsa and SieveStreaming performing best. Using $T \leq 1000$ for ThreeSieves seems to decrease the performance, which is reflected by the weaker guarantee of the algorithm. On Creditfraud, ThreeSieves performs *better* than Greedy with a relative performance above 100. Note, that only ThreeSieves showed this behavior, whereas the other algorithms never exceeded Greedy. As expected, a uniform random selection shows the weakest performance. SieveStreaming and SieveStreaming++ show identical behavior.

Please, note the logarithmic scale of the runtime. Here, we see that ThreeSieves and Random are by far the fastest methods. Using $T = 1000$ offers some performance benefit, but it is hardly justified by the decrease in maximization performance, whereas $T = 5000$ is only marginally slower but offers a much better maximization performance. SieveStreaming and SieveStreaming++ have very similar runtime, but are magnitudes slower than Random and ThreeSieves. Last, Salsa is the slowest method.

Regarding the memory consumption, please note again the logarithmic scale. Here, all versions of ThreeSieves use the least resources as our algorithm only stores a single summary in all configurations. These curves are identical with Random so that only 4 instead of 7 curves can be seen. SieveStreaming and their siblings use roughly two magnitudes more memory since they keep track of multiple sieves in parallel. As expected, SieveStreaming++ uses less memory than SieveStreaming which uses less memory than Salsa.

Performance over Different ε The behavior of the algorithms for different approximation ratios shows a slightly different picture than before. For larger $\varepsilon > 0.05$ the performance of the non-probabilistic algorithms remain relatively stable, but ThreeSieves performance starts to deteriorate. For small $\varepsilon \leq 0.05$ and larger T ThreeSieves and Salsa again perform best in all cases. Again, SieveStreaming and SieveStreaming++ show identical behavior. Regarding runtime and memory consumption we see a similar picture as before: ThreeSieves is by far the fastest method using the fewest resources followed by SieveStreaming(++) and Salsa. Again, note, that ThreeSieves requires the same amount of memory in all configurations and hence we find an overlap in plots.

We Conclude the Experiments In summary, ThreeSieves is competitive to the other algorithms and sometimes even outperforms them. The probabilistic guarantee of the algorithm comes along with a competitive performance in many cases while using fewer resources. In some cases ThreeSieves even outperforms the Greedy algorithm. ThreeSieves works best for small ε and large T. In contrast to the other algorithms, the resource consumption and overall runtime of ThreeSieves does not suffer from decreasing ε or increasing T.

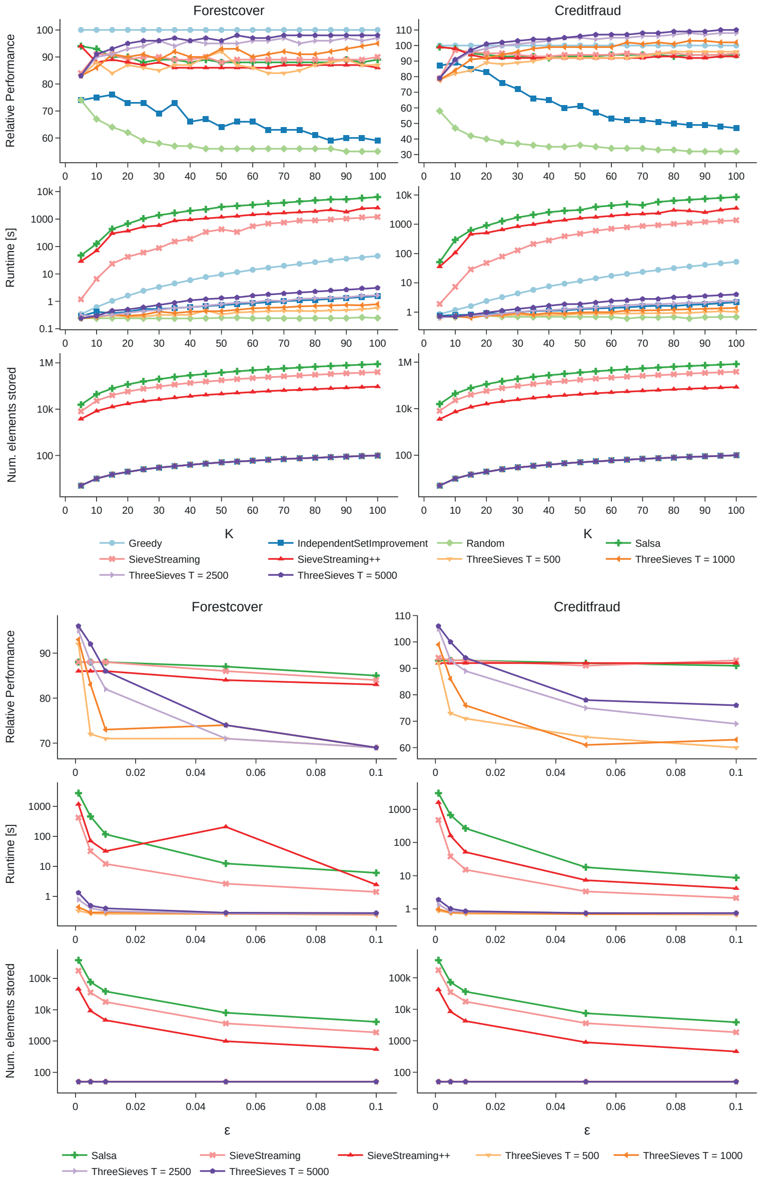


Fig. 3.1: Comparison of IndependentSetImprovement, SieveStreaming, SieveStreaming++, Salsa, Random, and ThreeSieves for different K values with fixed $\varepsilon = 0.001$ (top figure) and different ε with fixed $K = 50$ (bottom figure). The first row shows the relative performance to Greedy (larger is better), the second row shows the total runtime in seconds (logarithmic scale, smaller is better), and the third row shows the maximum memory consumption (logarithmic scale, smaller is better). Each column represents one data-set.

3.1.6 Conclusion

Data summarization is a valuable tool for humans to inspect and understand large amounts of data at a quick glance. For complex and long running processes these summaries must be selected online while the data generating process takes place. While the quality of a summary can be highly subjective to the task and person, submodular functions offer a well-established mathematical framework to produce small and comprehensible summaries for a variety of different tasks. In this Section, we discussed submodular functions and their maximization for data summarization. We focused on the task of stream summarization in which each item is evaluated only once and it must be decided on-the-fly whether it should be added to the summary or not. We reviewed existing algorithms and their theoretical properties in this realm. They are optimized towards the worst-case, whereas practical problems are often much more well-behaved, in particular the data inside the stream are most often independent and identically distributed (iid). This allows the `ThreeSieves` algorithm to compute good summaries with high probability. We experimentally showed that `ThreeSieves` not only outperforms the current state of the art, but also uses fewer resources by an order of magnitude. The algorithm is designed such that kernel functions can be chosen. This enables a more interactive data exploration for the human user, by, say, reviewing multiple summaries with different kernel functions in a very short period of time.

3.2 Coresets and Sketches for Regression Problems on Data Streams and Distributed Data

Alexander Munteanu

Abstract: Coresets and sketches are small data summaries for a given computational problem such as regression or clustering. They preserve the cost function for any possible solution up to little distortion and thus serve as a proxy for the original massive dataset during optimization or inference. They have strong aggregation properties such as linearity or mergeability and thus facilitate their construction for data streams as well as for distributed data. Once the data summary is computed, it can be analyzed using a classical algorithm and the result will be provably close to an optimal solution. In summary, this improves the efficiency and scalability and enables streaming and distributed computation using standard offline algorithms.

We show how linear sketching enables streaming and distributed data processing and show how even static off-line coreset constructions can be extended to those flexible computational settings via the Merge & Reduce principle. Next we survey classic sketching and coreset results for ordinary linear regression and show how those can be extended to more sophisticated models, such as Bayesian regression, generalized linear models, and dependency networks. We also show the limitations of data summarization via complementing lower bounds and how natural assumptions and parameterized beyond-worst-case analysis help to overcome those limitations.

3.2.1 Introduction

Developing highly efficient regression approaches is an important research direction that aims at making modern statistical regression methods scalable to large and high-dimensional data and also to settings where computational resources are scarce as is often the case in the Internet of Things (IoT). We pursue this goal via modern data reduction approaches: we have seen in Section 3.1 how direct sampling methods can summarize the items presented in a data stream. Here we focus on two further methods called *sketches* and *coresets*. Those three approaches are arguably the most promising and widely used methods to facilitate the analysis of mass data with provable accuracy guarantees. See [565] for an extensive survey. In recent years a new paradigm called *sketch-and-solve* has been established for dealing with mass data. The idea behind sketch-and-solve is to apply a simple and fast dimensionality reduction technique in a first step to compress the data to a significantly smaller *sketch* of at most polylogarithmic size. Next, as a second step, we feed the sketch as input to a standard solver for the problem. The theoretical challenge is to prove an approximation guarantee for the solution obtained from the sketch with respect to the original massively large dataset. The general algorithmic principle is shown in the following scheme:

$$\begin{tikzcd} X \arrow[r, "\Pi"] \arrow[d] & \Pi(X) \arrow[d] \\ f(\beta|X) \arrow[r, phantom, "\approx_\varepsilon"] & f(\beta|\Pi(X)). \end{tikzcd}$$

The classical way of data analysis is indicated by the left path, where we would feed the massive dataset X directly to the algorithm and perform the computationally demanding data analysis or learning task indicated by $f(\beta|X)$. This might not even be possible when the data does not fit in main memory or we encounter other computational restrictions. Instead, we follow the path to the right, where the massive dataset X is reduced via a dimensionality reduction mapping Π to obtain a significantly smaller data summary $\Pi(X)$ that is simple to calculate. The latter now fits into main memory and can be given as input to the classical algorithm for an efficient data analysis. The bottom line indicates that the result from analyzing the massive data is close to the result obtained from the sketch. A comprehensive example is given in [245] where 2 TB of data are compressed to only 140 MB with a parameter estimation error of less than 4×10^{-6} for a streamed Bayesian linear regression task.

In light of the sketch-and-solve paradigm, we focus on algorithmic approaches for the data reduction Π that can be efficiently implemented in streaming settings as well as in distributed environments. In particular, we develop methods to aggregate data and to reduce the number of observations using sketches via random linear projections and coresets obtained by importance sampling.

Sketching and coreset methods for regression on large-scale data are important areas of research with many interesting open questions. Although basic models are meanwhile well understood, research on more complex modern statistical and machine learning methods has just begun.

3.2.1.1 Brief Introduction to Coresets

Coresets are small, possibly weighted datasets that are designed to approximate an input dataset with respect to a computational problem. A survey on common techniques for obtaining coresets is given in [516]. Coresets usually depend on the considered objective function or on a broader class of objective functions. The first definitions were only implicitly given or were restricted to specific problems such as shape fitting or clustering [35, 295]. We give a more general definition.

Definition 2 (see [516]). *Let X be a set of points from a universe U and let Γ be a set of candidate solutions. Let $f : U \times \Gamma \to \mathbb{R}^{\geq 0}$ be a non-negative loss function. Then a set $C \subseteq X$ is an ε-coreset of X for f and some $\varepsilon \geq 0$, if*

$$\forall \gamma \in \Gamma : |f(X, \gamma) - f(C, \gamma)| \leq \varepsilon \cdot f(X, \gamma).$$

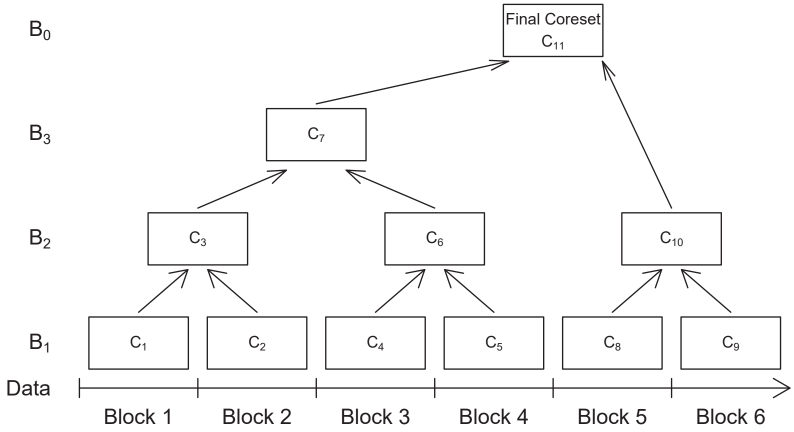


Fig. 3.2: Illustration of the Merge & Reduce principle from [246].

Note that the original point set is a perfectly accurate 0-coreset but has linear size. To be a useful data reduction, a coreset is required to be of sublinear size, e.g., polylogarithmic or even constant in the number of input points. The dependence on their dimension is usually allowed to be a small polynomial.

Coresets have been studied extensively for nearly two decades as a data aggregation and reduction tool to address scalability issues for a plethora of computational problems. Coresets have been developed for shape-fitting problems [6, 7, 33, 34, 218, 402], clustering [35, 216, 219, 453, 485], classification [294, 297, 594], ℓ_2-regression [147, 188, 189, 432], ℓ_1-regression [141, 142, 637], ℓ_p-regression [161, 709], M-estimators [144, 146], generalized linear models, [327, 501, 517, 594, 664], and other areas. We refer to [565] for an extensive survey and to [516] for a technical introduction to coresets.

Aggregation Properties and Merge & Reduce Most coreset constructions have strong aggregation properties as outlined in [295] for instance:

Definition 3. *Coresets are called* mergeable *if they satisfy the following properties:*

1. *If C_1 and C_2 are ε-coresets for input sets P_1 and P_2 respectively, then $C_1 \cup C_2$ is an ε-coreset for $P_1 \cup P_2$.*
2. *If C_1 is an ε-coreset for C_2, and C_2 is a δ-coreset for C_3, then C_1 is an $(\varepsilon + \delta)$-coreset for C_3.*

Given an off-line coreset construction that satisfies those properties, we can easily process data streams and distributed (or parallel) data via *Merge & Reduce* as a black box technique. Merge & Reduce was first introduced in [47] as a general method for extending static data structures to handle insertions. More recently, it has been adapted to work on coresets in the streaming setting [6, 295]. Nowadays, it is one of the main tools in the design of efficient streaming and distributed algorithms for the analysis of mass data. Though often only implicitly mentioned, Merge & Reduce has become a standard technique in the coreset literature. The **merge**(C_1, C_2) operation simply takes the union as in the first item of Definition 3 whereas the **reduce**(P) operation calls the off-line coreset construction algorithm on the point set P which can be used recursively to compute an ε-coreset from an ε-coreset etc. using the second item of the definition. Hereby, the error accumulates to εk after k recursive applications so one should control the value of $k = O(\log n)$ by, say, employing a binary tree construction as in Figure 3.2.

Figure 3.2 illustrates the principle of Merge & Reduce data stream algorithms. Note that all coresets are numbered in the order in which they are generated in a sequential data streaming application. First, Block 1 containing a fixed number of points is read from the stream into memory and the coreset C_1 is calculated. The same process yields coreset C_2 derived from the data contained in Block 2 of the stream. Since C_1, C_2 are siblings in the tree, they are combined into C_3 := **reduce**(**merge**(C_1, C_2)). At this point C_1, C_2 are not needed any more and are thus deleted from memory. The Merge & Reduce operations are indicated by the arrows in Figure 3.2. Now we proceed with C_4 derived from Block 3 and C_5 derived from Block 4. Since C_4, C_5 are siblings in the tree, they are combined into C_6 := **reduce**(**merge**(C_4, C_5)) and deleted. Again we have siblings C_3, C_6 on the same level, which are combined to C_7 := **reduce**(**merge**(C_3, C_6)) and deleted. The procedure is continued in the same manner until we reach the end of the stream. Say this is the case after processing Block 6. Note that at this point C_8, C_9 have been merged and reduced into C_{10} and have been deleted. The current state of the data structure is that it holds only coresets C_{10} in bucket B_2, i.e., on level 2, and C_7 in bucket B_3, i.e., on level 3, respectively. The buckets B_0 and B_1 are empty at this point and there are no further levels above level 3. Now a postprocessing step implicitly merges C_{11} = **reduce**(**merge**(C_7, C_{10})) via the working bucket B_0.

The construction can also be computed in a parallel or distributed setting. One possible scheme to achieve this, is to compute all coresets on the same level in parallel, starting with coresets C_1, C_2 C_4, C_5, C_8, C_9 on level 1 and proceeding with parallel computation of C_3, C_6, C_{10} on level 2 followed by C_7 on level 3 and finally deriving the final coreset C_{11} from C_7 and C_{10}.

Techniques that are similar to Merge & Reduce were employed in the area of physical design for relational databases [88]. Another interesting variant of Merge & Reduce directly combines statistical models rather than data summaries such as coresets [246]. We refer to Section 2.4.3 in Volume 3 of this book series for details.

3.2.1.2 Brief Introduction to Sketches

Sketching was introduced in the context of the theory of streaming algorithms. Popular examples include the `Count`-Sketch [123] the `CountMin`-Sketch [154], and the `Rademacher-`

Sketch [145]. Many contemporary sketches are variations or descendants of those basic techniques; see [708] for a survey and technical introduction. Similar to a coreset, a sketch is a succinct data summary, but it is not restricted to a subsample of the input or to representative substitute data points. Instead, any data structure of sublinear size with an efficient update procedure for processing new points may be regarded as a sketch. Usually, one encounters linear mappings, i.e., sketching matrices in the literature. Indeed, most known data stream algorithms are represented by linear sketches and there is some evidence that linear sketches are nearly optimal for such algorithms under certain conditions [429]. Linear sketches can be maintained dynamically in a data stream. Also, they have strong aggregation properties, which allow the combination of individual sketches—stemming from distributed data—to one single sketch for the entirety of the data. Sketching methods are much better positioned than coresets for handling high velocity streams, as well as highly unstructured massive databases [249, 628] and arbitrarily distributed data [648]. Linear sketches allow efficient applications in single pass sequential streaming and in distributed environments, see. e.g., [145, 354, 709]. Both, streaming and distributed computational settings are fundamental in the analysis of very large datasets and are very important for embedded systems and cyber-physical systems.

Linear sketches can be updated in the most dynamic streaming setting, which is commonly referred as the *turnstile* model, cf. [523]. In this model, we initialize a matrix X to be the all-zero matrix. The stream consists of $(key, value)$ updates of the form (i, j, v), meaning that X_{ij} will be updated to $X_{ij} + v$. Any entry can be defined by a single update or by a subsequence of not necessarily consecutive updates. For instance, the sequence $\ldots, (i, j, 25), \ldots, (i, j, -7), \ldots$ will result in $X_{ij} = 18$. Deletions are possible by using negative updates matching previous insertions. Due to linearity, linear sketches support operations such as adding, subtracting, and scaling entire databases X_j (i.e., matrices or vectors) efficiently in the sketch space, since $\Pi X = \Pi \sum_j \alpha_j X_j = \sum_j \alpha_j \Pi X_j$. For instance, if X_{t_1} and X_{t_2} are balances of bank accounts at time steps $t_1 < t_2$, then $\Pi T = \Pi X_{t_2} - \Pi X_{t_1}$ is a sketch of the transactions T within the period $t \in (t_1, t_2]$.

3.2.2 Our Contributions

Our research focused on developing streaming algorithms for frequentist and Bayesian linear regression as well as for generalized linear regression models. A common theme consists in developing data reduction techniques such as sketching via random linear projections or coresets via importance subsampling, retaining the statistical information up to little distortion. Hereby, we address resource restrictions such as memory access, communication cost, and runtime. Some highlights developed in the CRC 876 include coresets for specific classes of generalized linear models [501, 513, 515, 517] as well as graphical models [501]. We developed sketches for Bayesian linear regression models [245] and extended them towards hierarchical priors [247] and generalized normal distributions defined over ℓ_p-spaces [511, 513, 637]. We translated the Merge & Reduce principle from data summaries to maintaining statistical summaries in the streaming model [246] and introduced an asymptotic data stream model [303]. Another significant contribution lies in a dimensionality reduction for high-dimensional Bayesian optimization in sketching-based embeddings of low-dimensional subspaces [512]. An interesting further research direction is the development of dimensionality reduction techniques for reducing the width of neural networks and studying the limitations thereof [514].

3.2.2.1 Streaming Algorithms for Generalized Linear Regression

Generalized linear models (GLMs) extend classical linear regression to more flexible classes of generating distributions, cf. [479]. Usually, one assumes that the realizations of the dependent variable are generated from a member of the exponential family of distributions, based on the independent observations. Well-known examples of such distributions include the normal, binomial, Poisson, and gamma distributions. The expectation of the dependent target variable Y is connected to the linear predictor $X\beta$ via a link function h,

$$h(\mathbb{E}(Y)) = X\beta,$$

where X is the independent feature variable and β is the unknown parameter vector.

There is extensive work on sampling methods for approximating regression problems including ℓ_2-regression [188, 189] and ℓ_1-regression [141, 142, 637]. Those were generalized to ℓ_p-regression for all $p \in [1, \infty)$ [161, 709]. More recent works studied sampling methods for M-estimators [143, 144, 146] and generalized linear models [327]. We continued this line of research on coresets and sketches for logistic regression [515, 517] and p-generalized probit regression [513].

Logistic Regression Logistic regression is an important instance of a Generalized Linear Model [479]. The aim of logistic regression is to estimate the parameter β implicitly defining Bernoulli distributions based on the observed data. An exemplary task would be to assess the impact and interactions of variables in predicting the probability of patients suffering from a certain disease, based on their personal, physiological, and diagnostic data. This learning task is based on a fixed set of patient data $X \in \mathbb{R}^{n \times d}$ and corresponding labels $Y \in \{-1, +1\}^n$ indicating whether a patient is healthy or not. Folding the labels into the data we define row-vectors $Z_i = Y_i X_i$ for all $1 \le i \le n$.

Our first result in [517] shows the impossibility of compressing the data sublinearly in the input size, which holds in the worst case for any data reduction technique. To get around this limitation, we introduced a novel parameter that can be used to bound the complexity of compressing a dataset Z for logistic regression. This parameter is defined by

$$\mu(Z) = \sup_{\beta \in \mathbb{R}^d \setminus \{0\}} \frac{\|(Z\beta)^+\|_1}{\|(Z\beta)^-\|_1},$$

where $(Z\beta)^+$, $(Z\beta)^-$ comprise only the positive and negative entries of $Z\beta$, respectively. We call a dataset μ-complex if it satisfies $\mu(Z) \le \mu$. If the data is μ-complex for a small, not necessarily constant μ, then there exists an importance sampling and reweighting scheme based on the sensitivity framework of [219, 416] that produces an ε-coreset of sublinear size $O(\varepsilon^{-2}\mu\sqrt{n}d^{3/2}\log^{O(1)}(\mu nd))$ with high probability. A more involved recursive sampling scheme produces an ε-coreset of size $O(\varepsilon^{-4}\mu^3 d^3 \log^{O(1)}(\mu nd))$, which is beneficial if the data is well-behaved and the input size is particularly large. Those are the first provably sublinear coreset constructions for logistic regression.

The parameter $\mu(Z)$ has an intuitive statistical interpretation and might be of independent interest as detailed in [517]. It is not uncommon in practice that $\mu(Z)$ is small, since otherwise logistic regression exhibits methodological weaknesses.

Our experimental evaluation in [517] on real-world benchmark data shows that there is an efficient implementation based on a sketched QR-decomposition that is more accurate than uniform random sampling and state-of-the-art heuristic approaches such as described in [327] while being competitive in terms of runtime.

Meanwhile, the coreset size has been reduced to $\tilde{O}(\varepsilon^{-2}\mu^2 d)$ by replacing the leverage scores in the importance sampling distribution by ℓ_1-Lewis weights [462]. This has also improved the accuracy in experiments slightly, albeit at the cost of an increased runtime.

However, one limitation of known coreset constructions is that they require two passes over the data, one for approximating the importance sampling distribution and another for subsampling and collecting the data. The recursive improvement to polylogarithmic size, or calculating Lewis weights, requires even $O(\log\log n)$ passes. The Merge & Reduce framework is no remedy here due to the assumption of a small $\mu(Z)$. One might argue that a random order stream satisfies this condition for every batch of data, but in a worst-case setting we would have $\mu(Z_i) \to \infty$ for some batch Z_i, even in cases where $\mu(Z) = 1$.

Sketching Logistic Regression Towards creating a single-pass turnstile streaming algorithm for mild μ-complex data with all computational flexibilities, we developed the first linear sketch for logistic regression. Our main result [515] is a distribution over stacked sparse random matrices

$$\Pi = \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_{O(\log n)} \\ T \end{bmatrix}$$

Here, at each level i, S_i first subsamples a 2^{-i} fraction of the input points which are then hashed into a small number of buckets, where collisions are handled by summing the elements in the same bucket. The construction is complemented by a small uniform sampling matrix T. The resulting sketch reduces n input points in d dimensions to only $O(\text{poly}(\mu d \log n)) \times d$. We prove that ΠZ can be calculated over a turnstile stream in input sparsity time, i.e., $O(1)$ is spent on each non-zero element of the input. Moreover, with high probability over the random construction of Π, we have for $\tilde{\beta} \in \operatorname{argmin}_\beta f(\Pi Z \beta)$ that

$$f(Z\tilde{\beta}) \leq O(1) \min_{\beta \in \mathbb{R}^d} f(Z\beta),$$

where f denotes the logistic loss function [515]. The intuition behind this approach is that coordinates are grouped according to *weight classes* of similar loss that can be handled separately in the analysis. Weight classes with a small number of members will be approximated well on sketching levels with a large number of elements since roughly all members need to be subsampled to obtain a good estimate. Weight classes with many members will be approximated well on levels with a smaller number of subsamples. This is because if too many members survive the subsampling there will also be too many collisions under the uniform hashing, which would either lead to a large overestimate when those add up, or, due to asymmetry, would cancel each other and lead to large underestimations. Dealing with the asymmetry of the logistic loss was another issue that needed to be controlled. The error could not be bounded if the sign of an element was confused, since the ratio $\ell(x)/\ell(-x)$ is unbounded for the loss function $\ell(\cdot)$ of unconstrained logistic regression. Finally, there could be too many small contributions near zero. Logistic regression, unlike norms, assigns a non-zero constant loss to them. Their contribution can thus become significant. This is taken care of by the small uniform sample T of size $\tilde{O}(\mu d)$.

Poisson Regression Poisson regression is another instance of a GLM model, which aims at modeling count variables [479, 706]. A prominent example within the CRC 876 can be found in Section 4.1 in Volume 3 of this book series, where Poisson models are used to predict the number of vehicles per minute passing sensors of the highway ring around the city of Cologne. The predictions for a single sensor location are made based on the measurements at all other locations and the parameters learned from a Poisson regression model [284, 286]. This can be formalized as a Poisson dependency network (DN) [301]. Dependency networks are graphical models comprising a collection of GLMs, where each element of a set of d variables is regressed on all other variables. Dependency networks have several interesting applications surveyed in [501], such as collaborative filtering and density estimation, phylogenetic analysis, genetic analysis, network inference from sequencing data, and traffic modeling as well as topic modeling.

In our work [501], we have developed coresets for dependency networks. Assuming all GLMs in the dependency network to be ordinary linear regression models, we can subsample and reweight the input points as in [188] to construct a coreset. Surprisingly, we do not need to construct a coreset for each of the d GLMs separately. Instead, we can exploit the common subspace structure of all GLMs to show that it is sufficient to construct one single coreset of size $O(\varepsilon^{-2} d \log d)$.

With Poisson GLMs, the situation is different. Again, we can show that in the worst case, any data reduction technique produces either a summary of linear size or fails to approximate the objective function to within a large superconstant factor [501]. Reviewing the statistical modeling for count data, we note that the Poisson lognormal model is a statistical relaxation of the ordinary Poisson model [706]. It introduces a connection to linear ℓ_2-regression that we can exploit to show that a reweighted sample of size $O(\varepsilon^{-2} d \log^2 d)$ gives a good approximation of the consistent maximum likelihood estimator in this model [501].

Our experimental evaluation [501] shows that the importance sampling scheme outperforms uniform sampling for the normal GLMs. For the Poisson GLMs the result is not as remarkable and the log-likelihood approximation seems worse for large sample sizes at first glance. But as the subsample size drops below 20 %, our method captures more structure of the data. A remarkable, yet non-intuitive feature is that the approximation is capable of making better predictions than the optimal model [501]. Similar effects have been observed independently in the general setting of randomized linear algebra algorithms [461] and was attributed to an implicit regularization effect, since the distortion induced by the approximation prevents the model from overfitting the original data.

3.2.2.2 Sketches and Coresets for Bayesian Regression

Let us now focus on theoretical aspects of data compression for Bayesian regression. We point the interested reader to Section 2.4 in Volume 3 of this book series for more methodological results and applications. Bayesian regression does not assume a fixed *optimal* solution for a dataset as is required in the frequentist case. Instead, it introduces a distribution over the parameter space. The *likelihood* function $\mathcal{L}(Y|X, \beta)$ models the information that comes from the data. The *prior* distribution $p_{\text{pre}}(\beta)$ models problem-specific prior knowledge. Our goal is now to explore and characterize the *posterior* distribution $p_{\text{post}}(\beta)$, which, as a consequence of Bayes' theorem, is a compromise between the information from observed data and from the prior knowledge that we assume for the parameters[3]

$$p_{\text{post}}(\beta|X, Y) \propto \mathcal{L}(Y|X, \beta) \cdot p_{\text{pre}}(\beta).$$

Random Projections for Bayesian Regression Our work on random projections for Bayesian regression [245] extends previous work on frequentist ℓ_2-regression [145] to the Bayesian setting. Certain types of random projections studied in theoretical computer science form a so-called ε-subspace embedding. Those are linear sketches for ℓ_2-spaces,

[3] Here, $a \propto b$ means that there exists a constant $c > 0$ such that $a = cb$.

which preserve the ℓ_2-norm of all vectors in a linear subspace with little distortion. The guarantee we obtain is that there exists a distribution over sketching matrices Π with a reduced target dimension $O(d/\varepsilon)$ such that

$$\forall \beta \in \mathbb{R}^d : (1 - \sqrt{\varepsilon})\|X\beta\|_2 \leq \|\Pi X\beta\|_2 \leq (1 + \sqrt{\varepsilon})\|X\beta\|_2$$

holds with high probability over the random choice of Π. This implies that it preserves the ℓ_2-regression error up to a factor of $(1 + \varepsilon)$ [145], i.e., if we solve the compressed regression problem to obtain $\tilde{\beta} \in \operatorname{argmin}_{\beta\in\mathbb{R}^d} \|\Pi(X\beta - Y)\|$ then $\tilde{\beta}$ satisfies

$$\|X\tilde{\beta} - Y\|_2 \leq (1 + \varepsilon) \min_{\beta\in\mathbb{R}^d} \|X\beta - Y\|_2.$$

For Bayesian regression we also apply an ε-subspace embedding Π to compress the data matrix $[X, Y] \in \mathbb{R}^{n\times(d+1)}$ to a sketch $[\Pi X, \Pi Y] \in \mathbb{R}^{k\times(d+1)}$ for slightly larger $k \in O(\mathrm{poly}(d)/\varepsilon^2)$, whose dimensions notably do not depend on n. Our main finding is that the results of a *Bayesian* analysis on the sketch and on the original dataset are also similar up to little distortion, depending on the approximation parameter ε. More specifically, if we denote by $p = p_{\mathrm{post}}(\beta|X, Y)$ and $q = p_{\mathrm{post}}(\beta|\Pi X, \Pi Y)$ the posterior distribution on the original data and on the sketch respectively, then $p \approx_\varepsilon q$, i.e., they are close to each other. We can quantify the approximation via the Wasserstein distance [245]. This choice is especially appealing, because it relates the distance of probability measures to properties in the ℓ_2-space over which they are defined. For normal distributions this entails that their location parameters as well as their covariances are close to the original.

The aforementioned results were restricted to the most prominent case of Bayesian linear regression, namely to the basic case of a likelihood based on Gaussian distributions and a multivariate normal distribution as a prior. The model class of the prior includes the degenerate, but common non-informative choice of a uniform distribution over $\mathbb{R}^d$.

Hierarchical Models Hierarchical regression models offer an extension of the previous result to a broader class of prior distributions [247]. They present a modern statistical approach that is especially useful when information on different levels is present, e.g., in a meta-analysis, where raw data is available for some studies, but only averages for the others [699]. A hierarchical model is given by

$$p_{\mathrm{post}}(\beta, \theta|X, Y) \propto \mathcal{L}(Y|X, \beta) \cdot p_{\mathrm{pre}}(\beta|\theta) \cdot p_{\mathrm{hyper}}(\theta),$$

where the prior on β on the first level depends on a *hyperparameter* θ that is again modeled via a *hyper-prior* $p_{\mathrm{hyper}}(\theta)$ on the second level of the hierarchy. Such models can be naturally extended to model arbitrary, deep or broad hierarchies, and to model numerous different populations.

Generalized Normal Prior Distributions Generalized normal priors are another modern statistical extension that we study [247, 511]. They result from generalizing the inducing norm from ℓ_2 to ℓ_p for $p \in [1, \infty)$. Their probability density function is given by

$$f(x) = \frac{p}{2\varsigma\Gamma(1/p)} \exp\left(-\frac{|x-\mu|^p}{\varsigma^p}\right),$$

where μ is a location parameter and ς is a scale parameter. The parameter p determines the shape and heaviness of the tails. Special cases include the normal distribution for $p = 2$, the Laplace distribution for $p = 1$, and the uniform distribution on $[\mu - \varsigma, \mu + \varsigma]$ for $p \to \infty$. Generalized normal distributions have been suggested and employed as a robust alternative to model deviations from normality [438] and to model a Bayesian analogue of LASSO regression [554]. Alternatively, they can also be employed to model a higher sensitivity to outliers [513]. This is also important in the context of correcting statistical models [518].

Generalized Normal Likelihood Distributions Generalized normal likelihoods can also be treated with subspace embeddings. We note that the first such generalization for ℓ_1 was developed in the CRC 876 [637]. The case $p \in [1, \infty)$ can be approximated in a similar way as in the case of normal distributions via further generalized subspace embedding techniques for ℓ_p [709]. However, this is technically more challenging [511]. One complication is that the embedding sizes are much larger for $p > 2$ than for $p \leq 2$. The other problem is that the distortion is as large as $O((d \log d)^{1/p})$ rather than $(1 \pm \varepsilon)$. We thus use the random projection only in a preprocessing step [511] to obtain a so-called *well-conditioned basis*, which can be thought of as an ℓ_p-analogue to an orthonormal basis for ℓ_2. From this we can derive sampling probabilities such that by taking $O(d^{2p+3} \log^2 d \log(1/\varepsilon)\varepsilon^{-2})$ reweighted random samples, we achieve the desired $(1 \pm \varepsilon)$ distortion. This is in line with [709] for $p > 2$ but is slightly weaker for $p \in [1, 2]$. However, our simpler unified algorithm applies universally to both cases. Similar methods were recently developed for obtaining coresets for the p-generalized probit model [513] and are currently being extended to the Bayesian setting.

3.2.2.3 Bayesian Optimization in Embedded Subspaces

Bayesian optimization (BO) has emerged as a powerful technique for the global optimization of black-box functions that are expensive to evaluate [80, 235, 624]. Here ‘black-box’ means that we may evaluate an unknown but fixed objective function f at any point to observe its value, possibly with noise but without derivative information. The goal is to find

$$x^\star \in \operatorname{argmin}_{x \in \mathcal{C}} f(x)$$

over a set $\mathcal{C}$, the domain of optimization, which can represent constraints, such as a box-constraint $\mathcal{C} = [-1, 1]^D$ on a large D-dimensional domain, for instance.

The advantages of Bayesian optimization are sample efficiency, provable convergence to a global optimum, and a low computational overhead. A critical limitation is the number of parameters that BO can optimize over. This is especially true for the most common form of BO that uses Gaussian Process (GP) regression as a surrogate model for the objective function. Thus, it is not surprising that expanding BO to higher-dimensional search spaces is widely acknowledged as one of the most important goals in the field [235]. Our work [512] advances the field, both, in the theory of high-dimensional Bayesian optimization and in improving practical performance.

The idea of Bayesian optimization is to learn a Gaussian process surrogate model on the previous evaluations in order to gain knowledge on where to evaluate next by a simpler optimization of an *acquisition criterion*, e.g., the Expected Improvement (EI). Under the assumption that the objective function depends essentially only on a low d_e-dimensional *effective* subspace of an ambient high-dimensional space, we used a sparse subspace embedding matrix to perform the optimization in an intermediate subspace of dimension $O(d_e^2/\varepsilon^2)$. This solved several open problems in the area [512]:

1. It fixed the problem of large dilations that caused previous Gaussian embedding matrices to project the evaluation points out of the feasible region of optimization.
2. We provided a rigorous proof that the underlying Gaussian process is well approximated in terms of its mean and variance functions, which indicates that the sample efficiency is preserved.
3. We extended the result under mild assumptions to several highly non-linear kernel spaces, which may be of independent interest.
4. It is computationally much faster than previous and contemporary approaches due to the sparse embedding.
5. It performs among the best algorithms in practice even when the low-dimensional assumption is not satisfied, cf. [201].

We refer to Section 2.5 in Volume 3 for more research and applications using Bayesian optimization.

3.2.3 Conclusion

We introduced the concepts of coresets and sketching, which are methods for summarizing data in such a way that the reduced dataset retains provable approximation guarantees for a given computational or statistical learning problem. This enables analyzing data in resource-constrained environments such as data streams and distributed systems, sensor networks etc., which are common in embedded systems and cyber-physical systems. By reducing the data before their aggregation or analysis, our methods help to save computation time and memory requirements and support communication awareness. Consequently, this also saves resources on a lower technical level, for instance energy and bandwidth.

Originating from the theory of computing community in the early twenty-first century, those methods have paved their way into the machine learning and statistical communities over the last decade ever since the *Big Data* hype. From there, they are anticipated to spread into all kinds of technical and application domains in the near future. This also underlines the importance of integrating them into contemporary undergraduate and graduate training programs.

Research on data reduction techniques, such as coresets and sketching, is an evergrowing field from theoretical and from applied perspectives. The limitations and possibilities for relatively simple but important base problems like linear regression are now well-understood. But it remains open and challenging in many cases to extend research to more sophisticated and computationally more demanding methods such as Bayesian statistics, and neural networks.

We anticipate great advances in the field of Bayesian statistics. The advantages of those methods lie in their theoretical statistical foundation, the interpretability of their models, and their built-in quantification of uncertainty. However, normally Bayesian methods require horrendous amounts of resources. Our fundamental research has shown initial approaches for making those methods scalable and resource-efficient, and leaving still a lot of potential for future research.

4 Structured Data

In this chapter, we show methods and techniques that learn models for structured data in resource-aware environments. In practice, data models can often be structured as a graph where different data points are represented as nodes and the relationship between data points is captured by edges. Graphs occur in many applications because they serve well to represent objects of the physical world as compositions of parts. Molecules, for instance, can be described by a graph where the atoms are represented by nodes and their bonds by the edges. Another example are mathematical formulas, whose composition is semantically well modeled by graphs (see Section 4.5). Moreover, interactions between nodes of a graph can even be structured over time leading to spatio-temporal probabilistic graphs (see Section 4.1).

Once a particular type of a graph model is determined the models can be trained to do machine learning tasks such as classifying graphs or, when we interpret a graph as a transitional system, predicting the probability of a change from one state to another. The learning methods we use in this chapter can be divided mainly into *discriminative* Graph Neural Networks (GNNs) and *generative* Random Fields. GNNs use a learning approach that is derived from Convolutional Neural Networks (CNNs) by aggregating information of the neighborhood of each node through a message passing function (see Sections 4.2, 4.3, 4.5). Random Fields are a probabilistic model that captures the dependencies between multiple random variables and is trained to answer queries for a probability of event A under the condition that event B already happened (see Section 4.1). GNNs and Random Fields are different methods but both can be used to express the same kind of problems. For example, to infer conditional probabilities for each event we can use multiple GNNs in a layered approach [376]. However, the way in which they take care of the computational resources is rather different.

Here is an overview of this chapter. In Section 4.1, a new model is proposed to train spatio-temporal networks with Random Fields called the *Spatio-Temporal Random Field*. This model reduces the memory consumption without loss of the accuracy through a theoretically well based universal reparameterization. In Section 4.2, the *Weisfeiler-Leman algorithm* is explained with a focus on theoretical runtimes and the scalability of the algorithm. Then, the connection between the Weisfeiler-Leman algorithm and learning methods using graph kernels and GNNs, is surveyed. In Section 4.3, a unified framework for differentiable message passing in GNNs is introduced, and techniques for increasing its scalability are proposed. Section 4.4 proposes a framework to compute cuts in directed graphs with high quality, which scales well in shared memory and can be used in semi-supervised learning as well as in data compression. Section 4.5 presents a new technique to search for scientific papers, which uses mathematical formulas instead of words. A GNN is trained on a huge dataset extracted from arXiv and it is shown that the model scales well in practice.

https://doi.org/10.1515/9783110785944-004

4.1 Spatio-Temporal Random Fields

Nico Piatkowski
Katharina Morik

Abstract: Parameter sharing is a key technique in various state-of-the-art machine learning approaches. The underlying idea is simple yet effective. Given a highly overparametrized model whose input data obeys some repetitive structure, multiple subsets of parameters are tied together. On the one hand, this reduces the number of parameters, which simplifies the corresponding estimation problem. On the other hand, information is transferred from one part of the data space to another, thus allowing the model to learn patterns that never explicitly occurred in the training data. In the context of resource constrained data analysis, the primary interest lies in the reduced memory requirements, induced by the lower parameter space dimension and a presumably lower sample complexity. In this contribution, the concept that underlies parameter sharing is transferred to the spatio-temporal domain. More precisely, a re-parametrization of undirected probabilistic graphical models, known as Markov Random Fields (MRFs) is proposed for non-stationary time series of finite length. MRFs are equivalent to deep latent variable models [568] but obey an easier-to-interpret structure. Data for such spatio-temporal models arises naturally in distributed sensor networks. The corresponding machine learning models are, however, far too large to be processed directly at the sensor level. Re-parametrized probabilistic models exhibit a very sparse parameter space that facilitates probabilistic inference directly from a compressed model. This section studies different variants of the underlying re-parametrization and compares them in numerical experiments on benchmark data. Furthermore, we propose how the learning procedure can be embedded directly into a sensor network: proximal optimization is applied in a distributed setting. It turns out that the parameter optimization is purely local and that communication between sensor nodes is required only for the gradient computation. Different real-world applications, including traffic models and sensor network models underpin the practical relevance of compressed Spatio-Temporal Random Fields (STRF).

4.1.1 Introduction

Spatio-temporal sensor data is an archetypical instance of structured data. Inherent dependencies that span over space and time constitute demanding challenges when aiming for reliable models with reasonable resource requirements. Here, we consider the task of *spatio-temporal state prediction*, where the spatio-temporal structure is represented by an undirected graph $G = (V, E)$ that is either known or inferred from data. Nodes within the network represent locations at different points in time t from a finite time horizon T. Based on a set of N partially observed joint realizations, a generative model $\mathbb{P}_{\boldsymbol{\theta}}$ is learned, where $\boldsymbol{\theta}$ is the trainable parameter. This task arises frequently in the analysis of sensor networks e.g., communication networks [577] or satellite image data [229]. For the sake of clarity, modeling the traffic in a highway network will serve as our running example. That is, the model must answer queries for all parts of the network and all points in time. Examples of such predictions are:

- Given the traffic densities of all roads in a street network at discrete time points t_1, t_2, t_3 (e.g., 8 o'clock on Monday, Tuesday, Wednesday): indicate the probabilities of traffic levels on a particular road A at some other time point, not necessarily following the given ones (e.g., 7 o'clock on Thursday).
- Given a traffic jam at place A at time t_s: output other places with a probability higher than 0.7 for the state "jam" in the time interval of $t_s < t < t_t$.

One particular interest lies in learning probabilistic models for answering such queries in resource-constrained environments. This addresses huge amounts of data on fast computing facilities moderate data volume on embedded or ubiquitous devices. Results and methods that are presented in this contribution are based on [566] and [567].

4.1.2 Previous Work

In this section, an overview of previous contributions to spatio-temporal modeling is given. The task of *traffic forecasting* is often solved by simulations [467]. This presupposes a model instead of learning it. In the course of urban traffic control, events are merely propagated that are already observed, e.g., a jam at a particular highway section results in a jam at another highway section, or the prediction is based on a physical rule that predicts a traffic jam based on a particular congestion pattern [287]. Many approaches apply statistical time series methods such as auto-regression and moving average models [705]. They do not take into account spatial relations but restrict themselves to the prediction of the state at one location given a series of observations at this particular location. An early approach, that of Whittaker, Garside, and Lindveld [703], relies on the street network topology for deriving spatial relations. The training is done via Kalman filters, which imply a strictly linear conditional independence structure, that is not expressive enough for answering queries like the ones stated above. A statistical relational learning approach to traffic forecasting uses explicit rules for modeling spatio-temporal dependencies [441]. Here, training is done by a Markov Logic Network delivering conditional probabilities of congestion classes. The discriminative model is restricted to binary classification tasks and the spatial dependencies need to be given by hand-tailored rules. Moreover, the model is not sparse and training is not scalable. Even for a small number of sensors, training takes hours of computation. When the estimation of models for spatio-temporal data on ubiquitous devices is considered, such as when learning to predict smartphone usage patterns based on time and visited places, minutes are the order of magnitude in demand. Hence, even this advanced approach does not yet meet the demands of the spatio-temporal prediction task in resource-constrained environments.

Some geographically weighted regression or non-parametric k-Nearest Neighbor (kNN) methods model a task similar to spatio-temporal state prediction [263, 477, 743]. The regression expresses the temporal dynamics and the weights express spatial distances. Another way to introduce the spatial relations into the regression is to encode the spatial network into a kernel function [440]. The kNN method by [409] models correlations in spatio-temporal data not only by their spatial but also by their temporal distance. As stated for the spatio-temporal state prediction task, the particular place and time in question need not be known in advance, because the lazy learner kNN determines the prediction at query time. However, this approach does not deliver probabilities along with the predictions, either. For some applications, traffic prognoses for car drivers, a probabilistic assertion is not necessary. However, in applications of disaster management, the additional information regarding likelihood is desirable.

As is easily seen, generative Markov models fit the task of spatio-temporal state prediction. For notational convenience, let us assume only one variable X. Any *generative probabilistic model* represents the joint $\mathbb{P}(X, Y)$ and allows us to derive $\mathbb{P}(Y|X) = \frac{\mathbb{P}(X,Y)}{\mathbb{P}(X)}$ as well as $\mathbb{P}(X|Y) = \frac{\mathbb{P}(X,Y)}{\mathbb{P}(Y)}$. In contrast, *discriminative probabilistic models* represent $\mathbb{P}(Y|X)$ directly and must be trained specifically for each Y—this property is inherent since each realization of Y requires a different normalization constant. In our example a distinct model would need to be trained for each place. Hence, a huge set of discriminative models would be necessary to express one generative model. A discussion of discriminative versus generative models can be found in a study by [531]. Here, we refer to the capability of interpolation (e.g., between points in time) of generative models and their informativeness in delivering probability estimates instead of merely binary decisions.

Spatial relations are naturally expressed by *graphical models*. For instance, discriminative graphical models such as Conditional Random Fields (CRFs) have been used for object recognition over time [182], while generative graphical models such as Markov Random Fields (MRFs) have been applied to video or image data [322, 723]. The number of training instances does not influence the model complexity of MRFs. However, the number of parameters can easily exceed millions. In particular when using MRFs for spatio-temporal state prediction, the numerous spatial and temporal relations soon lead to inefficiency.

We have argued in favor of using generative graphical models that model both, spatial and temporal dependencies, at the same time. However, some problems have until now prohibited this:

- The original parametrization is not well suited for producing sparse models.
- Trained models tend to overfit to the training data.

– Training high-dimensional models is not feasible.

In the following, we shall review existing work on graphical models (Section 4.1.3) and regularization methods (Section 4.1.4) so that we can then introduce a new method for spatio-temporal state prediction that does not suffer from the listed disadvantages.

4.1.3 Graphical Models

The formalism of probabilistic graphical models provides a unifying framework for capturing complex dependencies among random variables, and building large-scale multivariate statistical models [692]. Let $G = (V, E)$ be an undirected graph with the set of vertices V and the set of edges $E \subset V \times V$. Note that the subset relation is strict, since self-edges are not allowed. Moreover, we represent undirected edges as sets (as opposed to ordered tuples). For each node (or vertex) $v \in V$, let $\boldsymbol{X}_v$ be a random variable, taking values x_v in some space $\mathcal{X}_v$. The concatenation of all $n = |V|$ variables yields a multivariate random variable $\boldsymbol{X}$ with state space $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n$. Training delivers a full probability distribution over the random variable $\boldsymbol{X}$. Let $\boldsymbol{\phi}$ be an *indicator function* or *sufficient statistic* that indicates if a configuration $\boldsymbol{x}$ obeys a certain event $\{\boldsymbol{X}_\alpha = \boldsymbol{x}_\alpha\}$ with $\alpha \subseteq V$. We use the short-hand notation $\{\boldsymbol{x}_\alpha\}$ to denote the event $\{\boldsymbol{X}_\alpha = \boldsymbol{x}_\alpha\}$. The functions of $\boldsymbol{x}$ defined in the following can be also considered as functions of $\boldsymbol{X}$. We replace $\boldsymbol{x}$ by $\boldsymbol{X}$ when it makes their meaning clearer. Restricting α to vertices and edges,[1] one gets

$$\boldsymbol{\phi}_{\{v=x\}}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x}_v = x \\ 0 & \text{otherwise,} \end{cases} \quad \boldsymbol{\phi}_{\{(v,w)=(x,y)\}}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } (\boldsymbol{x}_v, \boldsymbol{x}_w) = (x, y) \\ 0 & \text{otherwise} \end{cases}$$

with $\boldsymbol{x} \in \mathcal{X}$, $\boldsymbol{x}_v \in \mathcal{X}_v$ and $y \in \mathcal{X}_w$. Let us now define vectors for collections of those indicator functions:

$$\begin{aligned} \boldsymbol{\phi}_v(\boldsymbol{x}) &:= \left[\boldsymbol{\phi}_{\{v=x\}}(\boldsymbol{x})\right]_{x \in \mathcal{X}_v}, \\ \boldsymbol{\phi}_{(v,w)}(\boldsymbol{x}) &:= \left[\boldsymbol{\phi}_{\{(v,w)=(x,y)\}}(\boldsymbol{x})\right]_{(x,y) \in \mathcal{X}_v \times \mathcal{X}_w}, \\ \boldsymbol{\phi}(\boldsymbol{x}) &:= \left[\boldsymbol{\phi}_v(\boldsymbol{x}), \boldsymbol{\phi}_e(\boldsymbol{x}) : \forall v \in V,\ \forall e \in E\right]. \end{aligned} \tag{4.1}$$

The vectors are constructed for fixed but arbitrary orderings of V, E and $\mathcal{X}$. The dimension of $\boldsymbol{\phi}(\boldsymbol{x})$ is thus $d = \sum_{v \in V} |\mathcal{X}_v| + \sum_{(v,u) \in E} |\mathcal{X}_v| \times |\mathcal{X}_u|$. Now, consider a dataset $\mathcal{D} = \left\{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^N\right\}$ with instances $\boldsymbol{x}^i$. Each $\boldsymbol{x}^i$ consists of an assignment to every node in the graph. It defines a full joint state of the random variable $\boldsymbol{X}$.

[1] In general, one may consider indicator functions not only for nodes and edges, but for all cliques (fully connected subgraphs) in G. Our description still applies to higher order models, since we can convert them into models using only nodes and edges [692, Appendix E].

The quantities

$$\hat{\boldsymbol{\mu}}_{\{v=x\}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\phi}_{\{v=x\}}(\boldsymbol{x}^i), \quad \hat{\boldsymbol{\mu}}_{\{(v,w)=(x,y)\}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\phi}_{\{(v,w)=(x,y)\}}(\boldsymbol{x}^i) \tag{4.2}$$

are known as *empirical moments* and they reflect the empirical frequency estimates of the corresponding events. We say that a given probability density function p with base measure[2] ν and expectations $\mathbb{E}_p\left[\boldsymbol{\phi}_{\{x_\alpha\}}(\boldsymbol{x})\right]$ is *locally consistent* with data $\mathcal{D}$ if and only if p satisfies the *moment matching condition*

$$\mathbb{E}_p\left[\boldsymbol{\phi}_{\{x_\alpha\}}(\boldsymbol{x})\right] = \hat{\boldsymbol{\mu}}_{\{x_\alpha\}}, \forall \alpha \in V \cup E,$$

i.e. the density p is consistent with the data w.r.t. the empirical moments.

This problem is underdetermined in that there are many densities p that are consistent with the data, so that we need a principle for choosing among them. The principle of maximum entropy is to choose, among the densities consistent with the data, the densities $p^\star$ whose *Shannon entropy* $\mathcal{H}(p)$ is maximal. $\mathcal{H}$ is given by

$$\mathcal{H}(p) := -\int_{\mathcal{X}} p(\boldsymbol{x}) \log_2\left(p(\boldsymbol{x})\right) d\nu(\boldsymbol{x}).$$

This is turned into the constrained optimization problem

$$\max_{p \in \mathbb{P}} \mathcal{H}(p) \text{ subject to } \mathbb{E}_p\left[\boldsymbol{\phi}_{\{x_\alpha\}}(\boldsymbol{x})\right] = \hat{\boldsymbol{\mu}}_{\{x_\alpha\}}, \quad \forall \alpha \in V \cup E.$$

It can be shown that the optimal solution $p^\star$ takes the form of an exponential family of densities

$$p_{\boldsymbol{\theta}}(\boldsymbol{X} = \boldsymbol{x}) = \exp[\langle \boldsymbol{\theta}, \boldsymbol{\phi}(\boldsymbol{x}) \rangle - A(\boldsymbol{\theta})],$$

parametrized by a vector $\boldsymbol{\theta} \in \mathbb{R}^d$. Note that the parameter vector $\boldsymbol{\theta}$ and the sufficient statistics vector $\boldsymbol{\phi}(\boldsymbol{x})$ have the same length d. The term

$$A(\boldsymbol{\theta}) := \log \int_{\mathcal{X}} \exp[\langle \boldsymbol{\theta}, \boldsymbol{\phi}(\boldsymbol{x}) \rangle] d\nu(\boldsymbol{x})$$

is called *log partition function*. It is defined with respect to a reference measure ν such that $\mathbb{P}(X \in S) = \int_S p_{\boldsymbol{\theta}}(x) d\nu(x)$ for any measurable set S. Expanding $\boldsymbol{\phi}(\boldsymbol{x})$ by means of Equation 4.1 reveals the usual density of pairwise undirected graphical models, also known as *pairwise MRFs*

$$\begin{aligned} p_{\boldsymbol{\theta}}(\boldsymbol{X} = \boldsymbol{x}) &= \frac{1}{\exp A(\boldsymbol{\theta})} \prod_{v \in V} \exp[\langle \boldsymbol{\theta}_v, \boldsymbol{\phi}_v(\boldsymbol{x}) \rangle] \prod_{(v,w) \in E} \exp[\langle \boldsymbol{\theta}_{(v,w)}, \boldsymbol{\phi}_{(v,w)}(\boldsymbol{x}) \rangle] \\ &= \frac{1}{\Psi(\boldsymbol{\theta})} \prod_{v \in V} \psi_v(\boldsymbol{x}) \prod_{(v,w) \in E} \psi_{(v,w)}(\boldsymbol{x}). \end{aligned}$$

[2] Notice that when the underlying state space $\mathcal{X}$ is discrete, then ν is the counting measure and we may identify the density p with the measure $\mathbb{P}$.

Here, $\Psi = \exp A$ is the cumulant-generating function of $p_{\boldsymbol{\theta}}$, and ψ_α refers to the *potential functions*.

Inference, that is, computing the marginal probabilities or maximum a-posteriori states of each vertex, can be carried out by message propagation algorithms [404, 560, 690], variational methods [692], or quadrature-based methods [572, 573]. In order to fit the model on some dataset, the model parameters have to be estimated. If the dataset contains only fully observed instances, the parameters may be estimated by the maximum likelihood principle. The estimation of parameters in the case of partially unobserved data is a challenging topic on its own. Here, we assume that the dataset $\mathcal{D}$ contains only fully observed instances. The *likelihood* $\mathcal{L}$ and the *average log-likelihood* ℓ of parameters $\boldsymbol{\theta}$ given a set of i.i.d. data $\mathcal{D}$ are defined as

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{D}) := \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(\boldsymbol{x}^i) \quad \text{and} \quad \ell(\boldsymbol{\theta};\mathcal{D}) := \frac{1}{N}\sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^i) = \langle \boldsymbol{\theta}, \hat{\boldsymbol{\mu}} \rangle - A(\boldsymbol{\theta}). \tag{4.3}$$

The latter is usually maximized due to numerical inconveniences of $\mathcal{L}$. The most frequently applied optimization methods are iterative proportional fitting [160], gradient descent and quasi-newton methods such as L-BFGS or the conjugate gradient [538]. Section 4.1.5 will show how to model spatio-temporal dependencies within this formalism.

4.1.4 Regularization

As we can see, the number of parameters in $\boldsymbol{\theta}$ grows quite rapidly as we consider more complex graphical models. A large number of parameters is generally not preferable, since it may lead to overfitting, and it resists the implementation of a memory-efficient predictor. Therefore, some regularization is necessary to achieve a sparse and robust model.

Popular choices of regularizers are the l_1 and l_2 norms of the parameter vector, $\|\boldsymbol{\theta}\|_1$ and $\|\boldsymbol{\theta}\|_2$. By minimizing the L_1 norm, we coerce the values for less informative parameters to zero (similar to LASSO [660]), and by the l_2 norm we find smooth functions parametrized by $\boldsymbol{\theta}$ (similar to the penalized splines [559]). Using both together is often referred to as the *elastic net* [748]. For graphical models, elastic nets appeared in the context of structure learning (estimating the neighborhoods) [156] in a manner similar to the approach of [484]. For the state prediction task, there exist two short workshop papers [569, 571] using the elastic net. However, their analytical and empirical validation of such an approach is rather limited.

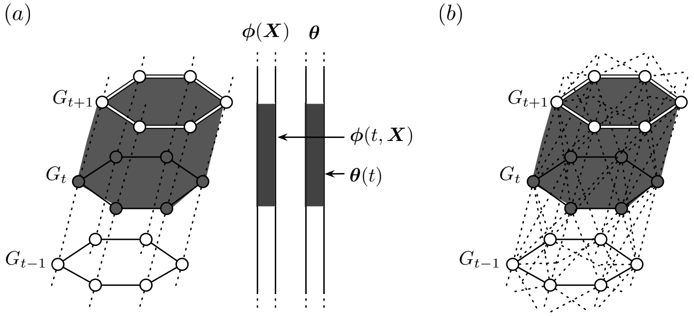


Fig. 4.1: A spatio-temporal model consisting of multiple snapshot graphs G_t for $t = 1, 2, \ldots, T$. The spatial and temporal edges are represented by solid and dotted lines, respectively. (a) A layer L_t is shown as the shaded region with simple temporal edges (L_t does not include the elements of G_{t+1}), along with the corresponding sufficient statistic and parameter subvectors $\boldsymbol{\phi}(t, X)$ and $\boldsymbol{\theta}(t)$. (b) An extended model with "crossing" temporal edges between consecutive snapshots. This extended model is adopted in our experiments.

4.1.5 From Linear Chains to Spatio-Temporal Models

Sequential undirected graphical models, also known as linear chains, are a popular method in the natural language processing community [407, 654]. There, consecutive words or corresponding word features are connected to a sequence of labels that reflects an underlying domain of interest like entities or part of speech tags. If we consider a sensor network G that generates measurements over space such as a word, then it would be appealing to think of the instances of G at different time points, like words in a sentence, to form a temporal chain $G_1 - G_2 - \cdots - G_T$. We will now present a formalization of this idea followed by some obvious drawbacks. Hereafter, we will discuss how to tackle those drawbacks and derive a tractable class of generative graphical models for the spatio-temporal state prediction task.

We first define the part of the graph corresponding to the time step t as the *snapshot graph* $G_t = (V_t, E_t)$, for $t = 1, 2, \ldots, T$. Each snapshot graph G_t replicates a given *spatial graph* $G_0 = (V_0, E_0)$, which represents the underlying physical placement of sensors, i.e., the spatial structure of random variables that does not change over time. We also define the set of spatio-temporal edges $E_{t-1;t} \subset V_{t-1} \times V_t$ for $t = 2, \ldots, T$ and $E_{0;1} = \emptyset$, that represent dependencies between adjacent snapshot graphs G_{t-1} and G_t, assuming a Markov property among snapshots, so that $E_{t;t+h} = \emptyset$ whenever $h > 1$ for any t. Note that the actual time gap between any two time frames t and $t + 1$ can be chosen arbitrarily.

The entire graph, denoted by G, consists of the snapshot graphs G_t stacked in the order of time frames $t = 1, 2, \ldots, T$ and the temporal edges connecting them: $G := (V, E)$ for $V := \cup_{t=1}^{T} V_t$ and $E := \cup_{t=1}^{T} \{E_t \cup E_{t-1;t}\}$. We sketch the structure of G in Figure 4.1.

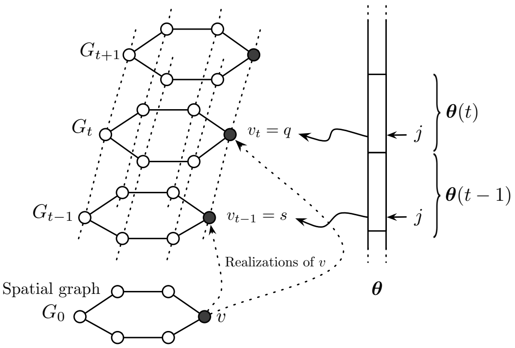


Fig. 4.2: An example of indexing for a node and state pair over time. A sensor modeled by the node v in the spatial graph G_0 shows its measurements v_{t-1} and v_t at time frames $t-1$ and t, respectively. The pairs $v_{t-1} = s$ and $v_t = q$ are located at the same index j in the subvectors $\boldsymbol{\theta}(t-1)$ and $\boldsymbol{\theta}(t)$.

For the sake of a simple description, we define a *layer* L_t as the partial subgraph of G containing all vertices of V_t and all edges of $E_t \cup E_{t;t+1}$, for $t = 1, 2, \ldots, T$. For instance, a layer L_t is depicted as a shaded region in Figure 4.1. Let $a \in \mathcal{X}_v$ and $b \in \mathcal{X}_w$ and define the subvectors of $\boldsymbol{\phi}(X)$ and $\boldsymbol{\theta}$ that correspond to a layer L_t as follows:

$$\begin{aligned} \boldsymbol{\phi}(t, \boldsymbol{X}) &:= (\boldsymbol{\phi}_{v=a}(\boldsymbol{X}_v), \boldsymbol{\phi}_{(v,w)=(a,b)}(\boldsymbol{X}_v, \boldsymbol{X}_w) \mid v \in L_t,\ (v,w) \in L_t,), \\ \boldsymbol{\theta}(t) &:= (\boldsymbol{\theta}_{v=a}, \boldsymbol{\theta}_{(v,w)=(a,b)} \mid v \in L_t,\ (v,w) \in L_t). \end{aligned} \tag{4.4}$$

By construction, the layers $L_1, L_2, \ldots, L_T$ define a non-overlapping partitioning of a graph G, which allows us to write

$$\langle \boldsymbol{\phi}(\boldsymbol{X}), \boldsymbol{\theta} \rangle = \sum_{t=1}^{T} \langle \boldsymbol{\phi}(t, \boldsymbol{X}), \boldsymbol{\theta}(t) \rangle.$$

The subvectors $\boldsymbol{\phi}(t, \boldsymbol{X})$ and $\boldsymbol{\theta}(t)$ have the same length $d' := d/T$ for all $t = 1, 2, \ldots, T$. Note that the subvectors should be "aligned", in the sense that the jth elements in all subvectors must point to the same node:state or edge:states pair over time. We illustrate this in Figure 4.2.

The spatial graph G_0 and the sizes of the vertex state spaces $\mathcal{X}_v$ determine the number of model parameters d. In order to compute this quantity, we consider the construction of G (as shown in Figure 4.1 (b)) from G_0. First, all vertices v and all edges (u, v) from G_0 are copied exactly T times and added to $G = (V, E)$, whereas each copy is indexed by time step t, i.e. $v \in V_0 \Rightarrow v_t \in V_t, 1 \le t \le T$ and likewise for the edges. Then, for each vertex $v_t \in V$ with $t \le T-1$, a temporal edge (v_t, v_{t+1}) is added to G. Finally, for each edge $(v_t, u_t) \in E$ with $t \le T-1$, the two spatio-temporal edges (v_t, u_{t+1}) and (v_{t+1}, u_t) are also added to G. The number of parameters per vertex v is $|\mathcal{X}_v|$ and accordingly $|\mathcal{X}_v||\mathcal{X}_u|$ per edge (v, u). Thus, the total number of model parameters is

$$\begin{aligned} d = \sum_{v \in V_0} \sum_{t=1}^{T} |\mathcal{X}_{v_t}| + \sum_{v \in V_0} \sum_{t=1}^{T-1} |\mathcal{X}_{v_t}|\,|\mathcal{X}_{v_{t+1}}| + \sum_{(u,v) \in E_0} |\mathcal{X}_{v_T}|\,|\mathcal{X}_{u_T}| \\ + \sum_{(u,v) \in E_0} \sum_{t=1}^{T-1} (|\mathcal{X}_{v_t}|\,|\mathcal{X}_{u_{t+1}}| + |\mathcal{X}_{v_{t+1}}|\,|\mathcal{X}_{u_t}| + |\mathcal{X}_{v_t}|\,|\mathcal{X}_{u_t}|)\,. \end{aligned} \tag{4.5}$$

If we assume that all vertices $v, u \in V$ share a common state space and that state spaces do not change over time, i.e. $\mathcal{X}_{v_t} = \mathcal{X}_{u_{t'}}, \forall v, u \in V, 1 \leq t, t' \leq T$, the expression simplifies to

$$d \;=\; \underbrace{T|V_0|\,|\mathcal{X}_{v_t}|}_{\text{\# of vertex parameters}} \;+\; \underbrace{\big[(T-1)(|V_0| + 3|E_0|) + |E_0|\big]\,|\mathcal{X}_{v_t}|^2}_{\text{\# of edge parameters}}$$

with some arbitrary but fixed vertex v_t. Note that the last two assumptions are only needed to simplify the computation of dimension d; the spatio-temporal random field that is described in the following section is not restricted by any of these assumptions.

This model now truly expresses temporal and spatial relations between all locations and points in time for all features. However, the memory requirements of such models are quite high due to the large problem dimension. Even loading or sending models may cause issues when mobile devices are the platform. Furthermore, the training does not scale well because of step-size adaption techniques that are based on sequential (i.e., non-parallel) algorithms.

4.1.6 Spatio-Temporal Random Fields

Now we describe how we modify the naive spatio-temporal graphical model discussed above. We have two goals in mind: (i) to achieve compact models retaining the same prediction power, and (ii) to find the best of such models via scalable distributed optimization.

4.1.6.1 Towards Better Sparsification

The memory consumption of MRFs is dominated by the size of its parameter vector: the graph G can be stored within $\mathcal{O}(|V| + |E|)$ space (temporal edges do not have to be constructed explicitly), and the size of intermediate variables required for inference is $\mathcal{O}(2|E||\mathcal{X}_v|)$. That is, if $|\mathcal{X}_v| \geq 2$ for all v, the dimension d in Equation 4.5 and therefore the memory consumption of the parameter vector are always a dominant factor. Also, since each parameter is usually accessed multiple times during inference, it is desirable to have them in a fast storage, e.g. a cache memory.

An important observation on the parameter subvector $\boldsymbol{\theta}(t)$ is that it is unlikely to be a zero vector when it models an informative distribution. For example, if the nodes can have one of the two states {high, low}, suppose that the corresponding parameters at time t satisfy $[\boldsymbol{\theta}(t)]_v = 0$ for all v and equally for all edge weights. Then it implies $\mathbb{P}(\boldsymbol{X}_v = \text{high}) = \mathbb{P}(\boldsymbol{X}_v = \text{low})$, a uniform marginal distribution. The closer the parameters of a classical MRF tend towards $\mathbf{0}$, the closer are the corresponding marginals to the uniform distribution.

When all consecutive layers are sufficiently close in time, the transition of distributions over the layers will be smooth in many real-world applications. But the optimal $\boldsymbol{\theta}$ is likely to be a dense vector, and it will require a large memory and possibly a long time to make predictions with it as we deal with large graphical models. This creates the necessity for a different parametrization.

4.1.6.2 Reparametrization

In our reparametrization, we consider a piecewise linear representation of $\boldsymbol{\theta}(t)$ with new parameter vectors $\boldsymbol{\Delta}_{\cdot i} \in \mathbb{R}^{d'}$ for $i = 1, 2, \dots, T$,

$$\boldsymbol{\theta}(t) = \sum_{i=1}^{t} \frac{1}{t-i+1}\boldsymbol{\Delta}_{\cdot i}, \quad t = 1, 2, \dots, T. \tag{4.6}$$

Our motivation is best shown by the differences in $\boldsymbol{\theta}$ between two consecutive layers, $\Delta_{(t-1):t} := \boldsymbol{\theta}(t) - \boldsymbol{\theta}(t-1) = \boldsymbol{\Delta}_{\cdot t} - \sum_{i=1}^{t-1} \frac{1}{(t-i+1)(t-i)}\boldsymbol{\Delta}_{\cdot i}$. That is, the difference (slope) is mostly captured by the first term $\boldsymbol{\Delta}_{\cdot t}$, and by the remainder terms $\boldsymbol{\Delta}_{\cdot (t-i)}$ with quadratically decaying weights in $\mathcal{O}(i^{-2})$, for $i = 1, 2, \dots, t$. We note that a simpler alternative might be setting $\boldsymbol{\theta}(t) = \sum_{i=1}^{t} \boldsymbol{\Delta}_{\cdot i}$, but our approach leads to better conditions in optimization which allow for faster convergence.

With the new parameters, if the changes between two consecutive layers are near zero, that is, $\boldsymbol{\theta}(t) \approx \boldsymbol{\theta}(t-1)$, then we expect $\boldsymbol{\Delta}_{\cdot t} \approx 0$. This is a novel property of the new parametrization, since with the classical parameters $\boldsymbol{\theta}$ the condition does not necessarily entail $\boldsymbol{\theta}(t) \approx 0$. In other words, $\boldsymbol{\Delta}_{\cdot t} = 0$ implies no changes in the distribution from $t-1$ to t, but $\boldsymbol{\theta}(t) = 0$ implies the distribution at t suddenly becoming a uniform distribution, regardless of the previous state at layer $t-1$. An example is illustrated in Figure 4.3.

Since we have defined $\boldsymbol{\theta}$ as a concatenation of vectors $\boldsymbol{\theta}(1), \boldsymbol{\theta}(2), \dots, \boldsymbol{\theta}(T)$, the reparametrization reads as follows:

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}(1) \\ \boldsymbol{\theta}(2) \\ \vdots \\ \boldsymbol{\theta}(T) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Delta}_{\cdot 1} \\ \frac{1}{2}\boldsymbol{\Delta}_{\cdot 1} + \boldsymbol{\Delta}_{\cdot 2} \\ \vdots \\ \sum_{i=1}^{T} \frac{1}{t-i+1}\boldsymbol{\Delta}_{\cdot i} \end{bmatrix}, \quad \boldsymbol{\Delta} := \begin{bmatrix} | & | & & | \\ \boldsymbol{\Delta}_{\cdot 1} & \boldsymbol{\Delta}_{\cdot 2} & \cdots & \boldsymbol{\Delta}_{\cdot T} \\ | & | & & | \end{bmatrix}.$$

For convenience, we define the *slope matrix* $\boldsymbol{\Delta} \in \mathbb{R}^{d' \times T}$ as above, which contains $\boldsymbol{\Delta}_{\cdot 1}$, $\boldsymbol{\Delta}_{\cdot 2}, \dots, \boldsymbol{\Delta}_{\cdot T}$ as its columns. In the following we sometimes use the notations $\boldsymbol{\theta}(\boldsymbol{\Delta})$ and $\boldsymbol{\theta}(t, \boldsymbol{\Delta})$, whenever it is necessary to emphasize the fact that $\boldsymbol{\theta}$ and $\boldsymbol{\theta}(t)$ are functions of

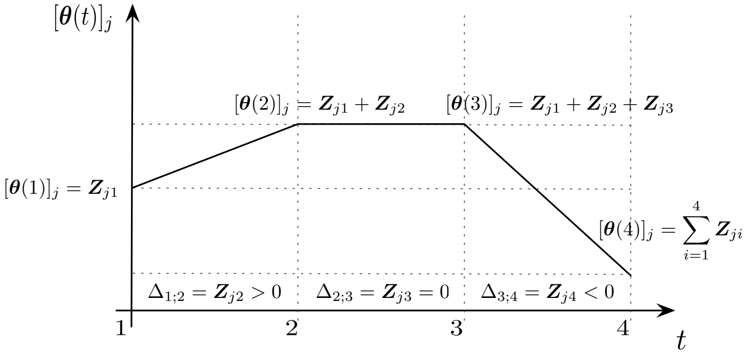

$[\boldsymbol{\theta}(t)]_j$
$[\boldsymbol{\theta}(2)]_j = \boldsymbol{Z}_{j1} + \boldsymbol{Z}_{j2}$
$[\boldsymbol{\theta}(3)]_j = \boldsymbol{Z}_{j1} + \boldsymbol{Z}_{j2} + \boldsymbol{Z}_{j3}$
$[\boldsymbol{\theta}(1)]_j = \boldsymbol{Z}_{j1}$
$[\boldsymbol{\theta}(4)]_j = \sum_{i=1}^{4} \boldsymbol{Z}_{ji}$
$\Delta_{1;2} = \boldsymbol{Z}_{j2} > 0$
$\Delta_{2;3} = \boldsymbol{Z}_{j3} = 0$
$\Delta_{3;4} = \boldsymbol{Z}_{j4} < 0$
1
2
3
4
t

Fig. 4.3: A simplified example of the reparametrization of $[\boldsymbol{\theta}(t)]_j$, the jth element in the subvector $\boldsymbol{\theta}(t)$, over the timeframes $t = 1, 2, 3, 4$. We store slopes $\boldsymbol{\Delta}_{jt}$ instead of the actual values of the piecewise linear function $[\boldsymbol{\theta}(t)]_j$ between two consecutive timeframes $t-1$ and t (except for $\boldsymbol{\Delta}_{j1}$ which works as an intercept). Near-zero slopes $\boldsymbol{\Delta}_{jt} \approx 0$ ($\boldsymbol{\Delta}_{j3} = 0$ above) can be removed from computation and memory.

$\boldsymbol{\Delta}$ under the new parametrization. Finally, another property of our reparametrization is that it is linear. Therefore an important property for optimization carries over: $A(\boldsymbol{\theta}(\boldsymbol{\Delta}))$ is convex in $\boldsymbol{\Delta}$ as $A(\boldsymbol{\theta})$ is convex in $\boldsymbol{\theta}$ [692].

We note that due to the summation in Equation 4.6 our reparametrization with $\boldsymbol{\Delta}$ introduces some additional overhead compared with the classical parametrization with $\boldsymbol{\theta}$. In particular, whenever an algorithm has to read a value from $\boldsymbol{\theta}$, it has do be decompressed instantly, which adds asymptotic complexity $\mathcal{O}(T)$ to every access. However, if we obtain a *sparse representation* with $\boldsymbol{\Delta}$, then it can be stored in small memory (possibly even in CPU cache memory) and therefore the chances for cache misses or memory swapping will be reduced. This becomes an important factor when, say, we deploy a learned model to applications running on mobile devices. Chapter 7 presents approaches to memory-aware learning in other classes of learning methods.

4.1.6.3 Analysis

We define the l_1 and l_2 regularizers for the slope matrix $\boldsymbol{\Delta}$ as follows,

$$\|\boldsymbol{\Delta}\|_1 := \sum_{j=1}^{d'} \|\boldsymbol{\Delta}_{j\cdot}\|_1, \quad \|\boldsymbol{\Delta}\|_F^2 := \sum_{j=1}^{d'} \|\boldsymbol{\Delta}_{j\cdot}\|_2^2. \tag{4.7}$$

The two regularizers induce sparsity and smoothness respectively, as we have discussed in Section 4.1.4. The difference is that due to the reparametrization, now differences between parameters $\boldsymbol{\theta}(t-1)$ and $\boldsymbol{\theta}(t)$ are penalized, not the actual values they contain, which are unlikely to be zero.

The proposed reparametrizations can result in large improvements regarding a model's memory consumption. Clearly, the amount of reduction depends on the specific dataset. It is hence even more astonishing that the reparametrization itself can be applied without any harm—it can represent any natural parameter. Let us consider a proper definition of our former intuition. For the sake of generality, let C be any clique (e.g., an edge) of the underlying graph.

Definition 4 (Piecewise Linear Reparametrization [567]). *Let G be a spatio-tem-poral graph of length T, and let $\boldsymbol{D}(h) \in [0;1]^{h \times h}$ be a lower unitriangular[3] matrix. Any MRF with graph G and piecewise linear clique-wise reparametrization*

$$\boldsymbol{\theta}_{C=\boldsymbol{x}'} = \eta_{\boldsymbol{D}(h)}(\boldsymbol{\Delta}_{C=\boldsymbol{x}'}) = \boldsymbol{D}(h)\boldsymbol{\Delta}_{C=\boldsymbol{x}'} \tag{4.8}$$

where $h = T - (\max\{t' \mid v(t') \in C\} - \min\{t' \mid v(t') \in C\})$ is called a spatio-temporal random field.

Based on that definition, we can derive some useful properties.

Lemma 5 (Universality of the Reparametrization). *The spatio-temporal repara-metrization is universal. That is, the piecewise linear reparametrization is a bijection.*

Proof Indeed, any $\boldsymbol{\Delta} \in \mathbb{R}^d$ can be mapped to some $\boldsymbol{\theta} \in \mathbb{R}^d$ by multiplication with $\boldsymbol{D}$ according to Definition 4. To see that the converse also holds, note that for each $t \in [T]$, $\det \boldsymbol{D}(h) = \prod_{i=1}^{t} \boldsymbol{D}(h)_{i,i} = 1$, due to unitriangularity. Each $\boldsymbol{D}(h)$ is thus invertible and so is the block diagonal matrix $\boldsymbol{D}^{\circ}$. So for any given natural parameter $\boldsymbol{\theta}_{C=\boldsymbol{y}}$, we can find the corresponding reparametrization via $\boldsymbol{\Delta}_{C=\boldsymbol{y}} = \boldsymbol{D}^{-1}\boldsymbol{\theta}_{C=\boldsymbol{y}}$. That is, $\eta_{\boldsymbol{D}}$ is bijective and hence universal. ■

Since $\eta_{\boldsymbol{D}}$ is universal, any natural parameter can be represented via some $\boldsymbol{\Delta}$. Moreover, $\eta_{\boldsymbol{D}}$ is a linear function of $\boldsymbol{\Delta}$. The convexity of a function is preserved by composing it with a linear function. Hence, the reparametrized negative average log-likelihood $\ell(\eta_{\boldsymbol{D}}(\boldsymbol{\Delta}); \mathcal{D}) = A(\eta_{\boldsymbol{D}}(\boldsymbol{\Delta})) - \langle \eta_{\boldsymbol{D}}(\boldsymbol{\Delta}), \tilde{\boldsymbol{\mu}} \rangle$ is a convex function of $\boldsymbol{\Delta}$.

Up to now, we have not saved any memory since $\boldsymbol{\Delta}$ and $\boldsymbol{\theta}$ have the same dimension. By imposing l_1- and l_2-regularization on the reparametrized objective, we arrive at the problem

$$\min_{\boldsymbol{\Delta} \in \mathcal{R}^d} \underbrace{A(\eta_{\boldsymbol{D}}(\boldsymbol{\Delta})) - \langle \eta_{\boldsymbol{D}}(\boldsymbol{\Delta}), \tilde{\boldsymbol{\mu}} \rangle + \frac{\lambda_2}{2}\|\boldsymbol{\Delta}\|_F^2 + \lambda_1\|\boldsymbol{\Delta}\|_1}_{\ell^{\mathrm{ST}}(\boldsymbol{\Delta};\mathcal{D})} . \tag{4.9}$$

The following theorem shows that the intuition that we used to design our reparametrization has indeed the desired effect—it allows us to convert redundancy into sparsity by detecting negligible changes in consecutive natural parameters. Moreover, a polynomial number of samples suffices to achieve a small estimation error with high probability.

[3] An unitriangular matrix is triangular and all entries on its main diagonal are 1.

Theorem 6 (STRF Consistency). *Consider a random variable $\boldsymbol{X}$ with exponential family density, parameter $\boldsymbol{\theta}^\star \in \mathbb{R}^d$ whose reparametrization has minimal norm among all equivalent parameters, and a generalized sequence structure of length T. We are given a dataset $\mathcal{D}$ with $N = |\mathcal{D}|$ samples from $\boldsymbol{X}$. Suppose $\|\nabla^2 A(\boldsymbol{\theta}^\star)^{-1}\|_\infty \leq \kappa$ and $\|\boldsymbol{\Delta}\|_\infty \leq \gamma$, and set $\lambda_1 = 4T\sqrt{\log(d)/N}$ and $\lambda_2 = \gamma^{-1}\lambda_1$. If $N \geq 324\kappa^4 d^{12} \log(d)/(T-d^2)^2$, then, for an arbitrary decay matrix $\boldsymbol{D}$:*

- *the distance between the true parameter $\boldsymbol{\theta}^\star$ and the estimate $\eta_{\boldsymbol{D}}(\hat{\boldsymbol{\Delta}})$ is bounded, i.e.,*

$$\|\eta_{\boldsymbol{D}}(\hat{\boldsymbol{\Delta}}) - \boldsymbol{\theta}^\star\|_\infty \leq 3\kappa d^2 \lambda_1 \ ,$$

- *any sparsity in the estimate implies some redundancy in the true parameter, i.e., $\hat{\boldsymbol{\Delta}}_{C=\boldsymbol{x}'}(t) = 0 \Rightarrow$*

$$\begin{aligned} & |\boldsymbol{\theta}^\star_{C=\boldsymbol{x}'}(t-1) - \boldsymbol{\theta}^\star_{C=\boldsymbol{x}'}(t)| \\ \leq \quad & \frac{3d^2\kappa\lambda_1}{T} + (t-1)\left(\max_{i=1}^{t-1} |\hat{\boldsymbol{\Delta}}_{C=\boldsymbol{x}'}(i)| + \frac{3d^2\kappa\lambda_1}{T}\right) \ , \end{aligned}$$

for any clique C and time-point t. Both statements hold with probability at least $1-(2/d)$.

A proof for this statement can be found in [567].

4.1.7 Experimental

We evaluate the performance of our suggested method on two real-world datasets, where each set is described by a spatial graph $G_0 = (V_0, E_0)$ with a set of sensors V_0 and connections E_0, and a set of historical sensor readings $\mathcal{D}$. We evaluate two approaches: MRFs with the original parametrization (MRF) and the spatio-temporal random fields[4] (STRF) presented in this section.

First we discuss the model training. We investigate the prediction quality and sparsity of resulting models with respect to regularization parameters. We also present the impact of separable optimization on training time. Next, the quality of prediction on test sets is discussed, regarding the sparsity (and thereby the size in memory) of trained models. Finally, we discuss the qualitative results regarding the interpretability of the STRF model.

Throughout the experiments, our STRF algorithm has produced solutions satisfying our target optimality of $< 10^{-5}$ within ten iterations. A description of the traffic and temperature datasets as well as the quality measures (accuracy Acc and number-of-non-zero-ratio NNZ) used for this evaluation can be found in [566].

[4] An implementation is part of the Python package pxpy which is available at https://pypi.org/project/pxpy.

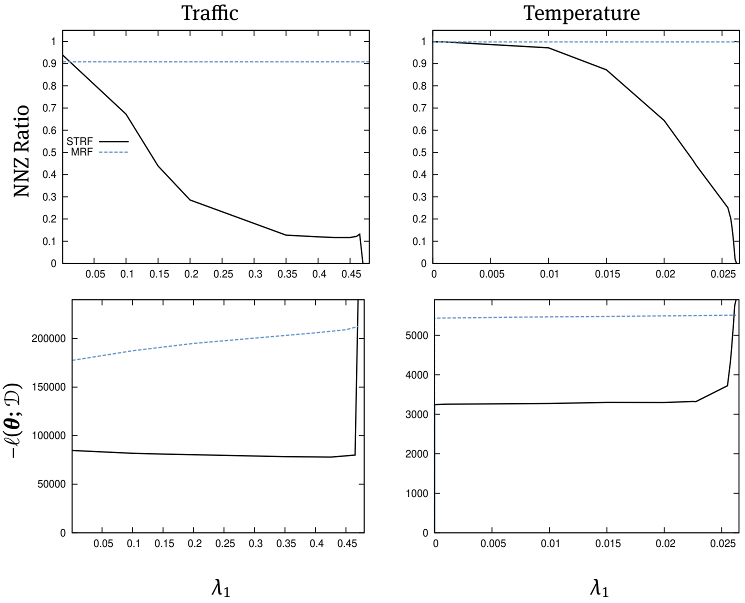


Fig. 4.4: The effect of regularization on models for varying sparsity parameter λ_1 (left: traffic data, right: temperature data, top: NNZ ratio, bottom: negative log-likelihood). All measurements were obtained after ten iterations, which was enough for STRF to reach the target optimality.

4.1.8 Regularized Training of Spatio-Temporal Random Fields

In our model, the l_2 regularizer imposes "smoothness" on the dynamics of parameters over time, providing a controllable way to avoid overfitting noisy observations. The degree of smoothness is controlled by λ_2, whereas the compression ratio is controlled by λ_1. Positive values of λ_2 help in our method, since the curvature estimation becomes better conditioned.

4.1.8.1 Sparsity of Trained Models and Their Training Accuracy

Figure 4.4 shows the performance of STRF (our method) and MRF (classical parametrization) in terms of the negative log-likelihood and the NNZ ratio for a range of values for λ_1. The parameter λ_2 was fixed to 10^{-1} (the characteristics were almost identical for various λ_2 values we tried in the range of $[0, 1]$). For MRF, we augmented the objective with the l_1 and l_2 regularizers discussed in Section 4.1.4, and then applied a subgradient descent method with fixed step size ($\eta = 10^{-2}$). Our results show that (i) the subgradient method does not properly perform regularization for MRF, regardless of the choices of (λ_1, λ_2); (ii) the negative log-likelihood decreases as λ_1 is increased, which is expected because at the strongest l_1 regularization will force all marginals to be uniform distributions; (iii) our method STRF identifies sparse models accordingly to given regularization strength, while retaining similar likelihood values to MRF. More precisely, focusing on the curves for STRF, likelihood keeps improving until λ_1 reaches 0.47. Beyond this value, the model is compressed too much, losing its prediction power. Overall, the pair $(\lambda_1, \lambda_2) = (0.4655, 1.0)$ with NNZ ratio 0.101573 has been identified as a good choice for the traffic data, and the pair $(\lambda_1, \lambda_2) = (0.0255, 1.0)$ with NNZ ratio 0.248136 has been identified as a good choice for the temperature data, since both lead to sparse models with reasonable likelihood values. We use these values in the following experiments.

Since the number of edge parameters is a dominant factor in the dimension d of the parameter space, it would be desirable that STRF sufficiently compresses edge parameters. Considering the NNZ ratio of vertex and edge parameters separately, it turns out that STRF has such a property: with the good parameter values above, the NNZ ratio of vertices is about 0.95, whereas that of the edges is about 0.09.

4.1.9 Prediction on Test Sets

Here we investigate (i) the test-set performance of the sparse models, obtained with the good parameter values of λ_1 and λ_2 found in training, and (ii) how the sparsity of trained models affect the testing time.

The test-set accuracy of the models, obtained by the regularization parameters described in Section 4.1.8.1, is presented in Figure 4.5. Here our method STRF, the classical MRF, the kNN algorithm with several values of k, and the random guessing method, are compared. The prediction quality of the models produced by STRF is almost identical to that of MRF, although the STRF models are much smaller in size (10.2 % and 24.8 % of the MRF models in size, for traffic and temperature, respectively). The kNN algorithm sometimes performs better than STRF and MRF, but remember that kNN cannot capture probabilistic relations and requires access to full training data, which is not the case for STRF and MRF.

4.1.10 Conclusion

In this contribution, we presented an improved graphical model designed for the efficient probabilistic modeling of spatio-temporal data. It is based on a combination of parametrization and regularization, such that the estimated parameters are sparse and the estimated marginal probabilities are smooth without losing prediction accuracy. We investigated the sparsity, smoothness, prediction accuracy, and scalability of the model on real-world datasets. The experiments showed that often around 10 % of the original model size suffices to achieve almost the same prediction accuracy. Moreover, the method is amenable to parallelization and scales well with an increasing number of CPUs.

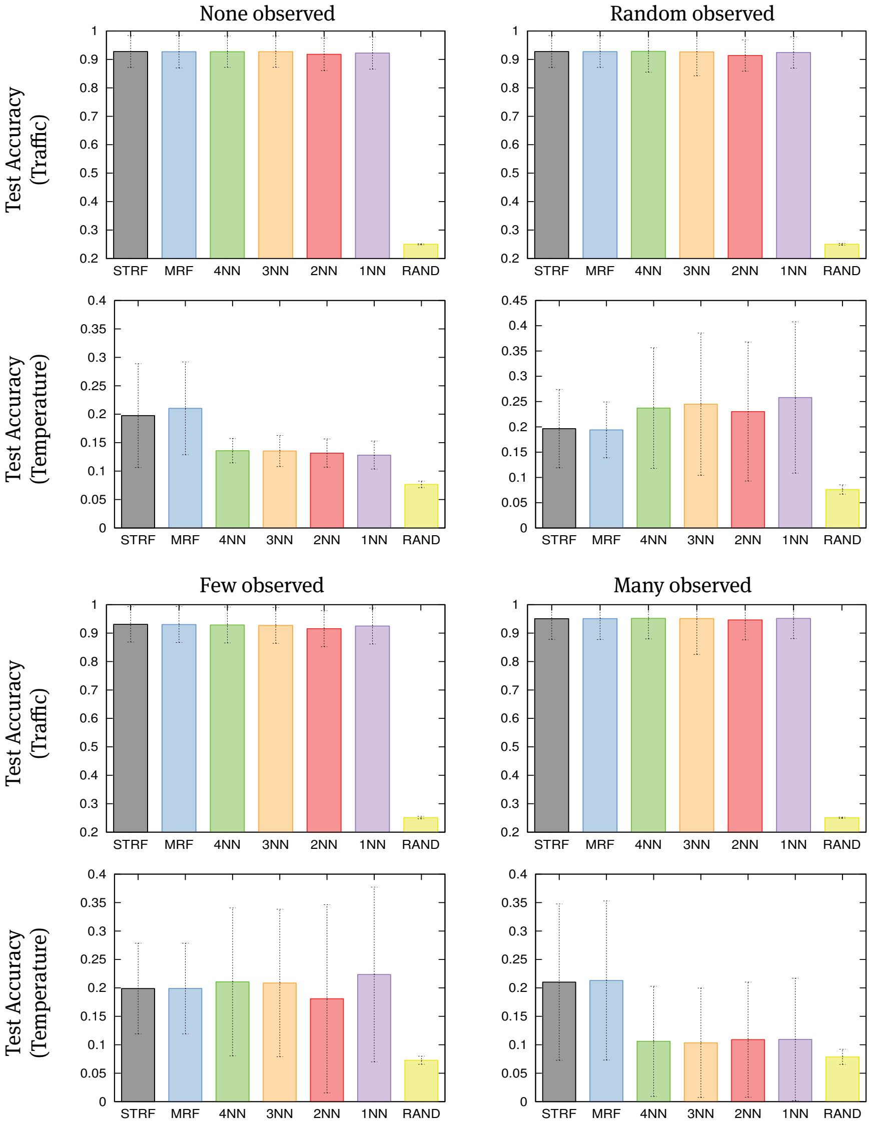


Fig. 4.5: Test accuracy of STRF, MRF, and *k*-nearest neighbor algorithm on the traffic dataset for four scenarios: unconditioned (first column, first two rows), random observed layers (second column, first two rows), conditioned on Monday (first column, last two rows), conditioned on Monday to Saturday (first column, last two rows).

4.2 The Weisfeiler-Leman Method for Machine Learning with Graphs

Nils Kriege
Christopher Morris

Abstract: The Weisfeiler-Leman method is a classic heuristic for graph isomorphism testing, which iteratively encodes vertex neighborhoods of increasing radius by vertex colors. Two graphs whose vertex colors do not match are called non-isomorphic. The method is fundamental for recent advances in machine learning with graphs, e.g., graph kernels and graph neural networks. This contribution overviews the development of graph kernels based on the Weisfeiler-Leman algorithm, which are among the most successful graph kernels today. We describe the Weisfeiler-Leman heuristic for graph isomorphism testing, from which the classical Weisfeiler-Leman subtree kernel directly follows. Further, we summarize the theory of optimal assignment kernels and present the Weisfeiler-Leman optimal assignment kernel for graphs and the related Wasserstein Weisfeiler-Leman graph kernel. We discuss kernel functions based on the k-dimensional Weisfeiler-Leman algorithm, a strict generalization of the Weisfeiler-Leman heuristic. We show that a local, sparsity-aware variant of this algorithm can lead to scalable and expressive kernels. Moreover, we survey other kernels based on the principle of Weisfeiler-Leman refinement. Finally, we shed some light on the connection between Weisfeiler-Leman-based kernels and neural architectures for graph-structured input.

4.2.1 Introduction

Graph-structured data is ubiquitous across application domains ranging from chemo- and bioinformatics [40, 647] to image [633] and social network analysis [193]. In drug discovery, molecules are represented as graphs [379] and the search for promising drug candidates that bind to a specific target protein can be greatly accelerated by machine learning methods suitable for graph data. Moreover, proteins themselves [64] as well as their interactions and complexes [646] (also see 2.6 in Volume 3) can be adequately modeled as graphs. The increasing amount of data in these areas offers enormous potential in studying diseases and their cures. However, due to the size and complexity of the data, automated methods for their analysis are required.

To develop successful machine learning models in these domains, we need techniques that can exploit the rich information inherent in the graph structure and the feature information contained within vertices and edges. In recent years, numerous approaches have been proposed for machine learning with graphs—most notably, methods based on graph kernels [398] and graph neural networks (GNN) [122, 252, 272].

Here, graph kernels based on the 1*-dimensional Weisfeiler-Leman algorithm* (1-WL) [28, 271], and corresponding GNNs [509, 714] have recently advanced the state of the art in supervised node and graph learning.

The 1-WL was introduced as a heuristic for the graph isomorphism problem and is widely used as a subroutine in graph isomorphism and canonization algorithms following the individualization-refinement paradigm [480]. It allows recognizing two graphs as non-isomorphic. More precisely, 1-WL assigns colors to the nodes of two graphs in an iterative process, such that isomorphic graphs are assigned matching node colors. Whenever two graphs obtain different colorings, they are guaranteed to be non-isomorphic. However, two graphs with matching colors may still be non-isomorphic. The abilities and limitations of the 1-WL for this task have been studied for decades and are well understood [271]. In machine learning with graph-structured data, the goal is less clear, and a general objective is to compute a meaningful similarity between graphs. Two graphs that are non-isomorphic but differ only by one edge, say, should still be considered highly similar. In practical applications, it has been observed that the Weisfeiler-Leman technique is often suitable to approximate computationally demanding graph similarity measures based on the minimum number of edit operations required to transform one graph into the other [397, 646]. (See 2.6 in Volume 3 for details.) Moreover, Weisfeiler-Leman type algorithms are remarkably successful in machine learning tasks. However, their abilities and limitations in these applications are not well understood and are the subject of current research.

Here, we give an overview of the recent progress of graph kernels based on the Weisfeiler-Leman paradigm. That is, we review the 1-WL and its more expressive generalization, the k-WL. Starting from the Weisfeiler-Leman subtree kernel [627], a simple graph kernel based on the 1-WL, we survey the area with a focus on assignment-based kernels and an extension based on the k-WL. Moreover, we overview the connections between the Weisfeiler-Leman algorithm and graph neural networks.

4.2.2 Preliminaries

In the following, we introduce notation and give the necessary background on graph s. As usual, let $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ for $n \geq 1$, and let $\{\!\{\ldots\}\!\}$ denote a multiset.

4.2.2.1 Graphs

A *graph* G is a pair (V, E) with a finite set of *vertices* V and a set of *edges* $E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$. We denote the set of vertices and the set of edges of G by $V(G)$ and $E(G)$, respectively. For ease of notation, we denote the edge $\{u, v\}$ in $E(G)$ by (u, v) or (v, u). In the case of *directed graphs* the order of the nodes is distinguished and $E \subseteq \{(u, v) \in V \times V \mid u \neq v\}$. A *labeled graph* G is a triple (V, E, l) with a label function $l\colon\ V(G) \cup E(G) \to \Sigma$, where Σ is some finite alphabet. Then $l(v)$ is the *label* of v in $V(G) \cup E(G)$. The

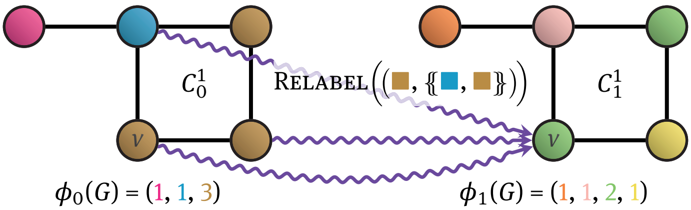


Fig. 4.6: Illustration of the coloring scheme of the 1-WL.

neighborhood of v in $V(G)$ is denoted by $\delta(v) = N(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$. Let $S \subseteq V(G)$ then $G[S] = (S, E_S)$ with $E_S = \{(u, v) \in E(G) \mid u, v \in S\}$ is the subgraph of G *induced* by S. A *tree* is a connected graph without cycles. A *rooted tree* is a tree with a designated vertex called *root* in which the edges are directed such that they point away from the root. Let p be a vertex in a rooted tree; we call its out-neighbors *children* with parent p.

We say that two graphs G and H are *isomorphic* if there exists a bijection $\varphi\colon V(G) \to V(H)$ that preserves the edges, i.e., (u, v) is in $E(G)$ if and only if $(\varphi(u), \varphi(v))$ is in $E(H)$ for all u and v in $V(G)$. If G and H are isomorphic, we write $G \simeq H$ and call φ an *isomorphism* between G and H. Moreover, we call the equivalence classes induced by $\simeq$ *isomorphism types*. In the case of labeled graphs, we additionally require that $l(v) = l(\varphi(v))$ for all v in $V(G)$ and $l((u, v)) = l((\varphi(u), \varphi(v)))$ for all (u, v) in $E(G)$.

4.2.2.2 Kernels

A *kernel* on a non-empty set $\mathcal{X}$ is a symmetric, positive semidefinite function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Equivalently, a function k is a kernel if there is a *feature map* $\phi\colon \mathcal{X} \to \mathcal{H}$, where $\mathcal{H}$ is a Hilbert space endowed with the inner product $\langle \cdot, \cdot \rangle$, such that $k(x, y) = \langle \phi(x), \phi(y) \rangle$ for all x and y in $\mathcal{X}$. Let $\mathcal{G}$ be the set of all graphs, then a kernel on $\mathcal{G}$ is called a *graph kernel*.

4.2.3 The Weisfeiler-Leman Algorithm

The 1-WL is a classical heuristic for the graph isomorphism problem [28, 273, 700]. Here, we formally introduce the 1-WL and its generalization, the k-WL, which form the basis for the graph kernels described in the following sections.

4.2.3.1 The 1-dimensional Weisfeiler-Leman Algorithm

Intuitively, the 1-WL aims to capture the structure of a graph by iteratively aggregating labels or *colors* of adjacent vertices. Two equally colored vertices get a different color if their neighborhood is colored differently. See Figure 4.6 for an illustration.

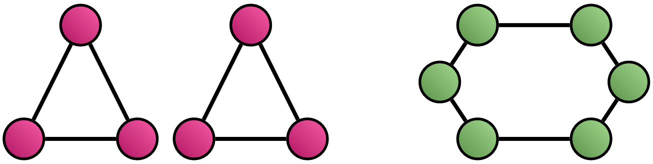


Fig. 4.7: Two graphs that cannot be distinguished by the 1-WL.

Formally, let (G, l) be a labeled graph. In each iteration $i \geq 0$, the algorithm computes a *coloring* $C^1_i \colon V(G) \to \mathbb{S}$, where $\mathbb{S}$ is some arbitrary codomain. In the first iteration, we color the vertices according to the labeling l, i.e., $C^1_0(v) = l(v)$ for v in $V(G)$. For $i \geq 0$, C^1_{i+1} is defined by

$$C^1_{i+1}(v) = \textsc{Relabel}\left(C^1_i(v), \{\!\!\{ C^1_i(w) \mid w \in \delta(v) \}\!\!\}\right).$$

Here, Relabel is an injection that maps the pair consisting of the current color and the multiset of colors of adjacent vertices to a new color. Hence, two vertices with the same color in iteration i get a different color in the next iteration if the number of neighbors colored with a certain color is different. Observe that it is straightforward to extend the 1-WL to labeled, directed graphs. We run the algorithm until convergence, i.e.,

$$C^1_i(v) = C^1_i(w) \iff C^1_{i+1}(v) = C^1_{i+1}(w),$$

holds for all v and w in $V(G)$. We call the partition of $V(G)$ induced by C^1_i the *stable partition*. For such i, we define $C^1_\infty(v) = C^1_i(v)$ for v in $V(G)$. For two graphs G and H, we run the algorithm in "parallel" on both graphs. Then the 1-WL *distinguishes* between them if

$$|V(G) \cap (C^1_\infty)^{-1}(c)| \neq |V(H) \cap (C^1_\infty)^{-1}(c)|,$$

for some color c in the codomain of C^1_∞. If the 1-WL distinguishes two graphs, the graphs are not isomorphic.

4.2.3.2 *k*-dimensional Weisfeiler-Leman Algorithm

The 1-WL is not able to distinguish between all pairs of non-isomorphic graphs. See Figure 4.7 for such a pair. The k-WL is a natural generalization of the 1-WL, which gets more powerful by coloring k-tuples defined over the set of vertices.

Formally, let G be a graph, and let $k \geq 2$. Moreover, let $\mathbf{v}$ be a tuple in $V(G)^k$, then $G[\mathbf{v}]$ is the subgraph induced by the components of $\mathbf{v}$, where the vertices are labeled with integers from $\{1, \ldots, k\}$ corresponding to indices of $\mathbf{v}$. In each iteration $i \geq 0$, the algorithm computes a *coloring* $C^k_i \colon V(G)^k \to \mathbb{S}$, where $\mathbb{S}$ is some arbitrary codomain. In the first iteration ($i = 0$), two tuples $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ get the same color if the map $v_i \mapsto w_i$ is an isomorphism between $G[\mathbf{v}]$ and $G[\mathbf{w}]$. Now, for $i \geq 0$, C^k_{i+1} is defined by

$$C^k_{i+1}(\mathbf{v}) = \textsc{relabel}(C^k_i(\mathbf{v}), M_i(\mathbf{v})),$$

where the multiset

$$M_i(\mathbf{v}) = \big(\{\!\{C_i^k(\phi_1(\mathbf{v}, w)) \mid w \in V(G)\}\!\}, \ldots, \\ \{\!\{C_i^k(\phi_k(\mathbf{v}, w)) \mid w \in V(G)\}\!\}\big), \tag{4.10}$$

and

$$\phi_j(\mathbf{v}, w) = (v_1, \ldots, v_{j-1}, w, v_{j+1}, \ldots, v_k).$$

That is, $\phi_j(\mathbf{v}, w)$ replaces the j-th component of the tuple $\mathbf{v}$ with the vertex w. We run the algorithm until convergence, i.e.,

$$C_i^k(\mathbf{v}) = C_i^k(\mathbf{w}) \iff C_{i+1}^k(\mathbf{v}) = C_{i+1}^k(\mathbf{w}),$$

for all $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ holds, and call the partition of $V(G)^k$ induced by C_i^k the *stable partition*. For such i, we define $C_\infty^k(\mathbf{v}) = C_i^k(\mathbf{v})$ for $\mathbf{v}$ in $V(G)^k$. The procedure of determining if two graphs are non-isomorphic is the same as for the 1-WL. With increasing k the algorithm gets more and more powerful [117]. That is, for each $k \geq 2$ there exists a pair of graphs that the k-WL cannot distinguish but the $(k+1)$-WL can.

Let A and B be two heuristics for the graph isomorphism problem, e.g., the k-WL, then we write $A \sqsubseteq B$ ($A \sqsubset B$, $A \equiv B$), if algorithm A is more powerful (strictly more powerful, equally powerful) than B in terms of distinguishing non-isomorphic graphs. Using this notation we write

$$(k+1)\text{-WL} \sqsubset k\text{-WL},$$

for $k \geq 2$, to state the result mentioned in the last paragraph.

4.2.4 Kernels Based on the Weisfeiler-Leman Algorithm

The Weisfeiler-Leman algorithm forms the basis for some of the most successful graph kernels. Here, we give an overview on kernels based on the 1-WL, followed by kernels based on the k-WL. Moreover, we survey other kernels related to the Weisfeiler-Leman paradigm.

4.2.4.1 Weisfeiler-Leman Subtree Kernel

The idea of the *Weisfeiler-Leman subtree graph kernel* [627] is to compute the 1-WL for $h \geq 0$ iterations resulting in a label function $C_i^1 \colon V(G) \to \mathbb{S}_i$ for each iteration $0 \leq i \leq h$. Now after each iteration, we compute a *feature vector* $\phi_i(G)$ in $\mathbb{R}^{|\mathbb{S}_i|}$ for each graph G. Each component $\phi_i(G)_c$ counts the number of occurrences of vertices labeled with c in $\mathbb{S}_i$. The overall feature vector $\phi_{\mathrm{WL}}(G)$ is defined as the concatenation of the feature vectors of all h iterations, i.e.,

$$\phi_{\mathrm{WL}}(G) = \big[\phi_0(G), \ldots, \phi_h(G)\big].$$

The Weisfeiler-Leman subtree kernel for h iterations is then computed as

$$k_{\mathrm{WL}}(G,H) = \langle \phi_{\mathrm{WL}}(G), \phi_{\mathrm{WL}}(H) \rangle,$$

where $\langle \cdot, \cdot \rangle$ denotes the standard inner product. The running time for a single feature vector computation is in $\mathcal{O}(hm)$ and $\mathcal{O}(Nhm + N^2hn)$ for the computation of the gram matrix for a set of N graphs [627], where n and m denote the maximum number of vertices and edges over all N graphs, respectively.

4.2.4.2 Weisfeiler-Leman Optimal Assignment Kernels

The Weisfeiler-Leman subtree kernel counts pairs of vertices with the same label. A different approach is to *assign* each vertex of G to a vertex of H. Constructing an assignment that maximizes the structural overlap and agreement of vertex attributes is a general concept for comparing graphs and also forms the basis of *graph matching* or *network alignment*. This principle was proposed to obtain graph kernels, where the similarity between two vertices is determined by an arbitrary base kernel [236]. However, it was soon observed that the resulting similarity measure is in general not positive semidefinite [685]. Subsequent research has identified a specific class of base kernels, for which the similarity derived from optimal assignments is guaranteed to be a valid kernel, i.e., positive semidefinite [395]. We summarize the theory of valid assignment kernels and then describe how a suitable base kernel can be obtained from the 1-WL.

Valid Optimal Assignment Kernels We consider the general setting, where the elements of two sets are to be assigned to each other. Let $[\mathcal{X}]^n$ denote the set of all n-element subsets of a set $\mathcal{X}$ and $\mathfrak{B}(X,Y)$ the set of all bijections between X and Y in $[\mathcal{X}]^n$ for n in $\mathbb{N}$. The *optimal assignment kernel* $K_{\mathfrak{B}}^k$ on $[\mathcal{X}]^n$ is defined as

$$K_{\mathfrak{B}}^k(X,Y) = \max_{B \in \mathfrak{B}(X,Y)} \sum_{(x,y)\in B} k(x,y), \tag{4.11}$$

where k is a *base kernel* on $\mathcal{X}$. For the application to sets of different cardinality, the smaller set can be augmented by dummy elements d with $k(d,\cdot) = 0$.

Similar to the concept of an ultrametric, which must satisfy the strong triangle inequality, the so-called *strong kernel* was introduced as a kernel satisfying $k(x,y) \geq \min\{k(x,z), k(z,y)\}$ for all x, y, z in $\mathcal{X}$. It was shown that the function $K_{\mathfrak{B}}^k$ is a valid kernel if k is a strong kernel [395]. Strong kernels are equivalent to kernels obtained from a hierarchical partition of their domain. Formally, let T be a rooted tree such that the leaves of T are the elements of $\mathcal{X}$ and $\omega\colon V(T) \to \mathbb{R}_{\geq 0}$ a weight function. We refer to the tuple (T,ω) as a *hierarchy*. A hierarchy on $\mathcal{X}$ induces a similarity $k(x,y)$ for x and y in $\mathcal{X}$ as follows. For v in $V(T)$ let $P(v) \subseteq V(T)$ denote the set of vertices in T on the path from v to the root r. Then the similarity between x and y in $\mathcal{X}$ is

$$k(x,y) = \sum_{v \in P(x) \cap P(y)} \omega(v).$$

For every strong kernel k there is a hierarchy that induces k and, vice versa, every hierarchy induces a strong kernel [395].

The optimal assignment kernel of Equation 4.11 can be computed in linear time from the hierarchy (T, ω) of the base kernel k by histogram intersection. For a node v in $V(T)$ and a set $X \subseteq \mathcal{X}$, let X_v denote the subset of X that is contained in the subtree rooted at v. Then the optimal assignment kernel is

$$K_{\mathfrak{B}}^{k}(X, Y) = \sum_{v \in V(T)} \min\{|X_v|, |Y_v|\} \cdot \omega(v), \tag{4.12}$$

which can be seen as the histogram intersection kernel for appropriately defined histograms representing the sets X and Y under the strong base kernel k [395].

Optimal Assignment Kernels from the 1-WL The 1-WL produces a hierarchy on the vertices of a (set of) graphs, where the ith level consists of nodes $\mathbb{S}_{i+1}$ with an artificial root at level 0. The parent-child relationships are given by the color refinement process, where the root has children $\mathbb{S}_1$. This hierarchy with a uniform weight function induces the strong base kernel

$$k(u, v) = \sum_{i=0}^{h} k_\delta(C_i^1(u), C_i^1(v)), \quad k_\delta(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

on the vertices. The kernel counts the number of iterations required to assign different colors to the vertices and reflects the extent to which the vertices have a structurally similar neighborhood. The optimal assignment kernel with this base kernel is referred to as *Weisfeiler-Leman optimal assignment kernel* and was shown to achieve better accuracy results in many classification experiments than the Weisfeiler-Leman subtree kernel. Moreover, the weights of the hierarchy associated with a strong base kernel can be optimized via multiple kernel learning [396].

4.2.4.3 Wasserstein Weisfeiler-Leman Graph Kernels

Related to assignment kernels are techniques based on the Wasserstein distance. Given two vectors a and b in $\mathbb{R}_+^n$ with entries that sum to the same value and a ground cost matrix D in $\mathbb{R}_+^{n \times n}$, the *Wasserstein distance* (or *earth mover's distance*, *optimal transport distance*)[5] is

$$W(a, b) = \min_{T \in \Gamma(a,b)} \langle T, D \rangle, \quad \Gamma(a, b) = \left\{ T \in \mathbb{R}_+^{n \times n} \colon T\mathbf{1} = a, T^\top \mathbf{1} = b \right\}, \tag{4.14}$$

where $\Gamma(a, b)$ is the set of so-called *transport plans* and $\langle \cdot, \cdot \rangle$ denotes the Frobenius dot product. Although $\Gamma(a, b)$ allows doubly stochastic matrices, the Wasserstein distance

[5] Depending on the context, slightly different definitions are used in the literature. Often, they require that a and b be distributions.

is a generalization of the min-version of Equation 4.11. The ground cost matrix, providing the dissimilarity between entries of a and b, has a role analogous to the base kernel.

The Wasserstein distance can be applied to the vertices of two graphs using ground costs obtained by 1-WL [663]. The entries of D are given by

$$d(u,v) = \frac{1}{h+1}\sum_{i=0}^{h}\rho(C_i^1(u), C_i^1(v)), \quad \rho(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise.} \end{cases} \tag{4.15}$$

Equation 4.15 is closely related to Equation 4.13 and can be regarded as its associated normalized distance. The Wasserstein distance $W(a, b)$ of Equation 4.14 is then combined with a distance-based kernel [283], specifically a variant of the Laplacian kernel. The resulting function was shown to be positive semidefinite. The authors also proposed extending the 1-WL to continuous attributes replacing discrete colors with real-valued vectors. Then, the ground costs of the Wasserstein distance are obtained from the Euclidean distance between these vectors. However, in this case, it is not guaranteed that the resulting function is positive semidefinite.

The Weisfeiler-Leman assignment kernel and the Wasserstein Weisfeiler-Leman kernel employ the 1-WL and improve the classification accuracy observed in practice on many datasets over the Weisfeiler-Leman subtree kernel. However, they are not more powerful in distinguishing non-isomorphic graphs. One approach to obtain kernels more expressive in this sense is to use the k-WL.

4.2.4.4 Kernels Based on the k-WL

The k-WL was also used to derive graph kernels [504, 506]. Essentially, the kernel computation works the same way as in the 1-dimensional case, i.e., a feature vector is computed for each graph based on color counts. To make the algorithm more scalable, the authors of [506] resorted to color all subgraphs on k vertices instead of all k-tuples, resulting in a less expressive algorithm. Moreover, the authors proposed that only a subset of the original neighbors be considered to exploit the sparsity of the underlying graph. Further, they offered a sampling-based approximation algorithm to speed up the kernel computation for a large graph, showing that the kernel can be approximated in constant time, i.e., independent of the number of vertices and edges, with an additive approximation error. Finally, they showed empirically that the proposed kernel beats the Weisfeiler-Leman subtree kernel on a subset of tested benchmark datasets.

Similarly, Morris, Rattan, and Mutzel [504] proposed graph kernels based on the k-WL. Again they proposed a local variant of the k-WL, named δ-k-LWL, that only considers a subset of the original neighborhood. However, they considered k-tuples and proved that a variant of their method is slightly more powerful than the original k-WL while taking the original graph's sparsity into account. That is, instead of Equation 4.10, the δ-k-LWL uses

$$M_i^{\delta}(\mathbf{v}) = \big(\{\!\!\{C_i^{k,\delta}(\phi_1(\mathbf{v}, w)) \mid w \in \delta(v_1)\}\!\!\}, \dots, \{\!\!\{C_i^{k,\delta}(\phi_k(\mathbf{v}, w)) \mid w \in \delta(v_k)\}\!\!\}\big).$$

Hence, the labeling function is defined by

$$C^{k,\delta}_{i+1}(\mathbf{v}) = \textsc{relabel}(C^{k,\delta}_{i}(\mathbf{v}), M^{\delta}_{i}(\mathbf{v})). \tag{4.16}$$

Empirically, they show that one of their variants of the k-WL achieves a new state of the art across many standard benchmark datasets while being several orders of magnitude faster than the k-WL.

4.2.4.5 Other Kernels Based on the Weisfeiler-Leman Algorithm

In the following, we survey other graph kernels that build on the Weisfeiler-Leman paradigm.

Weisfeiler-Leman Kernel Framework A general technique to modify and strengthen graph kernels is to modify their labels such that additional information is encoded. This can be achieved by computing the first $h \geq 0$ colors $C^1_0, \ldots, C^1_h$ of the 1-WL [627]. Then, given an arbitrary graph kernel used as base kernel, the corresponding Weisfeiler-Leman kernel is the sum of the base kernel applied to pairs of graphs with the label C^1_i for i in $\{0, \ldots, h\}$. The *Weisfeiler-Leman subtree kernel* described in Section 4.2.4.1 is obtained for a base kernel counting common vertex labels. Another instance of the approach commonly used is obtained by using the shortest-path kernel [65].

Hash Graph Kernel Framework In chem- or bioinformatics, edges and vertices of graphs are often annotated with real-valued information, e.g., physical measurements [508]. Previous graph kernels that can take these attributes into account are relatively slow and employ the kernel trick [65, 222, 394]. Therefore, these approaches do not scale to large graphs and datasets. Moreover, kernels such as the Weisfeiler-Leman subtree kernel cannot adequately deal with such continuous information due to its discrete nature. To overcome this, the hash graph kernel framework was introduced [507]. The idea is to iteratively turn the continuous attributes into discrete labels using randomized hash functions. This allows the application of fast, explicit graph feature maps, e.g., the Weisfeiler-Leman subtree kernel, which are limited to discrete annotations. In each iteration, the algorithm samples new hash functions and computes the feature map. Finally, the feature maps of all iterations are combined into one feature map. In order to obtain a meaningful similarity between attributes in $\mathbb{R}^d$, one requires that the probability of collision $\Pr[h_1(x) = h_2(y)]$ of two independently chosen random hash functions $h_1, h_2 \colon \mathbb{R}^d \to \mathbb{N}$ equals an adequate kernel on $\mathbb{R}^d$. Equipped with such a hash function, approximation results were derived for several state-of-the-art kernels that can handle continuous information [507]. In particular, we derived a variant of the Weisfeiler-Leman subtree kernel, which can handle continuous attributes. The extensive experimental study showed that instances of the hash kernel framework achieve state-of-the-art classification accuracies while being orders of magnitudes faster than kernels that were specifically designed to handle continuous information.

Neighborhood Aggregation in Graph Kernels The idea of neighborhood aggregation is widely used, and there are often subtle differences in definition. For completeness, we mention several graph kernels following this general idea. The *neighborhood hash kernel* [314] is similar in spirit to the Weisfeiler-Leman subtree kernel, but represents simple labels by bit-vectors and uses logical operations and hashing to encode the direct neighborhood for efficiency. Propagation kernels proposed in [528] provide a generic framework to define kernels on graphs based on an information propagation scheme for labels and attributes. Propagation, e.g., based on random walks, is performed individually on the two input graphs and a kernel is obtained by comparing label distributions after every propagation step. In the case of continuous (multi-dimensional) attributes, a single hash function is used to obtain a discrete label. In [537] a general message passing framework for kernels was proposed, where the concept of optimal assignments (see Section 4.2.4.2) was introduced in the neighborhood aggregation step. *Persistent Weisfeiler-Leman kernels* [597] combine 1-WL with persistent homology to extract topological features such as cycles. Recent theoretical results that link 1-WL to graph homomorphisms [167] were used to define graph kernels that have the same expressive power as the 1-WL, but a different feature space [533].

4.2.5 Graph Neural Networks and Their Connection to the Weisfeiler-Leman Algorithm

GNNs emerged as an alternative to graph kernels for graph classification and other machine learning tasks on graphs such as node classification or regression. Standard GNNs can be viewed as a neural version of the 1-WL, where colors are replaced by continuous feature vectors and neural networks are used to aggregate over node neighborhoods [252, 292, 375]. In effect, the GNN framework can be viewed as implementing a continuous form of graph-based “message passing”, where local neighborhood information is aggregated and passed on to the neighbors [252]. By deploying a trainable neural network to aggregate information in local node neighborhoods, GNNs can be trained in an end-to-end fashion together with the parameters of the classification or regression algorithm, possibly allowing for greater adaptability and better generalization compared with the kernel counterpart of the classical 1-WL.

A GNN model consists of a stack of neural network layers, where each layer aggregates local neighborhood information, i.e., features of neighbors, around each node and then passes this aggregated information on to the next layer. See Figure 4.8 for an illustration of the architecture.

In the following, we formally define GNNs and outline their connection to the Weisfeiler-Leman algorithm. Let $G = (V, E, l)$ be a labeled graph with an initial node coloring $f^{(0)}\colon V(G) \to \mathbb{R}^{1\times d}$ that is *consistent* with l. This means that each node v is annotated with a feature $f^{(0)}(v)$ in $\mathbb{R}^{1\times d}$ such that $f^{(0)}(u) = f^{(0)}(v)$ if and only if $l(u) = l(v)$. Alternatively, $f^{(0)}(v)$ can be an arbitrary real-valued feature vector associated with v. Examples include continuous atomic properties in chemoinformatic applications or

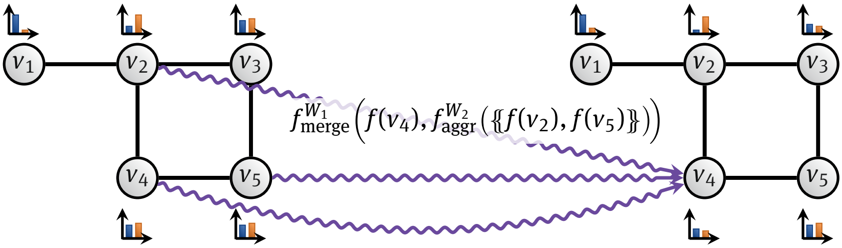

v_1
v_2
v_3
v_4
v_5
$f_{\text{merge}}^{W_1}\left(f(v_4), f_{\text{aggr}}^{W_2}(\{\!\{f(v_2), f(v_5)\}\!\})\right)$
v_1
v_2
v_3
v_4
v_5

Fig. 4.8: Illustration of the feature aggregation scheme of GNNs. The new feature of the node v_4 is computed from its old feature and the features of its neighbors v_2 and v_5.

vector representations of text in social network applications. A basic GNN model can be implemented as follows [292]. In each layer $t > 0$, we compute a new feature

$$f^{(t)}(v) = \sigma\left(f^{(t-1)}(v) \cdot W_1^{(t)} + \sum_{w \in N(v)} f^{(t-1)}(w) \cdot W_2^{(t)}\right) \tag{4.17}$$

in $\mathbb{R}^{1\times e}$ for v, where $W_1^{(t)}$ and $W_2^{(t)}$ are parameter matrices from $\mathbb{R}^{d\times e}$, and σ denotes a component-wise non-linear function, e.g., a sigmoid or a ReLU.[6]

One may also replace the sum defined over the neighborhood in the above equation by different permutation-invariant, differentiable functions, e.g., mean or max, and one may substitute the outer sum by, say, a column-wise vector concatenation [252]. Thus, in full generality a new feature $f^{(t)}(v)$ is computed as

$$f_{\text{merge}}^{W_1}\left(f^{(t-1)}(v), f_{\text{aggr}}^{W_2}(\{\!\{f^{(t-1)}(w) \mid w \in N(v)\}\!\})\right), \tag{4.18}$$

where $f_{\text{aggr}}^{W_2}$ aggregates over the set of neighborhood features and $f_{\text{merge}}^{W_1}$ merges the node's representations from step $(t-1)$ with the computed neighborhood features. Both $f_{\text{aggr}}^{W_2}$ and $f_{\text{merge}}^{W_1}$ may be arbitrary differentiable functions, e.g., neural networks, and, by analogy to Equation 4.17, we denote their parameters as W_1 and W_2, respectively.

A vector representation f_{GNN} over the whole graph can be computed by aggregating the vector representations computed for all nodes, e.g.,

$$f_{GNN}(G) = \sum_{v \in V(G)} f^{(T)}(v),$$

where $T > 0$ denotes the last layer. More refined approaches use differential pooling operators based on sorting [736] or soft assignments [724]. To adapt the parameters W_1 and W_2 of Equations 4.17 and 4.18 to a given data distribution, they are optimized in an end-to-end fashion (usually via stochastic gradient descent) together with the parameters of a neural network used for classification or regression. Efficient GPU-based implementations of many GNN architectures can be found in [225] and [696]. See also Section 4.3.

[6] For clarity of presentation we omit biases.

4.2.5.1 Connections to the Weisfeiler-Leman Algorithm

A recent line of work [468, 509, 714] connects the power or expressivity of GNNs to that of the Weisfeiler-Leman algorithm. The results show that GNN architectures generally do not have more power to distinguish between non-isomorphic (sub)graphs than the 1-WL.

Formally, let (G, l) be a labeled graph, and let $\mathbf{W}^{(t)} = \left(W_1^{(t')}, W_2^{(t')}\right)_{t' \leq t}$ denote the GNN parameters given by Equations 4.17 and 4.18 up to iteration t. We encode the initial labels $l(v)$ by vectors $f^{(0)}(v)$ in $\mathbb{R}^{1\times d}$ using a 1-hot encoding. The first theoretical result shown in [509] states that the GNN architectures do not have more power to distinguish between non-isomorphic (sub-)graphs than the 1-WL. More formally, let $f_{\text{merge}}^{W_1}$ and $f_{\text{aggr}}^{W_2}$ be any two functions chosen in Equation 4.18. For every encoding of the labels $l(v)$ as vectors $f^{(0)}(v)$, and for every choice of $\mathbf{W}^{(t)}$, the coloring C_i^1 of the 1-WL always refines the coloring $f^{(t)}$ induced by a GNN parameterized by $\mathbf{W}^{(t)}$.

Theorem 7. *Let (G, l) be a labeled graph. Then for all $t \geq 0$ and for all choices of initial colorings $f^{(0)}$ consistent with l, and weights $\mathbf{W}^{(t)}$,*

$$c_l^{(t)} \sqsubseteq f^{(t)} .$$

The second result of [509] states that there exists a sequence of parameter matrices $\mathbf{W}^{(t)}$ such that GNNs have the same power in terms of distinguishing non-isomorphic (sub-)graphs as the 1-WL. This even holds for the simple architecture Equation 4.17, provided we choose the encoding of the initial labeling l in such a way that linearly independent vectors encode different labels.

Theorem 8. *Let (G, l) be a labeled graph. Then for all $t \geq 0$ there exists a sequence of weights $\mathbf{W}^{(t)}$, and a 1-GNN architecture such that*

$$c_l^{(t)} \equiv f^{(t)} .$$

Hence, in the light of the above results, GNNs may be viewed as an extension of the 1-WL, which in principle have the same power but are more flexible in their ability to adapt to the learning task at hand and can handle continuous node features.

4.2.5.2 Higher-order Graph Neural Networks

The above results also have been lifted to the k-dimensional case. For example, Maron, Ben-Hamu, Serviansky, and Lipman [468] devised an architecture based on simple matrix operations that has the same power as the 3-WL. In a recent work, Morris, Rattan, and Mutzel [504] devised neural architectures, denoted δ-k-LGNN, that resemble the construction for GNNs.

Formally, given a labeled graph G, let each tuple $\mathbf{v}$ in $V(G)^k$ be annotated with an initial feature $f^{(0)}(\mathbf{v})$ determined by the isomorphism type of $G[\mathbf{v}]$. In each layer $t > 0$, we compute a new feature $f^{(t)}(\mathbf{v})$ as

$$f^{W_1}_{\text{merge}}\Big(f^{(t-1)}(\mathbf{v}), f^{W_2}_{\text{agg}}\big(\{\!\{f^{(t-1)}(\phi_1(\mathbf{v}, w)) \mid w \in \delta(v_1)\}\!\}, \dots,$$
$$\{\!\{f^{(t-1)}(\phi_k(\mathbf{v}, w)) \mid w \in \delta(v_k)\}\!\}\big)\Big),$$

in $\mathbb{R}^{1\times e}$ for a tuple $\mathbf{v}$, where $W_1^{(t)}$ and $W_2^{(t)}$ are learnable parameter matrices from $\mathbb{R}^{d\times e}$.[7] Moreover, $f^{W_2}_{\text{merge}}$ and the permutation-invariant $f^{W_1}_{\text{agg}}$ can be arbitrary differentiable functions, responsible for merging and aggregating the relevant feature information, respectively. Note that one can naturally handle discrete node and edge labels as well as directed graphs. The following result shown in [504] demonstrates the expressive power of the δ-k-LGNN in terms of distinguishing non-isomorphic graphs.

Theorem 9. *Let (G, l) be a labeled graph. Then for all $t \geq 0$ there exists a sequence of weights $\mathbf{W}^{(t)}$ such that*

$$C^{k,\delta}_t(\mathbf{v}) = C^{k,\delta}_t(\mathbf{w}) \iff f^{(t)}(\mathbf{v}) = f^{(t)}(\mathbf{w}).$$

Hence, for all graphs, the following holds for all $k \geq 1$:

$$\delta\text{-}k\text{-LGNN} \equiv \delta\text{-}k\text{-LWL}.$$

4.2.6 Conclusion and Future Work

The Weisfeiler-Leman method has been studied for decades in graph theory and recently turned out to be an essential technique in machine learning with graphs [505], achieving high accuracy on many real-world datasets [508]. While the Weisfeiler-Leman algorithm's expressivity limits machine learning methods to distinguishing non-isomorphic graphs, the generalization abilities of such methods are understood to a lesser extent, indicating an avenue for future research. Moreover, heterogeneous networks with different edge types or graphs annotated with temporal information will become increasingly important. The adaption of the Weisfeiler-Leman paradigm to such settings has only recently been considered, e.g., for temporal graphs [544], and the development of new suitable learning methods has only just begun.

[7] For clarity of presentation we omit biases.

4.3 Deep Graph Representation Learning

Matthias Fey
Frank Weichert

Abstract: Learning with graph-structured data such as molecules, social, biological, and financial networks, requires effective representations that successfully capture their rich structural properties. In recent years, numerous approaches have been proposed for machine learning on graphs — most notably, approaches based on graph kernels and *Graph Neural Networks (GNNs)*. Graph neural networks exploit relational inductive biases of the underlying data by following a differentiable neural message passing scheme, and show-case promising performance on a variety of different tasks due to their expressive power in capturing different graph structures. However, despite the indisputable potential of GNNs in learning such representations, one of the challenges that have so far precluded their wide adoption in industrial and social applications is the difficulty to scale them to large graphs. In particular, the embedding of a given node depends recursively on all its neighbor's embeddings, leading to high inter-dependency between nodes that grows exponentially with respect to the number of layers.

Here, we demonstrate the generality of message passing through a unified framework that is suitable for a wide range of operators and learning tasks. This generality of message passing led to the development of *PyTorch Geometric*, a well-known deep learning library for implementing and working with graph-based neural network building blocks. Furthermore, we discuss scalable approaches for applying graph neural networks to large-scale graphs. In particular, we show that scalable approaches based on sub-sampling of edges or non-trainable propagations weaken the expressive power of message passing. In order to overcome this restriction, we present *GNN AutoScale*, a framework for scaling arbitrary message passing neural networks to large graphs. GNN AutoScale prunes entire sub-trees of the computation graph by utilizing historical node embeddings from prior training iterations while provably being able to maintain the expressive power of the original architecture.

4.3.1 Introduction

Graphs are widely used for abstracting complex systems of interacting objects, such as social networks, knowledge graphs, molecular graphs, and biological networks, as well as for modeling 3D objects, manifolds, and source code [320]. To develop successful machine learning models in these domains, we need techniques that can exploit the rich information inherent in the graph structure, as well as the feature information contained within a graph's nodes and edges. Recently, graph neural networks emerged as a powerful approach and the de facto standard for representation learning on graphs. GNNs are able to capture local graph structure and feature information in a trainable fashion to derive powerful node representations suitable for a given task at hand [291, 455]. To achieve this, they follow a simple neighborhood aggregation procedure or neural message passing scheme motivated by two major perspectives: The generalization of classical CNNs to irregular domains, and their strong relations to the Weisfeiler-Lehman algorithm [226, 509, 715] (see Section 4.2.5).

The recent work in the fields of *geometric deep learning* and *relational representation learning* provides a large number of graph-based operators, which allows for precise control of the properties of extracted graph-based features [134, 225, 252, 292, 375, 378, 588, 683, 697, 714, 715]. Nonetheless, all those recent operators can be described by a simple message passing formulation, leading to a unified framework suitable across a wide range of operators and learning tasks [252]. The generality of message passing led to the development of the *PyTorch Geometric* library, a deep learning framework for implementing and working with graph-based neural networks [225].

While GNNs have become better understood and models have become more sophisticated, advancements in this field should be more noticeable with access to increasing data. However, applying mini-batch training of GNNs is challenging since the embedding of a given node depends recursively on all its neighbor's embeddings, leading to high inter-dependency between nodes that grows exponentially with respect to the number of layers [455]. Several recent works address this problem via different sampling techniques (leading to sub-sampling of edges) [455, 600], or by decoupling propagations from predictions [234, 321, 378, 710, 726]. Although empirical results suggest that the aforementioned methods can scale GNN training to large graphs, these techniques are either restricted to shallow networks, non-exchangeable operators, or reduced expressivity. In particular, existing approaches consider only specific GNN operators and it is not yet well known whether these techniques can be successfully applied to the wide range of GNN architectures available.

In the next sections, we will discuss and introduce the aforementioned general neural message passing framework, and show how common GNN operators fit into this scheme. We proceed by introducing the PyTorch Geometric library [225], which makes it easy to implement those GNN operators in practice. Furthermore, we present our *GNN AutoScale* framework for scaling arbitrary message passing GNNs to large-scale graphs [224].

4.3.2 Representation Learning on Graphs via Neural Message Passing

We begin by refining the general neural message passing scheme from Section 4.2.5 that is utilized in state-of-the-art graph neural networks and, along the way, introduce the necessary notation and background. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ or $\boldsymbol{A} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ denote a *graph* with node feature vectors $\boldsymbol{x}_v$ for all $v \in \mathcal{V}$ and (optional) edge features $\boldsymbol{e}_{v,w}$ in case

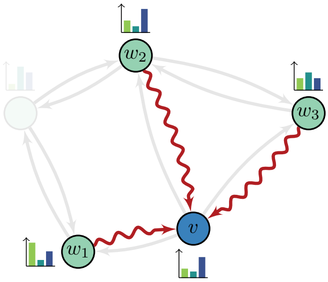


Fig. 4.9: Message passing flow in a GNN layer. Each direct neighbor of a node crafts a message that is sent along the given edge. Each node aggregates their incoming messages to update its current node representation.

$(v, w) \in \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Here, we are mostly interested in learning final node representations $\boldsymbol{h}_v \in \mathbb{R}^D$ for all $v \in \mathcal{V}$ in an end-to-end fashion that are suitable for a given downstream task (such as node, link, or graph classification). In node classification, each node $v \in \mathcal{V}$ is associated with a label y_v, and the goal is to learn a representation $\boldsymbol{h}_v$ from which y_v can be easily predicted. In link prediction, we want to find the missing links in an incomplete graph, and we can directly use $\boldsymbol{h}_v$ and $\boldsymbol{h}_w$, $v, w, \in \mathcal{V}$, for predicting the existence of an edge between the given node pair. In graph classification, each individual graph is associated with a label y, and we can use $\{\!\{\boldsymbol{h}_v : v \in \mathcal{V}\}\!\}$ alltogether to predict the label y in a permutation-invariant fashion.

Graph neural networks operate on graph-structured data $\mathcal{G}$ by following a *neural message passing scheme*, where a representation of a node is iteratively updated by aggregating representations of its neighbors [252]. After L iterations of aggregation, the representation of a node captures both structural *and* feature information within its L-hop neighborhood. Formally, the $(\ell + 1)$-th layer of a GNN is defined as

$$\boldsymbol{h}_v^{(\ell+1)} = \boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell+1)}\left(\boldsymbol{h}_v^{(\ell)}, \left\{\!\!\left\{\left(\boldsymbol{h}_w^{(\ell)}, \boldsymbol{h}_v^{(\ell)}, \boldsymbol{e}_{w,v}^{(\ell)}\right) : w \in \mathcal{N}(v)\right\}\!\!\right\}\right) \tag{4.19}$$

$$= \textsc{Update}_{\boldsymbol{\theta}}^{(\ell+1)}\left(\boldsymbol{h}_v^{(\ell)}, \bigoplus_{w \in \mathcal{N}(v)} \textsc{Message}_{\boldsymbol{\theta}}^{(\ell+1)}\left(\boldsymbol{h}_w^{(\ell)}, \boldsymbol{h}_v^{(\ell)}, \boldsymbol{e}_{w,v}^{(\ell)}\right)\right) \tag{4.20}$$

where $\boldsymbol{h}_v^{(\ell)}$ represents the feature vector of node v obtained in layer ℓ and $\mathcal{N}(v) = \{w : (w, v) \in \mathcal{E}\}$ defines the neighborhood set of v. We initialize $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$. Since different nodes can have identical feature vectors, a GNN operates on *multisets* $\{\!\{\ldots\}\!\}$, defined as a 2-tuple $\mathcal{X} = (\mathbb{P}^d, c)$, where $\mathbb{P}^d$ denotes the underlying set of $\mathcal{X}$ and $c\colon \mathbb{P}^d \to \mathbb{N}_{\geq 1}$ counts its multiplicity. A general illustration of this message passing flow is given in Figure 4.9. Most recent GNN operators $\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}$ can be decomposed into differentiable and parametrized $\textsc{Message}_{\boldsymbol{\theta}}^{(\ell)}$ and $\textsc{Update}_{\boldsymbol{\theta}}^{(\ell)}$ functions parametrized by weights $\boldsymbol{\theta}$, as well as permutation-invariant aggregation functions $\bigoplus$, e.g., taking the sum, mean or maximum of features [225]. Message and Update can be chosen in different ways, depending on the task at hand. For example, Message functions can transform incoming features either linearly or non-linearly [252, 588, 697]; aggregative functions can model static [714], structure-dependent [375], or data-dependent aggregations [683]; and UPDATE is typically used to preserve central node information via skip-connections [292] or residuals [134, 378].

Ideally, a maximally powerful GNN could distinguish non-isomorphic graph structures by mapping them to different representations in the embedding space. In recent studies [509, 714], it has been shown that the representational power of GNNs is bounded by the capacity of the *Weisfeiler-Leman (WL)* graph isomorphism test [701], (see Section 4.2.3), which uniquely refines the coloring of a node $c_v^{(\ell)} : \mathcal{V} \to \Sigma$ based on the colors of their neighbors. In fact, a GNN's expressiveness is equivalent to the WL test if all its layers $\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}$ are injective, i.e., if they *never* map two different neighborhoods to the same representation. As a result, numerous GNN operators have been proposed that are equally powerful as the WL test [155, 714], as well as higher-order variants to increase their representational power even further [67, 227, 468, 504, 509, 519] (see Section 4.2). We now briefly review how current state-of-the-art GNN operators fit into the given neural message passing scheme (omitting final non-linearities due to simplicity).

Graph Neural Networks (GNN) [375] can be considered as one of the pioneers of graph-structured deep learning methods, and they are motivated by a first-order approximation of spectral graph convolutions. Its underlying GNN operator uses a symmetrically normalized mean aggregation of linearly transformed neighboring node representations

$$\boldsymbol{h}_v^{(\ell+1)} = \overbrace{\frac{1}{c_{v,v}} \boldsymbol{W} \boldsymbol{h}_v^{(\ell)} + \underbrace{\sum_{w \in \mathcal{N}(v)}}_{\bigoplus} \underbrace{\frac{1}{c_{w,v}} \boldsymbol{W} \boldsymbol{h}_w^{(\ell)}}_{\textsc{Message}_{\boldsymbol{\theta}}^{(\ell+1)}}}^{\textsc{Update}_{\boldsymbol{\theta}}^{(\ell+1)}}, \tag{4.21}$$

where $c_{w,v} = \sqrt{\deg(w)+1}\sqrt{\deg(v)+1}$ with $\deg(\cdot)$ denoting node degree, and $\boldsymbol{W}$ being a trainable weight matrix.

Graph Attention Networks (GAT) [683] builds upon the idea of GCNs where the structure-dependent normalization coefficients are replaced by an anisotropic, learnable aggregation guided by attention

$$\boldsymbol{h}_v^{(\ell+1)} = \overbrace{\alpha_{v,v} \boldsymbol{W} \boldsymbol{h}_v^{(\ell)} + \underbrace{\sum_{w \in \mathcal{N}(v)}}_{\bigoplus} \underbrace{\alpha_{w,v} \boldsymbol{W} \boldsymbol{h}_w^{(\ell)}}_{\textsc{Message}_{\boldsymbol{\theta}}^{(\ell+1)}}}^{\textsc{Update}_{\boldsymbol{\theta}}^{(\ell+1)}}, \tag{4.22}$$

where attention coefficients are computed via

$$\alpha_{w,v} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\boldsymbol{a}^\top \left[\boldsymbol{W}\boldsymbol{h}_v^{(\ell)}, \boldsymbol{W}\boldsymbol{h}_w^{(\ell)}\right]\right)\right)}{\sum_{k \in \mathcal{N}(v) \cup \{v\}} \exp\left(\mathrm{LeakyReLU}\left(\boldsymbol{a}^\top \left[\boldsymbol{W}\boldsymbol{h}_v^{(\ell)}, \boldsymbol{W}\boldsymbol{h}_k^{(\ell)}\right]\right)\right)}. \tag{4.23}$$

with additional trainable parameters $\boldsymbol{a}$.

Spline-Based Convolutional Neural Networks [226] utilize edge information $\boldsymbol{e}_{w,v}$ to learn a data-dependent filter matrix

$$\boldsymbol{h}_v^{(\ell+1)} = \overbrace{\boldsymbol{W}\boldsymbol{h}_v^{(\ell)} + \underbrace{\sum_{w \in \mathcal{N}(v)}}_{\bigoplus} \underbrace{\boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{e}_{w,v})\boldsymbol{h}_w^{(\ell)}}_{\textsc{Message}_{\boldsymbol{\theta}}^{(\ell+1)}}}^{\textsc{Update}_{\boldsymbol{\theta}}^{(\ell+1)}} \tag{4.24}$$

via a parametrized and continuous B-Spline kernel function $\boldsymbol{g}_{\boldsymbol{\theta}}(\cdot)$.

Graph Isomorphism Networks (GIN) [714] make use of sum aggregation and MLPs to obtain a maximally powerful GNN operator

$$\boldsymbol{h}_v^{(\ell+1)} = \overbrace{\mathrm{MLP}_{\boldsymbol{\theta}}\left((1+\epsilon)\,\boldsymbol{h}_v^{(\ell)} + \underbrace{\sum_{w \in \mathcal{N}(v)}}_{\bigoplus} \underbrace{\boldsymbol{h}_w^{(\ell)}}_{\textsc{Message}_{\boldsymbol{\theta}}^{(\ell+1)}}\right)}^{\textsc{Update}_{\boldsymbol{\theta}}^{(\ell+1)}}, \tag{4.25}$$

where ϵ is a trainable scalar in order to distinguish neighbors from central nodes.

Principal Neighborhood Aggregation (PNA) [155] networks leverage mulitple aggregators combined with degree-scalers to better capture graph structural properties

$$\boldsymbol{h}_v^{(\ell+1)} = \overbrace{\boldsymbol{W}_2\left[\boldsymbol{h}_v^{(\ell)}, \bigoplus_{w \in \mathcal{N}(v)} \underbrace{\boldsymbol{W}_1\left[\boldsymbol{h}_v^{(\ell)}, \boldsymbol{h}_w^{(\ell)}\right]}_{\textsc{Message}_{\boldsymbol{\theta}}^{(\ell+1)}}\right]}^{\textsc{Update}_{\boldsymbol{\theta}}^{(\ell+1)}}, \tag{4.26}$$

where $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ denote trainable weight matrices, and

$$\bigoplus = \underbrace{\begin{bmatrix} 1 \\ s(\deg(v), 1) \\ s(\deg(v), -1) \end{bmatrix}}_{\text{Scalers}} \otimes \underbrace{\begin{bmatrix} \text{mean} \\ \text{min} \\ \text{max} \end{bmatrix}}_{\text{Aggregators}}, \tag{4.27}$$

with $\otimes$ being the tensor product and

$$s(d, \alpha) = \left(\frac{\log(d+1)}{\frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \log(\deg(v) + 1)} \right)^{\alpha} \tag{4.28}$$

denoting degree-based scalers. Having introduced the basic concepts of message passing within GNNs, we now look more closely at their practical implementation (Section 4.3.3) and resource efficiency (Section 4.3.4).

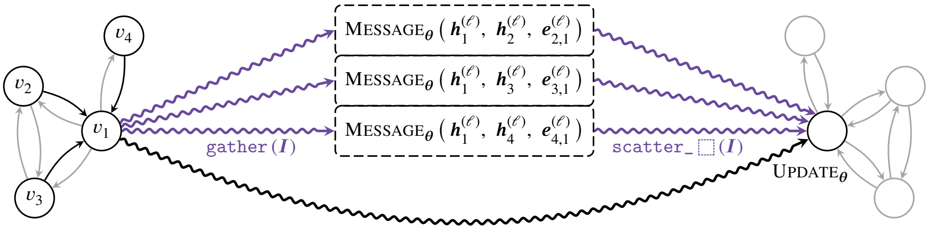

v_4
v_2
v_1
v_3
$\textsc{Message}_\theta\left(\boldsymbol{h}_1^{(\ell)},\ \boldsymbol{h}_2^{(\ell)},\ \boldsymbol{e}_{2,1}^{(\ell)}\right)$
$\textsc{Message}_\theta\left(\boldsymbol{h}_1^{(\ell)},\ \boldsymbol{h}_3^{(\ell)},\ \boldsymbol{e}_{3,1}^{(\ell)}\right)$
$\textsc{Message}_\theta\left(\boldsymbol{h}_1^{(\ell)},\ \boldsymbol{h}_4^{(\ell)},\ \boldsymbol{e}_{4,1}^{(\ell)}\right)$
gather($\boldsymbol{I}$)
scatter_□($\boldsymbol{I}$)
$\textsc{Update}_\theta$

Fig. 4.10: Computation scheme of a GNN layer by leveraging gather and scatter methods based on edge indices $\boldsymbol{I}$, hence alternating between node parallel space and edge parallel space.

4.3.3 PyTorch Geometric: Implementing Graph Neural Networks

The practical implementation of graph neural networks is challenging, as high GPU throughput needs to be achieved on highly sparse and irregular data of varying size. Here, we introduce and discuss the *PyTorch Geometric* library [225], a library for deep learning on irregularly structured data, built upon PyTorch [555]. In addition to general graph data structures and processing methods, it contains a variety of recently published methods from the domains of relational learning and 3D data processing. PyTorch Geometric achieves high data throughput by leveraging sparse GPU acceleration, by providing dedicated CUDA kernels, and by introducing efficient mini-batch handling for input examples of different sizes. All implemented methods support both CPU and GPU computations and follow an immutable data flow paradigm that enables dynamic changes in graph structures through time. PyTorch Geometric is released under the MIT license and is available on GitHub.[8] It is thoroughly documented and provides accompanying tutorials and examples as a first starting point.[9]

In PyTorch Geometric, we represent a graph $\mathcal{G} = (\boldsymbol{X}, (\boldsymbol{I}, \boldsymbol{E}))$ by a node feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ of N nodes holding F features, and a sparse adjacency tuple $(\boldsymbol{I}, \boldsymbol{E})$ of E edges, where $\boldsymbol{I} \in \mathbb{N}^{2 \times E}$ encodes edge indices in COOrdinate (COO) format and $\boldsymbol{E} \in \mathbb{R}^{E \times D}$ (optionally) holds D-dimensional edge features. All user-facing APIs, e.g., data-loading routines, multi-GPU support, data augmentation, and model instantiations are heavily inspired by PyTorch to keep them as familiar as possible.

In practice, the realization of Equation 4.20 can be achieved by gathering and scattering node features and a vectorized elementwise computation of Message and Update functions, as visualized in Figure 4.10. Although working on irregularly structured input, this scheme can be heavily accelerated by the GPU. In contrast to implementations via sparse matrix multiplications, the usage of gather and scatter proves to be advantageous for low-degree graphs and non-coalesced input, and allows for the integration of central node and multi-dimensional edge information directly while aggregating.

[8] GitHub repository: https://github.com/rusty1s/pytorch_geometric.

[9] Documentation: https://pytorch-geometric.readthedocs.io.

We implement different reductions for the scattering of neighboring node features via dedicated CUDA kernels, although execution on other hardware is applicable as well. For more, see Chapter 6.

We provide the user with a general `MessagePassing` interface to allow for rapid and clean prototyping of new research ideas. In order to use the interface, users only need to define the methods $\textsc{Message}_{\boldsymbol{\theta}}$, i.e., `message`, and $\textsc{Update}_{\boldsymbol{\theta}}$, i.e., `update`, and choose an aggregation scheme $\bigoplus$. For implementing `message`, node features are automatically mapped to the respective source and target nodes. Almost all recently proposed neighborhood aggregation functions can be lifted to this interface, including (but not limited to) the methods already integrated in PyTorch Geometric. Overall, PyTorch Geometric currently bundles over 40 different GNN operators proposed in literature, as well as over 15 complete models.

(Hierarchical) Pooling PyTorch Geometric also supports graph-level outputs as opposed to node-level outputs by providing a variety of graph-level pooling functions [435, 687, 736]. To further extract hierarchical information and to allow deeper GNN models, various pooling approaches can be applied in a deterministic or data-dependent manner [118, 175, 205, 241, 588, 633, 697, 724].

Mini-Batch Handling Our framework supports batches of multiple graph instances (of potentially different size) by automatically creating a single (sparse) block-diagonal adjacency matrix and concatenating feature matrices in the node dimension. Therefore, neighborhood aggregation methods can be applied without any modifications, since no messages are exchanged between disconnected graphs. In addition, an automatically generated assignment vector ensures that node-level information is not aggregated across graphs as when executing global aggregation operators.

Processing of Datasets We provide a consistent data format and an easy-to-use interface for the creation and processing of datasets, both for large datasets and for datasets that can be kept in memory during training. In order to create new datasets, users just need to read/download their data and convert it to the PyTorch Geometric data format via the respective `process` method. In addition, datasets can be modified by the use of `transforms`, which take in separate graphs and transform them, say, for data augmentation, for enhancing node features with synthetic structural graph properties, in order to automatically generate graphs from point clouds or to sample point clouds from meshes.

Empirical Evaluation We evaluated the correctness of the implemented methods by performing a comprehensive comparative study in homogeneous evaluation scenarios, reaching state-of-the-art performance on several graph benchmark tasks. For example, experiments for the semi-supervised node classification performance of common GNN

Tab. 4.1: Performance (accuracy and standard deviation) of semi-supervised node classification experiments for fixed and random splits across 100 runs.

Method	Cora		CiteSeer		PubMed	
	Fixed	Random	Fixed	Random	Fixed	Random
Cheby [164]	81.4±0.7	77.8±2.2	70.2±1.0	67.7±1.7	78.4±0.4	75.8±2.2
GCN [375]	81.5±0.6	79.4±1.9	71.1±0.7	68.1±1.7	79.0±0.6	77.4±2.4
GAT [683]	83.1±0.4	81.0±1.4	70.8±0.5	69.2±1.9	79.0±0.3	77.5±2.3
SGC [710]	81.7±0.1	80.2±1.6	71.3±0.2	68.7±1.6	78.9±0.1	76.5±2.4
ARMA [52]	82.8±0.6	80.7±1.4	**72.3**±1.1	68.9±1.6	78.8±0.3	77.7±2.6
APPNP [378]	**83.3**±0.5	**82.2**±1.5	71.8±0.5	**70.0**±1.4	**80.1**±0.2	**79.4**±2.2

architectures are easily finished within 1–2 seconds per run, either using fixed or random training splits. Table 4.1 presents the results of state-of-the-art GNNs on several citation datasets [621, 718]. Notably, all experiments show a high reproducibility of the reported results.

4.3.4 Scalable and Expressive Graph Neural Networks

While the full-gradient in GNNs is straightforward to compute since we have access to *all* hidden node representations of *all* layers, this is not feasible in large-scale graphs due to memory limitations and slow convergence [455]. Therefore, given a loss function ϕ, it is desirable to approximate its full-batch gradient stochastically

$$\nabla \mathcal{L} = \frac{1}{|\mathcal{V}|}\sum_{v\in\mathcal{V}} \nabla\phi(\boldsymbol{h}_v^{(L)}, y_v) \approx \frac{1}{|\mathcal{B}|}\sum_{v\in\mathcal{B}\subseteq\mathcal{V}} \nabla\phi(\boldsymbol{h}_v^{(L)}, y_v), \tag{4.29}$$

which considers only a mini-batch $\mathcal{B} \subseteq \mathcal{V}$ of nodes for loss computation. However, this stochastic gradient is still expensive to compute due to the exponentially increasing dependencies of node representations over layers, a phenomenon known as *neighbor explosion* [292]. Specifically, the representation of a given node depends recursively on all its neighbor's representations, and the number of dependencies grows exponentially with respect to the number of layers.

Recent works try to alleviate this problem by proposing various different sampling techniques [455], which can be broadly categorized as node-wise, layer-wise, and subgraph sampling strategies. In general, these techniques can all be viewed as different variants of dropping edges [600]. *Node-wise sampling* [126, 292] recursively samples a fixed number k of 1-hop neighbors, leading to an overall bounded L-hop neighborhood of $\mathcal{O}(k^L)$ for each node. In contrast to tracking down inter-layer connections, *layer-wise sampling* techniques independently sample nodes for each layer, leading to a constant sampled size in each layer [126, 323, 747]. Here, variance is further reduced via importance sampling or adaptive sampling techniques. In *subgraph sampling* [137, 731, 732], a full GNN is run on an entire subgraph $\mathcal{G}[\mathcal{B}]$ induced by a sampled batch of nodes $\mathcal{B} \subseteq \mathcal{V}$. Notably, most of these sampling approaches eliminate the neighbor explosion problem, but there are challenges to preserving the edges that present a meaningful topological structure.

Another line of work is based on the idea of decoupling propagations from predictions [234, 378, 710, 726]. Here, input node features are first enhanced by performing several rounds of propagation via, say, the normalized Laplacian matrix or the personalized matrix, before they are inputted into an Multilayer Perceptron (MLP) to perform the final prediction. While this scheme enjoys fast training and inference time, it cannot be applied to any GNN, especially because the propagation is non-trainable. Recently, Huang, He, Singh, Lim, and Benson [321] proposed a simple post-processing step to correct and smooth the predictions of a simple graph-agnostic model via label propagation. While this step is orthogonal to recent GNN advancements, it can only be applied in transductive learning scenarios.

It is well known that the most powerful GNNs adhere to the same representational power as the WL test [701] in distinguishing non-isomorphic structures [509, 714], i.e., $\boldsymbol{h}_v^{(L)} \neq \boldsymbol{h}_w^{(L)}$ in case $c_v^{(L)} \neq c_w^{(L)}$, where $c_v^{(L)}$ denotes a node's coloring after L rounds of color refinement. However, in order to leverage such expressiveness, a GNN needs to be able to reason about structural differences across neighborhoods directly *during* training. It has been shown that GNNs that scale by sub-sampling edges are not capable of doing so [224]:

Proposition 10. *Let $\boldsymbol{f}_{\boldsymbol{\theta}}^{(L)} : \mathcal{V} \to \mathbb{R}^d$ be a L-layered GNN as expressive as the WL test in distinguishing the L-hop neighborhood around each node $v \in \mathcal{V}$. Then there exists a graph $\boldsymbol{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ for which $\boldsymbol{f}_{\boldsymbol{\theta}}^{(L)}$ operating on a sampled variant $\tilde{\boldsymbol{A}}$, $\tilde{a}_{v,w} = \begin{cases} \frac{|\mathcal{N}(v)|}{|\tilde{\mathcal{N}}(v)|}, & \textit{if } w \in \tilde{\mathcal{N}}(v) \\ 0, & \textit{otherwise} \end{cases}$, produces a non-equivalent coloring, i.e., $\tilde{\boldsymbol{h}}_v^{(L)} \neq \tilde{\boldsymbol{h}}_w^{(L)}$ while $c_v^{(L)} = c_w^{(L)}$ for nodes $v, w \in \mathcal{V}$.*

Therefore, a special interest lies in the question if there exist scalable GNN variants that are as expressive as their full-batch counterpart.

4.3.4.1 Scaling Graph Neural Networks via Historical Embeddings

We now introduce the *GNNAutoScale (GAS)* framework [224], which scales graph neural networks by pruning entire sub-trees of the computation graph and filling the missing information by utilizing historical embeddings acquired in previous training iterations [126, 153], leading to constant GPU memory consumption with respect to input node size without dropping any data. Since GNNAutoScale accounts for all data, it provably is able to maintain the expressive power of the underlying graph neural network.

Let $\boldsymbol{h}_v^{(\ell)}$ denote the node embedding in layer ℓ of a node $v \in \mathcal{B}$ in a mini-batch $\mathcal{B} \subseteq \mathcal{V}$. For the general message scheme given in Equation 4.20, the execution of $\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell+1)}$

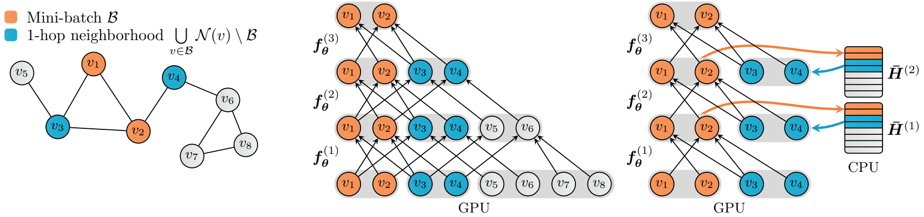


Fig. 4.11: Mini-batch processing of GNNs with historical embeddings. 🟧 denotes the nodes in the current mini-batch and 🟦 represents their direct 1-hop neighbors. For a given mini-batch (left), GPU memory and computation costs increase exponentially with GNN depth (middle). The usage of historical embeddings avoids this problem as it *prunes* entire sub-trees of the computation graph, which leads to constant GPU memory consumption with respect to input node size (right). Here, nodes in the current mini-batch *push* their updated embeddings to the history $\bar{\boldsymbol{H}}^{(\ell)}$, while their direct neighbors *pull* their most recent historical embeddings from $\bar{\boldsymbol{H}}^{(\ell)}$ for further processing.

can be formulated as:

$$
\begin{aligned}
\boldsymbol{h}_v^{(\ell+1)} &= \boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell+1)}\left(\boldsymbol{h}_v^{(\ell)}, \left\{\!\!\left\{ \boldsymbol{h}_w^{(\ell)} : w \in \mathcal{N}(v) \right\}\!\!\right\}\right) \\
&= \boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell+1)}\left(\boldsymbol{h}_v^{(\ell)}, \left\{\!\!\left\{ \boldsymbol{h}_w^{(\ell)} : w \in \mathcal{N}(v) \cap \mathcal{B} \right\}\!\!\right\} \cup \left\{\!\!\left\{ \boldsymbol{h}_w^{(\ell)} : w \in \mathcal{N}(v) \setminus \mathcal{B} \right\}\!\!\right\}\right) \\
&\approx \boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell+1)}\left(\boldsymbol{h}_v^{(\ell)}, \underbrace{\left\{\!\!\left\{ \boldsymbol{h}_w^{(\ell)} : w \in \mathcal{N}(v) \cap \mathcal{B} \right\}\!\!\right\}}_{\textbf{(1)}\text{ Local embeddings}} \cup \underbrace{\left\{\!\!\left\{ \bar{\boldsymbol{h}}_w^{(\ell)} : w \in \mathcal{N}(v) \setminus \mathcal{B} \right\}\!\!\right\}}_{\textbf{(2)}\text{ Historical embeddings}}\right)
\end{aligned}
$$

Here, we separate the neighborhood information of the multiset into *two* parts: **(1)** the local information of neighbors $\mathcal{N}(v)$ that are part of the current mini-batch $\mathcal{B}$, and **(2)** the information of neighbors that are not included in the current mini-batch. We then approximate the embeddings of out-of-mini-batch nodes with their historical embeddings denoted as $\bar{\boldsymbol{h}}_w^{(\ell)}$. After each step of training, the newly computed embeddings $\boldsymbol{h}_v^{(\ell+1)}$ are pushed to the history and serve as historical embeddings $\bar{\boldsymbol{h}}_w^{(\ell+1)}$ in future iterations.

A high-level illustration of its computation flow is visualized in Figure 4.11. It can be seen that in the original data flow without historical embeddings the required GPU memory increases as the model gets deeper. After a few layers, embeddings for the entire input graph need to be stored, even if only a mini-batch of nodes is considered for loss computation. By contrast, historical embeddings eliminate this problem by approximating entire sub-trees of the computation graph. The required historical embeddings are pulled from an offline storage, instead of being re-computed in each iteration, which keeps the required information for each batch local. For a single batch $\mathcal{B} \subseteq \mathcal{V}$, the GPU memory footprint for one training step is given by $\mathcal{O}(|\bigcup_{v \in \mathcal{B}} \mathcal{N}(v) \cup \{v\}| \cdot L)$ and thus only scales linearly with the number of layers L. The vast majority of data (the histories) can be stored in RAM or hard drive storage rather than GPU memory.

In contrast to existing scaling solutions based on the sub-sampling edges, GAS provides the following advantages:

1. **GAS trains over all the data:** In GAS, a GNN will make use of all available graph information, i.e., *no* edges are dropped, which results in lower variance and more accurate estimations. Nonetheless, for a single epoch and layer, each edge will be only processed once, putting its time complexity $\mathcal{O}(|\mathcal{E}|)$ on par with its full-batch counterpart. Notably, more accurate estimations will further strengthen gradient estimation during backpropagation. Specifically, the model parameters will be updated based on the node embeddings of *all* neighbors since $\partial \boldsymbol{h}_v^{(\ell+1)} / \partial \boldsymbol{\theta}$ also depends on $\{\!\{\bar{\boldsymbol{h}}_w^{(\ell)} : w \in \mathcal{N}(v) \setminus \mathcal{B}\}\!\}$.
2. **GAS enables constant inference time complexity:** The time complexity of model inference is reduced to a constant factor, since we can directly use the historical embeddings of the last layer to derive predictions for test nodes.
3. **GAS is simple to implement:** Our scheme does not need to maintain recursive layer-wise computation graphs, which makes its overall implementation straightforward and comparable to full-batch training. Only minor modifications are required to *pull* information from and *push* information to the histories.
4. **GAS provides theoretical guarantees:** In particular, if the model weights are kept fixed, $\bar{\boldsymbol{h}}_v^{(\ell)}$ eventually equals $\boldsymbol{h}_v^{(\ell)}$ after a fixed amount of iterations [126].

While sampling strategies loose expressive power due to the sub-sampling of edges, scalable GNNs based on historical embeddings leverage *all* edges during neighborhood aggregation. Therefore, a special interest lies in the question if history-based GNNs are as expressive as their full-batch counterpart. Here, a maximally powerful *and* scalable GNN needs to fulfill the following two requirements: **(1)** it needs to be as expressive as the WL test in distinguishing non-isomorphic structures, and **(2)** it needs to account for the approximation error $\|\bar{\boldsymbol{h}}_v^{(\ell-1)} - \boldsymbol{h}_v^{(\ell-1)}\|$ induced by the usage of historical embeddings. Since it is known that there exists a wide range of maximally powerful GNNs [155, 509, 714], we can restrict our analysis to the latter question.

Theorem 11. *Let $\boldsymbol{f}_{\boldsymbol{\theta}}^{(L)}$ be an L-layered GNN. If the historical embeddings do not run too stale, i.e., $\|\bar{\boldsymbol{h}}_v^{(\ell-1)} - \boldsymbol{h}_v^{(\ell-1)}\| \leq \epsilon$, then there exist $\textsc{Message}_{\boldsymbol{\theta}}^{(\ell)}$ and $\textsc{Update}_{\boldsymbol{\theta}}^{(\ell)}$ functions, $\ell \in \{1, \ldots, L\}$, such that there exists a map $\phi \colon \mathbb{R}^D \to \Sigma$ so that $\phi(\tilde{\boldsymbol{h}}_v^{(L)}) = c_v^{(L)}$ for all $v \in \mathcal{V}$.*

Informally, Theorem 11 (proof in Fey et al. [224]) indicates that scalable GNNs using historical embeddings are able to distinguish non-isomorphic structures (that are distinguishable by the WL test) directly during training, which is what makes reasoning about structural properties possible.

Nonetheless, to allow for high expressiveness, we need to tighten the upper bound of the approximation error induced by the usage of historical embeddings. As denoted before, the output embeddings of $\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell+1)}$ are exact if $|\bigcup_{v \in \mathcal{B}} \mathcal{N}(v) \cup \{v\}| = |\mathcal{B}|$, i.e., all neighbors of nodes in $\mathcal{B}$ are also part of $\mathcal{B}$. However, in practice, this can only be guaranteed for full-batch GNNs. Motivated by this observation, we aim to minimize the inter-connectivity between sampled mini-batches, i.e., $\min |\bigcup_{v \in \mathcal{B}} \mathcal{N}(v) \setminus \mathcal{B}|$, which minimizes history accesses and therefore reduces overall staleness in return.

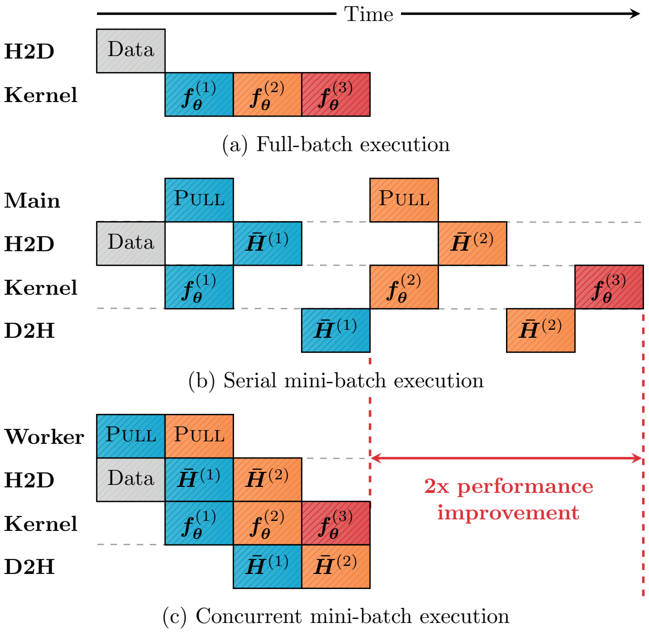
We make use of graph partitioning methods such as Metis [175, 361] to achieve this goal. It aims to construct partitions over the nodes in a graph such that intra-links within clusters occur much more frequently than inter-links between different clusters. Intuitively, this results in a high chance that neighbors of a node are located in the same cluster. Notably, modern graph clustering methods are both fast and scalable with time complexities given by $\mathcal{O}(|\mathcal{E}|)$, and only need to be applied once, which leads to an insignificant computational overhead in the pre-processing stage. However, this approach leads to the acceleration of training, since the number of neighbors outside of $\mathcal{B}$ is heavily reduced, and pushing information to histories leads to contiguous memory transfers.

4.3.4.2 Fast Historical Embeddings

Our approach accesses histories to account for any data outside the current mini-batch, which requires frequent data transfers to and from the GPU. Therefore, a special interest lies in the optimization of pulling from and pushing to the histories. We achieve that by making use of *non-blocking* device transfers. Specifically, we immediately start pulling historical embeddings for each layer asynchronously at the beginning of each optimization step, which ensures that GPUs do not run idle while waiting for memory transfers to complete. A separate worker thread gathers historical information into one of multiple pinned CPU memory buffers (denoted by Pull), from where it can be transfered to the GPU via the usage of CUDA streams without blocking any CPU or CUDA execution. Synchronization is done by synchronizing the respective CUDA stream before inputting the transferred data into the GNN layer. The same strategy is applied for pushing information to the history. Considering that the device transfer of $\bar{\boldsymbol{H}}^{(\ell-1)}$ is faster than the execution of $\boldsymbol{f}_{\boldsymbol{\theta}}^{(\ell)}$, this scheme does not lead to any runtime overhead when leveraging historical embeddings and can be twice as fast as its serial non-overlapping counterpart, (cf. Figure 4.12). We have implemented our non-blocking transfer scheme with custom C++/CUDA code to avoid Python's global interpreter lock.

4.3.4.3 Experimental Evaluation

For training large-scale GNNs, GPU memory consumption directly dictates the scalability of the given approach. In Fey et al. [224], we confirmed that GNNs trained via GAS are able to learn expressive node representations, closely resemble the performance of their non-scaling counterparts, and reach state-of-the-art performance on large-scale graphs. Here, we show how GAS maintains a low GPU memory footprint while, in contrast to other scalability approaches, accounting for all information present in the data. We directly compare the memory usage of GCN+GAS training with the memory usage of



(a) Full-batch execution

(b) Serial mini-batch execution

(c) Concurrent mini-batch execution

Fig. 4.12: Illustrative runtime performances of a serial and concurrent mini-batch execution compared with a full-batch GNN execution. In the full-batch approach (a), all necessary data is first transferred to the device via the HOST2DEVICE (H2D) engine, before GNN layers are executed in serial inside the kernel engine. As depicted in (b), a serial mini-batch execution suffers from an I/O bottleneck, in particular because each kernel engine has to wait for memory transfers to complete. The concurrent mini-batch execution (c) avoids this problem by leveraging an additional worker thread and overlapping data transfers, leading to a two times performance improvements compared with a serial execution, which is on par with the standard full-batch approach.

full-batch GCN [375], mini-batch GRAPHSAGE [292], and CLUSTER-GCN [137] training on three large-scale datasets [320, 731] (Table 4.2). Notably, GAS is easily able to fit the required data on the GPU, while the memory consumption only increases linearly with the number of layers. Although CLUSTER-GCN maintains an overall lower memory footprint than GAS, it will only utilize a fraction of the available information, i.e., about 23 % on average.

We now analyze how GAS enables large-scale training due to fast mini-batch execution. Specifically, we are interested in how our concurrent memory transfer scheme reduces the overhead induced by accessing historical embeddings from the offline storage. For this, we evaluate running times of a 4-layer GIN model on synthetic graph data, which allows fine-grained control over the ratio between inter- and intraconnected nodes (Figure 4.13). Here, a given mini-batch consists of exactly 4000 nodes, which are randomly intraconnected to 60 other nodes. We vary the number of inter-connections (connections to nodes outside of the batch) by adding out-of-batch nodes that are

Tab. 4.2: GPU memory consumption (in GB) and the amount of data used (%) across different GNN execution techniques. GAS consumes low memory while making use of all the data.

	# nodes **# edges**	**717K** **7.9M**	**169K** **1.2M**	**2.4M** **61.9M**
	Method	**Yelp**	**ogbn-arxiv**	**ogbn-products**
3-layer	Full-batch	9.44GB/ 100 %	2.11GB/ 100 %	31.53GB/ 100 %
	GraphSAGE	2.19GB/ 14 %	0.93GB/ 33 %	4.34GB/ 5 %
	Cluster-GCN	0.23GB/ 13 %	0.22GB/ 40 %	0.23GB/ 16 %
	GAS	0.79GB/ 100 %	0.34GB/ 100 %	0.59GB/ 100 %
4-layer	Full-batch	12.24GB/ 100 %	2.77GB/ 100 %	41.10GB/ 100 %
	GraphSAGE	4.31GB/ 19 %	1.55GB/ 37 %	11.23GB/ 8 %
	Cluster-GCN	0.30GB/ 13 %	0.29GB/ 40 %	0.29GB/ 16 %
	GAS	1.07GB/ 100 %	0.46GB/ 100 %	0.82GB/ 100 %

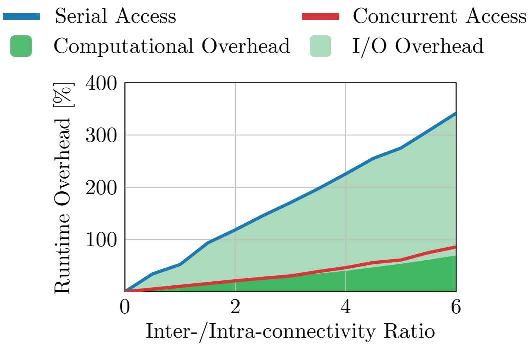


Fig. 4.13: Runtime overhead between serial and concurrent history access patterns in relation to the inter-/intraconnectivity ratio of mini-batches. The overall runtime overhead is further separated into computational overhead (overhead of aggregating additional messages) and I/O overhead (overhead of pulling from and pushing to histories). Our concurrent memory transfer scheme reduces historical-caused overhead by a wide margin.

randomly connected to 60 nodes inside the batch. Notably, the naive serial memory transfer increases runtimes up to 250 %, which indicates that frequent history accesses can cause major I/O bottlenecks. By contrast, our concurrent access pattern shows *no* I/O overhead *at all*, and the overhead in execution time is solely explained by the computational overhead of aggregating far more messages during message propagation. Considering the increased amount of additional data available, this overhead is marginal, in particular because most real-world datasets come with inter-/intra-connectivity ratios between 0.1 and 2.5 [224]. Further, the additional overhead of computing Metis partitions in the pre-processing stage is negligible. Computing the partitioning of a graph with 2M nodes takes only about 20–50 seconds (depending on the number of clusters).

4.3.5 Conclusion

We introduced graph neural networks for graph machine learning based on deep learning techniques. We demonstrated that graph neural networks follow a general message passing scheme, which is suitable for a wide range of operators and learning tasks. The generality of message passing is show-cased in the PyTorch Geometric library, a well-known deep learning library for implementing and working with graph-based neural networks. Furthermore, we discussed scalable approaches for applying graph neural networks to large-scale graphs. In particular, we showed that scalable approaches based on the sub-sampling of edges or non-trainable propagations weaken the expressive power of message passing. By contrast, our proposed framework, AutoScale, overcomes this restriction by utilizing historical node embeddings while being both fast and memory-efficient to train. While this scheme allows scalable graph machine learning on single or multiple GPUs on the same machine, additional considerations need to be taken into account when data is laid out in a distributed fashion (Section 8.2).

4.4 High-Quality Parallel Max-Cut Approximation Algorithms for Shared Memory

Nico Bertram
Jonas Ellert
Johannes Fischer

Abstract: We engineer parallel algorithms for approximating the maximum cut in a large directed graph. Our general approach is to first partition the graph into p parts, where p denotes the number of processing elements. The individual processors then independently compute an approximation to their local part of the graph using high-quality sequential approximation algorithms. In a final step, a single MAX-DICUT instance of size $\mathcal{O}(p^2)$, capturing the interprocessor edges, is defined and solved exactly, using fast parallel Integer Program solvers or slow approximation algorithms that compute a good approximation. By partitioning the input graph into $p' > p$ parts, we get a smooth trade-off between cut quality and running time. We also show applications of our algorithm in parallel grammar-based text compression.

4.4.1 Introduction

Data that occurs in real-world applications can often be structured as *graphs* where data points are represented as nodes and relationships between different data points are captured by edges. Graphs occur in many applications, e.g., road networks, relationships in social networks [193], and bioinformatics [40].

The problem of finding a partitioning of a *directed graph* into two subsets S and T such that the sum of the edge-weights between the two subsets is maximized is one of the classical NP-complete problems. We denote this problem with MAX-DICUT. It is closely related to its counterpart in *undirected graphs*, MAX-CUT, which was shown to be NP-complete by Karp [359]. In fact, MAX-DICUT seems much harder than MAX-CUT since every instance of MAX-CUT can be easily transformed into an instance of MAX-DICUT. It can be shown that this transformation also defines a reduction from a MAX-CUT instance to a MAX-DICUT instance which shows the NP-hardness of MAX-DICUT. That means that there is probably no polynomial time algorithm to compute an optimal solution for MAX-DICUT.

One common approach in theory and practice is to solve such hard problems by using approximation algorithms. These algorithms allow for a multiplicative error α with $0 < \alpha < 1$ so that the computed cut is in the worst case by a factor of α worse than the optimal cut. We call this factor α the *performance guarantee* of an algorithm.

One simple randomized algorithm assigns each node with probability $\frac{1}{2}$ either to S or T, which leads to a solution with an expected performance guarantee of $\frac{1}{4}$. This algorithm can be derandomized with the method of *conditional expectations* [590, 643]. Buchbinder et al. described a linear time algorithm with a performance guarantee of $\frac{1}{3}$ [90] that can also be randomized to achieve an expected performance guarantee of $\frac{1}{2}$.

Currently, the best-known performance guarantee of 0.874 uses a formulation of MAX-DICUT as an Integer Program that is then relaxed into a *Semidefinite Program* and achieves a performance guarantee of 0.79607 [256]. This algorithm can be derandomized as well [459]. The performance guarantee was later further improved to 0.859 [750]. The best-known performance guarantee of 0.874 was achieved by further improving this approach [426]. In case that the *Unique Games Conjecture* [372] is true, the performance guarantee can be improved up to 0.878.

There are also attempts to solve MAX-CUT by using a machine learning approach. Gu and Yang described a deep neural network combined with learning strategies such as supervised learning and reinforcement learning [275]. Yao et al. [719] used Graph Neural Networks [291] to solve MAX-CUT and compared it with the algorithm by Goemans for undirected graphs [256] and a local search procedure [59]. The results for both machine learning approaches are promising. However, they were only evaluated for small graphs; how to apply these approaches for directed graphs remains open.

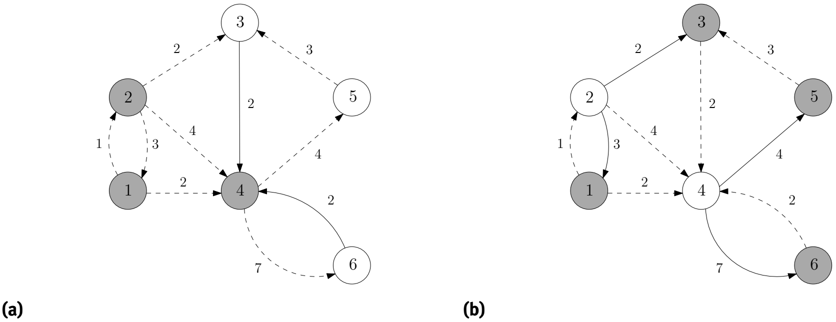
MAX-CUT can also be used in a graph-based semi-supervised learning approach. Wang et al. [695] showed that a bivariate cost function can be reduced to a constrained MAX-CUT formulation. Since this formulation has a number of linear constraints on the nodes and the edge weights can be negative, most approximation algorithms cannot be used directly. The authors propose using a greedy gradient MAX-CUT algorithm, instead.

To our knowledge, no algorithm exists that produces a MAX-DICUT with high quality and performs well in shared memory. One approach to developing such algorithms is to use a graph partitioner [100] to partition a graph into k parts of roughly equal size in terms of node balancing or edge balancing. On each of the k parts we can run a sequential algorithm to compute a local solution with high quality that we have to merge in a final step. In this contribution, we first describe some elementary algorithms to compute a MAX-DICUT in a graph. Then, we engineer a framework that computes a MAX-DICUT with high quality in shared memory that uses the pattern described above. We also show how we can use a parallel MAX-DICUT with high quality in grammar-based string compression to improve the compression ratio.

Parts of this work have already been published in [49].

4.4.2 Preliminaries

First, we define cuts in directed graphs. Then, we describe some important approximation algorithms for MAX-DICUT.



(a)

(b)

Fig. 4.14: A graph with two example cuts. The nodes that are in S are colored in white and the nodes that are in T are colored in gray. The edges that are not counted for the cut are dashed. The cut in (a) has the value 4. The cut in (b) has the value 16, which is the optimal value.

4.4.2.1 Notations

Here, we define the necessary notations for graphs and cuts in directed graphs. A *directed* and *weighted graph* G is a tuple (V, E, w) where $V = \{1, ..., n\}$ is the set of *vertices*, $E \subseteq V^2$ is the set of *edges* with $|E| = m$ and $w\colon E \to \mathbb{R}_{>0}$ defines the non-negative *weights* of each edge.

A *cut* in a directed and weighted graph $G = (V, E, w)$ is a partitioning of V into the subsets S and T so that the sum of the edge-weights for edges with origin in S and target in T is maximized. The value of a cut with respect to S and T is defined by $C(S, T) = \sum_{i\in S, j\in T} w(i, j)$. We omit S and T in case they are clear by the context. The maximum cut is then defined by $C_{max} = \max_{S,T} C(S, T)$. We call an edge (u, v) with $u \in S$ and $v \in T$ a *cutting edge*. In Figure 4.14 we see examples of cuts in a directed graph.

4.4.2.2 Algorithms

In the following, we will describe the algorithms we implemented in our framework. First, we describe a naive random approximation and its derandomization. Then, we describe the algorithm by Goemans and Williamson.

Random Partitioning One simple algorithm to produce a partitioning of a graph G is to decide for each node v independently with probability $\frac{1}{2}$ whether we assign v to S or T. This algorithm calculates a cut with an expected performance guarantee $\frac{1}{4}$ in linear time.

Theorem 12. *The described algorithm calculates a cut with an expected performance guarantee of $\frac{1}{4}$ in $\mathcal{O}(n)$ time.*

Proof. Since we assign each node in constant time either to S or T, the running time is $\mathcal{O}(n)$. Now, we have to show the performance guarantee. Let $G = (V, E, w)$ be a directed graph. Let $W = \sum_{i,j \in V} w_{ij}$. First, we observe that

$$C_{max} \leq W. \tag{4.30}$$

Next, let $e = (u, v) \in E$ be an arbitrary edge. This edge is a cutting edge only if $u \in S$ and $v \in T$. Since we assigned each node randomly with probability $\frac{1}{2}$ to either side of the partition, the probability that e is a cutting edge is exactly $\frac{1}{4}$. So in expectation our calculated cut has the value $E[C] = \frac{1}{4}W$. By Equation 4.30 it follows that $\frac{1}{4}W \geq \frac{1}{4}C_{max}$ and lastly $\frac{E[C]}{C_{max}} \geq \frac{1}{4}$. □

Derandomization The random algorithm described above can be derandomized so it deterministically produces a cut with a performance guarantee of $\frac{1}{4}$. This can be done with the method of *conditional expectations* [590, 643]. Suppose we already placed nodes $1, \ldots, i-1$ into either S or T. We denote as $E[C \mid 1, \ldots, i-1]$ the expected value of the cut when we place nodes $i, \ldots, n$ at random into each partition. Now, we want to assign node i to either S or T. Let us assume that the value $E[C \mid 1, \ldots, i-1] \geq \frac{1}{4}C_{max}$ (when $i = 0$ this assumption is trivially satisfied). Intuitively, we try to put i into the partition that results in the best expected outcome. Since $E[C \mid 1, \ldots, i-1] \geq \frac{1}{4}C_{max}$, at least one of both decisions has to result in an expected value of the cut of $\frac{1}{4}C_{max}$. We can calculate the expected increase for each decision for node i as follows:

$$A = \sum_{\substack{j<i \\ j \in T}} w_{ij} + \frac{1}{2} \sum_{j>i} w_{ij} \tag{4.31}$$

$$B = \sum_{\substack{j<i \\ j \in S}} w_{ji} + \frac{1}{2} \sum_{j>i} w_{ji} \tag{4.32}$$

Equation 4.31 describes the expected increase of the cut when we place i into S and Equation 4.32 describes the expected increase of the cut when we place i into T. In both sums the first term refers to the already calculated partitioning of $1, \ldots, i-1$. The second term refers to the expected value when we assign $i+1, \ldots, n$ at random to either S or T. When we choose the maximum of A and B, we have $E[C \mid 1, \ldots, i] \geq \frac{1}{4}C_{max}$ and, when we assigned all nodes, $E[C] \geq \frac{1}{4}C_{max}$.

Theorem 13. *The described algorithm calculates a cut with a performance guarantee of $\frac{1}{4}$ in $\mathcal{O}(m)$ time.*

Goemans and Williamson Algorithm In the following, we describe the Goemans and Williamson algorithm [256] that in its original description had a performance guarantee of 0.79607 but was further improved up to 0.874 [426]. To illustrate the algorithm, we describe the algorithm for *undirected* graphs that has a performance guarantee of 0.878 and show at the end how to modify the algorithm for *directed* graphs.

First, we need some additional notation. By Prob[A] we denote the probability that event A happens. The function sgn(x) denotes the *sign* function that is defined as

$$\operatorname{sgn}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0. \end{cases}$$

The general idea of the algorithm is to solve a relaxed formulation of Max-Cut as an integer quadratic program (IQP) and then assign each node to either S or T depending on the computed solution. The interesting part about this algorithm is that we relaxed our formulation to a *semidefinite program*. This method, first introduced by Goemans and Williamson, leads to improved performance guarantees for other problems as well such as Max-2-SAT [256].

Let $G = (V, E, w)$ be a directed graph. We start with the following formulation of Max-Cut as IQP:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2}\sum_{i<j} w_{ij}(1 - x_i x_j) \\ \text{subject to} \quad & x_i \in \{-1, 1\} \quad \forall i \in \{1, \dots, n\} \end{aligned} \tag{4.33}$$

Each node i is represented by a variable x_i that has value -1 when i is placed into S and 1 when i is placed into T. When we look at the term $(1-x_i x_j)$, we can see that it evaluates to 2 if nodes i and j are in different partitions and 0 otherwise. Hence, each cutting edge is counted twice, which is why we normalize the calculated value by $\frac{1}{2}$. Solving Equation 4.33 is still NP-hard but now we examine the properties of this formulation when we relax its variables to vectors of dimension n. Let S_n be the n-dimensional unit sphere. In Equation 4.34 we see the relaxed formulation.

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2}\sum_{i<j} w_{ij}(1 - v_i v_j) \\ \text{subject to} \quad & |v_i| \in S_n \quad \forall i \in \{1, \dots, n\} \end{aligned} \tag{4.34}$$

Note, that the optimal solution of Equation 4.34 is an upper bound of the optimal solution of Equation 4.33 because every solution of Equation 4.33 is also a solution of Equation 4.34 (we can transform x_i to a vector v_i by setting the first element to x_i and every other value to 0). In Figure 4.15a we see five vectors that for simplicity's sake are embedded in the unit circle. At first glance, it is hard to see how we should divide the vectors into the partitions S and T. Intuitively, v_1 and v_2 are relatively similar to each other so it should be more likely that they are placed in the same partition as v_2 and v_4. This similarity can be expressed by the scalar product of vectors u and v, which is defined by $u \cdot v = |u| \cdot |v| \cdot \cos(\alpha) = \cos(\alpha)$ where α is the angle between u and v.

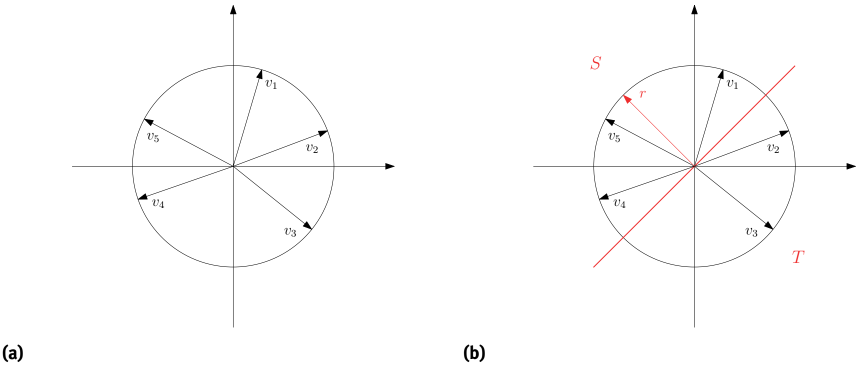


Fig. 4.15: An example for how to assign nodes to either S or T. In (a) we see a solution to Equation 4.34. In (b) we see a random vector r that defines a partitioning of the nodes. Since v_1, v_4 and v_5 lie on the same side of r, we set $S = \{1, 4, 5\}$ and $T = \{2, 3\}$. For simplicity, all vectors are embedded in the 2-dimensional unit circle.

To compute an optimal partitioning is still hard but we can compute a partitioning that results in a good solution with high probability. We choose a random vector r uniformly distributed over S_n. With r we can define a partitioning by putting all vectors v_i that lie on the same side of r into S i.e., $S = \{i \mid v_i \cdot r \geq 0\}$ and all other vectors into T i.e. $T = \{i \mid v_i \cdot r < 0\}$. This partitioning is visualized in Figure 4.15b. Intuitively, it is more likely that with this partitioning similar vectors are placed in the same partition, which should result in a good solution. This intuition can be formalized in the following lemma.

Lemma 14. *Let v_i and v_j be vectors that are optimal solutions of Equation 4.34 and $r \in S_n$ be a random vector drawn uniformly from the n-dimensional unit sphere. Then*

$$Prob[sgn(v_i \cdot r) \neq sgn(v_j \cdot r)] = \frac{1}{\pi} \arccos(v_i \cdot v_j).$$

By Lemma 14 our calculated Cut has an expected value of

$$E[C] = \frac{1}{\pi} \sum_{i<j} w_{ij} \arccos(v_i \cdot v_j).$$

From the fact that $\frac{\arccos(v_i \cdot v_j)}{\pi} \geq \alpha \frac{1}{2}(1 - v_i v_j)$ with $\alpha > 0.878$ we can derive the following theorem:

Theorem 15. *Let v_i and v_j be vectors that are optimal solutions of Equation 4.34. Then*

$$E[C] \geq \alpha \frac{1}{2} \sum_{i<j} w_{ij}(1 - v_i v_j) \geq \alpha C_{max}.$$

Now, we still have to show how to get an optimal solution for Equation 4.34. We can transform this formulation into a *semidefinite program (SDP)*. First, we have to define *positive semidefinite matrices*.

Definition 16. *Let $M \in \mathbb{R}^{n \times n}$ be a symmetric matrix. M is called* positive semidefinite *if all of its eigenvalues are non-negative. If M is positive semidefinite, we denote this by $M \succeq 0$.*

The following important property holds.

Lemma 17 ([257, 410])**.** *Let $M \in \mathbb{R}^{n \times n}$ be a positive semidefinite matrix. There exists a matrix $B \in \mathbb{R}^{n \times n}$ such that $M = B^T B$. We can calculate B in $\mathcal{O}(n^3)$ time with a Cholesky Decomposition.*

A *semidefinite program* has the following form where $A_1, ..., A_m, B_1, ...B_m \in \mathbb{R}^{n \times n}$ are constant matrices and $b_1, ..., b_m \in \mathbb{R}$. The variable matrices $X_i \in \mathbb{R}^{n \times n}$ have the constraint that they should be positive semidefinite matrices. To multiply matrices, we use the *Frobenius inner product* defined by: $A \bullet B = \sum_{i,j} A_{ij} B_{ij}$.

$$\begin{aligned} \text{maximize} \quad & A_1 \bullet X_1 + ... + A_m \bullet X_m \\ \text{subject to} \quad & X_i \succeq 0 \quad \forall i \in \{1, \dots, m\} \\ & B_i \bullet X_i = b_i \quad \forall i \in \{1, \dots, m\} \end{aligned} \tag{4.35}$$

Optimal solutions for a SDP can be computed in $\mathcal{O}(n^c \log(\frac{1}{\epsilon}))$ time for some $c > 0$ by using *interior point methods* [343] where $\epsilon > 0$ is an error parameter.

To convert Equation 4.34 into an SDP in the form of Equation 4.35, we set $y_{ij} = v_i \cdot v_j$. We observe that y_{ij} describes the cosine of the angle between vectors v_i and v_j and $y_{ij} = y_{ji}$. So the matrix

$$Y = \begin{pmatrix} y_{11} & y_{11} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nn} \end{pmatrix}$$

is a symmetric matrix that describes the cosine of the angles between every vector. The element y_{ii} describes the length of vector v_i. Since every vector lies on the unit sphere S_n, we add the condition that $y_{ii} = 1$ for every $i \in \{1, \dots, n\}$. So the formulation of MAX-CUT as SDP is as follows.

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} W \bullet ((1)_{n \times n} - Y) \\ \text{subject to} \quad & Y \succeq 0 \\ & y_{ii} = 1 \quad \forall i \in \{1, \dots, n\} \end{aligned} \tag{4.36}$$

Here W is defined as the weight matrix of G and $(1)_{n\times n}$ is the matrix that contains 1 in each component. By Lemma 17 we can obtain an optimal set of vectors v_i with a Cholesky Decomposition in time $\mathcal{O}(n^3)$.

Now, we show how to modify our formulations to get an approximation algorithm that results in a cut with a performance guarantee of 0.79607 for MAX-DICUT. We start again with the formulation of MAX-DICUT as IQP:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{4}\sum_{i<j} w_{ij}(1 + x_0x_i - x_0x_j - x_ix_j) \\ \text{subject to} \quad & x_i \in \{-1, 1\} \quad \forall i \in \{1, \dots, n\} \end{aligned} \tag{4.37}$$

Here, we introduce an additional variable x_0 that marks which side lies in S. More precisely, if x_0 is equal to -1 all other nodes i with $x_i = -1$ should be assigned to S and to T otherwise. The term $(1 + x_0x_i - x_0x_j - x_ix_j)$ evaluates to 4 if x_i is assigned to S and 0 otherwise. That is why we have to normalize with the value $\frac{1}{4}$. Similar to the undirected MAX-CUT, we relax the variables in Equation 4.37 so that they are n dimensional vectors.

$$\begin{aligned} \text{maximize} \quad & \frac{1}{4}\sum_{i<j} w_{ij}(1 + v_0v_i - v_0v_j - v_iv_j) \\ \text{subject to} \quad & |v_i| \in S_n \quad \forall i \in \{1, \dots, n\} \end{aligned} \tag{4.38}$$

For its formulation as a SDP we use the matrices $X \in \mathbb{R}^{n+1\times n+1}$ and $Y, Z \in \mathbb{R}^{n\times n}$ that are defined as follows:

$$X = \begin{pmatrix} y_{00} & y_{01} & \dots & y_{0n} \\ y_{10} & y_{11} & \dots & y_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n0} & y_{n1} & \dots & y_{nn} \end{pmatrix} \quad Y = \begin{pmatrix} y_{11} & y_{11} & \dots & y_{1n} \\ y_{21} & y_{22} & \dots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \dots & y_{nn} \end{pmatrix} \quad Z = \begin{pmatrix} y_{10} & y_{10} & \dots & y_{10} \\ y_{20} & y_{20} & \dots & y_{20} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n0} & y_{n0} & \dots & y_{n0} \end{pmatrix}$$

Then MAX-DICUT can be formulated as SDP as follows:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{4}W \bullet ((1)_{n\times n} + Z - Z^T - Y) \\ \text{subject to} \quad & X \succeq 0 \\ & y_{ii} = 1 \quad \forall i \in \{0, \dots, n\} \end{aligned} \tag{4.39}$$

When we compute a solution for Equation 4.39, we choose a random vector vector r uniformly distributed over S_n. Since v_0 marks the side for the partition S, we assign all nodes i where v_i and v_0 lie on the same side to S. More precisely, we set $S = \{i \mid \text{sgn}(v_i \cdot r) = \text{sgn}(v_0 \cdot r)\}$ and $T = \{i \mid \text{sgn}(v_i \cdot r) \neq \text{sgn}(v_0 \cdot r)\}$.

By analyzing the algorithm similarly to the undirected MAX-CUT, we find that our algorithm has a performance guarantee of 0.79607. The following theorem summarizes our results.

Theorem 18. *The described algorithm calculates a cut with a performance guarantee of* 0.79607 *in polynomial time.*

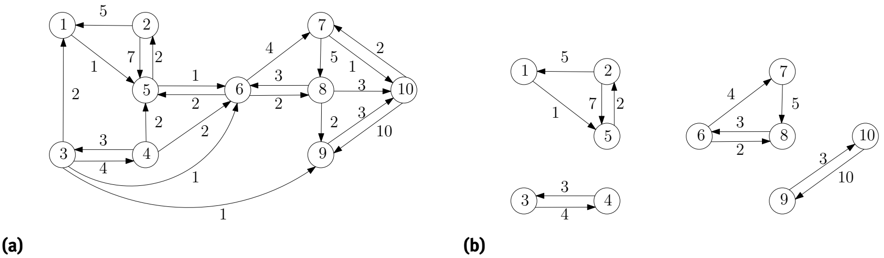


Fig. 4.16: The input graph in (a) is partitioned into 4 subgraphs, which can be seen in (b), so that the sum of the edge-weights between the subgraphs is minimized.

4.4.3 Framework

In this section we introduce a parallel framework that computes a Max-Dicut with high quality in shared memory. First, we give an overview of the whole framework before we introduce each step individually.

Our approach is to partition an input graph G into k subgraphs G_i of roughly equal size so that the dependency between the subgraphs is minimized i.e. the edge-weights between the subgraphs are minimized. Then on each computed subgraph, we run in parallel a sequential Max-Dicut algorithm to compute several local cuts. In a final step, we have to merge the locally computed cuts. We achieve this by defining a new graph on which Max-Dicut is solved where each node represents a partition computed by the local Max-Dicut algorithms.

4.4.3.1 Graph Partitioning

The first step of our framework is to partition the input graph $G = (V, E, w)$ into k subgraphs so that we can run a Max-Dicut algorithm on each subgraph independently. Our goal is to include as much edge information as possible into each subgraph to improve the quality of the computed Max-Dicut. We achieve this by maximizing the sum of the edge-weights in each subgraph or, vice versa, by minimizing the sum of the edge-weights between the subgraphs. That is to say, we want to minimize $\sum_{i,j\in\{1,\dots,k\}} E_{ij}$ where E_{ij} is the sum of the edge-weights between subgraph G_i and G_j. To compute a well balanced graph partitioning in shared memory, there already exist several approaches. In our framework we use the graph partitioner KaHIP [13], which partitions G into the subgraphs $G_i = (V_i, E_i, w), i \in \{1, \dots, k\}$ so that each subgraph has roughly equal size, i.e. we allow for a multiplicative error ϵ so that $|V_i| \leq (1+\epsilon)\left\lceil \frac{|V|}{k} \right\rceil$. We also use a partitioning algorithm that naively divides either the nodes or edges into k chunks of equal size. We could also use Metis, as in Section 4.3. However, KaHIP outperforms Metis and is better suited for our application. In Figure 4.16 we see an exemplary input graph that is partitioned into 4 subgraphs.

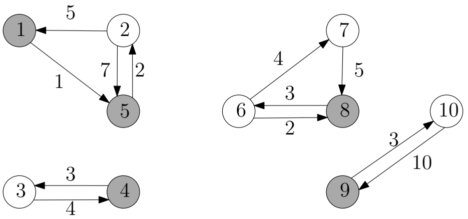


Fig. 4.17: On each computed subgraph in Fig. 4.16 we compute a Max-Dicut. The nodes that are in S_i are colored in white and the nodes that are in T_i are colored in gray.

4.4.3.2 Compute Local Solutions

After partitioning the input graph into multiple subgraphs, we run in parallel a sequential MAX-DICUT algorithm on each subgraph to compute a local cut for each subgraph. We can compute cuts with higher quality when we use algorithms with better performance guarantees. Since these algorithms are also slower, we have to consider which algorithm achieves the best trade-off between the quality of the cut and the runtime of the framework.

We have implemented the randomized algorithm with an expected performance guarantee of $\frac{1}{4}$ and its derandomized variant, the algorithm with a performance guarantee of $\frac{1}{3}$ that was introduced by Buchbinder [90], and the algorithm with an expected performance guarantee of 0.79607 by Goemans and Williamson [256]. As an example, we show in Figure 4.17 the optimal MAX-DICUT for all subgraphs that were computed in Figure 4.16.

4.4.3.3 Merging

In a final step, we have to merge the computed local cuts into a global cut. A naive approach is to define $S = \bigcup_i S_i$ and $T = \bigcup_i T_i$ as the trivial cut. The problem with this approach is that it does not consider the edges between the subgraphs. It might be possible that it is more advantageous to swap the subsets S_i and T_i in the global graph or even to put S_i and T_i into the same partition. To consider each possible combination of merging the cuts, we reduce the problem of merging the local solutions to another MAX-DICUT instance. We build a complete graph H with $2k$ nodes in which each node represents a locally computed partition S_i or T_i. Let X and Y be two nodes of H. We add an edge (X, Y) to H with weight $\sum_{i \in X, j \in Y} w(i, j)$. Then, we can run a MAX-DICUT algorithm on H. Since the graph has only $2k$ nodes, we can use an expensive algorithm to compute an exact solution. In our framework we implemented a simple brute-force algorithm and an algorithm that solves the formulation of MAX-DICUT as an Integer Program. We can also use the approximation algorithm with a performance guarantee of 0.79607 by Goemans and Williamson. In Figure 4.18 we see how the local cuts that were computed in Figure 4.17 are merged into a new MAX-DICUT instance. Then, we compute the global solution on this instance.

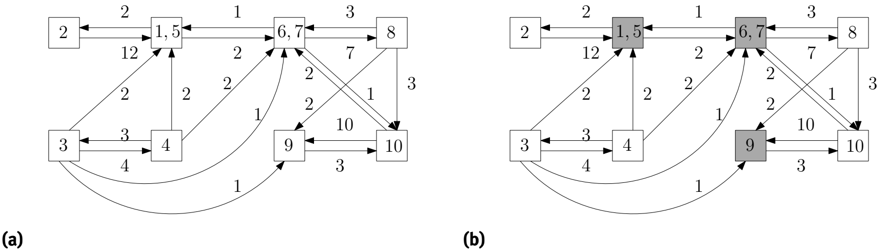


Fig. 4.18: We compute a new Max-Dicut instance with 8 nodes (Fig. 4.18a) by merging the nodes that are in the same partition after Fig. 4.17. We run an exact Max-Dicut algorithm on this instance and compute the global Max-Dicut in Fig. 4.18b. The nodes that are in S are colored in white and the nodes that are in T are colored in gray.

Tab. 4.3: A summary of our used input graphs.

Graph	$\|V\|$	$\|E\|$
recomp_dna1GB_5	28 245	1 439 986
road-luxembourg-osm	114 600	239 332
rt-retweet-crawl	1 112 703	4 557 704

4.4.4 Evaluation

In this section we evaluate our framework. We conducted our experiments on the LiDO3-Cluster of the Technical University of Dortmund[10] on a node with an Intel Xeon CPU E5-2640 processor (20 cores, 2.4 GHz, L1 32K, L2 256K, L3 256M) with 64 GB of RAM. The code was written in C++ and compiled using GCC 8.4 using OpenMP for parallelization.

We evaluate our framework on the input graphs that are summarized in Table 4.3. The graph recomp_dna1GB_5 was generated from a *recompression* tool [342] by using the 1 GiB prefix of the text dna.txt from the Pizza & Chili text corpus.[11] Here, the nodes represent the characters from the alphabet and we have an edge (a, b) if the pair ab appears in the text. The weight of the edge represents the number of occurrences of the pair ab. The graphs road-luxembourg-osm and rt-retweet-crawl were taken from *Network Repository* [604]. The graph road-luxembourg-osm is a road network of Luxembourg and the graph rt-retweet-crawl is a Retweet graph of Twitter where each node represents a Twitter user and we have an edge between two users when one user retweets a tweet from the other user.

[10] https://www.lido.tu-dortmund.de/cms/de/LiDO3/index.html, accessed June 9, 2022.

[11] http://pizzachili.dcc.uchile.cl/, accessed June 9, 2022.

4.4.4.1 Experiments

For our experiments, we evaluate each part of our framework separately. First, we compare our partitioning algorithms KaHIP, NodeSlice, and EdgeSlice. Then, we compare our local Max-Dicut algorithms Derandomization, Buchbinder, and Goemans. For the Goemans algorithm, we compare two variants: one solves the SDP exactly, which we call *Goemans*, the other solves the SDP with a small error where we set $\epsilon = 0.01$, which we will call *Goemans*($\epsilon = 0.01$). For our merging algorithms, we compare the Buchbinder algorithm, the two Goemans variants described above, and an exact algorithm that solves an *integer linear program* (ILP). When we evaluate the algorithms of one part of our framework, all other parts are fixed, i.e. for the partitioning we use KaHIP, as the local Max-Dicut algorithm we use Buchbinder, and as merging algorithm we use Goemans ($\epsilon = 0.01$). We conducted all of our experiments five times and took the average of each result for the computed cut as well as the runtime for each step in the framework. We divide the graphs into as many as 2048 parts. Up until 16 parts, we use the same amount of cores as the number of parts. For more than 16 parts, we constantly use 16 cores.

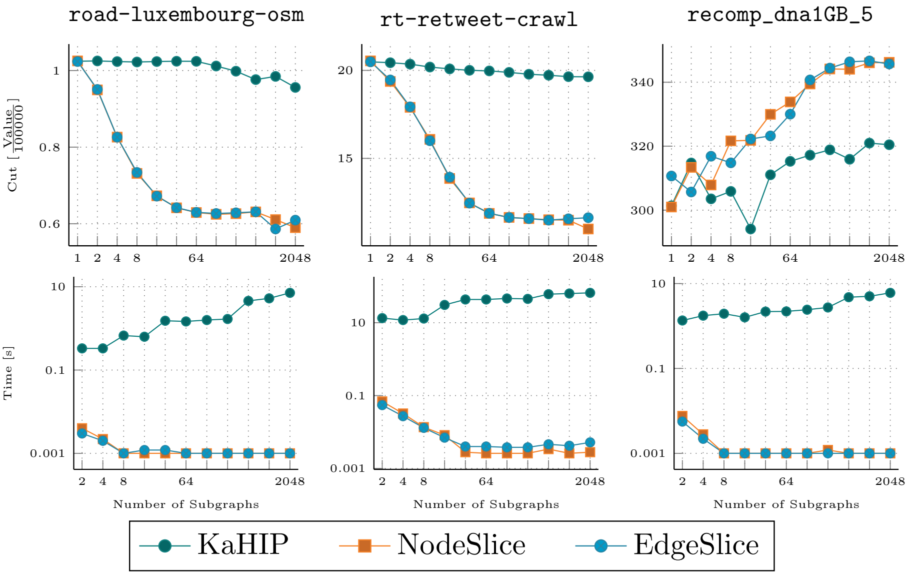


Fig. 4.19: The computed cut and the runtime for our partitioning algorithms while the other steps of the framework are fixed algorithms. Missing data points indicate either that the runtime of the whole framework exceeded the time limit or that the memory exceeded the RAM.

In Figure 4.19, we can see our results for our partitioning algorithms. We see that using KaHIP as a partitioner results in an almost constant cut quality for each number of subgraphs for the graph `road-luxembourg-osm` and `rt-retweet-crawl` while the cut quality when using the naive partitioning algorithms NodeSlice and Edgeslice gets worse when we partition the graph into more subgraphs. However, on the graph `recomp_dna1GB_5` the cut quality when using NodeSlice and EdgeSlice scales better than KaHIP. The runtime of KaHIP is on all inputs significantly slower than NodeSlice and EdgeSlice and does not scale as well as the naive algorithms.

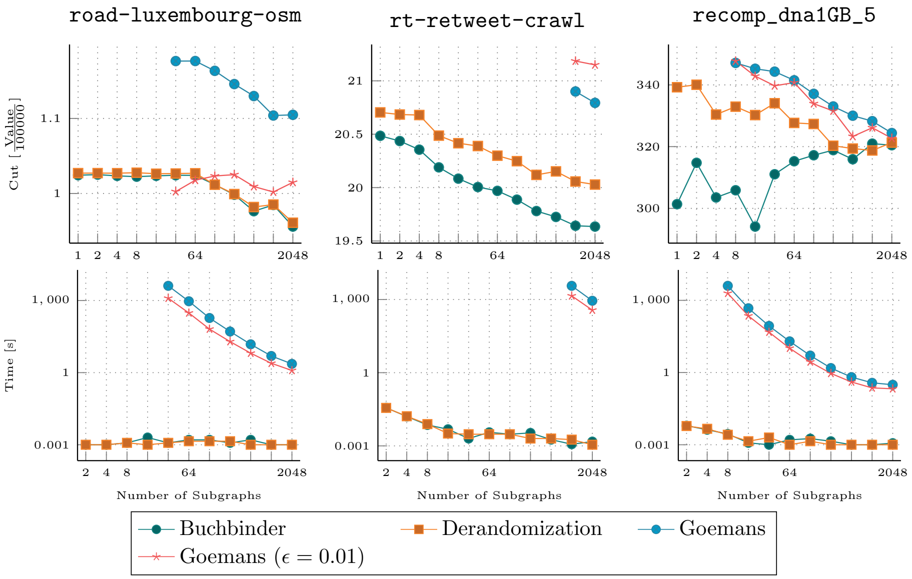


Fig. 4.20: The computed cut and the runtime for our local MAX-DICUT algorithms while the other steps of the framework are fixed algorithms. Missing data points indicate either that the running time of the whole framework exceeded the time limit or the memory exceeded the RAM.

In Figure 4.20, we can see our results for the local MAX-DICUT algorithms. We can see that by using the variants of the Goemans algorithm our framework produces the overall best cut quality. However, these algorithms only compute a solution when we partition the graph into a large number of subgraphs or when we have smaller subgraphs. For larger subgraphs, the Goemans algorithm either takes too long or consumes too much memory. The runtime of the Goemans algorithms is significantly larger than the linear-time algorithms but it gets faster the smaller the subgraphs get.

In Figure 4.21, we can see our results for the merging algorithms. Note that, since our framework uses some random variables, the computed quality may vary between different configurations. Overall, the merging has only a small effect on the cut quality. As one would expect, the exact algorithm that solves an ILP gives the best solution most of the times, closely followed by the exact Goemans algorithm. Using Goemans ($\epsilon = 0.01$), our framework produces a good cut quality most of the times,as well. However, for 128 parts or more the cut quality gets significantly worse on `road-luxembourg-osm`.

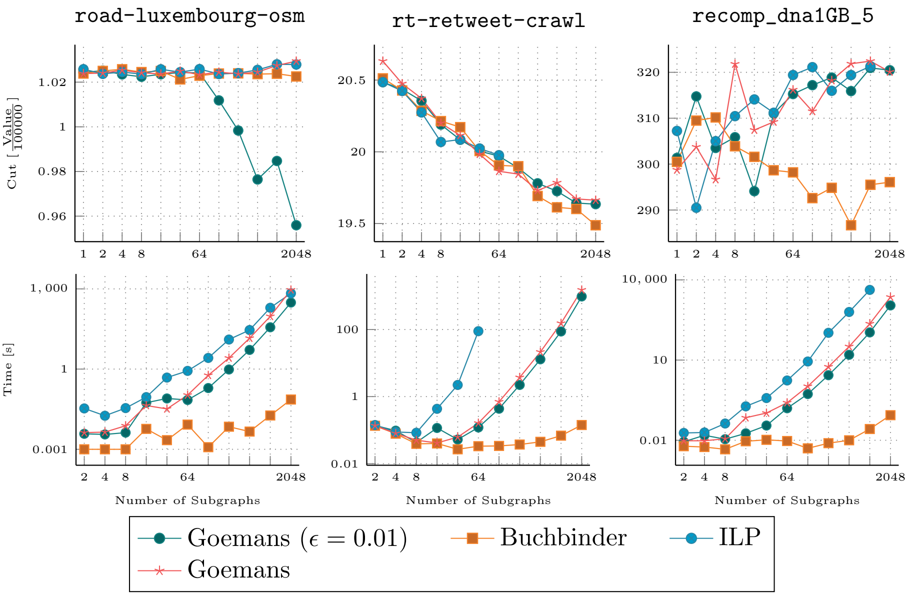


Fig. 4.21: The computed cut and the runtime for our merging algorithms; the other steps of the framework are fixed algorithms. Missing data points indicate either that the runtime of the whole framework exceeded the time limit or that the memory exceeded the RAM.

The runtime of ILP is overall the slowest and on `rt-retweet-crawl` becomes too slow for 128 and more parts. The Goemans variants are faster than IPL but are still slower than Buchbinder. We can see that ILP and the Goemans variants slow the larger the Graph H becomes.

4.4.5 Application in String Compression

Max-Dicut is used in building a succinct data structure over strings to answer *Longest Common Extension (LCE) queries* efficiently. An LCE query over a string S asks for two positions i and j for the longest common prefix of the suffixes starting at position i and j.

To answer such queries efficiently, one can use the *recompression* technique that was described by Jez [342]. With this technique, a string S is compressed into a context-free grammar that generates exactly S. Then, we build an LCE data structure [330] on top of the grammar. The memory usage is $\mathcal{O}(z\log(\frac{n}{z}))$ and the query time is $\mathcal{O}(\log(n))$ where z is the size of the Lempel-Ziv 77 factorization [746] and n is the size of S.

During the compression of S into a context-free grammar, we try to find pairs ab and build a rule $X \to ab$ so that as many pairs are covered by a rule. To do that, we build a directed graph G in which each node represents a character of S and we insert

Tab. 4.4: The results of different recompression algorithms. We compare the running time in seconds and the compression ratio (compressed text length divided by original text length) for 8 cores on different texts taken from the Pizza & Chili text corpus. In all experiments we use 200 MiB prefix for each text. We mark in bold text the best result on the respective text. Additionally, we provide the size of and the alphabet size σ for each text.

		`max-dicut_recomp`		`lp_recomp`	
Text	σ	**Time [s]**	**ratio**	**Time [s]**	**ratio**
`cere`	5	290.75	**4.9 %**	**21**	4.91 %
`dna`	16	286.2	42.42 %	**26**	**42.29 %**
`einstein.en`	139	211.75	**0.17 %**	**26**	**0.17 %**
`english`	239	277.4	**41.23 %**	**36**	42.72 %
`para`	5	256.6	**6.81 %**	**22**	6.82 %
`sources`	230	288.2	**37.79 %**	**33**	39.91 %

an edge from a to b if the pair ab appears in S. Then, a cut in G represents a partition of the characters into two subsets Σ_1 and Σ_2 so that we can compress as many pairs ab with $a \in \Sigma_1$ and $b \in \Sigma_2$ as possible without overlapping pairs. Accordingly, there is a direct correlation between the quality of the computed cut and the compression ratio of S.

We integrated our framework for computing a Max-Dicut in a tool that computes the compression with *recompression* in shared memory. We compare the algorithm `max-dicut_recomp` that uses our Max-Dicut framework with the algorithm `lp_recomp` that computes first a naive Max-Cut (S, T) and then compares $C(S, T)$ and $C(T, S)$ in the directed graph and takes the largest value. Additionally, `lp_recomp` tries to take the solution that produces lesser production rules.

Again, we conducted our experiments on the LiDO3-Cluster of the Technical University Dortmund on a node with an Intel Xeon CPU E5-2640 processor (20 cores, 2.4 GHz, L1 32K, L2 256K, L3 256M) with 64 GB of RAM. We compared the compression ratio and runtime for 8 cores of our algorithms on a number of texts taken from the Pizza & Chili text corpus.[12] We repeated our experiments five times and took the average as the final result.

Table 4.4 shows our results. We can see that `max-dicut_recomp` achieves on almost all texts a similar or better compression ratio than `lp_recomp`. On `english` and `sources` the compression ratio increases by 1–2 %. However, to increase the compression ratio for `max-dicut_recomp` we need around 10 times more runtime than `lp_recomp` on all texts on 8.

[12] http://pizzachili.dcc.uchile.cl/, accessed June 9, 2022.

4.4.6 Conclusion

In this section we described a framework that calculates a high quality Max-Dicut in shared-memory that is also easily extendable. We implemented our framework and evaluated it in shared-memory on real-world graphs. The experiments showed that our graph partitioning algorithm KaHIP does not scale well in shared-memory so we plan to use other partitioning algorithms in the future. The best configuration of our framework is to partition our graph into small graphs and use Goemans as our local Max-Dicut algorithm.

We also integrated our framework into a software that calculates a grammar-based compression. By using our framework, we achieve in most cases better compression rates. However, our new algorithm is much slower than other compression algorithms.

4.5 Millions of Formulas

Lukas Pfahler

Abstract: Amid the increase in the number of research publications, the search for relevant papers has become tedious. In particular, searches across disciplines or schools of thinking are not supported. This is mainly due to the retrieval in terms of keyword queries, as technical terms differ in different sciences and at different times. Relevant articles might better be identified by their mathematical problem descriptions. Just looking at the equations in a paper already gives a hint to whether the paper is relevant. Hence, we propose a new approach for the retrieval of mathematical expressions based on machine learning. We design an unsupervised representation learning task that combines embedding learning, contrastive learning, and self-supervised learning. We want our learned representation to allow the automatic identification of related, relevant mathematical expressions. Using graph convolutional neural networks we embed mathematical expressions in low-dimensional vector spaces that allow efficient nearest-neighbor queries. To train our models, we collect a huge dataset with over 29 million mathematical expressions from over 900 000 publications on arXiv.org. The math is converted into an XML format, which we view as graph data. In this data, we are able to automatically identify equalities and inequalities that we can use for training and testing of our models. Furthermore, our empirical evaluations involve a dataset of manually annotated search queries show the benefits of using embedding models for mathematical retrieval. This contribution is based on a conference paper [563] and more details can be found in [562].

4.5.1 Introduction

Machine learning has contributed to many a search engine success story. Unfortunately, the search is most often based on words or text. Technical terms in different disciplines, however, may have different meanings or the same meaning may be referred to by different terms. For instance, various usages of Bayes' law occur in different scientific fields and can be found under different names. For instance in astrophysics, it is known as *information field theory* [200]. Without a knowledge of physics or the use of the name *Bayes*, the law is easily recognized by the formula $P(d|s) = P(d,s)/P(s)$ in any paper. Another example is a 1925 paper by Ising in the physics journal under the title *Ferromagnetismus*. Today, the Ising model is also popular in machine learning, but it is referred to first as the *Hopfield network* and later as the *Boltzmann machine*. This illustrates the aspect of time: words for particular topics change over time. The language of Ising's paper is German; the paper introducing Jensen's inequality in 1906 is written in French. Again, the inequality $f((a+b)/2) \le f(a)/2 + f(b)/2$ can be easily understood, in both cases. We conclude that the most compact and comprehensive way to transport the main ideas of scientific manuscripts in disciplines like computer science or physics are the equations used. Thus it should also be the way we formulate our search queries when searching for scientific manuscripts. In order to judge the relevance of mathematical expressions for a search query, a system has to generalize between different notations and match the parts of equations, that describe the same concepts, even if they appear in a different form. A human reader resorts to domain knowledge acquired over years of training in his field to judge relevance. We wonder how machine learning models with access to vast amounts of mathematical content can help to automatize this process.

In this work, we propose using graph neural networks to learn a representation of mathematical expressions that captures semantic relatedness. To this end, we design two unsupervised learning tasks, one classic embedding learning task based on contextual similarity and one self-supervised learning task inspired by masked-language models. We curate a dataset of over 28.9 million equations from over 900 000 papers on arXiv.org and represent the equations as graphs with one-hot encoded features. Then we train our models on this large collection of equations. We compile an evaluation dataset with annotated search queries from several different disciplines and showcase the usefulness of our approach for deploying a search engine for mathematical expressions.

4.5.2 Math Search and KDD

Mining and indexing mathematical expressions in document collections is a challenging task, mostly tackled in the information retrieval community [277, 745]. We outline how the problem of math search is treated with the tools from Knowledge Discovery in Data and data mining and present related work on the machine learning methods we chose for our approach.

Representation The first question we have to consider is how to represent mathematical expressions. Approaches can be divided into two categories: those for visual representation and those for semantic representation. The former category is focused on the layout of an expression. The most prominent choices are LaTex, a Turing-complete language used in the publications on arXiv.org, and `Presentation MathML`[13], an XML dialect for displaying math on the web that we chose in this work. The latter category includes Content MathML and OpenMath, two similar XML dialects that focus on semantics rather than layout, and domain-specific languages for symbolic math solvers

[13] https://www.w3.org/TR/MathML3/.

like Mathematica that also allow the manipulation and transformation of formulas. To the best of our knowledge, no large, public collection of semantic math expressions exists and, unfortunately, converting math from a display-representation, where data is available in large quantities, to a semantic representation, which seems more appropriate for searching, is a non-trivial task. Available solutions either use rules and heuristics, e.g. the converter `ml2om` that translates LaTeX to OpenMath[661], or also apply machine learning [693]. We chose to apply machine learning methods directly on the `Presentation MathML` representation. The bottom line of the representation question is that math is expressed in trees, either XML or other parse trees. Our previous work [564] may be the notable exception to this: we chose to represent equations as fixed-size bitmaps. While one could argue that this is an unsuitable choice, the multitude of machine learning or computer-vision approaches that successfully transform images of typeset [170] or hand-written [15, 460] math back to tree-based representations suggests that bitmap representations preserve all required information of tree-based approaches.

Similarity Measure The second question is how we compute similarity between formulas. Zanibbi et al. distinguish text-based, tree-based, and spectral approaches [729]. Text-based approaches transform tree-structured math into a sequence by preorder traversal, say, and then estimate the similarity using methods for sequences such as cosine similarities of bags-of-words or the length of the largest common substring. Tree-based approaches focus on matching trees or subtrees. Typically computing similarities using sub-structures, either sub-sequences or sub-trees, involves solving dynamic-programming problems. Spectral approaches work on paths or partial subtrees in the trees. An example is the work by Zhong and Zanibbi [745], which indexes root-leaf paths of operator trees. From matches of the root-leaf paths, they compute the largest common subexpression to score the similarity of two equations. To convert math from LaTeX to the semantic representation of operator trees, the authors use ca. 100 grammar rules created by domain experts.

A new trend is to use machine learning to learn a similarity measure. A machine learning model maps an equation to a dense, low-dimensional vector. The similarity between these so-called embeddings can be computed via their inner product, which enables fast indexing using a variety of index structures, including faiss and annoy, designed for efficiently handling millions of these dense, low-dimensional vectors. Mansouri et al. [464] propose that equations be embedded using fastText, a method originally designed for computing word embeddings, while in our previous work [564] we compute embeddings with a similar embedding learning task and convolutional neural networks (see Section 4.2).

Graph Convolutional Neural Networks We have proposed an embedding model based on Graph Neural Networks (GNN) [563] introduced in this contribution. They are an appealing model choice for this task, as like classic Convolutional Neural Networks

(CNNs) for image processing, they compute feature maps based on local neighborhoods and thus can work on relations between symbols in formulas. While in CNNs we have features associated with each pixel in the pixel grid and neighborhoods are defined by this grid, in GNNs we have features associated with each node of the graph and neighborhoods are defined by the edges in the graph. We define graph structures $x = (X, E)$ as a tuple of node-features X and edges E. Let $|x|$ denote the number of nodes in x. We assume that $X \in \mathbb{R}^{|x| \times d}$ where X_i are the features of the i-th node. A GNN maps an input graph to an output with transformed feature vectors in a $d^{'}$-dimensional output space but with identical edge structure. We use the graph network to compute a vector-valued embedding for mathematical expressions by an average-pooling operation that aggregates all node-embeddings of a graph into a single graph-embedding.

Additionally we investigate the use of transformer architectures [681], more specifically of Bidirectional Encoder Representations (BER)T models [173], for the task of embedding mathematical expressions into vector spaces. Transformers can be viewed as GNNs on a fully connected graph where each layer aggregates neighborhoods using self-attention [681].

Self-Supervised Learning We further draw inspiration from a recently proposed class of representation learning tasks called self-supervised learning. Self-supervised learning tasks are unsupervised learning tasks, where parts of the inputs are used to construct proxy tasks. The representations learned in these proxy-tasks can then be used in downstream tasks. For instance, we can rotate images and train a model to predict the rotation angle, as proposed by Gidaris et al. [251]. Using massive amounts of unlabeled data readily available, we can fit models that solve a task like this.

We are particularly interested in , where parts of the input are hidden from a model and the model's task is to predict the hidden parts. This was made popular by the BERT model for pretraining natural language representations [173] and has since then been adopted to other inputs such as pretraining for image classification with convolutional neural networks [669]. We construct a masking task for mathematical expressions and use graph convolutional neural networks to predict the masked parts.

4.5.3 The Data

We outline how we gather data from arxiv.org and transform it to graph structured data for our graph convolutional neural network.

4.5.3.1 Dataset

We are working on data obtained from arxiv.org, a service where scientists can upload their manuscripts or pre-prints without reviewing process. We have downloaded all the

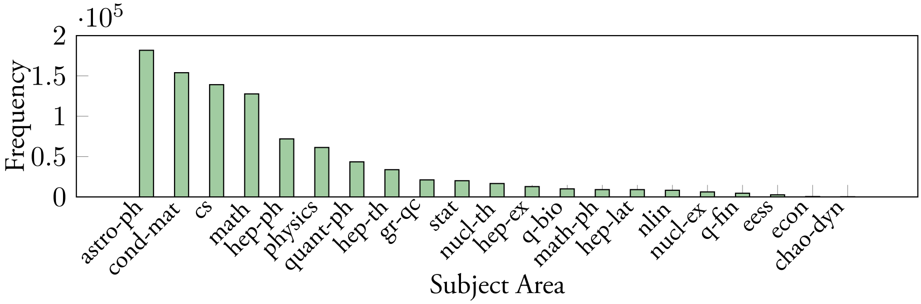


Fig. 4.22: Number of papers per subject area in our sample.

LaTeX sources of publications up to April 2019 from the official bulk data repositories.[14] This way we have obtained 934,287 papers. As we can see in Figure 4.22, the large majority of these papers are from disciplines where mathematical expressions are an important part of the publications. The most prominent subject areas are astrophysics, condensed-matter physics, high energy physics, computer science, and mathematics.

From all publications, we extract mathematical expressions by using regular expressions for the most common math-environments such as 'equation', 'align', etc. We do not use inline math snippets but focus on expressions that stand on their own, as they tend to describe more important concepts. Furthermore we extract user-defined commands and macros. Using the library `Katex`[15] we compile the raw LaTeX-equations to the XML-based `MathML` format. Out of all papers downloaded, 760 041 papers contain at least one equation that we were able to convert to `MathML`. In total we have a dataset of 28 973 591 `MathML` equations. Furthermore we have used regular expressions to find arXiv-ids in the bibliographies of the paper to build a citation graph. In total, 540 892 papers have an outgoing edge, with a total number of edges of 4 553 297. Since we only detect those references that use an arXiv-id in, say, an texttturl, our citation graph is only a subgraph of the true citation graph.

To ensure reproducibility we provide the scripts used for processing the public arXiv data dump, extracting the mathematical expressions and converting them to `MathML` as well as collecting meta-data and citations at https://github.com/Whadup/arxiv_library[16].

[14] https://arxiv.org/help/bulk_data_s3.

[15] http://katex.org.

[16] You can find the datasets used in this study at http://github.com/Whadup/arxiv-learning. We also share our citation graph, which might be interesting in other applications.

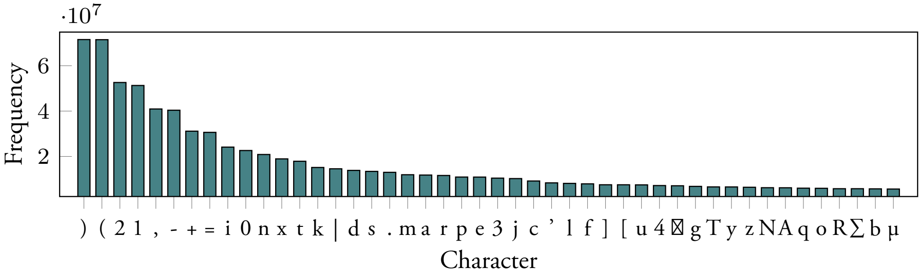


Fig. 4.23: The 50 most frequent characters in math environments.

4.5.3.2 Data Representation

In order to feed the MathML to a graph convolutional neural network, we have to convert it to a graph with vectorial node features. The MathML standard defines around 30 different XML-tags such as <mi> for math identifiers or <mo> for math operators. Some of these tags use attributes, to change font or spacing, say. Leaf nodes contain text such numbers, parenthesis, or letters (Greek, Latin, etc...). We view the XML structure as a tree and use its nodes and edges and derive features based on tags, attributes, and text. For each node we use one-hot encoded feature vectors of dimensionality 256. First, we represent each node as a single token, where the token is derived by concatenating tag, attributes and text and use the 256 most frequent tokens that capture the majority of tokens in the data. Attribute values often contain numbers, e.g., for changing the font-size. We round these numbers to one decimal place to reduce the number of possible values. In addition to the one-hot encoded features, we store the position of the node among its sibling nodes.

Then, for the use with transformer models, we compute a sequential representation of our tree-structured data by a pre-order traversal of the tree.

4.5.4 Learning to Find Related Equations

In this section we will introduce the graph convolutional neural network used for computing embeddings and present two unsupervised learning tasks used for training the network. indexsubsubsectionModel for Embedding Formulas

Graph Neural Network We define a graph convolutional neural network for the task of embedding mathematical expressions into a low-dimensional vector space. The raw MathML is converted to graphs with vectorial features as described in Section 4.5.3.2. We propose using a special first layer that combines the one-hot encoded information at a node with the decimal position attribute. Following Vaswani et al. [681], we encode the position of the i-th node $p_i \in \mathbb{N}$ using positional embeddings. We use fixed sinusoid embeddings [681] denoted by $E(p_j)$, but in order to still allow the model to control the influence of the positional embeddings, we introduce a learnable scaling coefficient α initialized to 1.

$$x_i^{(1)} = \text{ReLU}\left(\sum_{j\in\mathcal{N}(i)\cup i} W^{(1)}x_j + \alpha E(p_j) + b^{(1)}\right)$$

The first layer is followed by 3 fully-connected graph convolution layers of width 512, where the l-th layer is defined by

$$x_i^{(l)} = \text{ReLU}\left(\sum_{j\in\mathcal{N}(i)\cup i} W^{(l)}x_j^{(l-1)} + b^{(l)}\right)$$

which linearly transforms all nodes using a weight matrix $W^{(l)}$, adds a bias term $b^{(l)}$, aggregates by computing the sum over all neighbors $\mathcal{N}(i)$ and applies the ReLU activation component-wise. All graph convolution layers output feature maps with 512 dimensions. In our tree-structured data we assume all edges are bi-directional; hence the set of neighbors consists of the parent node and all child nodes. We apply batch-normalization before the first and third graph convolution layer. For the remainder of this paper, let $\phi(x) \in \mathbb{R}^{|x|\times 512}$ denote the output of the last graph convolution layer given the input x. To obtain a single embedding for an input graph, we compute the mean of all node features. This mean is transformed in another linear layer to reduce the dimensionality to 64. For the remainder of this paper, let $\bar{\phi}(x) \in \mathbb{R}^{64}$ denote this embedding of x.

When scoring similarities between embeddings with margin losses, we need to control the norm of the embeddings, otherwise the notion of adherence to a margin becomes meaningless. Ding et al. [179] and others have proposed normalizing all embeddings to unit length. We propose a softer normalization inspired by batch normalization[335] that also allows us to obtain embeddings with norms smaller than 1. For every training batch of graphs, we compute the mean of the norm as well as its standard deviation. Then we inversely scale each embedding by the mean plus the standard deviation. This way, most embeddings have a norm smaller than 1. We keep a running average of the means and standard deviations. At inference time, we use these running averages for scaling.

Transformer The original transformer model — as proposed by Vaswani et al. [681] — is slightly modified in BERT [173], which only uses encoder layers. In our work we use the same transformer model architecture as BERT — including the same encoder layers, activation functions, optimization algorithms, and learning rate schedules. The transformer architecture introduces the multi-head-attention layers as the key mechanism for learning the relations between each pair of tokens in the input sequence. This is applicable on mathematical formulas too, because understanding the relations between the symbols of a mathematical formula is crucial for understanding the meaning of the formula. We also extend the vocabulary by the special classification, separation, masking, and unknown token—as did [173]—in order to predict masked tokens and thereby integrate in the model the ability to correct mathematical expressions.

We explore three differently-sized variants of the BERT architectures for embedding mathematical expressions. While BERT has a hidden-size-dependent number of attention heads, we keep them constant. We set the number of different multi-head-attention heads *D* to 4. By doing so the hidden size *H* has the largest impact on the performance of the multihead attention. As for the intermediate projection size, we kept this always bigger than the hidden size so that we can have a linear projection on a higher space. The resulting models are summarized in Table 4.5.

Tab. 4.5: Math-BERT model configurations.

Model	L	H	I	D	Params	GFLOP
SMALL	4	128	768	4	1.2 m	0.7
BASE	8	256	768	4	6.0 m	3.6
LARGE	12	512	768	4	25.0 m	15

4.5.4.1 Representation Learning Tasks

We propose that our embeddings are trained using two self-supervised learning tasks simultaneously by adding their respective losses.

Contextual Similarity For learning relations between equations, we rely on the established contextual similarity task that was first made popular by word embeddings [493] and has hence been used in many representation learning approaches, including our approach [564] for learning similarities between equations. The main idea is that objects that frequently appear in shared contexts are related. We define the context of mathematical expressions as the paper containing the equation and conjecture that two equations are related if they appear in the same paper, as originally proposed in [564]. We extend this approach and further define two equations as related if one paper references the other using a citation graph. This way we hope to connect equations that describe the same context but use different notation. In addition, we discriminate between sampling expressions from the same paper and from the same section. We hope that within sections, equations are more related to each other. For obtaining positive examples of related equations, we

1. sample a paper uniformly at random and select an expression from this paper uniformly at random;
2. randomly select whether we sample from the same section, same paper or along a citation;
3. sample a positive example using that method; when we cannot find a positive example using that method, we jump back to (1).

For learning similarities we also require negative examples. To obtain these, we sample a paper uniformly at random and select an expression from this paper uniformly at random. The random process that generates these weak labels for similarity learning introduces a lot of noise, as many equations we claim to be related are in fact unrelated and some of the pairs we say are unrelated are related. We leave the investigation of more advanced sampling schemes to future work.

Using the sampled equations x with positive x^+ and negative partners x^-, we apply similarity learning. We have to choose a suitable loss function and investigate two different losses: Triplet and Histogram. The triplet loss [38] that we have previously used [564], contrasts the similarity between a positive pair of examples and a negative pair of examples and demands that the similar pair has a higher similarity by a user-defined margin Δ, usually set to 1.

$$\ell_t(x, x^+, x^-) = \max(0, \Delta - \langle\bar{\phi}(x), \bar{\phi}(x^+)\rangle + \langle\bar{\phi}(x), \bar{\phi}(x^-)\rangle) \tag{4.40}$$

We have proposed using the histogram loss as first published by Ustinova and Lempitsky[676]. It does not work on a triplet of equations, but on a mini-batch of size m positive pairs X^+ and a batch of negative pairs X^- with respect to anchor examples X. We collect all similarities between positive pairs in a vector $s^+ = (\langle\bar{\phi}(x_i), \bar{\phi}(x_i^+)\rangle)_{i=1,\ldots,m}$ and of all negative pairs in s^-. We divide the interval $[-1, 1]$ into $R-1$ equally-sized bins with boundaries $-1 = t_1, t_2, \ldots, t_R = 1$ and width $\Delta = 2/(R-1)$ and build histograms for the positive similarities and the negative similarities. Now we demand that the positive histogram leans more toward the $+1$ similarity than the negative histogram. We formalize this intuition as

$$\ell_h(s^+, s^-) = \frac{1}{m^2}\sum_{r=1}^{R}\sum_{r'=1}^{r}\left(\sum_{i=1}^{m}\delta_r[s_i^-]\right)\left(\sum_{i=1}^{m}\delta_{r'}[s_i^+]\right) \tag{4.41}$$

where instead of hard assignments, we use the triangular kernel

$$\delta_r[s] = \begin{cases} (s - t_{r-1})/\Delta \text{ if } s \in [t_{r-1}, t_r] \\ (t_{r-1} - s)/\Delta \text{ if } s \in [t_r, t_{r+1}] \\ 0 \text{ otherwise} \end{cases}$$

to put similarities into bins. This way we obtain a differentiable loss function. We hope that histogram loss is more robust with regard to the massive noise in our labels as each positive example is contrasted with all negative examples.

Masking Task We propose extending the contextual similarity task by another task and optimizing the sum of both tasks for training our embedding models. The main idea of our second task is, that the symbols in mathematical expressions do not appear independent from each other, but have strong dependencies. Thus if we hide a fraction of the symbols in an equation, we should be able to approximately reconstruct the hidden symbols from the remaining symbols. This task is reminiscent of masked language modeling tasks made popular by BERT [173] for natural language processing. In order to successfully solve this task, a model has to learn about the frequencies of symbols and their dependencies from the data, as is illustrated in Figure 4.24.

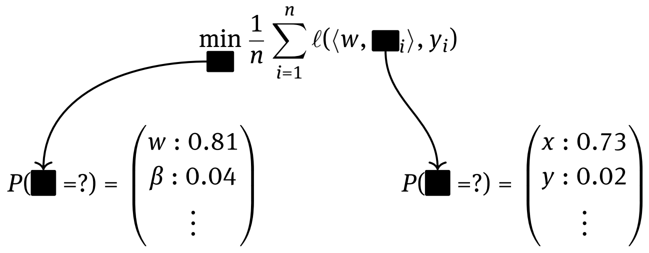

$\min_{\blacksquare} \frac{1}{n} \sum_{i=1}^{n} \ell(\langle w, \blacksquare_i \rangle, y_i)$
$P(\blacksquare =?) = \begin{pmatrix} w : 0.81 \\ \beta : 0.04 \\ \vdots \end{pmatrix}$
$P(\blacksquare =?) = \begin{pmatrix} x : 0.73 \\ y : 0.02 \\ \vdots \end{pmatrix}$

Fig. 4.24: Example of the masking task with fictional values.

More formally, we proceed as follows. For each input graph x with features X, we randomly set the feature vector of 15 % of the nodes to all zero obtaining the graph $x_{\blacksquare}$. Then we compute $\phi(x_{\blacksquare}) \in \mathbb{R}^{|x| \times 512}$. Now for each masked node, we solve a classification task: given $\phi_i(x_{\blacksquare})$, predict the right token, i.e. the combination of XML-tag, XML-attributes, and character. This classification task is solved using a single linear layer of dimensionality 256 with softmax-activation and cross-entropy-loss.

$$\ell_i = \ell(\text{softmax}(W)\phi_i(x_{\blacksquare}) + b, X_i)$$

The loss is only evaluated for the masked tokens and we compute the mean over all masked tokens to obtain a loss value for $x_{\blacksquare}$.

Adding this task to the contextual similarity task has the additional advantage that we now learn a representation that not only captures context information, but also preserves information about the raw input symbols.

4.5.4.2 Data Augmentation

Data augmentation eases the generalization of machine learning models and is particularly popular for image classification tasks where we can augment images by randomly rotating, scaling, padding, etc. For mathematical expressions, we propose the following random data augmentation. Since we know that a renaming of symbols in equations rarely changes the semantic, we propose randomly permuting the character features of all nodes that correspond to a math identifier, encoded in `<mi>` tags according to the `MathML` standard. For each equation we process, we sample a number of flips from a Poisson distribution with an expected value of 32. Then starting with the identity permutation that does not change the order of our 192 features, we construct a permutation with the desired number of flips by incrementally exchanging two random characters.

4.5.5 Experimental Results

In this section we perform an experimental evaluation of our embedding model. In particular, we focus on the use-case of a search engine for mathematical expressions. We begin by investigating the effects of the individual components of our model on a small, closed subset of the data. Then we investigate the effectiveness of our method on all 29.9 million equations.

4.5.5.1 Analysis on the Machine Learning Subset

We begin our analysis only on arXiv publications where the primary subject classification is machine learning (`cs.LG`). This is a natural choice, as we have some expertise to judge the quality of our results, a task which we are in no way equipped for across all subject fields.

Of these 9936 publications, we sample two subsets, train and test sizes of 7949 and 1987, respectively, and a total number of equations of 237 335 and 54 767, respectively. We use the train-set for building our embedding models and use the test-set to investigate generalization properties.

For training, we sample 1 million triplets (x, x^+, x^-). Of these triples, 45.9 % have a positive pair from the same section, 42.2 % from the same paper, and 13.9 % along an edge in the citation graph. We sample 100k triplets for testing with similarly distributed positive examples.

We perform an ablation study on our proposed embedding model and compare it with prior work. This section investigates the influence of our design choices. We decided (a) to use the histogram loss instead of the triplet loss, and (b) to add a masking task, (c) to data augmentation.

We measure the ranking score, i.e. the fraction of all triples in the training data where same-class pairs of equations have higher similarities than across-class pairs. As we see in Table 4.6, our evaluations indicate that all of our design choices contribute favorably to the overall performance on hold-out data, as deactivating any component decreases the score. We note that the biggest gain is achieved by switching from triplet-loss to histogram-loss. We believe that this is due to the massive noise in our labels.

We also compare with our previous model [564] and see that we beat it by a small margin. However, this comparison is not entirely fair, as their model was trained on a

Tab. 4.6: Ablation Study.

Influence factor	Ranking hold-out	Ranking eval	Accuracy eval
Full model	76.5 (±0.0)	57.7 (±0.0)	60.6 (±0.0)
No histogram loss	72.5 (−4.0)	49.6 (−8.1)	30.9 (−29.7)
No masking	75.2 (−1.3)	54.3 (−3.4)	50.0 (−10.6)
No augmentation	75.3 (−1.2)	53.6 (−4.1)	50.0 (−10.6)
Bitmap CNN original[564]	76.2 (−0.3)	71.9 (+14.2)	68.3 (+7.7)
Bitmap CNN retrained	70.0 (−6.5)	50.0 (−7.7)	52.9 (−7.7)

larger dataset of around 25 000 papers, probably including some of the papers in our test set. We use their code to re-train on our subset of equations and yield a substantial margin of 6.5 percentage points.

We also use our previous evaluation data [564]. It consists of 103 equations labeled into 13 categories related to machine learning including k-means, LSTMs, empirical risk minimization, etc. Since only bitmaps are available, we transcribe the equations manually. There are three issues with this evaluation set. First, it is too small to produce significant numbers. Second, some equations in the dataset appear in the training data. This is not only the case for our subset, but also for the training data used in [564]. Third, many equations within a category are obviously from the same paper, hence we have seen some of the pairs in our training data. Nevertheless we use the evaluation data. Indeed in our use-case of search engines, the crawled equations will always be in the training data and only the user queries will be unseen equations. In a way, we simulate this with the evaluation data.

Following the original experimental protocol, we measure the 1-nearest-neighbor accuracy obtained in leave-one-out validation (named Accuracy) as well as the above Ranking score. In Table 4.6, we again see that our model is only surpassed by the pre-trained model that uses a larger training dataset. This motivates the use of a much larger dataset.

4.5.5.2 Large-Scale Experiments

For training on all the papers in our dataset, we sample two different sets of training triplets, one with 5 million triplets and one with 20 million triplets. We train our models on a Nvidia GTX1080 GPU with 8 GB memory, which allows us to process mini-batches of 128 triplets, or 384 equations. During training, we process around 1300 triplets per second, not counting the time for reading data from hard disk. In total, one of the 20 epochs of training on 20 million triplets takes 6:30h on our system. We use annoy to construct an index for approximate nearest neighbor retrieval. In total, our index uses 13 GB of hard disk storage to manage all mathematical expressions in our dataset.

Tab. 4.7: Evaluation Scores.

Dataset	Ranking eval	Accuracy eval
1mio ML-subset triplets	57.7	60.6
5mio full ArXiv triplets	**76.2**	80.9
20mio full ArXiv triplets	75.3	**84.0**
Bitmap CNN original[564]	71.9	68.3

Before we evaluate our models in a search engine study, we again check the performance on the aforementioned evaluation data. The results in Table 4.7 indicate the power of using large amounts of training data, although it is unclear if using 20 million training triplets is an advantage over using only 5 million. Our large-scale models beat all the models trained on smaller amounts of data. Even though the smaller models were trained on only machine learning-related data, we obtain better scores on the machine learning evaluation data by training on all disciplines.

Let us now inspect two example search queries. In Figures 4.25 and 4.26 we see the two examples from the introduction, Bayes law and Ising models, and their respective nearest neighbors under our model trained on 5 million triplets. We see that we can find other definitions of Bayes' law as well as the related law of total probability. When we perform a query for the Ising model and look at the first 20 results, we find papers where the model is called the Boltzmann machine as well as papers that refer to the Ising model. This illustrates the power of querying for mathematical expressions instead of using keywords.

4.5.5.3 Search Engine Study

Finally we want to study the usefulness of our embedding approach for a search engine application more systematically. Traditionally, validating search engines using the measures precision or recall requires relevance scores for each result for each evaluation query. We see that this requires much manual annotation work since we have to manually identify each relevant equation for each query. Unfortunately, we were not able to find available evaluation data. The best fit is the NTCIR-12 task evaluation data [729] consisting of 37 annotated queries. But this is not appropriate for our ap-

Query:	$P(d \mid s) = \frac{P(d,s)}{P(s)}$
1st result:	$P(s \mid d) = \frac{P(d\|s)P(s)}{P(d)}$
4th result:	$P(d) = \int P(d \mid s)P(s)ds$

Fig. 4.25: Example: Bayes' law. We report the first result and the first result that does not show Bayes' law, but, in this case, the related law of total probability. The first result is from: R. H. Leike, T. A. Enßlin, *Charting nearby dust clouds using Gaia data only*, 2019.

Query:	$\sum_{i<j} w_{ij}\, s_i\, s_j + \sum_i \theta_i\, s_i$
’Boltzmann’ Result:	$E = -\sum_i b_i\, s_i - \sum_{i<j} w_{ij} s_i s_j.$
’Ising’ Result:	$\mathcal{H} = -\sum_{i<j} C_{ij} J_{ij} \sigma_i \sigma_j - \sum_i h_i \sigma_i$

Fig. 4.26: Example: Ising model. We find equations related to both Ising models and Boltzmann machines. First result is from: Weinstein, *Learning the Einstein-Podolsky-Rosen correlations on a Restricted Boltzmann Machine*, 2017. Second result is from: Ferrari et al., *Finite size corrections to disordered systems on Erdős–Rényi random graphs*, 2013.

proach, as most queries are a combination of math as well as keywords. When we ignore the keywords, the remaining query becomes very generic, for instance $x + y$, which makes it very unlikely that we accurately find the articles labeled as relevant. In addition, the overall focus of the NTCIR-12 task is recovery of exact matches, whereas our focus is on retrieving *related* expressions.

Consequently, we curate and publish our own evaluation dataset. To reduce the manual annotation labour, we want to apply a heuristic for the relevance judgement. To this end, we have asked our colleagues, many from disciplines other than computer science and data science, to provide us with equations that we should query. For each equation, they provide a set of keywords or keyphrases that should appear in the section around the result. If one of the keywords is present, we count the result as correct. In this way, we can evaluate our search result without manually checking result lists. If a keyword has more than 10 characters, we also count it, if we find a substring that has a Levenshtein distance less than 2. In total, we have 53 evaluation queries publicly available and editable online.[17]

We inspect two different information retrieval metrics that do not require the number of relevant documents in advance: Precision@k and unnormalized Mean Average Precision. Precision@k is defined as the fraction of relevant documents within the first k results. We report it for lists of 10, 100, and 1000 results and compute its mean over our evaluation queries.

Unnormalized Mean Average Precision is derived from the standard mean average precision metric. Since we do not now the number of relevant documents in advance, we omit this term, limit the search to a maximum of 1000 results, and obtain the following definition

$$\text{uMAP} = \sum_{k=1}^{1000} P(k)\Delta_k$$

where $P(k)$ is Precision@k and Δ_k specifies if the k-th result is relevant. Again we compute the mean over all evaluation queries. Compared with Precision@k, uMAP

[17] Crowd-sourced evaluation data can be accessed and edited here: https://www.overleaf.com/8721648589nrjxgwmtzfvm.

Tab. 4.8: Search Engine Performance

	P@10	P@100	P@1000	uMAP
BoW	0.4567	0.3170	0.2083	106.17
5Mio	**0.5038**	**0.3817**	**0.2984**	**165.04**
20Mio	0.4547	0.3709	0.2897	156.51

considers the order of the search results and rewards relevant results early in the result lists.

For reference, we include retrieval based on a bag-of-words (BoW) representation. To this end, we use our data representation as in Section 4.5.3.2, but compute the sum over all nodes in the graph to obtain a single 256-dimensional vector of the whole tree. We retrieve the nearest neighbors using cosine similarity.

In Table 4.8, we see that our approach beats the bag-of-words margin, in particular for larger values of k. We see for Precision@10, the performance between BoW and our embedding model is very similar. This is because for many queries the top-10 results are mostly near-perfect matches that are easily identified. However when looking at more results, we are able to find almost 50 % more relevant equations.

4.5.5.4 Retrieval of Equalities and Inequalities

We have extracted equalities and inequalities in the test set of our data using regular expressions. Using a simple heuristic, we filter the resulting (in-)equalities, such that left-hand-side (LHS) and right-hand-side (RHS) do not differ in length dramatically, thereby eliminating formulas such as definitions, where the LHS is only a single symbol. We derive three different datasets, one with only equalities (LHS and RHS split at "="), one with inequalities (split at < and ≤) and one with mixed relations (split at =<>≤ and ≥). This data allows us to use the LHS of the (in-)equalities as queries in hopes of retrieving RHS. We have made our finetuning-data available at https://whadup.github.io/arxiv_learning/ as well.

Following other machine learning-based approaches for mathematical retrieval [464, 563, 564], we use our models to encode formulas into a dense vector space and retrieve results using approximate nearest neighbor search [48]. In the case of our BERT models, we use output embedding of the CLS token as the representation for the whole formula and finetune the model to output meaningful embeddings for this first token. We finetune our models on half of the available data and test on the remaining half.

Finetuning Task We propose using contrastive learning to learn to identify the RHS given the LHS. The learning task in contrastive learning is identifying the right partner for each input in a minibatch of datapoints. Hence the representation learning problem is formulated as a classification problem. Let $X^l, X^r \in \mathbb{R}^{m\times d}$ contain the output embeddings of a minibatch of LHSs and RHSs. We normalize each embedding to unit length and denote the normalized embeddings by $\bar{X}^l$ and $\bar{X}^r$. We use the InfoNCE loss[547], i.e. the negative log-likelihood of softmax probabilities parameterized by the pairwise cosine similarities between the LHSs and RHSs:

$$\ell_\tau(\bar{X}^l, \bar{X}^r) = m^{-1} \sum_{i=1}^{m} \log \frac{\exp(\langle \bar{X}_i^l, \bar{X}_i^r \rangle / \tau)}{\sum_{j \neq i} \exp(\langle \bar{X}_i^l, \bar{X}_j^r \rangle / \tau)} \tag{4.42}$$

where $\tau > 0$ is a hyperparameter that controls the temperature of the output probability distribution, which we set to 10^{-2}. The contrastive learning task is more difficult for larger batchsizes m, as there are more candidate RHSs to chose from and thus the underlying classification problem becomes more difficult. But it has been shown that the utility of the model increases for larger batchsizes [134, 496], which we also investigate in our application.

Baseline Models In addition to our models we include several baseline models:

- First, we test a simple *bag-of-words* (BoW) model that is trained on a bag of `MathML` tree nodes. This model does not use a pre-training phase, but is only tuned on the finetuning data. The BoW model maps the sparse BoW representation to a d-dimensional vectorial embedding though a single matrix multiplication. In comparison with our BERT models, we do not restrict the vocabulary size of the inputs. The representation is trained using the same contrastive learning task with InfoNCE loss. We test $d \in \{64, 128, 256\}$ and report the best result after varying learning rates and number of training epochs in a grid search.
- Second we evaluate a pretraining approach based on the BoW model. The word-embedding-based approach *fastText* by Joulin et al.[350] is trained by predicting which tokens appear in the contexts together. Mansoury et al. use it for learning embeddings of formulae by serializing a `MathML` layout tree similar to the one we use in this work [464], hence we include it in our comparison. We finetune these embeddings by learning a linear mapping into a d-dimensional vector space, $d \in \{64, 128, 256\}$ using the same contrastive learning task.

We begin by training our models and the baseline models with a minibatch-size of 1024. Then we also investigate the effect of varying the batch size. Our implementations of all methods is available at http://github.com/Whadup/arxiv-learning.

Results For testing, we compute embeddings for all LHSs and RHSs in the test data and store them in an index structure. We use annoy [48], an indexing method for an approximate nearest-neighbor search based on an ensemble of random projection trees. We use an ensemble of 16 trees with default hyperparameters, but we found that the results were very insensitive to our particular parameter choices.

Then we query the k-nearest neighbors, $k \in \{1, 10, 100\}$, for each formula from the test set and check if the corresponding other side of the (in-)equality is in the result set. This way we can compute recall values to measure the quality of our embeddings.

Tab. 4.9: Results of the mathematical retrieval experiment. We report recall@K for $K \in \{1, 10, 100\}$.

Model	**Equalities (36864)**			**Relations (40960)**			**Inequalities (13312)**		
	R@1	**R@10**	**R@100**	**R@1**	**R@10**	**R@100**	**R@1**	**R@10**	**R@100**
SMALL-PRE	0.379	0.577	0.714	0.432	0.626	0.749	0.397	0.661	0.795
SMALL-NO-PRE	0.400	0.584	0.700	0.405	0.632	0.769	0.371	0.632	0.769
BASE-PRE	**0.511**	0.71	0.805	0.503	0.697	0.791	0.484	0.765	0.86
BASE-NO-PRE	0.434	0.623	0.729	0.446	0.63	0.734	0.409	0.682	0.797
LARGE-PRE	0.507	0.704	0.799	0.496	0.683	0.777	0.489	0.765	0.864
LARGE-NO-PRE	0.452	0.637	0.737	0.46	0.640	0.736	0.427	0.703	0.817
BOW	0.483	0.653	0.739	0.491	0.658	0.743	0.503	0.738	0.821
FASTTEXT [350]	0.480	0.650	0.739	0.488	0.651	0.742	0.488	0.713	0.810
GNN [563]	0.507	**0.833**	**0.884**	**0.512**	**0.834**	**0.883**	**0.504**	**0.870**	**0.922**

We summarize our findings in Table 4.9. Our BERT approach substantially outperforms both the BoW approaches, without (BOW) and with pretraining (FASTTEXT). This suggests that our model is capable of matching formulas based on characteristics that go beyond merely counting the number of matching tokens. However, the graph neural network GNN outperforms the sequential models in most scenarios, sometimes even substantially. It is, however, noteworthy that of the transformer models, the mid-size model is most useful.

For the mid-size and large models we observe the benefit of pretraining, as models that were trained from scratch perform worse than their pretrained counterparts. For the small models we do not consistently see this effect.

Overall, the recall at 10 for our approaches is already pretty high, which indicates that our representation learning on structured data is useful in search engine applications where users generally want to inspect only a small number of results.

4.5.6 Conclusion

Finding relevant literature across disciplines is essential for research. The search results should contain papers that are both relevant and stimulating. Very often, a look at the formulas in a paper gives a compact description of the problems and solutions it discusses. Hence, the goal is to find related papers based on the mathematical expressions. This task is different from mathematical information retrieval, but it shares the problem of determining the right representation of mathematical expressions.

In order to handle the large amounts of data that are common in search engine applications, we need models that allow efficient computation of the vector representations. Our approach based on graph-neural networks is a good fit for this demand as it makes use of the sparsely connected input graphs. As such it is much more computationally efficient than the other transformer models that we considered in this contribution.

We have demonstrated that representation learning on structured input is a useful approach for mathematical retrieval. Self-supervised and embedding learning successfully learned real-valued representations of tree-structures that allow efficient nearest-neighbor searches.

5 Cluster Analysis

An important process when analyzing a new dataset is to understand the dataset properties, characteristics, and contents. An essential ingredient here is the so-called "domain expertise", the knowledge about domain-specific pecularities of the data to get them preprocessed into an appropriate form for analysis. But even a domain expert that understands the meaning of the data may not be aware of some of its characteristics. In Exploratory Data Analysis (EDA), the data has now been preprocessed, cleaned, and transformed into an appropriate shape for further analysis—a tidy tabular form, say, and scaled such that we can apply distance measures. We can now explore the dataset to identify interesting substructures, that may either already be known (and may be a good candidate for labeling for later classification), that may be unknown, though irrelevant to the problem at hand, or that may ideally be not yet known, but interesting. Finding such novel knowledge about the data is known as the "data mining" step in the "Knowledge Discovery in Databases" (KDD) process. New knowledge represents the nuggets of gold that we are looking for in our mountains of data. There are several kinds of patterns that we may be looking for: these could be frequent patterns (such as combinations or sequences), anomalies (also called outliers, as we assume these objects to be rare deviations from normal data), and clusters.

In this chapter, we focus on clusters which are subsets of the dataset that are more coherent within the group, and that exhibit a larger deviation between these groups. Depending on the notions of coherence and deviation, we can arrive at very different notions of clusters – and hence of algorithms. Additionally, models and algorithms may differ by assumptions such as whether the data must be partitioned into disjoint subsets, or whether clusters may overlap. Clusters may form hierarchies, or may only be noticeable in particular subspaces or projections. Some methods reduce clusters to a single central point (for example the omnipresent k-means, but also k-medoids and many more), while other models allow non-convex clusters of arbitrary shape as long as the data is connected (for example, density connected in DBSCAN and Support Vector Clustering, but also in spectral clustering). There exists a huge zoo of algorithms, which may produce very different results. The appropriate choice of model and algorithm depends on the problem to solve. In many cases, k-means is a poor choice even though it may appear to be the easiest to use. It will always assign points to the nearest cluster center and it does not even handle clusters with varying diameter well.

Our focus in this book is on resource efficiency, which includes many aspects in clustering, as it is a very expensive problem. Many clustering problems are NP-hard to solve exactly, hence finding the optimum solution is infeasible for large datasets. Instead, it is common to use heuristics such as the standard algorithm for k-means that will only find a local fix point solution (which may not even be a local optimum), or to approximate the data.

https://doi.org/10.1515/9783110785944-005

In Section 5.1, we will focus on k-medoids clustering, which conceptually is related to k-means clustering but not limited to squared Euclidean distances. The usual algorithms for this problem require runtime and memory quadratic in the number of data points, and hence are not considered to be very efficient by the usual expectations for clustering (given that the popular k-means algorithms are considered to be linear in the number of data points N). The solution discussed here exploits that datasets can be sparse (i.e., have many missing values) when working with structured data and not coordinate data, so that we can then avoid working with a full matrix.

In Section 5.2, we consider the input to the k-clustering problems to consist of time series or sequences of points, that can both be interpreted as polygonal curves. Since the order of the points matters, our choice of the distance measure is the Fréchet distance, which is widely known by the "dog on a leash" analogy: assume one trajectory is that of the dog, and the other that of the owner. What is the minimum length of a leash needed to be able to always connect the two trajectories, and hence the maximum distance these two objects must have had? However, in this case a single distance computation is expensive, and hence we may not have the resources to compute all pairwise Fréchet distances. It is widely believed that for two trajectories with m vertices each no algorithms with running time $O\left(m^{2-\eta}\right)$ exist, for any constant $\eta > 0$. To improve resource efficiency, we investigate algorithms for clustering such curves by approximating them in a more compact form that allows the bounding of the distances.

Section 5.3 improves the scalability and resource efficiency of hierarchical clustering (where the common AGNES algorithm is of complexity $O(N^3)$), by aggregating the data into a tree-based summary data structure. These summaries are built in linear time and use only constant memory, and we show how these summaries can then be clustered using different distances and algorithms. The clustering process then depends on the size (bounded to a constant) of the summary storage, and given m summaries, the complexity then is $O(N + m^2)$ if we employ improved algorithms such as the nearest-neighbor chain algorithm. Because these data summaries can be built in a single pass over the dataset with constant memory, the resulting methods are well suited for streaming data processing on edge devices with limited resources. They can also be built in parallel on multiple processors and then aggregated afterwards, and are hence a good choice for aggregating big data in a cluster before continuing the analysis on a smaller system.

In Section 5.4, we change to yet another data type, namely Boolean matrix data that is then factorized. Matrix factorization is a central technique underlying many approaches ranging from spectral clustering to word embeddings (word2vec can be seen as a factorization of a word cooccurrence matrix). Boolean matrixes are a special case that indicates the presence and absence of items, such as words in documents, products in a market basket, or that indicate gene expression levels. While a matrix-based approach is relatively memory intensive to store, advances in modern hardware such as acceleration with graphics processors (GPUs) help enormously to improve the scalability of such approaches. Such processing capabilities have since become available for embedded systems in the form of GPUs in mobile CPUs (e.g., Kirin CPUs) as well as embedded tensor processing units (e.g., Google Coral Edge TPUs).

Cluster analysis is an explorative approach to data analysis, and hence is usually performed multiple times during the data analysis process. The results must not be considered an ultimate truth (a "validation" is usually not possible in a meaningful way for real data, unfortunately). But they can help to identify data properties and data problems (in particular with respect to preprocessing), and they can serve as an inspiration for further processing and analysis. For example, clusters may lead to the discovery of an appropriate classification of the data, though the individual clustering results tend to be too unreliable for a fully automatic classification, and users are better advised to label the data as desired manually after studying the clusters.

5.1 Sparse Partitioning Around Medoids

Lars Lenssen
Erich Schubert

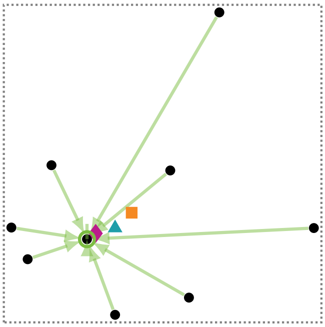
Abstract: Partitioning Around Medoids (PAM, k-medoids) is a popular clustering technique to use with arbitrary distance functions or similarities, where each cluster is represented by its most central object, called the medoid or the discrete median.In operations research, this family of problems is also known as the Facility Location Problem (FLP). FastPAM recently introduced a speedup for large k to make it applicable for larger problems, but the method still has a runtime quadratic in N. In this contribution, we discuss a *sparse and asymmetric* variant of this problem, which can be used on graph data such as road networks.

By exploiting sparsity, we can avoid the quadratic runtime and memory requirements, and make this method scalable to even larger problems, as long as we are able to build a small enough graph of sufficient connectivity to perform local optimization. Furthermore, we consider asymmetric cases, where the set of medoids is not identical to the set of points to be covered (or in the interpretation of facility location, where the possible facility locations are not identical to the consumer locations). Because of sparsity, it may be impossible to cover all points with just k-medoids for $k$values which are too small, which would render the problem unsolvable and would break common heuristics for finding a good starting condition. Hence, we consider determining k as a part of the optimization problem and propose to first construct a greedy initial solution with a larger k, then to optimize the problem by alternating between PAM-style “swap” operations where the result is improved by replacing medoids with better alternatives and “remove” operations to reduce the number of k until neither allows further improvements of the result quality.

We demonstrate the usefulness of this method on a problem from electrical engineering, with the input graph derived from cartographic data.

5.1.1 Introduction

The algorithm Partition Around Medoids (PAM, [363, 365]), also known as k-medoids, is a popular clustering algorithm used as alternative to k-means clustering when one wants to minimize other distances than squared errors distance. Similar to k-means, it aims at minimizing the sum of distances from a cluster center, but the cluster center in k-medoids is one of the data points and called a medoid, and the distance function here may be arbitrary. This increases the flexibility over k-means, which uses the arithmetic mean as the cluster center. The mean minimizes squared errors, and because of this



Euclidean (L_2), Manhattan (L_1), and squared Euclidean (L_2^2) distance sums to the different central points.

	L_2	L_2^2	L_1
■ Arithmetic Mean	6.909	**6.330**	8.674
▲ Per-axis Median	6.761	6.530	**8.267**
♦ Geometric Median	**6.712**	7.008	8.431
○ Euclidean Medoid	6.726	7.391	8.526

Fig. 5.1: Four different central points: the arithmetic mean ■, per-axis median ▲, geometric median ♦, and the Euclidean medoid ○.

k-means only minimizes Bregman divergences such as the squared Euclidean distance. Even on one-dimensional data, it does not minimize the linear error, which is easily seen from the difference between the arithmetic mean and the median. While k-means minimizes the sum-of-squared errors, k-medoids, with k representative medoids m_i minimizes the absolute error criterion (“Total Deviation”, TD):

$$\mathrm{TD} := \sum\nolimits_{i=1}^{k} \sum\nolimits_{x_c \in C_i} d(x_c, m_i) \tag{5.1}$$

where $d(x_c, m_i)$ is the distance between the data point x_c of cluster C_i and the medoid m_i; though the distance is not necessarily the Euclidean distance, and not necessarily a metric. The difference between the arithmetic mean, the per-axis median, the geometric median, and the medoid of a dataset is exemplified in Figure 5.1. It can be seen that the medoid is less sensitive to outliers than the arithmetic mean, and also that k-means does not minimize Euclidean distances (but the squared distances).

In operations research, the k-medoids problem is also known as the (discrete) facility location problem. Several variants of this problem have been researched there. The variants differ mainly in the objective function to be minimized. For example, k-center instead minimizes the maximum distance of all points to their assigned cluster centers. There has been substantial research in the area of finding approximation algorithms for all these different problems.

Unfortunately, the algorithms commonly used for k-medoids are not very scalable to large problems, as we will discuss in the next section.

5.1.2 Runtime Complexity of Partition Around Medoids

The k-medoids problem is NP-hard [357]; hence we have to resort to approximate solutions, using greedy and local optimization techniques. The PAM algorithm is such an approach: its initialization (known as `BUILD`) is a greedy approximation to the k-medoids problem, which afterwards is refined using a local search (called SWAP).

Greedy initialization chooses k times the point that reduces the error the most; local search then optimizes this solution by searching for the best way to swap one of the cluster centers with a non-center. While the name k-medoids resembles k-means, the standard PAM algorithm works differently from the standard k-means algorithm. A k-means-like strategy of alternating optimization for k-medoids has been proposed several times [299, 465, 553, 595], but was shown to produce worse solutions than a swap-based approach such as PAM [603, 616, 658]. Kanungo, Mount, Netanyahu, Piatko, Silverman, and Wu [355] proposed a swap-based approach to also improve the results of k-means, but it is rather expensive as we will see below.

Both the greedy initialization as well as the local search require that all pairwise distances be stored in a distance matrix. Greedy initialization performs k iterations, each of cost $O(N^2)$ to find the best medoid to add. PAM's swap evaluates $O(k(N-k))$ potential swaps, each with a reduced effort of $O(N-k)$ operations by computing only the change in the loss function. Hence each swap takes $O(k(N-k)^2)$ time to find, which already was an improvement over the naive approach in $O(k^2(N-k)^2)$. The resulting runtime complexity of PAM is $O(kN^2i)$, where i is the number of iterations until convergence for which little is known except that it usually is reasonably small, and likely has an unfavorably high worst case just as with k-means.

We have recently proposed improved versions of PAM named FastPAM [617] and FasterPAM [616], which provide a substantial speedup over PAM by eliminating the nested loop over the k-medoids. By greedily performing the first swap that improves the loss (instead of the best swap) and random initialization, we could decrease the runtime complexity to $O(N^2i)$ with an empirically much smaller i (but with a similar theoretical worst case).

Because both methods use each pairwise distance several times—and the method is in particular interesting to use with a more complex and hence expensive distance function—it is prohibitive to not use it with a pairwise distance matrix. Hence both methods also require $O(N^2)$ memory.

5.1.3 Sparse Partitioning Around Medoids

A large part of these pairwise distances may be unnecessary to know exactly. It is easy to see that given some assignment of points to medoids, and the maximum distance τ of this *assignment*, we could replace all values larger than τ in this *input* distance matrix with τ, and the solution would not change. Hence there is some natural "cut-off" to distances, and larger values do not contribute to the solution. If our distance function satisfies the triangle inequality, we may be able to omit computing some of these large distances (e.g., with the algorithm of Newling and Fleuret [530]).

In this research, we want to focus on a different scenario, where the cut-off may be given in advance (and may be different for each point), but the distance is not necessarily metric. An real-word example for such as problem will be introduced in Section 5.1.4.

While we can (and, effectively, will) treat distances considered uninteresting for the application as infinite (or sufficiently large) values, using a sparse storage or the distances only immediately reduces the memory usage, not the runtime. Unfortunately, this also easily breaks the optimization procedure, which relies on first finding a *feasible* initial solution, then performing local changes that *improve* the solution. A greedy strategy such as the one discussed above is usually not able to find a valid initial solution for a small k (and in particular, for a very small k the problem may become unsatisfiable with a finite loss). In such cases, the local optimization will also not help, as neighbor solutions will often still be invalid, and hence make no progress. This is most easily seen when the dataset consists of many components that are not connected with edges of finite length.

Instead of searching directly for a solution with k centers, we can solve a second problem of k-medoids clustering at the same time: how to choose k? As with k-means clustering, choosing the “optimal” k has eluded a general solution, and is mostly performed by some crude heuristic such as the infamous Elbow criterion, which is frequently misused.

If we allow the algorithm to vary k, we can much more easily find a valid initial solution (e.g., by choosing the best unconnected vertex until everything is covered). But of course this will usually yield a much higher number of clusters k than desired. But if we perform a multi-criteria optimization in the refinement phase, we may be able to reduce the number of clusters along with minimizing our main objective.

When varying k, we will obtain a Pareto front of solutions that are all optimal in one way or another. This can be formalized as solutions not “dominated” by any other solution in each criterion at the same time. To reduce the set of remaining candidate solutions, it is best if we have some additional constraints to satisfy based on the particular problem to solve.

5.1.4 Use Case: Simulation of Electrical Substation

We obtain networks using OSMOGrid, which implements ideas of distribution network generation of Kays et al. [366] on the basis of public data (OpenStreetMap, OSM). The electrical grid is modeled to follow the streets, and the buildings are used to model consumers. Power consumption is estimated based on zoning and building size, and used to simulate the load flow in the grid. We have made some graph simplifications in preparation for the problems presented below. We remove dead ends, and move the consumers locations (i.e., buildings) to the next point in the street network. Figure 5.2 shows the simulation based on the township Witten Stockum.

On the basis of this graph structure, there are different computational tasks in which resource-efficient clustering models are necessary. One of these tasks is the simulation of electrical substations within the graph. We want to identify the optimal positions of power substations, so that the electric losses in the network are minimized.

As the electric loss is related to load, voltage, and cable length, we approximate it using the distance between substations and their connected consumers, which is weighted by the consumer load. We describe this as a facility location problem, which comes from urban and public service planning. The objective function FL for facilities and demand points is

$$\mathrm{FL} = \sum\nolimits_{i\in \mathrm{Demands}} d(i, m(i)) + \sum\nolimits_{j\in \mathrm{Centers}} c(j) \quad , \tag{5.2}$$

with $c(j)$ as the cost of opening a facility, and $d(i, m(i))$ as the distance between consumer i and the assigned center $m(i)$. FL has strong similarities to the objective function of k-medoids. We take the facilities as the electrical substations and the consumers as the demand points. Figure 5.3 shows results of clustering the consumers with FasterPAM for $k = 4$ substations for the generated graph for Witten Stockum. We can observe that cluster assignment follows the road network, and consumers are not necessarily assigned to the closest center "as the bird flies".

Even with the FasterPAM improvements, the runtime complexity is $O(N^2 i)$ for N nodes in the graph and i iterations of the optimization procedure. The underlying OSM planet file contains about 1.2 TB of data. Even though we are only interested in modeling smaller areas of the world, we need to reduce complexity for solving the task for whole cities or regions in an acceptable runtime, as these will nevertheless contain several thousands of houses. We take advantage of some properties of a typical electrical network. We consider only nodes with at least 3 outgoing edges as possible optimal substation locations (except for disconnected points). The optimal position on

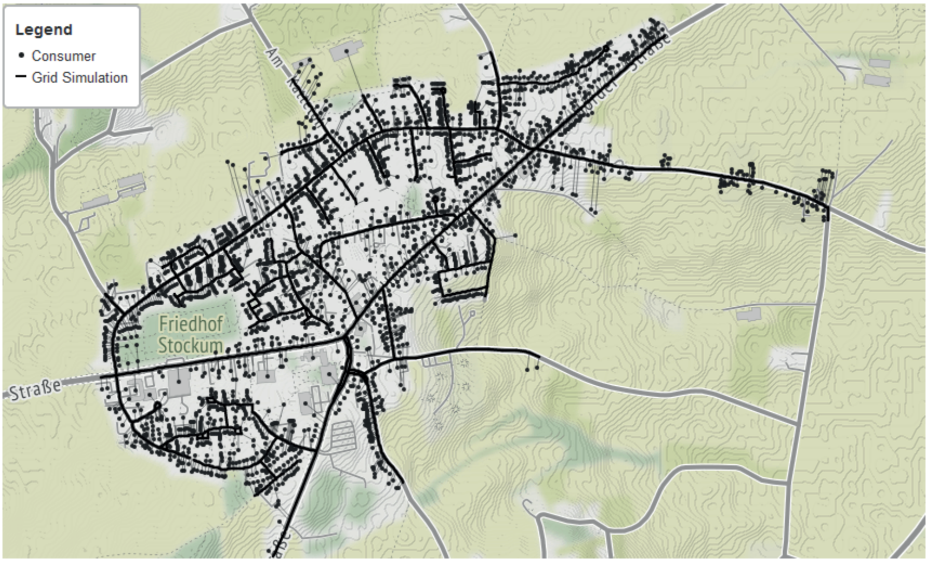


Fig. 5.2: Simulation of an electrical grid based on OSM data of Witten Stockum.

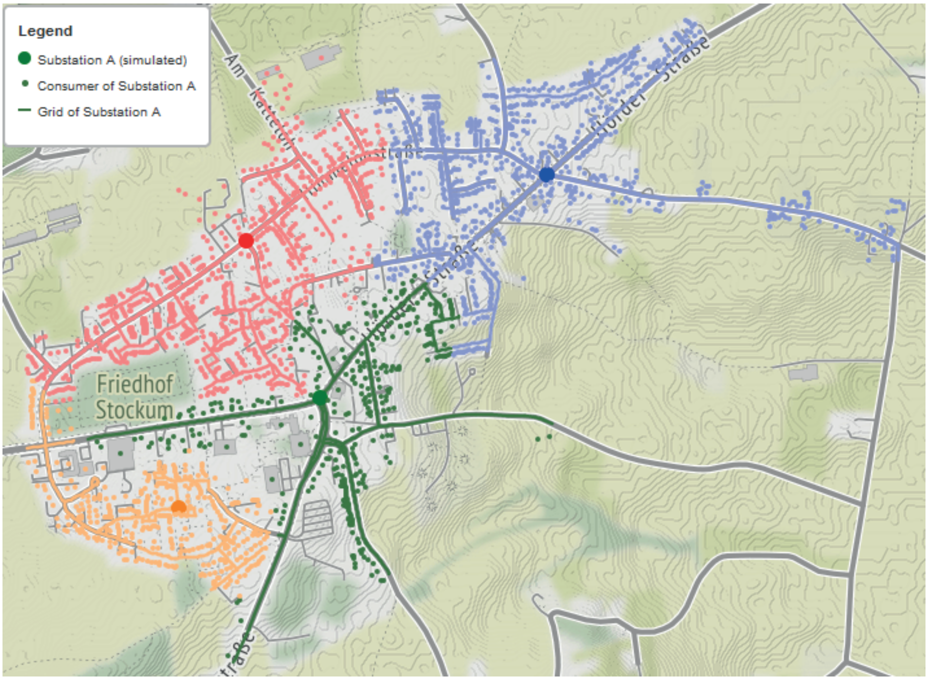


Fig. 5.3: Clustering of the demand points of the generated graph structure according to optimal substation locations with $k = 4$ using FasterPAM.

a single edge is trivial to calculate and is neglected. Hence, it is beneficial to formulate this as an asymmetric problem, where demand points and facility locations are no longer the same set. The distance matrix then no longer has to be calculated for all node pairs, but only for all demand points and substation location connections. This reduces the complexity to $O(Nmi)$ for N consumers, m possible substation locations, with $m < N$. If we further limit the maximum distance between a consumer and a substation (to limit the power losses), this distance matrix becomes *sparse*, i.e., we now have missing values that we can consider as infinite values. If we do not store these missing values and iterate using appropriate sparse data structures, we can expect to further reduce the runtime to $O((e+N+m)\cdot i)$ for e edges. Assuming a similar density of houses and roads everywhere, we can expect the number of edges e to be approximately linear in the *area* of the map we are processing.

5.1.5 Sparse *k*-Medoids

To use k-medoids clustering for problems with asymmetric and sparse input data, we have to adapt the objective function of k-medoids. We still want to minimize the "total deviation" of all data points $\{x_1, \dots, x_N\}$ from the current set of medoids $M \subseteq \{y_1, \dots, y_m\}$, but we no longer assume $M \subset X$ as in tranditional k-medoids. Furthermore, for some points, there currently may be no closest reachable medoid $m(x_i)$, and all distances from this x_i to all medoids $m_i \in M$ are undefined. In such cases, we have to incorporate a penalty $\pi(x_i)$ in our loss ℓ:

$$\ell := \sum\nolimits_{i=1}^{N} \begin{cases} \pi(i) & \text{if } m(x_i) = \text{undefined} \\ d(x_i, m(x_i)) & \text{otherwise} \end{cases} \tag{5.3}$$

Note that we allow the set M to change in size below. The penalty $\pi(i)$ can be used to trade the loss of not covering all possible data points against having larger distances. We do not further consider tuning this parameter below, but we instead use $\pi(i) = \pi = \text{const} \to \infty$ to enforce a complete coverage. Because such extreme values can cause numerical problems, our implementation always uses pairs (i, d) to store a loss (and a loss change): an integer i to count the number of unassigned points, and the sum of distances of assigned points d, such that mathematically we have $\ell = i \cdot \pi + d$, but do not suffer from numerical problems.

Based on the objective function, we introduce DynBUILD (Dynamic Asymmetric BUILD initialization) as an adaptation of the BUILD algorithm of Kaufman and Rousseeuw [363, 365] to asymmetric sparse input datasets. The greedy BUILD approach is supplemented by a dynamic increase of k, if after choosing k-medoids, some objects (still) are not reachable by the current set of medoids. The algorithm hence always choses at least k-medoids and covers all consumers. As a baseline, the strategy denoted Random simply uses a given percentage of points as initial cluster centers, and may hence yield an initial solution where constraints are violated, but our improved DynSWAP procedure will repair these while optimizing the assignment. Sparse++ is an adaptation of the well-known k-means++ [25] method to sparse data, where cluster centers are chosen in proportion to how many points they cover (again, we continue choosing additional centers until all constraints are satisfied).

We introduce DynSWAP (Dynamic SWAP for asymmetric sparse data) as a dynamic SWAP algorithm based on FasterPAM [616, 617], adapted to dynamically reduce k, while efficiently processing asymmetric and sparse input data. DynSWAP differs from FasterPAM's SWAP in two ways: To dynamically change k depending on the constraints, we check after each swap whether we can reduce k without violating a constraint (line 30) if the current object is not suitable for swapping but reduces the number of violated constraints; if added as a new medoid (line 34) then we make it an additional medoid. We deliberately chose to reduce k only if we also perform a swap, as to alternate between optimizing the existing medoids and learning the number of clusters k. Both checks are very efficient to implement, as we already know the removal loss change for all medoids ($\Delta\ell^{-m_1}, \ldots, \Delta\ell^{-m_k}$, also needed by the FastPAM improvement over PAM) and we also have in $\Delta\ell^{+}$ the loss change when adding a new medoid. We can remove the medoid m_i without breaking any constraint if the π component is zero: $\Delta\ell_{\pi}^{-m_i} = 0$, and making the current candidate y_j a new medoid is beneficial if its $\Delta\ell_{\pi}^{+} < 0$. Whenever adding, removing, or swapping a medoid, we need to update for all data points x_o the nearest medoid $n_1(o)$, the distance to the nearest medoid $d_{n_1}(o)$, and the distance to the second nearest medoid $d_{n_2}(o)$. This can be done more efficiently by updating the previous values, exactly as in FasterPAM. Based on this information, we can also update $\Delta\ell^{-m_1}, \ldots, \Delta\ell^{-m_k}$, which is the loss change for removing each medoid, efficiently: for each object, removing the nearest medoid incurs a loss change of $(0, d_{n_2}(o) - d_{n_1}(o))$ if there is a second nearest medoid, and $(\pi(o), -d_{n_1}(o))$ otherwise. Removing another medoid except the nearest medoid does not incur a loss change.

When computing the loss change for adding a new candidate medoid y_c, we initialize an array $\Delta\ell$ with the removal loss of each existing medoid, an optimization from FastPAM [617]. To avoid an inner loop over all medoids k, we also incorporate an idea from FasterPAM [616], namely to accumulate in the variable $\Delta\ell^+$ the loss change that applies to all medoids. An interesting property of $\Delta\ell^+$ is that it is the loss change for adding a new medoid, which we use for dynamically increasing the number of clusters, too. We benefit from sparsity in this approach because we do not have to consider objects that are not neighbors of the candidate y_c: the loss change by removing existing medoids has already been accounted for, and as they are not reachable from y_c, there is no loss change when adding the replacement medoid. Because of this, our loop

Algorithm 2: DynBUILD: Dynamic Asymmetric BUILD initialization

```
$\ell, M \leftarrow (\infty, \infty), \emptyset$
/* Choose the first medoid:                                          */
foreach $y_j$ do                                  // compute loss for each $y_j$
    $\ell_j \leftarrow (\sum_i \pi(i), 0)$         // everything is unassigned
    foreach $x_o \in N(y_j)$ do                    // check neighbors (sparse)
        $\ell_j \leftarrow \ell_j + (-\pi(o), d(x_o, y_j))$
    if $\ell_j < \ell$) then $\ell, M \leftarrow \ell_j, \{y_j\}$        // current best
/* Choose the remaining medoids:                                     */
for $i = 1 \ldots k-1$ do
    $\Delta\ell^\star, y^\star \leftarrow (0,0), \emptyset$      // storage for best solution
    foreach $y_j \notin M$ do
        $\Delta\ell_j \leftarrow (0,0)$                 // loss change accumulator
        foreach $x_o \in N(y_j)$ do                // check neighbors (sparse)
            $\delta\pi \leftarrow -\pi(o)$ if $d_{n_1}(o) = \infty$ else $0$
            $\delta d \leftarrow d(x_o, y_j) - d_{n_1}(o)$
            if $\delta\pi < 0$ or $\delta d < 0$ then $\Delta\ell_j \leftarrow \Delta\ell_j + (\delta\pi, \delta d)$
        if $\Delta\ell_j < \Delta\ell^\star$ then $\Delta\ell^\star, y^\star \leftarrow \Delta\ell_j, y_j$     // current best
    $\ell, M \leftarrow \ell + \Delta\ell^\star, M \cup \{y^\star\}$         // use best new medoid
    if $i = k-1$ and $\ell_\pi > 0$ then $k \leftarrow k+1$              // increase $k$
return $\ell, \{m_1, \ldots, m_k\}$
```

Algorithm 3: DynSWAP: Dynamic SWAP for asymmetric sparse data

```
$y_{\text{last}} \leftarrow$ invalid
foreach $x_o$ do compute $n_1(o)$, $d_{n_1}(o)$, $d_{n_2}(o)$
$\Delta\ell^{-m_1}, \ldots, \Delta\ell^{-m_k} \leftarrow$ compute loss change removing $m_i$
while still changing do
    foreach $y_c \notin \{m_1, \ldots, m_k\}$ do
        break outer loop if $y_c = y_{\text{last}}$          // no improvements found
        $\Delta\ell \leftarrow (\Delta\ell^{-m_1}, \ldots, \Delta\ell^{-m_k})$          // removal loss
        $\Delta\ell^+ \leftarrow 0$          // accumulator (FasterPAM)
        foreach $x_o \in N(y_c)$ do          // check neighbors (sparse)
            $d_{oc} \leftarrow d(x_o, y_c)$          // distance to candidate
            if $d_{n_1}(o) = \infty$ then          // $x_o$ not covered yet
                $\Delta\ell^+ \leftarrow \Delta\ell^+ + (-\pi(o), d_{oc})$
            else if $d_{oc} < d_{n_1}(o)$ then          // new nearest
                $\Delta\ell^+ \leftarrow \Delta\ell^+ + (0, d_{oc} - d_{n_1}(o))$
                if $d_{n_2}(o) = \infty$ then          // no second nearest
                    $\Delta\ell_{n_1(o)} \leftarrow \Delta\ell_{n_1(o)} + (-\pi(o), d_{n_1}(o))$
                else
                    $\Delta\ell_{n_1(o)} \leftarrow \Delta\ell_{n_1(o)} + (0, d_{n_1}(o) - d_{n_2}(o))$
            else if $d_{n_2}(o) = \infty$ then          // no second nearest
                $\Delta\ell_{n_1(o)} \leftarrow \Delta\ell_{n_1(o)} + (-\pi(o), d_{oc})$
            else if $d_{oc} < d_{n_2}(o)$ then          // new second nearest
                $\Delta\ell_{n_1(o)} \leftarrow \Delta\ell_{n_1(o)} + (0, d_{oc} - d_{n_2}(o))$
        $i \leftarrow \arg\min(\{\Delta\ell_i\})$          // best current medoid
        $\Delta\ell_i \leftarrow \Delta\ell_i + \Delta\ell^+$          // add accumulator
        if $\Delta\text{TD}_i < (0, 0)$ then          // eager swapping (FasterPAM)
            swap roles of medoid $m^\star$ and non-medoid $y_c$
            $\ell \leftarrow \ell + \Delta\ell_i$
            update $n_1(o)$, $d_{n_1}(o)$, $d_{n_2}(o)$, $\Delta\ell^{-m_1}, \ldots, \Delta\ell^{-m_k}$
            $y_{\text{last}} \leftarrow y_c$          // new stopping position
            // After each swap, try to reduce $k$:
            if $\min(\Delta\ell_\pi^{-m_1}, \ldots, \Delta\ell_\pi^{-m_k}) = 0$ then          // Dyn$^\downarrow$
                $r \leftarrow \arg\min(\{\Delta\ell^{-m_i}\})$
                remove medoid $m_r$
                update $n_1(o)$, $d_{n_1}(o)$, $d_{n_2}(o)$, $\Delta\ell^{-m_1}, \ldots, \Delta\ell^{-m_k}$
        else if $\Delta\ell_\pi^+ < 0$ then          // Dyn$^\uparrow$
            add new medoid $y_c$ as it fixes at least one constraint
            update $n_1(o)$, $d_{n_1}(o)$, $d_{n_2}(o)$, $\Delta\ell^{-m_1}, \ldots, \Delta\ell^{-m_k}$
            $y_{\text{last}} \leftarrow y_c$          // new stopping position
return $\ell$, $M$
```

only needs to iterate over the neighbors. For each neighbor x_o, we distinguish four cases: (1) the point is currently not yet covered, hence we gain $\pi(o)$ but incur $d(x_o, y_c)$ in line 12; (2) the new medoid is closer than all existing medoids and hence we gain $d_{n_1}(o) - d(x_o, y_c)$ in line 14. For the case of removing the nearest medoid, we have already included $d_{n_1}(o)$, and hence we have to cancel this out (either with $-\pi(o)$ or $d_{n_2}(o)$). If the new medoid is only second nearest, and there is (3) no previous second nearest, only the loss of removing the nearest medoid needs to be updated in line 20. If (4) a previous second nearest exists, but is farther than the new medoid, we also need to adjust the loss of removing the nearest medoid by the difference that arises from an assignment to the new medoid instead of to the previous second closest in line 22. Similar case distinctions—except for handling the case of an undefined second closest—can already be found in `FasterPAM` [616].

We observe that the two loops in lines 5 and 9 iterate over all edges, hence the complexity of the procedure is $O((e + N \cdot k) \cdot i)$, where e is the number of edges and i the number of iterations. In the street network example, we can argue that $e \in O(N)$ as we scale the approach to larger networks (as we would keep the maximum distance constant, but increase the area). Hence, this sparse k-medoids version scales linearly for this application. If we have a densely connected graph, then $e \in O(N^2)$ and the runtime matches that of standard FasterPAM.

5.1.6 Experiments

In our experiments, we expect to see a speedup compared with FasterPAM. We also want to check how the dynamic change of k works under consideration of constraints. We have to evaluate how well the Sparse k-medoids is able to find the smallest possible k still meeting the constraints. Hence, we analyze three initialization methods DynBUILD, Random, and Sparse++. Finally, we perform a qualitative evaluation by comparing our simulations with original substation locations from OSM.

Datasets To verify the algorithm, we need sufficiently large test datasets, and choose constraints to obtain sparse distance matrices. In this work, we focus on the processing and evaluation of energygrids generated by OSMOGrid. For quality evaluation, we choose areas where many substations are documented in OSM. Figure 5.4 shows a cutout of the electrical grid generated by OSMOGrid for the city of Witten, using the 127 substation locations from OSM (although this likely is not complete, as seen in Figure 5.4). We can then compare the quality of our calculated models to the model based on the real substations, but we need to remember that there may be additional substations missing in OSM, and that real power networks have grown historically, have to satisfy additional constraints, and are hence not optimal. For the purpose of generating “realistic” networks, it is desirable to achieve a comparable quality, without overfitting to the example data we have. On the dataset, we evaluate the dynamic

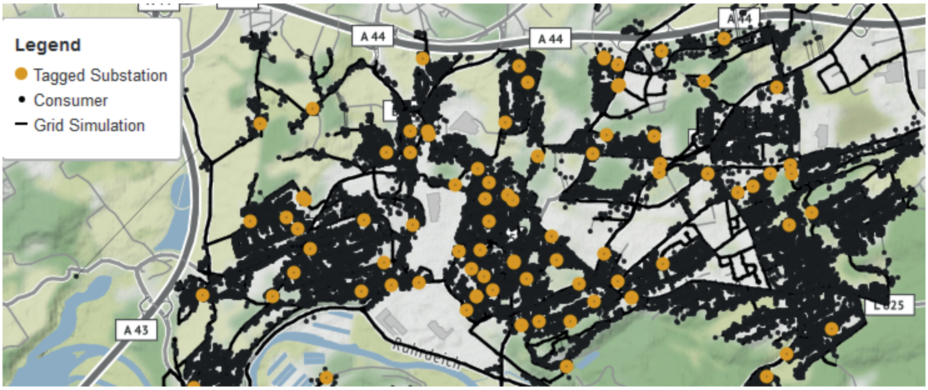


Fig. 5.4: Grid simulation and known substation locations in OSM for Witten. The road network contains 37 287 edges and 36 844 nodes, $N = 35\,713$ consumer and $m = 1130$ possible places for substations. The location of 127 substations is documented in OSM, but very likely several are missing, especially in the east.

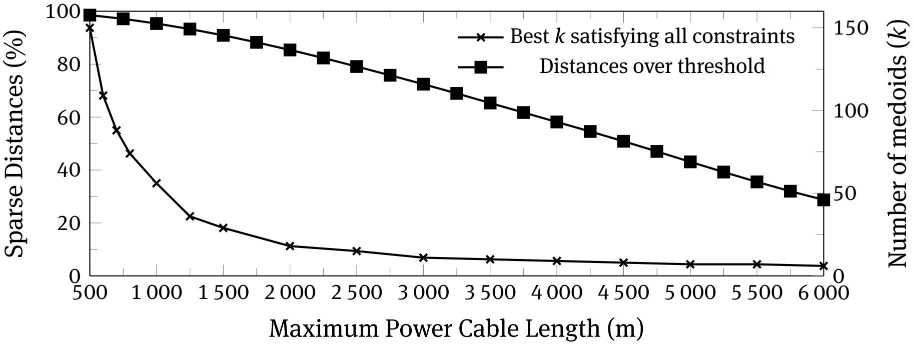


Fig. 5.5: Sparsity of the distance matrix depending on a maximum cable length constraint between consumer and substation for the simulation of Witten, and the minimum number of substations k for which no constraint is broken.

methods for choosing k. The “optimal” k depends on the constraints, and thus on the sparsity of the distance matrix. Figure 5.5 shows the smallest k to meet the constraint of a maximum cable length in the grid. With increasing cable length, the number of missing distances in the matrix decreases, but the best number of substations decreases much faster.

We evaluate the algorithms in the ELKI open-source toolkit [618] in Java. For comparability, we perform all computations in ELKI and use the original implementations of FasterPAM as a reference. This way we avoid side-effects caused by different implementations [399]. We run 100 restarts on an AMD EPYC 7302 processor using a single thread, and evaluate the average, maximum, and minimum values.

Tab. 5.1: Comparison of k depending on initialization and swap algorithms for generating a grid for Witten. All results are averaged over 100 restarts.

Algorithm		k after		Runtime in ms		Medoid	Success
Init.	SWAP	Init	SWAP	Init	SWAP	changes	
Random^{5}	Dyn$^{\downarrow}$	56	56	**0.2**	6919.2	205.3	0 %
	Dyn$^{\downarrow\uparrow}$	56	77.9	**0.2**	7862.3	287.9	100 %
Random^{10}	Dyn$^{\downarrow}$	113	82.8	**0.1**	11578.8	354.9	100 %
	Dyn$^{\downarrow\uparrow}$	113	82.5	**0.1**	11044.4	372.9	100 %
Sparse++	Dyn$^{\downarrow}$	182.3	80.3	16.3	8632.2	269.2	100 %
	Dyn$^{\downarrow\uparrow}$	182.2	80.6	3.6	8640.3	268.4	100 %
DynBUILD	Dyn$^{\downarrow}$	93	**76.3**	389.9	**4296.3**	141.9	100 %
	Dyn$^{\downarrow\uparrow}$	93	**76.2**	394.6	4320.3	141.4	100 %

Dynamic k To evaluate the quality with a variable k, we compare the solutions found by the algorithms to the best-known solution of all runs. We also compare the different initialization algorithms, and the variants of the dynamic SWAP. We measure the solution's k, the runtime of initialization and SWAP, and whether the result satisfies all constraints. Summarized results are shown in Table 5.1. Only Random^{5} initialization without dynamic increase of k fails to satisfy constraints. This was to be expected, because it only uses 5 % of the possible substations as medoids, but there does not appear to be a solution with just this many clusters. Because DynBUILD is deterministic, it always produced $k = 93$ clusters after initialization. After the SWAP phase, the average k was 76.2, which is 2.2 more than the best known $k = 74$ (we iterate in a randomized order in SWAP to avoid dependence on the input data order). Since the initial solution already satisfied all constraints, and SWAP preserves this property, k can only decrease. Among the random initializations, Random^{5} with DynSWAP$^{\downarrow\uparrow}$ found the best results on average. With $k = 77.9$ after the SWAP phase, the number of stations is on average $3.9\times$ larger than the best k. Finally, the SWAP after DynBUILD needs on average 48.4 % of the runtime of the SWAP after a random initialization. Due to random initialization, the average number of medoid changes during the SWAP increases significantly from 141 to 310, showing that DynBUILD has superior starting conditions compared with random sampling and Sparse++.

Runtime Speedup In order to evaluate the runtime of the different methods, we perform experiments for varying constraints and values of k. Figure 5.6 shows the total runtime (initialization and SWAP) for DynBUILD, Random^{5}, Random^{10}, and Sparse++ initialization. We again use the Witten dataset and choose the distance constraint such that all methods can achieve the desired k. We compare the runtime with the FasterPAM implementation with a random initialization (as recommended for FasterPAM). For DynBUILD we evaluate the SWAP with a dynamic decrease of k (Dyn$^{\downarrow}$) and for all random initialization the SWAP with both a dynamic increase and decrease of k (Dyn$^{\downarrow\uparrow}$).

We use a log scale on this plot because of the huge differences: the sparsity optimized DynSWAP over all initializations on average uses only 7 % of the runtime of the original FasterPAM with random initialization; the DynBUILD and DynSWAP$^{\downarrow}$ combination on average uses only 4 %. This was expected as FasterPAM has to process the much larger dense matrix. With increasing k, we can use a more sparse matrix here, which is the reason why the DynSWAP approaches become faster while FasterPAM becomes slower due to the higher number of clusters. The various random initializations differ only slightly in runtime, but require on average about twice as long as DynBUILD. In addition to the fast runtime, DynBUILD with Dyn$^{\downarrow}$SWAP also produces the lowest number of excess clusters compared with the best known k, with an average of 2.2 stations more than the best known $k = 74$.

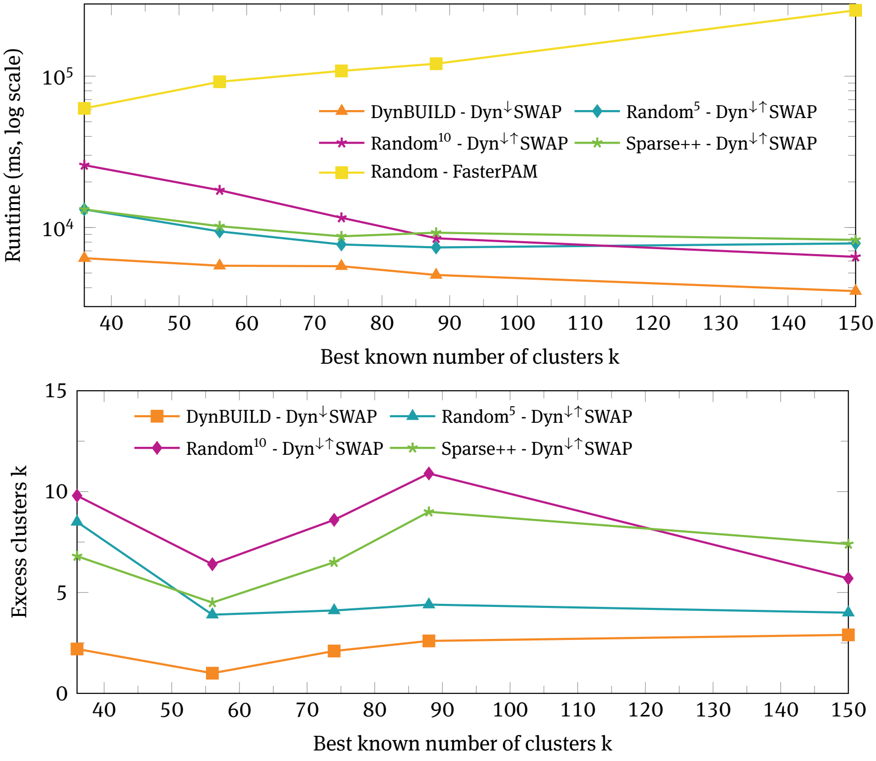


Fig. 5.6: Runtime of the initialization and SWAP for DynBUILD, Random^5, Random^{10}, and Sparse++ initialization depending on the best number of k for the simulation of the grid of Witten. For reference, the random initialization and SWAP runtime of the FasterPAM implementation is included, where the k chosen is the best one we know. The best k is controlled indirectly by the constraints set, as in Figure 5.5. In addition to the runtime, the deviation from the best known k after SWAP is also shown.

Quality In order to evaluate the resulting quality, we compare the optimized substation locations to the substations tagged in the OSM (these are likely incomplete). Table 5.2 shows the results for a target $k = 127$ compared with the loss of the 127 tagged substations, as shown in Figure 5.4. Sparse++ and Random^{10} initialization with Dyn$^{\downarrow\uparrow}$SWAP results in the lowest loss with 1.3149×10^7 and is 29 % lower than the loss of the tagged substations. The quality difference between the randomized initializations is not significant, however (the SWAP does a good enough job of always reaching

Tab. 5.2: Comparison of the loss of the grid with 127 tagged substations with the calculated loss for $k = 127$. All results are given as average values of 100 restarts. All constraints were satisfied after the SWAP phase.

Algorithm		Loss after SWAP ×10^7				Runtime in ms		Medoid changes
		avg.		min.				
Init.	SWAP	d	π	d	π	Init	SWAP	
Random^5	Dyn$^{\downarrow\uparrow}$	1.3155	**0**	**1.3083**	**0**	**0.1**	12418.6	469.6
Random^{10}	Dyn$^{\downarrow\uparrow}$	**1.3149**	**0**	1.3085	**0**	**0.1**	10395.5	339.8
Sparse++	Dyn$^{\downarrow\uparrow}$	**1.3149**	**0**	**1.3083**	**0**	16.4	10796.5	279.4
DynBUILD	Dyn$^{\downarrow}$	1.3171	**0**	1.3092	**0**	525.6	**9225.4**	294.9
Tagged substations		1.86	**0**					

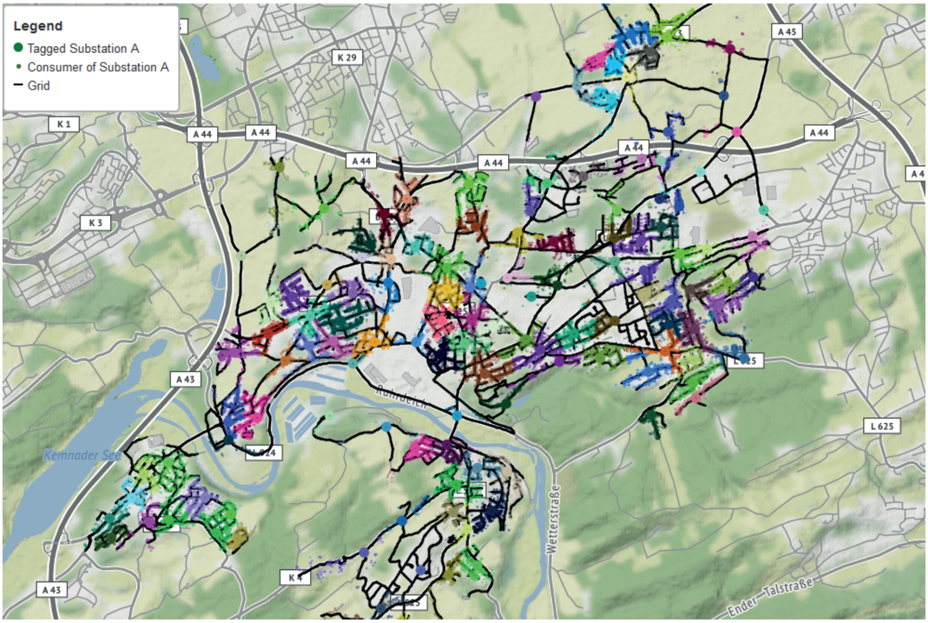


Fig. 5.7: Simulation of an electrical grid based on OSM data of Witten. 127 substations calculated with Sparse++ and Dyn$^{\downarrow\uparrow}$SWAP from Table 5.2. All consumers are color-coded to their nearest substation.

a good solution). The main difference here is in the runtimes, where the strategy of sampling more centers than necessary and then decreasing, seems superior to the others. The minimum loss over 100 restarts is obtained with Sparse++ and Random[5] with 1.3083×10^7. DynBUILD initialization with Dyn$^{\downarrow}$SWAP finds a slightly higher loss of 1.3271×10^7, but yields the fastest total runtime with 9 551.0 ms, despite using the slowest initialization by far.

All methods found significantly better solutions (1.31 vs. 1.86) than the "gold standard" solution given by the OSM tags. This had to be expected because of supposedly missing tags, but also because the real grids were grown over time and the substations were built one by one, and not automatically optimized. In our simulation, we have complete information about the grid structure and can thus calculate an optimal substation distribution (green field planning) that cannot be realistically achieved in practice, because the existing power network cannot simply be replaced and has to obey additional constraints. Nevertheless, the resulting networks can be useful for simulating power networks in different scenarios, such as when investigating the effect of significantly expanding the charging infrastructure for electric cars.

5.1.7 Outlook

In the experiments, we focused on the specific use case of energy grid simulation. Besides optimizing the FasterPAM approach for sparse problems, we have begun working on automatically finding the parameter k as part of the optimization problem. For this, we combined two losses in our loss function: one corresponding to the cost of poorly handled locations (which could also be outliers), and the other part being the classic k-medoids problem. It would be easy to incorporate an additional cost term to the opening or closing of locations, and to weigh these costs differently. In this experiment, we used a strict requirement to cover all locations (i.e., $\pi \to \infty$), but using a smaller weight may yield interesting approximations.

So far, we have considered the problem of optimal substation positions without a maximum capacity of substations. In reality, there is a maximum load that can be served by a substation. In densely populated areas therefore, we may need more substations. This results in a capacitated facility location problem that contains such an additional capacity constraint, and is worth exploring in future work.

5.2 Clustering of Polygonal Curves and Time Series

Amer Krivošija

Abstract: Sensor measurements can be represented as points in $\mathbb{R}^d$. Ordered by the time-stamps of these measurements, these points yield a time series, that can be interpreted as a polygonal curve in the d-dimensional ambient space.

The Fréchet distance is a popular dissimilarity measure for curves, in its continuous and discrete versions. These are the dissimilarity measures of choice should the inner structure of the curves be observed. One of the limitations is the inherent complexity of the computation of the Fréchet distance. It is believed that no algorithms exist to compute the Fréchet distance between two curves with m vertices each (called complexity of the curve) in the running time that is subquadratic in m.

Clustering is a fundamental computational task on curves. We consider clustering in the (metric) spaces with the Fréchet distance. The research of the k-clustering problems on curves, with the bounded complexity of the cluster centers, was started by Driemel, Krivošija, and Sohler [185], whose results are limited to the one-dimensional ambient curves. These results started a series of publications, which we survey in the first part of this section.

Related to the k-clustering is the middle curve problem [12]. Buchin, Funk, and Krivošija [98] studied the computational complexity of this problem, based on the previous work by Buchin et al. [93, 95], and showed that the middle curve problem is **NP**-complete. This result is presented in the second part of this section.

5.2.1 Introduction

Sensors and other measuring devices generate vast amount of data every day. We consider the recorded data in the order of the measurements. Such data describes trends of an event (e.g. stock market, ECG, etc.), or trajectories of some object (e.g. bird migration, routes of ships, etc.). Sensor measurements ordered by their respective time stamps define a time series. By connecting sensor measurements, that are represented as points in $\mathbb{R}^d$, in that order using the straight-line segments, we can interpret the time series as a polygonal curve in the d-dimensional ambient space.[1] In this section we consider the clustering problems on the input consisting of polygonal curves, i.e. finding one or more representative curves such that some goal function of the input's distance to the representative curves is minimized. The resource constraint we consider

[1] An ambient space is the space surrounding an object, e.g. here $\mathbb{R}^d$.

is the algorithms' running time. A curve in the Euclidean space $\mathbb{R}^d$, for $d \in \mathbb{N}$, is a continuous function[2] $\tau : [1, m] \to \mathbb{R}^d$. A polygonal curve is a curve such that there are the values $1 = t_1 \leq t_2 \leq \ldots \leq t_m = m$, with $w_i = \tau(t_i)$ that we call vertices, and such that for all $i \in \{1, \ldots, m - 1\}$ each curve segment between $\tau(t_i)$ and $\tau(t_{i+1})$ is affine, i.e. line segment. W.l.o.g. we may assume that $t_i = i$, for all $i \in \{1, \ldots, m\}$, thus for all $x \in [0, 1]$ it is $\tau(i + x) = (1 - x) \cdot \tau(i) + x \cdot \tau(i + 1)$. The line segments between two consecutive vertices w_i and w_{i+1} are called edges. We identify the curves with their images $(\tau([1, m]) \subseteq \mathbb{R}^d)$. We work only with polygonal curves, thus we simply refer to τ as a *curve*, and write $\tau = \langle w_1, \ldots, w_m \rangle$. We say that such a curve τ has complexity m.

An alternative view on curves is provided by the data mining community that analyzes the signal measurements. A time series is a series $(w_1, t_1), \ldots, (w_m, t_m)$ of measurements $w_i \in \mathbb{R}^d$ of a signal taken at times $t_i \in \mathbb{R}$. We assume $1 = t_1 < t_2 < \ldots < t_m = m$ and m is finite. A time series may be viewed as a continuous function $\tau : [1, m] \to \mathbb{R}^d$ by linearly interpolating $w_1, \ldots, w_m$ in order of t_i, $i \in \{1, \ldots, m\}$, thus being a polygonal curve in the ambient space $\mathbb{R}^d$. This notation does not specify the points of time at which the measurements are taken. This is justified by the choice of the dissimilarity measures we work with, and thus we make no distinction between the notions of time series and curves in $\mathbb{R}^d$. We denote with Δ^d the set of all polygonal curves in the ambient space $\mathbb{R}^d$, and with Δ^d_m the set of all polygonal curves in $\mathbb{R}^d$ of complexity at most m.

The choice of the dissimilarity measure on the set of the curves is very important. By using the well-known Hausdorff distance (that treats curves as sets), two curves consisting of the same measurement values would be at distance 0, even if the order of the time stamps would be completely random. A natural way to compare the curves while observing their ordered structure is using the (continuous or discrete) Fréchet distance. The Fréchet distance is the minimal cost of transforming one curve into another, where the cost measure of the transformation is the maximum distance between the mapped points along both curves. This is often illustrated in the literature by the metaphor of the shortest leash that allows a man and a dog to run along the two curves, without ever moving backward.

Formally, let $\mathcal{H}$ denote the set of continuous and monotonically increasing functions $f : [1, m'] \to [1, m'']$ with the property that $f(1) = 1$ and $f(m') = m''$. The functions in $\mathcal{H}$ are bijections. For two given functions $\sigma : [1, m'] \to \mathbb{R}^d$ and $\tau : [1, m''] \to \mathbb{R}^d$, their (continuous) Fréchet distance is defined as

$$d_F(\sigma, \tau) = \inf_{f \in \mathcal{H}} \max_{t \in [1, m']} \|\sigma(t) - \tau(f(t))\|_2, \tag{5.4}$$

The Fréchet distance between two curves is defined as the Fréchet distance of their corresponding continuous functions. Note that any $f \in \mathcal{H}$ induces a bijection between the

[2] The domain $[1, m]$ can be replaced by an arbitrary interval $[a, b]$, with $a < b$.

two curves. We refer to the function f that realizes the Fréchet distance as a matching.[3] We say that the matching witnesses the Fréchet distance between the two curves.

The continuous Fréchet distance requires a mapping of the complete domain interval. A related dissimilarity measure is the discrete Fréchet distance, which requires only a mapping between the vertices of the input curves. Let two curves σ, τ in $\mathbb{R}^d$ be given by their sequences of vertices $\sigma = \langle v_1, \ldots, v_{m'} \rangle$ and $\tau = \langle w_1, \ldots, w_{m''} \rangle$. A traversal T of σ and τ is a sequence of pairs of indices (i, j) of vertices $(v_i, w_j) \in \sigma \times \tau$ such that

i) the traversal T starts with $(1, 1)$ and ends with (m', m''), and
ii) the pair (i, j) of T can be followed only by one of $(i + 1, j)$, $(i, j + 1)$ or $(i + 1, j + 1)$.

Every traversal is monotone. If $\mathcal{T}$ is the set of all traversals T of σ and τ, then the discrete Fréchet distance between σ and τ is defined as

$$d_{dF}(\sigma, \tau) = \min_{T \in \mathcal{T}} \max_{(i,j) \in T} \|v_i - w_j\|_2. \tag{5.5}$$

We can overload the notion and say that the traversal that realizes the discrete Fréchet distance is a matching.

A related dissimilarity measure to the discrete Fréchet distance is the *Dynamic Time Warping* (DTW) distance. The cost measure of the DTW transformation between two curves is *the sum* instead of the maximum over all pairs of matched points. DTW is often used in the machine learning community. However, DTW is not a metric, while both the continuous and the discrete Fréchet distance are metric on the set Δ^d [14, 197].[4] The metric properties are useful tools for theoretical analysis of the algorithms.

When discussing the Fréchet distance of two curves σ and τ, we assume for the sake of simplicity, that both of them are of complexity m. The Fréchet distance is commonly computed using the algorithm of Alt and Godau [14] for the continuous case (in time $O\left(m^2 \log m\right)$), and the algorithm of Eiter and Mannila for the discrete case (in time $O\left(m^2\right)$). The state-of-the-art algorithms have running times roughly quadratic in m [5, 92]. It is widely believed, based on the conditional lower bounds, that no algorithms to compute either distance measure exist with running time significantly better than $O\left(m^2\right)$.

Bringmann [77] showed that, unless SETH[5] fails, there is no $O\left(m^{2-\eta}\right)$ algorithm to compute the (continuous or discrete) Fréchet distance for any $\eta > 0$, in the ambient

[3] It may be that such a matching exists in the limit only. This technicality is removed using a slight perturbation of the function. See a proof in the paper by Buchin et al. [97].

[4] The continuous Fréchet distance is a pseudo-metric, since two different functions can be at the distance 0. This can be easily repaired by observing the equivalence classes of functions, and thus we say that the Fréchet distance is also a metric.

[5] The Strong Exponential Time Hypothesis (SETH) claims that there is no $\eta > 0$, such that there is an algorithm for all k, that answers if a formula in conjunctive normal form with N variables, and whose claims have at most k literals, is satisfiable, in time $O\left((2 - \eta)^N\right)$. SETH is a fruitful tool for showing conditional lower bounds. It was used to show similar claims for the DTW distance as well.

space $\mathbb{R}^d$, $d \geq 2$. This result was extended by Bringmann and Mulzer [79] for the discrete Fréchet distance and for $d = 1$. Finally, Buchin, Ophelders and Speckmann [94] showed that even if $d = 1$, no strongly subquadratic time algorithm exists to approximate the (continuous or discrete) Fréchet distance better than the factor 3, unless SETH fails.

5.2.2 (k, ℓ)-Center and (k, ℓ)-Median Clustering

Q*: Can one find k representative curves with at most ℓ vertices each?* ***A****: It is* ***NP****-hard to do this exactly, but it can be well-approximated in time linear in the number of the input curves.*

Given are a ground set $\mathcal{X}$ equipped with dissimilarity measure $\mathbf{d}$, and two positive integers k and n. For the well-known k-clustering problems we get a set $P \subset \mathcal{X}$ with $|P| = n$ as input, and we aim to find a set $C \subset \mathcal{X}$, with $|C| = k$, such that the elements of P are assigned (clustered) to a center from C, and such that some goal function is minimized. Three most often studied problems are the k-center, the k-median, and the k-means problem, where the maximum distance, the sum of the distances, and the sum of the squares of the distances, respectively, of the input elements to the assigned centers is minimized.[6] We call these problems the k-clustering problems.

These problems are well researched, both in Euclidean and in general metric spaces. We focus on the k-center and k-median problems. Both problems are **NP**-hard, both in Euclidean and in general metric spaces [213, 483]. k-center is **NP**-hard to approximate better than a factor 2 [213]. k-median in general metric spaces cannot be approximated better than a factor $1 + 2/e \approx 1.736$, unless $\mathbf{NP} \subseteq \mathrm{DTIME}\left[n^{O(\log\log n)}\right]$ [337]. Even the discrete k-median is **NP**-hard in Euclidean space, and thus implicitly in general metric spaces [550].

In the Euclidean space $\mathbb{R}^d$ there exists a series of $(1+\varepsilon)$-approximation algorithms to the k-clustering problems. Many of these are based on the concept of coresets; that is, a coreset is a (weighted) set smaller than the input that $(1+\varepsilon)$-approximates the clustering cost of the input with respect to any choice of k centers (strong coresets), or with respect only to the optimal choice of the k-centers (weak coresets). For a survey of the coreset methods for k-clustering, see the work of Munteanu and Schwiegelshohn [516].

We address now the state of the art for k-clustering problems in general metric spaces. For the k-center problem there exists a simple greedy 2-approximation algorithm by Gonzalez [264] (and also independently given by Hochbaum and Shmoys [317]), which is also optimal. Intuitively, the algorithm picks the first center from the input at random, and then $k - 1$ times the point from the input that maximizes the distances to the already chosen centers.

[6] For each of these problems the discrete version of the problem can be observed. There the set of centers C needs to be a subset of P. In particular, the discrete k-median is known as the k-medoid problem.

For the k-median in general metric spaces, it is the discrete version that is usually studied. Note that every α-approximation to the discrete case is a 2α-approximation to the unrestricted case, due to the triangle inequality. Chen [130] gave a $(10 + \varepsilon)$-approximation algorithm with running time $O\left(nk + k^7\varepsilon^{-5}\log^5 n\right)$. The approximation factor of Chen [130] was further improved in two papers, but with a running time that is no longer linear in n. Li and Svensson [433] gave a $(1 + \sqrt{3} + \varepsilon) \approx (2.732 + \varepsilon)$-approximation in time $O\left(n^{(1/\varepsilon)^2}\right)$. Byrka et al. [116] improved the result of Li and Svensson [433] to a $(2.675 + \varepsilon)$-approximation algorithm, with the running time $O\left(n^{(1/\varepsilon)\log(1/\varepsilon)}\right)$.

An important line of research is built upon the $(1 + \varepsilon)$-approximation algorithm for the k-median problem by Kumar, Sabharwal, and Sen [405] with running time $O\left(nd \cdot 2^{(k/\varepsilon)^{O(1)}}\right)$, based on the random sampling. Their result was originally developed for the Euclidean k-median problem. Kumar et al. [405] showed that a small uniform sample of a constant number of input points, independent of n: $O\left((1/\varepsilon)^{O(1)}\right)$, is sufficient to construct a candidate set of size $O\left(2^{(1/\varepsilon)^{O(1)}}\right)$, that contains a $(1 + \varepsilon)$-approximation for the 1-median problem (and then recursively construct a $(1 + \varepsilon)$-approximation to the k-median problem). Indyk and Thorup [333, 659] showed that to approximate the discrete metric 1-median on n points within a factor of $(1 + \varepsilon)$ a uniform sample of size $O\left((1/\varepsilon^2) \cdot \log n\right)$ is sufficient. Ackermann, Blömer, and Sohler [2] showed how this argument can be adapted to the metric spaces with finite doubling dimension,[7] which includes the continuous Euclidean space ℓ_2^d.[8]

Ackermann, Blömer, and Sohler [2] showed that a $(1 + \varepsilon)$-approximation to the k-median problem in general metric spaces can be efficiently found, if a $(1 + \varepsilon)$-approximation to the 1-median problem can be found by taking a random sample of constant size, and exactly solving the 1-median problem on the sample. This result holds not only for the metric spaces with finite doubling dimension (e.g. ℓ_2^d), but also for the (not necessarily metric) spaces, whose dissimilarity measure satisfies the *sampling property*. The above results, however, do not apply directly to the spaces with d_F or d_{dF} metric, due to the unbounded doubling dimension ([185]).

Before approaching the k-clustering problems in the metric space (Δ^d, d_F) or (Δ^d, d_{dF}) we need to address the overfitting problem: even if we are looking only for a single cluster representative (center) for the input of n curves in Δ_m^d under the Fréchet distance, the optimal solution can have the complexity $O(mn)$, as noted by Ahn et al. [12]. This is not desirable considering resource-constraints, and often unnecessary for the modeling of the real-world problems. Therefore, we adapt the classical problems

[7] The doubling dimension of a metric space is the smallest positive integer d such that every ball of the metric space can be covered by 2^d balls of half the radius, cf. [281].

[8] ℓ_2^d denotes the (vector) space $\mathbb{R}^d$ equipped with the Euclidean norm $\|\cdot\|_2$.

by bounding the complexity of the clustering center curves by a constant $\ell \in \mathbb{N}$, as introduced by Driemel et al. [185].

Formally, given a set of n curves $\mathcal{W} = \{\tau_1, \ldots, \tau_n\} \subseteq \Delta_m^d$ and parameters $k, \ell \in \mathbb{N}$, $\ell \geq 2$, that we assume to be constants, we define that the (k, ℓ)-clustering problem is to find a set of k curves $\mathcal{C} = \{\varsigma_1, \ldots, \varsigma_k\}$ taken from Δ_ℓ^d that minimizes one of the following cost functions:

$$\mathrm{cost}_\infty(\mathcal{W}, \mathcal{C}) = \max_{1 \leq i \leq n} \min_{1 \leq j \leq k} d_F\left(\tau_i, \varsigma_j\right), \tag{5.6}$$

$$\mathrm{cost}_1(\mathcal{W}, \mathcal{C}) = \sum_{i=1}^{n} \min_{1 \leq j \leq k} d_F\left(\tau_i, \varsigma_j\right). \tag{5.7}$$

We refer to the clustering problem as (k, ℓ)**-center** (Equation 5.6) and (k, ℓ)**-median** (Equation 5.7), respectively.

The (k, ℓ)-clustering problems are **NP**-hard. When k is part of the input, the hardness result was shown by Driemel et al. [185] for both the (k, ℓ)-center and the (k, ℓ)-median problems, by reduction from their classical counterparts in $\mathbb{R}^d$. In this case the (k, ℓ)-center problem is **NP**-hard to be approximated better than factor 2.

When ℓ is part of the input, then there is no polynomial-time approximation scheme for the (k, ℓ)-center problem, as shown by Buchin et al. [95], who reduced the problem from the *Shortest Common Supersequence* (SCS) problem (cf. the definition of the SCS problem on page 205). The approximation factor bound depends on the dimension of the ambient space d and on whether the Fréchet distance is discrete or continuous. The lower bound factors from Buchin et al. [95] are presented in Table 5.3. These bounds hold even if $k = 1$, i.e. for the smallest enclosing ball problem. The (k, ℓ)-median problem is **NP**-hard as well, if ℓ is part of the input. This was shown by Buchin, Driemel and Struijs [93] by reduction from the SCS problem. Before Driemel et al. [185] defined the (k, ℓ)-clustering problems, there existed only approaches to find a single representative curve for a set of n input curves. As such, Buchin et al. [96] looked for a median curve using only parts of the input curves; Har-Peled and Raichel [296] defined a mean curve minimizing the distance to the input curves; and Ahn et al. [12] defined the middle curve. We discuss the middle curves more in detail in Subsection 5.2.3.

Tab. 5.3: The lower bounds for the approximation factor of an approximation algorithm for the (k, ℓ)-center problem, if ℓ is part of the input [95].

	Continuous Fréchet distance	**Discrete Fréchet distance**
$d = 1$	1.5	2
$d \geq 2$	2.25	2.598

Driemel et al. [185] gave the first $(1 + \varepsilon)$-approximation algorithms for both the (k, ℓ)-center and the (k, ℓ)-median problems under the continuous Fréchet distance in the one-dimensional ambient space. Their results are based on the curve simplifications called signatures, that capture the important vertices of the curves, while keeping the continuous Fréchet distance to the original curves small. The signatures bound the search for the candidate cluster centers for both the (k, ℓ)-center and the (k, ℓ)-median problem. The signatures' technique, albeit limited to the one-dimensional ambient space, was used recently to obtain approximation algorithms for the near neighbor problem ([78, 186]). The techniques of Driemel et al. [185] provided only constant-factor approximation algorithms for the discrete Fréchet distance case.

In the multidimensional ($d \geq 2$) ambient space, there exists a constant-factor approximation algorithm for the (k, ℓ)-center problem by Buchin et al. [95]. They adapted the algorithm of Gonzalez [264] with an approximation factor of 3 for the discrete Fréchet distance (in time[9] $\tilde{O}(mn)$), and the factors 3 and 6 for $d = 2$ and $d > 2$ respectively, for the continuous Fréchet distance (in time $\tilde{O}(mn + m^3)$). The result of Buchin et al. [95] for the discrete Fréchet distance was later improved by Buchin, Driemel, and Struijs [93] into a $(1 + \varepsilon)$-approximation algorithm with running time $\tilde{O}(mn)$, for $d \geq 1$. They also gave an exact algorithm for $d \leq 2$ with running time $\tilde{O}((mn)^{2k\ell+1})$.

For the (k, ℓ)-median problem an improvement to the result of Driemel et al. [185] was given by Buchin, Driemel, and Struijs [93]. They gave a $(1 + \varepsilon)$-approximation algorithm for $d > 1$ under the discrete Fréchet distance in time $\tilde{O}(nm^{dkl+1})$. This result was further improved into a $(1 + \varepsilon)$-approximation algorithm under discrete Fréchet distance by Nath and Taylor [525], with running time $\tilde{O}(mn)$. Their approach extends to the k-median under the Hausdorff distance.

We note that for the (k, ℓ)-median problem, Driemel et al. [185] (for the continuous Fréchet distance) adapted the sampling property of Ackermann et al. [2] to guarantee the complexity of the sampled candidate curves. Nath and Taylor [525] (for the discrete Fréchet distance) circumvented the limitations of Ackermann et al. [2] by introducing the concept of *coverability*, which generalizes the notion of doubling dimension. However, it is an open question if the coverability holds for the continuous Fréchet distance.

To find a $(1+\varepsilon)$-approximation to the (k, ℓ)-median clustering under the continuous Fréchet distance for $d > 1$ is still an open problem. However, for $d > 1$ there are recent results by Meintrup, Munteanu, and Rohde [485], and by Buchin, Driemel, and Rohde [97], that both obtain a $(1 + \varepsilon)$-approximation solution to the (k, ℓ)-median clustering under the continuous Fréchet distance, but with a caveat. The result of Meintrup, Munteanu, and Rohde [485] assumes that the number of outlier input curves is bounded, which is a natural beyond-worst-case assumption. In the worst case, however, their bound guarantees only a factor $(2 + \varepsilon)$. The result of Buchin, Driemel, and Rohde [97] has no assumptions on the input, but yields a bicriteria approximation solution with complex-

[9] The tilde-notation $\tilde{O}(X)$ hides polylogarithmic factors in X, i.e. $\tilde{O}(X) = O\left(X\text{polylog}(X)\right)$.

ity of each center curve at most $2\ell - 2$, in time linear in n and polynomial in m. The work of Buchin, Driemel, and Rohde [97] avoided problems that occur in the previous work [2, 185, 525] using the *shortcutting* curve simplification technique, related to the signatures, to guarantee the good approximate medians, but at the cost of increasing the center curves' complexity.

We summarize the best-known results for the problems we discussed in this subsection in Table 5.4.

Tab. 5.4: The best-known approximation algorithms for the (k, ℓ)-center and the (k, ℓ)-median problems. For each result the reference, the approximation factor, and the runtime are given.

	d	**Continuous Fréchet distance**			**Discrete Fréchet distance**		
(k, ℓ)-**center**	1	[185]	$1 + \varepsilon$	$\tilde{O}(mn)$	[93]	$1 + \varepsilon$	$\tilde{O}(mn)$
	2	[95]	3	$\tilde{O}(mn + m^3)$	[93]	$1 + \varepsilon$	$\tilde{O}(mn)$
	> 2	[95]	6	$\tilde{O}(mn + m^3)$	[93]	$1 + \varepsilon$	$\tilde{O}(mn)$
(k, ℓ)-**median**	1	[185]	$1 + \varepsilon$	$\tilde{O}(mn)$	[525]	$1 + \varepsilon$	$\tilde{O}(mn)$
	≥ 2	?	?	?	[525]	$1 + \varepsilon$	$\tilde{O}(mn)$
Bicriteria	≥ 2	[97]	$1 + \varepsilon$	$O\left(m^{O(1)} n\right)$			
Outliers bounded	≥ 2	[485]	$2 + \varepsilon$	$O\left(m^{O(1)} n\right)$			

For the (k, ℓ)-means problem (an analogous extension of the known k-means problem) the techniques of Driemel et al. [185] yield a constant factor approximation algorithm (under both d_F and d_{dF}), but with a runtime polynomial in n ([401]). No other results on this problem are known. For the k-clustering problem under the DTW distance the only known theoretical result is the work of Brill et al. [76], who gave an exact algorithm for 1-median in one-dimensional space, but whose running time is exponential in m.

5.2.3 Middle Curve Clustering

***Q**: Can one find one representative curve using only vertices from the input curves?* ***A**: It is **NP**-hard to do so exactly.* Given are a set of n polygonal curves $\mathcal{W} = \{\tau_1, \ldots, \tau_n\} \subseteq \Delta_m^d$, a value $\delta \geq 0$, and a dissimilarity measure $\mathbf{d}$ for polygonal curves. We use $\mathbf{d} = d_{dF}$ as in the work of Ahn et al. [12], and for the continuous Fréchet distance $\mathbf{d} = d_F$ the definitions hold verbatim. An (unordered) **middle curve** at distance δ to $\mathcal{W}$ is a curve $\mu = \langle m_1, \ldots, m_\ell \rangle$ with vertices $m_i \in \bigcup_{\tau_j \in \mathcal{W}} \bigcup_{w \in \tau_j} \{w\}$, $1 \leq i \leq \ell$, such that it holds $\max\{d_{dF}(\mu, \tau_j) : \tau_j \in \mathcal{W}\} \leq \delta$.

If the vertices of a middle curve μ respect the order given by the curves of $\mathcal{W}$, then we call μ an **ordered middle curve**. Formally, for all $1 \le j \le n$, if the vertex $m_i \in \mu$ is matched to $w_o \in \tau_j$ realizing $d_{dF}\left(\mu, \tau_j\right)$, then for the vertices $m_{i'} \in \mu$, $i < i'$, it holds that $m_{i'} \in \left(\bigcup_{\tau_x \in \mathcal{W} \setminus \{\tau_j\}} \bigcup_{w \in \tau_x} \{w\}\right) \cup \left(\bigcup \{w_{o'} : w_{o'} \in \tau_j, o' > o\}\right)$. If the vertices of μ are matched to themselves in their original curves $\tau \in \mathcal{W}$ in the matching realizing $d_{dF}(\mu, \tau) \le \delta$, we have a **restricted middle curve**. There is a hierarchy of the three middle curve notions: an ordered middle curve is simultaneously an unordered middle curve. A restricted middle curve is simultaneously an ordered middle curve.

We define the decision problems corresponding to finding such a curve. Let a set of polygonal curves $\mathcal{W} = \{\tau_1, \dots, \tau_n\}$ and a $\delta \ge 0$ be given as input. The UNORDERED MIDDLE CURVE problem returns TRUE if and only if there exists a middle curve μ at distance δ to $\mathcal{W}$. The ORDERED MIDDLE CURVE and RESTRICTED MIDDLE CURVE return TRUE if and only if there exist an ordered and a restricted middle curve, respectively, at distance δ to $\mathcal{W}$. Otherwise, the problems return FALSE.

Ahn et al. [12] presented dynamic programming algorithms for each variant of the middle curve problem (under the discrete Fréchet distance). The running times of these algorithms for $n \ge 2$ curves of complexity (at most) m are $O\left(m^n \log m\right)$ for the unordered case, $O\left(m^{2n}\right)$ for the ordered case, and $O\left(m^n \log^n m\right)$ for the restricted middle curve case. However, there are no known algorithms to compute the middle curves under the continuous Fréchet distance. Ahn et al. [12] noted that for all three variants of the problem there is a simple 2-approximation, by taking any of the input curves. This holds for both d_F and d_{dF}, due to the triangle inequality.

The exponential running times (in n) of the three algorithms by Ahn et al. [12] yield the question if there is a lower bound for these problems. We present in this subsection the proof that all three variants of the MIDDLE CURVE problem are **NP**-complete (under both d_F and d_{dF}). This hardness result was given originally by Buchin, Funk, and Krivošija [98].

The technique for the proof that all variants of the MIDDLE CURVE are **NP**-hard is based on the proof by Buchin et al. [95] and Buchin, Driemel, and Struijs [93] for the **NP**-hardness of the smallest enclosing ball and 1-median problems for curves under Fréchet distance. Their proof is a reduction from the Shortest Common Supersequence (SCS) problem, which is known to be **NP**-hard, as shown by Pietrzak [578]. The SCS problem has as input a set $\mathcal{S} = \{S_1, \dots, S_n\}$ of n sequences over a binary alphabet $\Sigma = \{A, B\}$, and $t \in \mathbb{N}$. SCS returns TRUE if and only if there exists a sequence $S^\star$ of length at most t, that is, a supersequence[10] of all sequences in $\mathcal{S}$.

Our **NP**-hardness proof differs from the proof of Buchin et al. [93, 95] in three aspects. First, the mapping of the characters of the sequence is extended by additional points. Second, in order to validate all three variants of our problem, the conditions of the restricted middle curve have to be fulfilled, i.e. each vertex has to be matched to itself

[10] A sequence S' is a supersequence of the sequence S'' if S'' is a subsequence of S'.

(in the original curves). Third, our representative curve is limited to the vertices of the input curves. We show the reductions from SCS to the RESTRICTED MIDDLE CURVE, and from UNORDERED MIDDLE CURVE to SCS. The hierarchy of the middle curves concludes the circular equivalence proof for all three variants of the problem.

Given are a set $\mathcal{S} = \{S_1, \dots, S_n\}$ of sequences over $\Sigma = \{A, B\}$, and $t \in \mathbb{N}$ defining a SCS instance that returns TRUE. We construct for each sequence $S_i \in \mathcal{S}$ a polygonal curve in $\mathbb{R}$, and thereby a MIDDLE CURVE instance. We use the following points in $\mathbb{R}$:

$$\begin{aligned} v_{-3} &= -3, & v_{-2} &= -2, & v_{-1} &= -1, & v_0 &= 0, \text{ and} \\ v_1 &= 1, & v_2 &= 2, & v_3 &= 3. \end{aligned} \tag{5.8}$$

We use the notation $\left(v_i, \dots, v_j\right)^t$ to represent the concatenation of the sequence of vertices $v_i, \dots, v_j$ that is repeated t times. Each character in a sequence $S_i \in \mathcal{S}$ is mapped to a curve in $\mathbb{R}$ as follows:

$$\begin{aligned} \eta(A) &= \left\langle v_0 (v_{-1} v_1)^t v_{-2} v_{-3} v_{-2} (v_1 v_{-1})^t v_0 \right\rangle, \\ \eta(B) &= \left\langle v_0 (v_1 v_{-1})^t v_2 v_3 v_2 (v_{-1} v_1)^t v_0 \right\rangle. \end{aligned} \tag{5.9}$$

The curve $\eta(S_i)$ representing the sequence $S_i \in \mathcal{S}$ is constructed by concatenating the curves resulting from each character's mapping. The set of all resulting curves is denoted by $\mathcal{G} = \{\eta(S_i) \colon S_i \in \mathcal{S}\}$. We call the subcurves $\langle v_{-2} v_{-3} v_{-2} \rangle$ and $\langle v_2 v_3 v_2 \rangle$ **letter** A and **letter** B **gadgets**, respectively, and the subcurves between two letter gadgets (or at the beginning and at the end of curves) consisting of v_{-1}, v_1, and v_0 **buffer gadgets**.

We define the set $\mathcal{I}_t = \{(a, b) \in \mathbb{Z}^2 \colon a, b \geq 0, a + b = t\}$. A pair $(a, b) \in \mathcal{I}_t$ represents the number of A's and B's in a possible supersequence of length t. For some $(a, b) \in \mathcal{I}_t$ we construct the curves $\zeta(A^a)$ and $\zeta(B^b)$ in $\mathbb{R}$ with

$$\begin{aligned} \zeta(A^a) &= \left\langle v_1 (v_{-3} v_1)^a \right\rangle \\ \zeta(B^b) &= \left\langle v_{-1} (v_3 v_{-1})^b \right\rangle. \end{aligned} \tag{5.10}$$

We use these curves to construct the (UNORDERED and RESTRICTED, respectively) MIDDLE CURVE instance $\left(\mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}, 1\right)$ for a pair $(a, b) \in \mathcal{I}_t$. We prove that the SCS instance $(\mathcal{S}, t)$ returns TRUE if and only if there exists a pair $(a, b) \in \mathcal{I}_t$ such that $\left(\mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}, 1\right)$ is an UNORDERED and RESTRICTED, respectively, MIDDLE CURVE instance that returns TRUE. We consider the discrete Fréchet distance case first, and then discuss the differences for the continuous case.

Lemma 19. *If $(\mathcal{S}, t)$ is a SCS instance returning* TRUE, *then there exists a pair $(a, b) \in \mathcal{I}_t$ such that $\left(\mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}, 1\right)$ is a* RESTRICTED MIDDLE CURVE *instance for the discrete Fréchet distance that returns* TRUE.

Proof. If $(\mathcal{S}, t)$ is a SCS instance returning TRUE, then there exists a supersequence of the sequences in $\mathcal{S}$ with length at most t. Let $S^\star$ be this supersequence with letters $s_i^\star$, for $i \in \{1, \dots, t\}$.

We construct a curve $\mu(S^\star) = \langle m_1, \dots, m_{2t+1}\rangle$ using vertices of the curves in $\mathcal{G}$, such that $\mu(S^\star)$ represents $S^\star$. The vertex m_j for $j \in \{1, \dots, 2t+1\}$ is defined as:

$$m_j = \begin{cases} v_0 & j \text{ is odd,} \\ v_{-2} & j \text{ is even and } s^\star_{j/2} = A, \\ v_2 & j \text{ is even and } s^\star_{j/2} = B. \end{cases}$$

The vertices with even indices in $\mu(S^\star)$ represent the characters in $S^\star$ while the vertices with odd indices act as a buffer between them. For every $S_i \in \mathcal{S}$ there is the curve $\eta(S_i) \in \mathcal{G}$. We construct a traversal between $\eta(S_i)$ and $\mu(S^\star)$, that realizes $d_{dF}\left(\eta(S_i), \mu(S^\star)\right) \le 1$. Since S_i is a subsequence of $S^\star$, we iterate over the letters of $S^\star$ and S_i, starting from the first letter, and as long as there are letters in $S^\star$ do:

If the current letter in $S^\star$ and S_i is the same, map $v_0 \in \mu(S^\star)$ to the next buffer gadget in $\eta(S_i)$ (and the possible rest of the previously unused buffer gadget). Then, map $v_2 \in \mu(S^\star)$ to the letter A gadget in $\eta(S_i)$ (or map $v_{-2} \in \mu(S^\star)$ to the letter B gadget in $\eta(S_i)$). Move to the next letter in both $S^\star$ and S_i. Note that the buffer gadget in $\eta(S_i)$ is not yet mapped.

If the current letters in $S^\star$ and S_i differ, then map $v_0 \in \mu(S^\star)$ to the possible rest of the previous buffer gadget in $\eta(S_i)$, and:

- if we have A in $S^\star$ and B in S_i, then map $v_0 \in \mu(S^\star)$ to $v_0, v_1 \in \eta(S_i)$, and $v_{-2} \in \mu(S^\star)$ to $v_{-1} \in \eta(S_i)$. Move to the next letter in $S^\star$.
- if we have B in $S^\star$ and A in S_i, then map $v_0 \in \mu(S^\star)$ to $v_0, v_{-1} \in \eta(S_i)$, and $v_2 \in \mu(S^\star)$ to $v_1 \in \eta(S_i)$. Move to the next letter in $S^\star$.

If there are no more letters in S_i, then depending on the last letter in S_i we have the following cases (and in all cases, move to the next letter in $S^\star$ afterward):

- if A is the last letter in S_i, and we have A in $S^\star$, then map $v_0 \in \mu(S^\star)$ to $v_1 \in \eta(S_i)$, and $v_{-2} \in \mu(S^\star)$ to $v_{-1} \in \eta(S_i)$.
- if A is the last letter in S_i, and we have B in $S^\star$, then map $v_0, v_{-2} \in \mu(S^\star)$ to $v_{-1} \in \eta(S_i)$, and $v_0 \in \mu(S^\star)$ to $v_1 \in \eta(S_i)$.
- if B is the last letter in S_i, and we have A in $S^\star$, then map $v_0, v_2 \in \mu(S^\star)$ to $v_1 \in \eta(S_i)$, and $v_0 \in \mu(S^\star)$ to $v_{-1} \in \eta(S_i)$.
- if B is the last letter in S_i, and we have B in $S^\star$, then map $v_0 \in \mu(S^\star)$ to $v_{-1} \in \eta(S_i)$, and $v_2 \in \mu(S^\star)$ to $v_1 \in \eta(S_i)$.

We conclude with mapping $v_0 \in \mu(S^\star)$ to the unused rest of the last buffer gadget in $\eta(S_i)$. Notice that the vertices in $\mu(S^\star)$ are mapped to themselves in $\eta(S_i)$ (in the curves they are taken from), while vertices v_0, v_2, or v_{-2} in $\mu(S^\star)$ respect the order from the original curves, thus the conditions for a restricted middle curve are met. The distance between the mapped points is at most 1, thus we have $d_{dF}\left(\eta(S_i), \mu(S^\star)\right) \le 1$, for all $S_i \in \mathcal{S}$. See Figure 5.8 for an example.

Set a and b to the number of occurrences of A and B in $S^\star$ respectively, therefore it is $a + b = t$. Per definition $\mu(S^\star)$ contains v_{-2} exactly a times, thus we can match

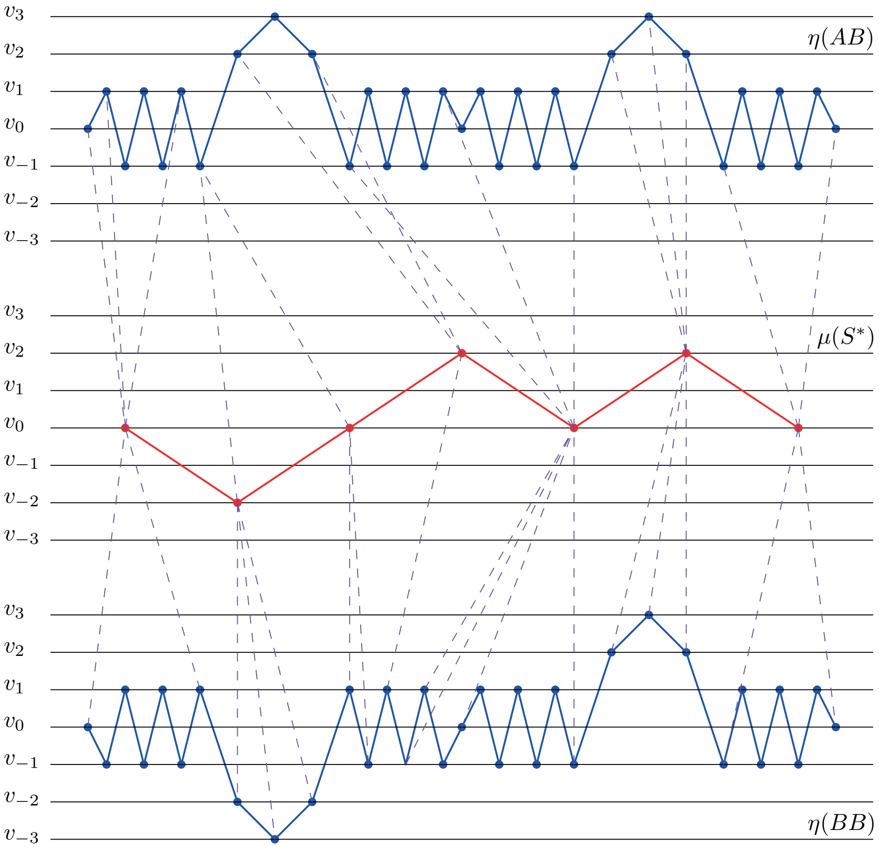


Fig. 5.8: Construction of the restricted middle curve $\mu(S^\star)$ at the distance 1, for the SCS instance $(\{AB, BB\}, 3)$, represented by the curves $\eta(AB)$ and $\eta(BB)$ (blue). The supersequence $S^\star = ABB$ is represented by the curve $\mu(S^\star) = \langle 0, -2, 0, 2, 0, 2, 0\rangle$ (red). A traversal realizing the distance is marked by dashed violet lines.

these v_{-2} to the vertices $v_{-3} \in \zeta(A^a)$, while the remaining vertices $v_0, v_2 \in \mu(S^\star)$ can be matched to the vertices $v_1 \in \zeta(A^a)$, respecting the order of the vertices on $\mu(S^\star)$ and $\zeta(A^a)$. Analogously there are b vertices $v_2 \in \mu(S^\star)$, and they can be mapped to the vertices $v_3 \in \zeta(B^b)$. The vertices $v_0, v_{-2} \in \mu(S^\star)$ can be mapped to $v_{-1} \in \zeta(B^b)$. Therefore it holds that $d_{dF}\left(\mu(S^\star), \zeta(A^a)\right) \leq 1$ and $d_{dF}\left(\mu(S^\star), \zeta(B^b)\right) \leq 1$. So $\mu(S^\star)$ is a restricted middle curve of $\mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}$ at distance 1, as claimed. □

One may ask why we need the curves $\zeta(A^a)$ and $\zeta(B^b)$ in Lemma 19. The next lemma resolves that question.

Lemma 20. *If there exists a pair $(a, b) \in \mathcal{I}_t$ such that $\left(\mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}, 1\right)$ is an* Unordered Middle Curve *instance for the discrete Fréchet distance that returns* true, *then $(\mathcal{S}, t)$ is a SCS instance that returns* true.

Proof. Given a pair $(a, b) \in \mathcal{I}_t$, let μ be an unordered middle curve of the set $\mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}$ at distance 1. We construct a sequence that represents the curve μ and prove that every $S_i \in \mathcal{S}$ is a subsequence of this sequence.

We observe a matching between μ and $\zeta(A^a)$ that realizes $d_{dF}\left(\zeta(A^a), \mu\right) \leq 1$. Since $\zeta(A^a)$ consists only of vertices v_1 and v_{-3}, and there cannot exist a point in $\mathbb{R}$ with distance of at most 1 to both of these vertices, every vertex in μ can only be matched to one vertex in $\zeta(A^a)$. Since for every two vertices v_{-3} in $\zeta(A^a)$ there is a v_1 vertex between them in $\zeta(A^a)$, a vertex in μ can be matched to at most one v_{-3} in $\zeta(A^a)$. The same holds for the vertices v_1 in $\zeta(A^a)$. Thus every vertex in μ is matched to exactly one vertex in $\zeta(A^a)$. Analogously every vertex in μ is matched to exactly one vertex in $\zeta(B^b)$.

We can partition the vertices of μ into $2a + 1$ subsets M_i^a, $i \in \{1, \ldots, 2a + 1\}$, where all vertices within one subset M_i^a are mapped to the i-th vertex in $\zeta(A^a)$ (in the matching realizing $d_{dF}\left(\zeta(A^a), \mu\right)$). Analogously we can partition the vertices of μ into $2b + 1$ subsets M_j^b, $j \in \{1, \ldots, 2b + 1\}$ (using the matching realizing $d_{dF}\left(\zeta(B^b), \mu\right)$). We combine these partitions into one. We call the subsets M_i^a that represent $v_{-3} \in \zeta(A^a)$ the A-**subsets**, and the subsets M_j^b that represent $v_3 \in \zeta(B^b)$ the B-**subsets**.

We note that there cannot exist a vertex in μ that is simultaneously in some A-subset and some B-subset, otherwise it would be at distance at most 1 to both v_3 and v_{-3}. We take over the A- and B-subsets into the new partition (and call them letter subsets). By construction there are $a + b = t$ letter subsets. The remaining vertices in μ – either before the first letter subset along μ, or after the last letter subset, or between two letter subsets form the pairwise disjunct buffer subsets, and thus together with letter subsets define a partition of the vertices of μ. There can be at most $t + 1$ buffer subsets, thus there are at most $2t + 1$ subsets in the constructed partition of the vertices of μ. Figure 5.9 shows an example of such a partition.

The sequence $S^\star$ can be constructed using the constructed partition of μ, by replacing the A-subsets with the letter A, and the B-subsets with the letter B. The buffer subsets are simply omitted. The sequence $S^\star$ has length t. We need to prove that $S^\star$ is a supersequence of all sequences in $\mathcal{S}$.

Let for some $S_i \in \mathcal{S}$ be $\eta(S_i) \in \mathcal{G}$ its representing curve. As μ is a middle curve of $\mathcal{G} \cup \{\zeta(A^i), \zeta(B^j)\}$ at distance 1, there exists a matching of $\eta(S_i)$ and μ that realizes $d_{dF}\left(\eta(S_i), \mu\right) \leq 1$. In this matching, a vertex in one A-subset (of the partition of the vertices of μ) cannot be matched to two vertices in different letter gadgets (in $\eta(S_i)$), since the buffer gadget separating two letter gadgets contains the vertex v_1, which cannot be matched to a vertex in an A-subset with distance at most 1. Analogously, a vertex in one B-subset cannot be matched to vertices in two different letter gadgets.

Each letter A gadget in $\eta(S_i)$ contains vertex v_{-3} which has to be matched to a vertex in an A-subset of the vertices of μ (otherwise by construction it would be at distance at most 1 to v_1). Analogously, each letter B gadget in $\eta(S_i)$ contains vertex v_3 which has to be matched to a vertex in a B-subset. Thus each letter gadget in $\eta(S_i)$ corresponds one-to-one to a letter subset in μ, and the sequence of letter gadgets in $\eta(S_i)$ corresponds to the sequence of letter subsets in μ. Therefore S_i is a subsequence of $S^\star$, as claimed. □

Lemma 19 and Lemma 20 show a viable reduction from the SCS problem, which is known to be **NP**-hard, to (every variant of) the Middle Curve problem. Given the SCS instance $(\mathcal{S}, t)$, the Middle Curve instance $\left(\mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}, 1\right)$ for a pair $(a, b) \in \mathcal{I}_t$ can be constructed in a time linear in the input size. As the number of possible pairs $(a, b) \in \mathcal{I}_t$ for a given supersequence of length t is linear in t, the number of different Middle Curve instances is also linear in t. Thus the reduction can be computed in a time polynomial in the input size of the SCS instance. Therefore, the following theorem holds for the discrete Fréchet distance.

Theorem 21. *Every variant of the Middle Curve problem for the discrete and the continuous Fréchet distance is* **NP***-hard.*

Like the proof of Buchin et al. [95], the shown reduction for the discrete Fréchet distance can be adapted to prove Theorem 21 for the continuous Fréchet distance, too. Lemma 22 and Lemma 23 take the place of Lemma 19 and Lemma 20, respectively. The rest of the proof is taken verbatim.

Lemma 22. *If* $(\mathcal{S}, t)$ *is an instance of the SCS that returns* true*, then there exists a pair* $(a, b) \in \mathcal{I}_t$ *such that* $\left(\mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}, 1\right)$ *is a Restricted Middle Curve instance for the continuous Fréchet distance that returns* true*.*

Proof. Given the SCS instance $(\mathcal{S}, t)$ returning true, Lemma 19 implies that there exists a pair $(a, b) \in \mathcal{I}_t$, such that $d_{dF}(\tau, \mu) \leq 1$ for all $\tau \in \mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}$, and for the restricted middle curve $\mu = \mu(S^\star)$ constructed in its proof. Since the discrete Fréchet distance is an upper bound for the continuous Fréchet distance, we have $d_F(\tau, \mu) \leq d_{dF}(\tau, \mu) \leq 1$ for all $\tau \in \mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}$. This means that μ is also a restricted middle curve for the continuous Fréchet distance. □

Lemma 23. *If there exists a pair* $(a, b) \in \mathcal{I}_t$ *such that* $\left(\mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}, 1\right)$ *is an Unordered Middle Curve instance for the continuous Fréchet distance that returns* true*, then* $(\mathcal{S}, t)$ *is a SCS instance that returns* true*.*

Proof. Given a pair $(a, b) \in \mathcal{I}_t$, let μ be an unordered middle curve of the set $\mathcal{G} \cup \{\zeta(A^a), \zeta(B^b)\}$ at distance 1. We adapt the proof of Lemma 20 to the continuous case. Since $d_F\left(\zeta(A^a), \mu\right) \leq 1$, there has to be a point q_a on the curve μ that is at distance at most 1 to the vertex $v_{-3} \in \zeta(A^a)$, for each such a vertex. Thus $q_a \in [-2, -3]$. But since

$d_F\left(\zeta(B^b), \mu\right) \le 1$, there has to be a point on $\zeta(B^b)$ at distance at most 1 to q_a, thus such a point is in $[-2, -1]$. Since all points on $\zeta(B^b)$ lie in $[-1, 3]$, it implies that that point has to be exactly at -1, thus $q_a = v_{-2}$. We call that point an A-subset of μ. It is possible that the curve μ contains several consecutive vertices at v_{-2}, and in that case the whole subcurve defined by such vertices is an A-subset of μ. Analogously, we conclude that for each $v_3 \in \zeta(B^b)$ there is a point $v_2 \in \mu$, and call it a B-subset of μ.

As in Lemma 20, we partition the curve μ into $2a + 1$ (or $2b + 1$) subcurves (subsets) M_i^a, $i \in \{1, \ldots, 2a + 1\}$ (or M_j^b, $j \in \{1, \ldots, 2b + 1\}$). If we enumerate them, the subcurves with even indices are A-subsets (resp. B-subsets) of μ, and the rest of the curve μ defines the subcurves with odd indices. Again, we combine these two partitions of μ into one, since no point on μ can be in both A- and B-subsets. The sequence $S^\star$ is constructed by replacing each letter subset in μ with the corresponding letter.

The rest of the proof of Lemma 20 follows, since for each $S_i \in \mathcal{S}$ and for the matching that realizes $d_F\left(\eta(S_i), \mu\right) \le 1$, it holds that a vertex in one A-subset in μ cannot be mapped to the vertex v_{-3} in two different letter B gadgets in $\eta(S_i)$, and each vertex $v_{-3} \in \eta(S_i)$ has to be mapped to a vertex in an A-subset. The analogous claim can be made for B-subsets. There is a one-to-one correspondence between the letter gadgets in $\eta(S_i)$ and the letter subsets in μ, thus S_i is a subsequence of $S^\star$. □

Using Theorem 21, we can now prove the **NP**-completeness of each variant of the Middle Curve decision problem. Given a Middle Curve instance $(\mathcal{P}, \delta)$ with $\mathcal{P}$ containing n curves of complexity m, we guess non-deterministically a middle curve μ of complexity ℓ. We can decide whether the Fréchet distance between μ and a curve $\tau \in \mathcal{P}$ is at most δ in time $O(m\ell)$ using the algorithm by Alt and Godau [14] for the continuous, and by Eiter and Mannila [197] for the discrete Fréchet distance. We note that the algorithm by Alt and Godau [14] has to be modified a bit, as it uses a random access machine instead of a Turing machine, as this allows the computation of square roots in constant time. But comparing the distances is possible by comparing the squares of the square roots, thus this results in a non-deterministic $O(nm\ell)$-time algorithm for the Unordered Middle Curve problem.

In order to decide the Ordered Middle Curve problem, it is necessary to compare the middle curve to the input curves, which is possible in time $O(nm)$. For the restricted Restricted Middle Curve problem the matching corresponding to the Frechet distance $\le \delta$ has to be known. This matching is a result of the decision algorithm by Alt and Godau [14]. Given this matching, it can be checked in time $O(m + \ell)$ if a vertex is matched to itself. This yields the following theorem.

Theorem 24. *Every variant of the Middle Curve problem for the discrete or continuous Fréchet distance is* **NP***-complete.*

5.2.4 Further Reading

In this subsection we reference for further reading the other theoretical clustering results that emerged from CRC876-A2, related to the topic of this section.

Driemel and Krivošija [184] investigated the relation between loss of the information on curves and saving of computing resources by embedding of Fréchet distance in the lower-dimensional spaces by random projections. The concept of the Fréchet distance can be extended to graphs and surfaces. Buchin et al. [99] gave an algorithm to compute the Fréchet distance between trees.

If the points $w_1, \dots, w_m \in \mathbb{R}^d$, that defined a curve when observed as the vertices in the order of their indices, are observed in the way that they define a discrete distribution over a finite number of locations in $\mathbb{R}^d$ where a point may appear, we have the clustering of probabilistic points. Here, the quality of the clustering centers is evaluated in expectation over the random input. In particular, for the problem of probabilistic 1-center clustering (smallest enclosing ball) the previously best algorithm was the PTAS of Munteanu et al. [218], that had time linear in the number of points, but exponential in the dimension d of the ambient space. This dependency on d was reduced to linear by Krivošija and Munteanu [402] using a novel combination of stochastic and subgradient descent techniques. This further enabled an application to the probabilistic version of the SVDD problem, with extensions to kernel spaces in even an infinitely large ambient dimension.

Related to the 1-center clustering problem, Bury, and Schwiegelshohn [102] studied the set similarity Jaccard center problem, where the input consists of a collection of sets. They showed that the problem is **NP**-hard and provided a PTAS.

If the input curves are only singleton points, then we have the Euclidean k-clustering problems in $\mathbb{R}^d$. One line of research on $(1 + \varepsilon)$-approximation algorithms to the k-median is based on the strong coresets (together with the framework of Kumar et al. [405], cf. Subsection 5.2.2). Previously the smallest known strong coresets of Feldman and Langberg [217] still were size dependent on the dimension d: their size was $O\left((dk \log k)/\varepsilon^2\right)$, which yielded the total running time for the k-median algorithm of $O\left(nd + 2^{\text{poly}(k,1/\varepsilon)}\right)$. Sohler and Woodruff [636] gave strong coresets for the Euclidean k-median of size independent of the dimension: $O\left((k^2 \log k)/\varepsilon^4\right)$. These coresets can be computed in time $\tilde{O}\left((n + d)\text{poly}(k/\varepsilon) + \exp(\text{poly}(k/\varepsilon))\right)$. After the result of Sohler and Woodruff [636] the bound on the size of the strong coresets for the k-median was further lowered, and the most recent improvement is the new framework of Cohen-Addad, Saulpic, and Schwiegelshohn [149], which is applicable for large variety of settings. For the Euclidean k-median problem the best-known coresets have size $O\left((k \log k)/\varepsilon^3\right)$, which is close to the lower bound of $\Omega(k/\varepsilon^2)$. Both results were given by Cohen-Addad et al. [148].

For the k-means problem Feldman, Schmidt, and Sohler [219] gave a method to reduce the strong coresets to a constant size independent on the dimension d of the input space. Cohen-Addad and Schwiegelshohn [150] studied classic k-median and k-means problems in the *beyond-worst-case* scenario. They gave local-search-based PTAS for the both problems for the stable input instances. Becchetti et al. [45] showed that $(1+\varepsilon)$-approximation of the cost of the k-means clustering can be obtained using a data-oblivious random projection onto roughly $\tilde{O}((\log k + \log\log n)/\varepsilon^6)$ dimensions, as well as using a data-dependent random projection onto roughly $\tilde{O}(\log k/\varepsilon^4)$ dimensions.

If the input points are not released simultaneously, but one at a time, then we have a streaming setting. For the k-median problem in the dynamic streaming scenario in the discrete Euclidean space, Braverman et al. [70] gave $O\left(dk/\varepsilon^2\right)$ space/time algorithm. All previous algorithms required space/time exponential in the dimension d. Cohen-Addad, Schwiegelshohn, and Sohler [151] investigated the diameter and the k-center problems in general metric spaces under the sliding window streaming scenario, and provided first constant-factor approximation algorithms. Fichtenberger et al. [228] designed an efficient data stream algorithm for the k-means problem that works well in practice, based on coresets and on the well-known BIRCH algorithm.

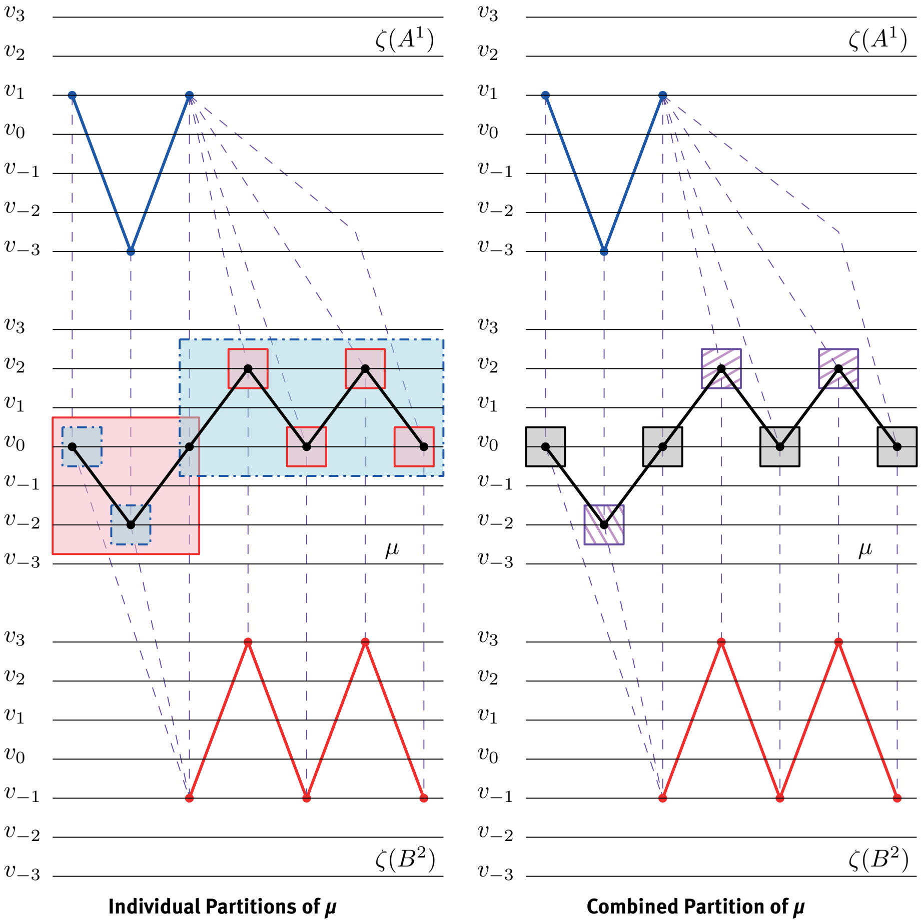


Fig. 5.9: A possible matching between the middle curve μ (black), and the curves $\zeta(A^1)$ (blue) and $\zeta(B^2)$ (red). **(a)**: The individual mappings and partition of μ based on $\zeta(A^1)$ and $\zeta(B^2)$ (blue and red boxes, respectively). **(b)**: The combined partition of μ. Letter parts in tiled violet (A - rising, B - falling tiling), buffer parts in gray.

5.3 Data Aggregation for Hierarchical Clustering

Erich Schubert
Andreas Lang

Abstract: Hierarchical Agglomerative Clustering (HAC) is likely the earliest and most flexible clustering method, because it can be used with many distances, similarities, and various linkage strategies. It is often used when the number of clusters the dataset forms is unknown and some sort of hierarchy in the data is plausible. Most algorithms for HAC operate on a full distance matrix, and therefore require quadratic memory. The standard algorithm also has cubic runtime to produce a full hierarchy. Both memory and runtime are especially problematic in the context of embedded or otherwise very resource-constrained systems. In this section, we present how data aggregation with BETULA, a numerically stable version of the well-known BIRCH data aggregation algorithm, can be used to make HAC viable on systems with constrained resources with only small losses on clustering quality, and hence allow exploratory data analysis of very large datasets.

5.3.1 Introduction

Hierarchical Agglomerative Clustering (HAC) is a popular clustering method that is especially useful if a hierarchy of clusters exists in the dataset. Initially, each data entry is seen as a cluster of one. At each hierarchy level, the two clusters with the least distance (c.f. Section 5.3.2) between them are combined until the whole dataset is in one cluster. Another commonly used name, Simple Agglomerative Hierarchical Nesting (SAHN), reflects this easy-to-understand core idea. The standard algorithm used for HAC, known as AGNES [364], requires the pairwise distances between all data points to be stored in a distance matrix, and when merging clusters, two columns and rows in this matrix are to be combined using the Lance-Williams equations [411, 412]. AGNES can be utilized with different primary distance functions, but also with different cluster distances (commonly called linkages), see Section 5.3.2. Hierarchical Agglomerative Clustering, like many other clustering methods, is a rather resource-hungry process commonly implemented using $O(N^2)$ memory and $O(N^2)$ to $O(N^3)$ time, depending on the exact algorithm implemented. One possibility to reduce the resource demands for big data or when using small embedded systems is data aggregation. The BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [737, 738] algorithm is a well-known data aggregation technique for clustering. BIRCH is a multi-step clustering algorithm that aggregates the data into a tree structure known as CF-tree before the actual clustering. We will first review some fundamentals of hierarchical clustering, and then discuss an improved version of BIRCH, called BETULA [413, 414], that avoids some numerical problems in the original BIRCH. We then show how it can be used to accelerate HAC for big data, and reduce its memory requirements.

5.3.2 Hierarchical Clustering Linkages

Because Hierarchical Agglomerative Clustering is based on the idea of always merging the two closest clusters, we need to define a suitable distance between clusters, not just between single points. Usually, we want this distance to be consistent with our distance between single points. This notion of "cluster distance" is commonly called the "linkage" criterion. The choice of linkages affect greatly how the resulting clusters look, but they also influence which algorithms can be used.

The two most widely known linkage strategies are single-link and complete-link, where the distance of two clusters is defined as the minimum or maximum distance of any two points. However, many other linkages have been proposed in literature, many as far back as the 1950s by, e.g., McQuitty [482], Sneath [634], Sokal and Sneath [638], and Wishart [707]. More recent proposals include Mini-Max [19] and medoid linkages [307, 499, 613]. Several (but not all) linkages can be expressed in terms of Lance-Williams recurrences [411, 412], which offer computational advantages. WPGMA (McQuitty) and WPGMC (median linkage) can only be defined in terms of a recurrence, and do not have a closed-form based only on the sets of points. The Lance-Williams formula is as follows:

$$d(A \cup B, C) = \alpha_A d(A, C) + \alpha_B d(B, C) + \beta d(A, B) + \gamma |d(A, C) - d(B, C)| \,. \tag{5.11}$$

Different linkage strategies can be defined in terms of the factors α_A, α_B, β, and γ as given in Table 5.5. These may depend on the sizes of the clusters A, B, and C, which we denote as n_A, n_B, and n_C. For brevity, we use the shorthand $n_{AB} := n_{A\cup B} = n_A + n_B$, and $n_{ABC} := n_{A\cup B\cup C} = n_A + n_B + n_C$. An additional—and often overlooked—detail is the initialization of the distance matrix. While single, complete, and group-average linkage work with any distance, the centroid, Ward, and median methods need to be initialized with squared distances and are closely tied to the Euclidean distance and variance. Ignoring this initialization difference (and interpretation of the output) can easily lead to incorrect results [521]. The reason becomes apparent when considering the objective function of the clustering, and the closed-form as in Equations 5.12 to 5.14. Consider single-linkage first (and, by substituting max for min, complete-linkage). Here the aim is to merge clusters A and B with the smallest distance between their points, i.e., with the smallest $d_{\text{single}}(A, C) := \min_{a\in A, c\in C} d(a, c)$. If both clusters consist of a single element, we obviously have $d_{\text{single}}(\{a\}, \{c\}) = d(a, c)$, and we can recursively compute this linkage using $d_{\text{single}}(A \cup B, C) = \min\{d_{\text{single}}(A, C), d_{\text{single}}(B, C)\}$. It is easy to see that the weights given in Table 5.5 correspond to using the minimum or maximum.

Tab. 5.5: Common linkages in terms of Lance-Williams factors

Linkage	α_A	α_B	β	γ	Init.
Single	1/2	1/2	0	−1/2	$d(a,b)$
Complete	1/2	1/2	0	1/2	$d(a,b)$
Group-average (UPGMA)	$\frac{n_A}{n_{AB}}$	$\frac{n_B}{n_{AB}}$	0	0	$d(a,b)$
McQuitty (WPGMA)	1/2	1/2	0	0	$d(a,b)$
Centroid (UPGMC)	$\frac{n_A}{n_{AB}}$	$\frac{n_B}{n_{AB}}$	$-\frac{n_A \cdot n_B}{{n_{AB}}^2}$	0	$d(a,b)^2$
Median (WPGMC)	1/2	1/2	−1/4	0	$d(a,b)^2$
Ward	$\frac{n_{AC}}{n_{ABC}}$	$\frac{n_{BC}}{n_{ABC}}$	$-\frac{n_C}{n_{ABC}}$	0	$d(a,b)^2$

Group-average linkage, also known as Unweighted Pair Group Method with Arithmetic mean (UPGMA), is another very intuitive linkage and is often considered one of the best to use in practice. The idea is to capture the average distance between elements from different clusters, i.e., $d_{\text{avg}}(A,C) := \frac{1}{n_A n_C}\sum_{a\in A}\sum_{c\in C} d(a,c)$. Clearly, for one-elemental clusters, we have $d_{\text{avg}}(\{a\},\{c\}) = d(a,c)$. The recursive computation formula is easy to derive:

$$\begin{aligned} d_{\text{avg}}(A\cup B, C) &= \frac{1}{n_{AB} n_C}\left(\sum_{a\in A}\sum_{c\in C} d(a,c) + \sum_{b\in B}\sum_{c\in C} d(b,c)\right) \\ &= \frac{1}{n_{AB} n_C}\left(n_A n_C d_{\text{avg}}(A,C) + n_B n_C d_{\text{avg}}(B,C)\right) \\ &= \frac{n_A}{n_{AB}} d_{\text{avg}}(A,C) + \frac{n_B}{n_{AB}} d_{\text{avg}}(B,C) \end{aligned} \tag{5.12}$$

The term “weighted” (going back to Sokal and Sneath [634, 638]) can be confusing: it refers to the influence each point has. In “unweighted” group average, each *object* has the same weight (and, hence, the weight of each cluster is proportional to the number of objects contained), whereas, in the “weighted” version i.e McQuitty and Median linkage, each *cluster* has the same weight (and, hence, each object in a larger cluster has a reduced weight). As easily seen in Table 5.5, both “weighted” versions correspond to their “unweighted” counterparts if we fix the cluster sizes to a constant $n_A = n_B := 1$, i.e., ignoring the cluster sizes when merging.

McQuitty’s Weighted Pair-Group Method with Arithmetic mean (WPGMA [482])can be recursively defined as $d_{\text{McQ}}(A\cup B, C) = \frac{1}{2}(d_{\text{McQ}}(A,C) + d_{\text{McQ}}(B,C))$, which introduces an unfortunate dependency on the “merge history” of the child clusters A and B. Given three objects a, b, c, merging a and b first, then with c may yield a different result than first merging one of the other pairs. A similar argument holds for median linkage (WPGMC), discussed below.

The Unweighted Pair-Group Method using Centroids (UPGMC), also known as centroid linkage, combines clusters by the distance of the cluster means $\mu_A = \frac{1}{|A|}\sum_{x\in A} x$,

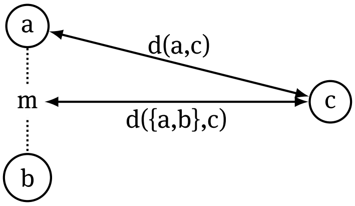


Fig. 5.10: An example showing why median and centroid linkages are non-monotone: the midpoint *m* of the merged cluster $\{a, b\}$ is closer to *c* than any of its clusters members *a* and *b* were.

i.e., it always merges the smallest $d_{\text{cent}}(A, C) = \|\mu_A - \mu_C\|$. Computing the distances between the means explicitly require many additional distance computations and hence is slower and less resource-efficient than a recurrent approach. But there is a special relationship between the mean, the variance, and squared Euclidean distance that we can exploit to compute this special case elegantly with a recurrence. We discuss this relationship, without loss of generality, only for univariate data, because squared Euclidean is simply the sum of the squared variates. We then have $\|\mu_A - \mu_C\|^2 = \mu_A^2 + \mu_C^2 - 2\mu_A\mu_C$ and obtain

$$\begin{aligned} d_{\text{cent}}(A \cup B, C) &= \tfrac{n_A}{n_{AB}} d_{\text{cent}}(A, C) + \tfrac{n_B}{n_{AB}} d_{\text{cent}}(B, C) - \tfrac{n_A n_B}{{n_{AB}}^2} d_{\text{cent}}(A, B) \\ &= \mu_{AB}^2 + \mu_C^2 - 2\mu_{AB}\mu_C = \|\mu_{AB} - \mu_C\|^2 \quad . \end{aligned} \tag{5.13}$$

This means that for squared Euclidean distances, we can compute the distance of the means without computing the means themselves. Hence, we need to initialize the distance matrix with squared Euclidean distances, and also need to interpret the resulting linkage distances as such squared values.

The idea of median linkage (or Weighted Pair-Group Method using Centroids, WPGMC) is to minimize the distance of the medians, $\|m_{A \cup B} - m_C\|$, where the median is recursively defined as $m_{A \cup B} = \frac{1}{2}(m_A + m_B)$, the midpoint of the previous medians. For squared Euclidean distances, we again have a recurrent formula: the derivation is exactly as for centroid linkage, but with fixed $n_A = n_B = 1$. Median linkage and centroid linkage have the oddity that the distance $d(A \cup B, C)$ can be less than the distance of $d(A, C)$, which can yield non-monotone dendrograms. If we draw a tree representing the cluster merges, and use the linkage distance as the height of a branch, the resulting tree does not monotonously grow. Such anomalies in the trees are also referred to as inversions, and can only exist if a linkage does not have the reducibility property of Bruynooghe [89]). Intuitively, this happens when the new center is between two well-separated clusters, and then closer to a third than either of the two, as illustrated in Figure 5.10. This can cause undesirable results, and these linkages should be used with care.

The popular Ward linkage optimizes the criterion [16, 364, 707]:

$$d_{\text{Ward}}(A, B) = \tfrac{2 n_A \cdot n_B}{n_{AB}} \left\|\mu_A - \mu_B\right\|^2 \tag{5.14}$$

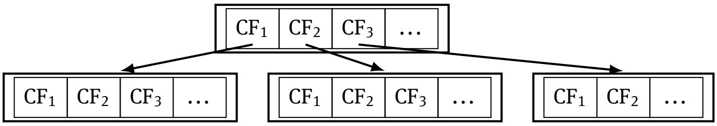


Fig. 5.11: Basic structure of a CF-Tree

The factor 2 in this equation ensures that $d_{\text{Ward}}(\{a\}, \{b\}) = \left\|a, b\right\|^2$, as desired for one-elemental clusters. This criterion can be described as the "minimum increase in the sum of squares" [582], which may come as a surprise given that the equation only uses the means, and does not appear to contain the sum of squares. The reader may have noticed that k-means clustering also minimizes the sum of squares. The main difference here is that Ward linkage imposes a hierarchical structure on the result, whereas k-means imposes a flat partitioning into k partitions. Usually, the result of the Ward linkage cut at k partitions will be (often substantially) worse than that of k-means (for the consistency reasons explained in Schubert [613] for the case of medoid linkage), but k-means results for varying k will usually not nest into a hierarchy of clusters. Equation 5.14 can be obtained from rewriting the increase in the sum of squares via the König-Huygens theorem:

$$\begin{aligned} d_{\text{Ward}}(A, B) &= \sum_{x \in A \cup B} \|x - \mu_{AB}\|^2 - \sum_{a \in A} \|a - \mu_A\|^2 - \sum_{b \in B} \|b - \mu_B\|^2 \\ &= \tfrac{2 n_A n_B}{n_{AB}} \|\mu_A - \mu_B\|^2 \end{aligned}$$

The Lance-Williams recurrence given in Tab. 5.5 follows (full derivation omitted):

$$\begin{aligned} d_{\text{Ward}}(A \cup B, C) &= \tfrac{2 n_{AB} n_C}{n_{ABC}} \|\mu_{AB} - \mu_C\|^2 = \tfrac{2 n_{AB} n_C}{n_{ABC}} \left\| \tfrac{n_A}{n_{AB}} \mu_A + \tfrac{n_B}{n_{AB}} \mu_B - \mu_C \right\|^2 \\ &= \tfrac{n_{AC}}{n_{ABC}} d_{\text{Ward}}(A, C) + \tfrac{n_{BC}}{n_{ABC}} d_{\text{Ward}}(B, C) - \tfrac{n_C}{n_{ABC}} d_{\text{Ward}}(A, B) \end{aligned}$$

5.3.3 The Cluster Feature Tree (CF-Tree)

We now briefly introduce the CF-Tree of the improved BETULA version [413, 414], which improves the numerical accuracy of the original BIRCH CF-Tree [737, 738].

The CF-Tree (Cluster Feature Tree) is a basic height-balanced tree storing cluster features (CF). Each BETULA cluster feature [414] is a triple

$$\text{CF} := (n, \mu, \text{SSE}) \tag{5.15}$$

where n in this context is the number of data points or their aggregated weight, μ denotes the mean vector, and SSE is the sum of squared deviations from the mean. Two

BETULA cluster features can be efficiently combined into one:

$$n_{AB} = n_A + n_B \tag{5.16}$$

$$\mu_{AB} = \mu_A + \tfrac{n_B}{n_{AB}}(\mu_B - \mu_A) \tag{5.17}$$

$$\mathrm{SSE}_{AB} = \mathrm{SSE}_A + \mathrm{SSE}_B + n_B(\mu_B - \mu_A)(\mu_B - \mu_{AB})\ . \tag{5.18}$$

A single data point x can be trivially represented by a Cluster Feature $(1, x, 0)$. The rules also follow from the König-Huygens theorem and can be found in Lang and Schubert [413]. The numerical inaccuracies of the original BIRCH approach were previously observed by Schubert and Gertz [614].

The CF-Tree is a height-balanced tree: each leaf is a cluster feature that represents data point(s). Inner nodes store the aggregated information of their children. The tree is built by sequentially inserting all data points. When adding a data point or cluster feature to the tree, it is inserted by traversing the tree and choosing the least distant node on each level. When a leaf entry is reached the data is added to the leaf entry if the absorption threshold (c.f. Section 5.3.4) is not violated. If the data cannot be added to an existing leaf entry, a new leaf entry is generated. The threshold can be set based on expert input which results in a tree of variable size but with a fixed accuracy guarantee. But because we can also add cluster features to the CF-Tree the same way, we can dynamically rebuild the tree from its leaf entries with an increased threshold once a selected maximum number of leaf entries is reached, to reduce the tree's memory usage. In this case, the tree is built within a fixed size range but with variable accuracy, which is beneficial for scenarios where we have memory resource constraints.

5.3.4 Distances for Cluster Features

Zhang et al. [737, 738] originally proposed several distance functions and absorption criteria for BIRCH cluster features. Both essentially measure a distance, but distance functions are used to choose insertion sub-trees, whereas absorption criteria are used to decide when to add to an existing node, or when to create a new node. As suggested by Lang and Schubert [414], we do not distinguish between distances and absorption criteria in the following, as there is no benefit to doing so.

Euclidean distance:

$$\mathrm{D0}(A, B) = \left\|\mu_A - \mu_B\right\| \tag{5.19}$$

Manhattan distance:

$$\mathrm{D1}(A, B) = \left\|\mu_A - \mu_B\right\|_1 \tag{5.20}$$

Inter-cluster distance:

$$\mathrm{D2}(A, B) = \sqrt{\tfrac{1}{n_A n_B} \sum_{x \in A} \sum_{y \in B} \left\|x - y\right\|^2} \tag{5.21}$$

Tab. 5.6: Linkage strategy for (squared) Euclidean distances and the corresponding BIRCH distance with their objective function.

Linkage	Closed form	BIRCH distance
UPGMA	$\frac{1}{n_A n_B} \sum_{x\in A} \sum_{y\in B} \|x-y\|^2$	D2^2
UPGMC	$\|\mu_A - \mu_B\|$	D0^2
Ward	$\frac{2n_A n_B}{n_{AB}} \|\mu_A - \mu_B\|^2$	$2 \cdot$ D4^2

Intra-cluster distance (= diameter absorption criterion):

$$\mathrm{D3}(A,B) = \sqrt{\tfrac{1}{n_{AB}(n_{AB}-1)} \sum_{x,y\in AB} \left\|x-y\right\|^2} \tag{5.22}$$

Variance-increase distance:

$$\mathrm{D4}(A,B) = \sqrt{\sum_{x\in AB} \left\|x-\mu_{AB}\right\|^2 - \sum_{x\in A} \left\|x-\mu_A\right\|^2 - \sum_{x\in B} \left\|x-\mu_B\right\|^2} \tag{5.23}$$

Radius absorption criterion:

$$\mathrm{R}(A,B) = \sqrt{\tfrac{1}{n_{AB}} \sum_{x\in AB} \left\|x-\mu_{AB}\right\|^2} \tag{5.24}$$

These distances can be computed efficiently based on the summary statistics stored in BETULA cluster features. The corresponding equations and their derivations can be found in Lang and Schubert [413].

5.3.5 Hierarchical Clustering with Cluster Features

While the CF-Tree itself is a form of hierarchical clustering, its levels, and inner structure are not in a form that is easily interpretable. Because of this, it is usually only used in data aggregation as preparation for the actual clustering, for which only the leaf entries are used. Naively, one could just use the centers of the leaf entries and use a standard hierarchical clustering algorithm. This approach discards the variance information of the clustering features.

The interesting observation now is that linkages and CF distances are not very different. We show that there is a correspondence between certain linkages and CF distances that can be exploited for clustering by incorporating additional information stored in the cluster features besides using only the centers. In Table 5.6 we summarize the identified relationships between linkage strategies known from literature and BIRCH distances with their respective object function. The most obvious similarity can be seen when looking at the Centroid-Euclidean-Distance (D0, Equation 5.19) and the Centroid-linkage (Equation 5.13), which are almost the same. The differences between Ward-linkage (Equation 5.14) and the Variance-increase-distance (D4, Equation 5.23) are only in the notation and that D4 squared is Ward, but since BETULA internally uses squared distances for computational reasons, this difference is trivial. The last linkage that can be expressed as a BETULA distance is UPGMA, which is effectively the squared Intercluster-distance (D2, Equation 5.21). This similarity becomes obvious when replacing the general equation with the one for UPGMA with the squared Euclidean distance:

$$d_{\text{UPGMA}}(A,B) = \tfrac{1}{n_A \cdot n_B} \textstyle\sum_{a \in A} \sum_{b \in B} d(a,b) \tag{5.25}$$

$$= \tfrac{1}{n_A \cdot n_B} \textstyle\sum_{a \in A} \sum_{b \in B} \left\| a - b \right\|^2 . \tag{5.26}$$

While WPGMA cannot have an exact match, we may nevertheless choose D2 and D0 as their respective "unweighted" counterparts because of their close relationship. With this knowledge, we can now meaningfully transition from cluster features into hierarchical clustering with the Lance-Williams formula by calculating the distance matrix based on the corresponding distances between the cluster features.

We can also do the opposite, and instead of using the classic linkage strategies, we can perform the following adaptation to hierarchical clustering of cluster features, while using the distance functions from Section 5.3.4 instead of a separate linkage strategy. As in standard hierarchical clustering (e.g., AGNES), we find the smallest non-diagonal value in the distance matrix to find the best next merge. But instead of combining distances using the Lance-William equation, we can instead combine the corresponding two cluster features using the update Equations 5.16 to 5.18, and compute new distances with respect to the new CF.

For both cases (Lance-Williams and CF distances), we can use the approach of Anderberg [16] and NN-chain [520] for acceleration. While the first does not improve the worst-case complexity of $O(|\text{CF}|^3)$, it typically performs closer to quadratic in runtime. The second may yield different results for non-reducible distances (c.f. [89], Centroid and Median linkage), but guarantees $O(|\text{CF}|^2)$ runtime; furthermore, it can be implemented with only linear memory for some linkages. As the CF-Tree allows us to reduce the data to a constant size less than $O(\sqrt{N})$ or $O(\sqrt[3]{N})$ (as applicable) cluster features, we can then perform hierarchical clustering in time linear in the original data input size N and within a constant memory limit, making this useful for resource-limited data processing.

5.3.6 Experiments

We evaluate hierarchical clustering with and without BETULA cluster features. We are interested in comparing the runtime and quality of aggregated and non-aggregated algorithms but do not compare different linkage strategies. As baselines, we use the Anderberg [16] and NN-Chain [520] algorithms (the latter in an implementation that only uses linear memory). For BETULA we allow a maximum of 25 000 leaf entries, such that no data aggregation takes place for the smallest datasets. Both of these HAC algorithms can be combined with BETULA in different ways. We use "full data" when not using BETULA aggregation; "CF centers" denotes the naive approach using the Euclidean distances of the cluster features centers and no weights (found in many implementations

Tab. 5.7: Average runtime in seconds and number of cluster features after BETULA initialization for different algorithms and data generators with $N = 50\,000$ and $d = 5$ dimensions.

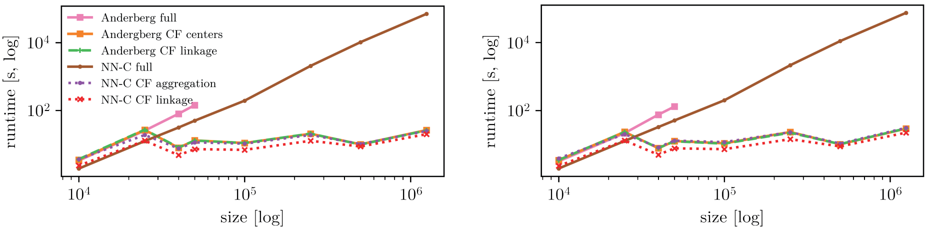
Algorithm	Input	Uniform		Gaussian	
		Runtime	|CF|	Runtime	|CF|
Anderberg	full data	142.96	-	147.05	-
Anderberg	CF centers	12.99	17482.4	4.88	9496.7
Anderberg	CF linkage	12.85	17482.4	4.71	9496.7
NN-Chain	full data	49.64	-	49.44	-
NN-Chain	CF aggregation	11.39	17482.4	4.36	9496.7
NN-Chain	CF linkage	7.18	17482.4	3.15	9496.7

of BIRCH). For "CF linkage", the initial distances are computed using the full cluster feature information, but afterward, the algorithm uses the Lance-Williams equations for hierarchical aggregation. The "CF aggregation" approach maintains cluster features throughout the hierarchical clustering process.

All algorithms are implemented in the Java framework ELKI [618]. By using the same framework for all implementations we try to minimize the effects caused by implementation differences, as recommended for comparing algorithms [399]. Each experiment was repeated 10 times with varying input orders on a single core of an AMD EPYC™ 7302 CPU. Because of our focus on improving the scalability, we may rely on synthetic data for this experiment. We sample data from both a 5-dimensional uniformly distributed hypercube and from a combination of 500 5-dimensional Gaussian clusters (as applicable). While the uniform distribution is supposed to adversely affect the aggregation quality of BETULA, the Gaussian clusters are well-suited for this type of aggregation.

First, we look at the runtime analysis with 50 000 data points, the biggest dataset the baseline Anderberg implementation can process.[11] As Table 5.7 shows, the NN-Chain algorithm can be significantly faster than the Anderberg algorithm (at least for this low-dimensional dataset). While we still largely limit the data aggregation of BETULA (set to a maximum of 25 000 leaves), the number of CFs obviously is the main contributor to the runtime, as seen when comparing the results on uniform data with those on Gaussians. The design of BETULA does not allow for the exact control of this number, but when the given maximum is reached a smaller tree is built from the current leaves. By choosing a smaller limit, an even larger speedup over the baseline algorithms would be possible. Because the data aggregation performed by BETULA is deterministic, the same input leads to the same tree, and hence the number of CF in Table5.7 is independent of the clustering step used afterward. Next, we look at the

[11] This is because the array size reaches the 2^{31} array length limit of Java.



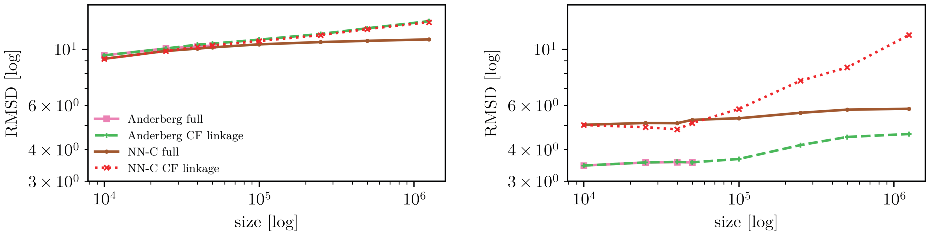
(a) UPGMC **(b)** Ward

Fig. 5.12: Runtime versus dataset size on uniformly distributed data.

Tab. 5.8: Root mean squared deviation for different linkages, algorithms, and datasets with $N = 50\,000$. All values are given as mean value plus minus standard deviation over 10 runs.

Algorithm	Input	Generation	UPGMC	UPGMA	Ward
Anderberg	full data	Uniform	10.34 ± 0.00	10.11 ± 0.00	10.12 ± 0.00
Anderberg	CF linkage	Uniform	10.53 ± 0.03	10.29 ± 0.02	10.27 ± 0.01
NN-Chain	full data	Uniform	10.17 ± 0.04	-	10.17 ± 0.04
NN-Chain	CF linkage	Uniform	10.36 ± 0.03	-	10.35 ± 0.02
Anderberg	full data	Gaussian	3.56 ± 0.00	3.51 ± 0.00	3.38 ± 0.00
Anderberg	CF linkage	Gaussian	3.56 ± 0.00	3.53 ± 0.03	3.40 ± 0.01
NN-Chain	full data	Gaussian	5.25 ± 0.05	-	5.25 ± 0.05
NN-Chain	CF linkage	Gaussian	5.09 ± 0.05	-	4.95 ± 0.08

scalability of our approach. Figure 5.12 shows the runtime of the algorithms for centroid (UPGMC) and Ward linkage on various dataset sizes in a log-log plot. We can see the quadratic increase in the runtime of the baseline NN-Chain and Anderberg algorithms. For the Anderberg baseline, only times up to 50 000 points are given because of Java array size restrictions, but scaling would be at least as bad as for the NN-Chain algorithm. The runtime for all variants that use BETULA for data aggregation seems to fluctuate around some constant value. This is caused by changes in the number of tree leaf CFs. Because the results are averaged over multiple permutations of the dataset, tree sizes and tree rebuilds are not constant for a particular dataset size. Even for big dataset sizes, the CF-tree construction phase, which has a runtime in $O(N)$, plays a minor role compared with the later hierarchical clustering phase and its $O(|\mathrm{CF}|^2)$ runtime; reading the input data once is unavoidable in most applications. The quality of a hierarchical clustering is hard to evaluate properly because it very much depends on the dataset and application. A thorough evaluation of a clustering on real data will usually require manual inspection by a domain expert. For our experiments, we chose to simply compare the variability of the clusters when cut into 500 clusters, assuming that a result with less spread also indicates a better clustering.



(a) Uniform

(b) Gaussian

Fig. 5.13: Root mean squared deviation versus dataset size using centroid (UPGMC) linkage for both dataset generators and $k = 500$.

Table 5.8 shows the root mean squared deviation (RMSD) for all relevant algorithms for the datasets with 50 000 data points. Here, the runtime improvements with BETULA were significant but the difference in quality for all algorithms is very small for the uniform dataset (within the variability caused by NN-Chain using a processing order different from Anderberg). The results of the evaluation on the Gaussian data warrants further discussion. On this dataset, which is favorable to the assumptions of BETULA, the negative effect of the data aggregation when combined with Anderberg is even smaller. There is no measurable difference for UPGMC and only slightly worse results for UPGMA and WARD. By contrast, the NN-Chain algorithm suffers from its known differences to the Anderberg algorithm (making greedy locally optimal choices, as opposed to choosing the global optimum).

Finally, we evaluate the scalability of our approach. Figure 5.13 shows the root mean squared deviation of the $k = 500$ clusters. For $N = 25\,000$, where no aggregation takes place, the results are the same, with and without BETULA. The results for the datasets with more entries are similar. When the number of cluster features used stays below 25 000, the quality only is impacted slightly. On the uniform data, the difference in quality between the algorithms is small. The Gaussian data shows that the difference between the NN-Chain and Anderberg algorithms is bigger than that of the data aggregation with BETULA. The only outlier is the combination of BETULA and NN-Chain, which shows a noticeably worse result.

5.3.7 Conclusion

In this section, we discussed how the scalability of hierarchical clustering can be improved by integrating data aggregation techniques from BIRCH (or its more stable variant BETULA). We show how hierarchical linkages relate to particular BIRCH distance criteria, and that some criteria improve the clustering for the same metric. We use this relation to accelerate the hierarchical clustering with small effects on the quality of the clustering while keeping most benefits of hierarchical approaches and expanding it to dataset sizes not practical for the standard approaches. This optimization allows the usage of hierarchical clustering on small or embedded systems with limited memory by using data aggregation to decouple the total data size from the input data size of the much more expensive hierarchical clustering step, leading to better scalability. While there is some loss in clustering quality, it is small enough for most use cases of explorative data analysis, i.e., we will still be able to make meaningful choices for the subsequent steps in our data analysis process.

5.4 Matrix Factorization with Binary Constraints

Sibylle Hess

Abstract: A natural strategy for dealing with big data is to compress it. Compression can be used as a preprocessing step, as known from dimensionality reduction tasks, or it can be used to identify underlying patterns in the data that extract the core information. Both learning tasks can be formulated as a matrix factorization. Here, we discuss those matrix factorizations that impose binary constraints on at least one of the factor matrices. Such factorizations are particularly relevant in the field of clustering, where the data is summarized by a set of groups, called clusters. Unfortunately, the optimization methods that are able to integrate binary constraints mostly work under one condition: *exclusivity*. For clustering applications this entails that every observation belongs to exactly one cluster, which is inept for many applications.

We propose a versatile optimization method for matrix factorizations with binary constraints without requiring additional constraints, such as exclusivity. Our method is based on the theory of proximal gradient descent and supports the use of GPUs. We show that our approach is suitable to discover meaningful clusters even in the prevalence of a high level of noise by means of synthetic and real-world data.

5.4.1 Introduction

In the field of clustering, and more generally in the field of data mining, one of the most relevant challenges is the optimization which is subject to binary constraints. In particular with respect to resource efficiency, binary constraints gain relevance due to the decreased storage requirements of binary models. Yet they also help to make data mining results interpretable. Is a picture showing a cat? Should a movie be recommended to this user? Binary results provide definite answers to the questions arising when solving data mining tasks.

Many methods are able to solve binary-constrained problems. However, they mostly work under one condition: *exclusivity*. Under this condition, we assume that if a picture shows a cat, then it cannot show a dog, or if a movie is assigned to one cluster (e.g., a genre), then it cannot belong to another cluster (i.e., to another genre). From these examples, we can easily observe that the exclusivity assumption does not always make sense. For example, a movie generally belongs to more than one cluster, e.g., a genre. The effect that exclusivity is unrealistic is most often observable for applications in high-dimensional data. The clustering of high-dimensional data requires a simultaneous feature selection to circumvent the *curse of dimensionality*, stating that all instances are approximately equally similar to each other in high dimensions. This introduces the task of *biclustering*, a simultaneous clustering of rows and columns, such as movies and users, features and observations. Users within a bicluster give similar ratings for the movies in the bicluster. Yet, a science-fiction fan (usually) does not exclusively like science-fiction movies. In this respect, the exclusivity assumption is clearly imposing stringent, unrealistic constraints.

Similar observations can be drawn in other biclustering applications. For instance, the biclustering of gene-expression data is employed to identify groups of genes and patients, that are strongly linked by similar expression levels. Such an analysis can discover functions of genes related to clinical traits. However, one gene generally does not have a single function in an organism, but is actually involved in multiple biological processes [580]. Turning it the other way around, not every gene necessarily plays a significant role in the considered conditions. In this case, the exclusivity assumption would force every gene to belong to one cluster. Hence, *outliers*, or *isolated objects*, could be improperly modeled in the presence of the exclusivity assumption.

A popular way to circumvent the difficulties of binary optimization is to either apply greedy heuristics to the combinatorial binary problem, or to relax the binary constraint into a nonnegative and/or orthogonal one (cf. Section 5.4.3). However, the heuristics can not guarantee theoretical properties and relaxed *fuzzy* clusters need to be post-processed into binary results, at which point theoretical guarantees are lost.

In this contribution we discuss two methods and propose a theoretically founded optimization of biclustering methods as part of the collaborative research center CRC 876. The first method, PAL-Tiling [309, 310, 311], optimizes a Boolean matrix factorization, indicating a clustering of binary data that particularly allows for the overlap of clusters and the modeling of outliers. The second method Broccoli (Binary RObust Co-Clustering Optimization through alternating LInearized minimization) [312] optimizes a biclustering of real-valued data to obtain models that can handle cluster overlap and the presence of outliers. Both methods employ a penalization approach, where a relaxed objective is optimized while the violation of binary constraints is penalized.

We highlight synthetic and qualitative experiments of the proposed methods, showing that both methods are able to detect biclusterings of various structures and are robust to the noise in the data and other parameters. The qualitative inspection reveals that both methods are able to derive meaningful clusters, which are interpretable by their modular structure.

5.4.2 Matrix Factorization – the Mother of Clustering

Even researchers who are not directly involved in matrix factorization probably know of two prominent instances: Singular Value Decomposition (SVD) and k-means clustering. SVD decomposes a given data matrix $D \in \mathbb{R}^{N \times d}$, gathering N observations of d features, into the product of three matrices $D = U\Sigma V^\top$. The matrices $U \in \mathbb{R}^{N \times N}$ and $V \in \mathbb{R}^{d \times d}$ are orthogonal, which means that they are invertible and the inverse is given by the transposed matrix. The matrix $\Sigma \in \mathbb{R}^{N\times d}$ is a rectangular diagonal matrix, having the singular values in decreasing order on the diagonal $\sigma_1 = \Sigma_{11} \geq \sigma_2 = \Sigma_{22} \geq \ldots \geq 0$. The singular values indicate the *importance* of the directions indicated in U and V. To see this, we write the SVD as a weighted sum of the outer products of columns in U and V (we denote column s of U or V with $U_{\cdot s}$ or $V_{\cdot s}$). Let $k = \min\{N, d\}$, then we have

$$D = \sigma_1 U_{\cdot 1} V_{\cdot 1}^\top + \ldots + \sigma_k U_{\cdot k} V_{\cdot k}^\top .$$

The columns of the orthogonal matrices U and V have all a norm of one. Hence, the singular values indicate the significance of every outer product $U_{\cdot s} V_{\cdot s}^\top$ for the approximation of D. If a singular value is equal to zero, then the corresponding outer product is not relevant for the representation of D. Likewise, singular values close to zero indicate expendable outer products. This opens up the possibility of compressing the matrix by a low-rank product, as with truncated SVD.

Truncated SVD computes for a given rank $r < \min\{N, d\}$ an approximation of the matrix D by solving the following optimization problem:

$$\min_{X,Y} \|D - YX^\top\|^2 \qquad \text{s.t. } Y \in \mathbb{R}^{N\times r}, X \in \mathbb{R}^{d\times r}. \tag{5.27}$$

The solution to this optimization problem is given by truncating the SVD to the first r columns of the factor matrices: $YX^\top = U_{\cdot\mathcal{R}} \Sigma_{\mathcal{R}\mathcal{R}} V_{\cdot\mathcal{R}}^\top$[12], where $\mathcal{R} = \{1, \ldots, r\}$. The truncated decomposition reflects the most important components of the data.

For example, if the data matrix is mean-centered (that is, the mean of all observations in D is equal to zero), then the columns of V indicate the principal components of the data and the squared singular values relate to the variances of the data in the direction of the principal components. Here, the low-dimensional projection of the data onto the principal components, given by $U_{\cdot\mathcal{R}} \Sigma_{\mathcal{R}\mathcal{R}}$, is often used as a dimensionality reduction technique.

The main drawback of SVD is its interpretability. In principle, the matrix X of Equation 5.27 denotes a pattern in the data. The degree with which an observation $D_{j\cdot}$ exhibits pattern $X_{\cdot s}$ is denoted by the value Y_{js}. Yet the mixture of positive and negative values in Y and X makes the interpretation of patterns and their occurrence difficult. This is why constraints on the matrices X and Y have been introduced.

One such constraint is the limitation to nonnegative factor matrices in Nonnegative Matrix Factorization (NMF). NMF was originally introduced by Paatero and Tapper [548] under the name positive matrix factorization. It gained attention since the publication from Lee and Seung [423], who showed that the nonnegative constraints and the resulting parts-based explanation of the data empower the interpretability of the results. The drawback of NMF is that the constraint to nonnegative values makes the polynomially solvable objective of truncated SVD NP-hard [682]. In particular, the amount of local minima increases with the introduction of nonnegative constraints. As a result, the optimization of NMF plays an important role in the quality of the obtained result.

[12] The matrix $U_{\cdot\mathcal{R}}$ contains all columns of U whose index is in the set $\mathcal{R}$.

Tab. 5.9: Overview of matrix factorization objectives for popular clustering and biclustering models. The matrix D is the data matrix and K is a positive semi-definite quadratic matrix, (e.g., the kernel matrix, or in the case of spectral clustering, the negative graph-Laplacian). We denote with $X^\dagger$ the Moore-Penrose inverse of X.

	Clustering objective
k-means	$\min_{X,Y} \lVert D - YX^\top \rVert^2 \quad \text{s.t.} \quad X \in \mathbb{R}^{d\times r}, Y \in \mathbb{1}^{N\times r}$
Kernel k-means (& Spectral Clustering)	$\min_{X,Y} \lVert U - YX^\top \rVert^2$ $\text{s.t.} \quad X \in \mathbb{R}^{d\times r}, Y \in \mathbb{1}^{N\times r}, K = UU^\top, U \in \mathbb{R}^{N\times k}$
Checkerboard	$\min_{X,C,Y} \lVert D - YCX^\top \rVert^2$ $\text{s.t.} \quad X \in \mathbb{1}^{d\times u}, Y \in \mathbb{1}^{N\times r}, C \in \mathbb{R}^{r\times u}$
Plaid	$\min_{X,Y} \lVert D - YY^\dagger D - DXX^\dagger + YCX^\top \rVert^2$ $\text{s.t.} \quad X \in \mathbb{1}^{d\times u}, Y \in \mathbb{1}^{N\times r}, C = Y^\dagger DX$
Block-Diagonal	$\min_{X,C,Y} \lVert D - YCX^\top \rVert^2$ $\text{s.t.} \quad X \in \mathbb{1}^{d\times r}, Y \in \mathbb{1}^{N\times r}, C = \operatorname{diag}(C_{11}, \dots, C_{rr})$
Binary	$\min_{X,Y} \lVert D - YX^\top \rVert^2 \quad \text{s.t.} \quad X \in \{0,1\}^{d\times r}, Y \in \{0,1\}^{N\times r}$
Boolean	$\min_{X,Y} \lVert D - Y \odot X^\top \rVert^2 \quad \text{s.t.} \quad X \in \{0,1\}^{d\times r}, Y \in \{0,1\}^{N\times r}$

The Relationship to Clustering NMF is considered to be fuzzy clustering. The matrix X denotes the patterns of feature values of the data and Y_{js} indicates the degree with which pattern $X_{\cdot s}$ belongs to observation $D_{j\cdot}$. Yet filtering the most important information from a fuzzy clustering still requires post-processing of the result. For example, to extract the observations that belong to the cluster with index s, a threshold has to be specified that defines how large a fuzzy cluster indicator Y_{js} has to be to indicate cluster membership. This post-processing step is alleviated if we introduce binary constraints to the matrix factorization objective.

Table 5.9 summarizes the matrix factorization objectives that define the correspondingly denoted clustering task. We see that the factor matrix Y is often constrained to be in the set $\mathbb{1}^{N\times r}$. This set denotes all partition indicator matrices

$$\mathbb{1}^{N\times r} = \{Y \in \{0,1\}^{N\times r} \mid |Y_{j\cdot}| = 1, 1 \leq j \leq N\},$$

where $|Y_{j\cdot}|$ denotes the L_1-norm of the j^{th} row of Y. A clustering indicated by the matrix $Y \in \mathbb{1}^{N\times r}$ assigns every observation $D_{j\cdot}$ to exactly one cluster: the cluster with index s for which $Y_{js} = 1$. We say that the clustering implements the *exclusivity constraint* in this case.

We see in Table 5.9 that many popular clustering methods are instances of matrix factorization with binary constraints. In particular, the popular k-means clustering computes a matrix factorization into the cluster assignment matrix Y, and the centroids, indicated by the columns of X. Also nonconvex clustering methods such as kernel k-means and spectral clustering are instances of matrix factorization. Here, the objective is to compute a k-means factorization on the factor U of a symmetric decomposition of the kernel matrix (or the graph Laplacian) $K = UU^\top$. This formulation of spectral clustering in terms of the objective of k-means closes a long-standing gap, which explains why the application of k-means on the eigendecomposition of the graph-Laplacian is so successful [308].

The models of Checkerboard, Block-Diagonal, and Plaid clustering are biclustering models. Likewise, Binary and Boolean matrix factorization compute biclusterings, which are even more suitable for binary data $D \in \{0, 1\}^{N\times d}$. A biclustering computes a simultaneous clustering of features and observations. Especially for high-dimensional data, where all observations tend to be equally similar, biclustering is applied. The idea of biclustering is that for every cluster of observations, a group of features is identified, such that the points cluster in the subspace spanned by the selected features.

A variant of the binary matrix factorization, where clusters are explicitly allowed to overlap, is the /indexMatrix factoization!BooleanBoolean matrix factorization. Here, the matrix multiplication is computed in Boolean algebra, yielding $1 \oplus 1 = 1$. Note that in binary matrix factorization the area where two biclusters overlap is approximated by $1 + 1 = 2$, and the area where three biclusters overlap is approximated by $1 + 1 + 1 = 3$, and so on. Hence, the overlap of binary biclusters introduces an approximation error to the binary data matrix. In Boolean algebra, this is not the case and we always obtain a binary matrix as the result of a Boolean product of binary matrices.

5.4.3 Reviewing Optimization for Constrained Matrix Factorizations

By and large, there are three main ways to handle the computational challenging task of clustering. If the exclusivity assumption is applied, then the objective can be optimized with the alternating minimization scheme known from Lloyd's k-means algorithm. If the exclusivity assumption is inept, as is often the case in biclustering applications, then the optimization usually relies on a relaxation of the binary constraints. This approach has the problem that a crisp cluster assignment has to be inferred in a postprocessing step. However, optimality guarantees are lost after this step. The third possibility is to apply (usually greedy) heuristics, which search for the optimizers in the binary space. However, the problem of heuristics is their lack of theoretical foundation. Usually, there are no guarantees over the found solution.

In the following, we review these approaches before we propose our optimization scheme, which is based on a relaxed objective with a penalization of non-binary values.

Alternating Minimization The exclusivity assumption enables an efficient alternating minimization that follows the scheme of *the* k-means algorithm [444]. In every iteration, one of the factor matrices is optimized while the other factor matrices are fixed. Here, the exclusivity assumption enables the analytical derivation of the optimizer in every iteration [243, 494]. That is, we do not need to apply gradient descent in

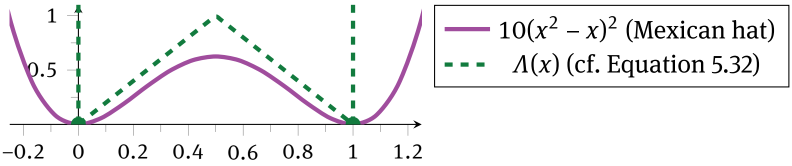


Fig. 5.14: Binary penalization functions: the Mexican hat function and Λ.

every optimization step, but we can directly state the optimum for one of the matrices. This facilitates an optimization subject to binary constraints. This optimization scheme has been implemented for checkerboard biclustering [139, 475, 694] and for diagonal biclustering [293, 640]. Koyutürk and Grama [391] and Li [434] propose alternating minimization schemes for binary matrix factorization, restricting one of the factor matrices to the exclusivity assumption. In this scenario, row-clusters are always nonoverlapping, but column-clusters may overlap, or vice versa. Alternating minimization is an elegant and theoretically founded optimization method, but its feasibility is restricted to clusterings with the exclusivity assumption.

Approaches Based on a Relaxation If we do not want to apply the exclusivity assumption, and want to reflect outliers or the overlap of clusters, then we cannot make use of the alternating minimization method in a computationally efficient way based on k-means (or at least we don't know how). In this case, an often employed strategy is to relax the binary constraints such that numerical optimization approaches can be applied. Most often, the binary constraints are relaxed to nonnegative constraints with hard or soft orthogonality constraints of the factor matrix columns [165, 176, 178, 536, 725, 733]. The more strongly orthogonality is enforced, the more the resulting fuzzy clustering resembles a clustering with the exclusivity assumption. In the most extreme case, the factor matrix columns are orthogonal and a binary cluster assignment is indicated by every nonzero entry. By contrast, if we allow for more fuzzy cluster indicators, then we allow for overlap in clusters, but a discretization of the fuzzy clusters is nontrivial.

One of the few attempts to solve the task for binary matrices without making use of the exclusivity assumption is the penalization approach proposed for binary biclustering [740, 741, 742]. So far, multiplicative updates have been used to minimize the approximation error together with a penalization term for non-binary values, called the Mexican hat function (cf. Figure 5.14).

The optimization of the relaxation-based approaches with multiplicative updates can be seen as a gradient descent method, where the step-size is chosen small enough such that the constraints are not violated. This results unfortunately in a slow convergence rate and a particularly strong sensitivity to initialization.

Heuristics Most heuristics follow a greedy approach, where the clusters are added one by one and the next cluster is selected in a greedy manner. Greedy methods are not necessarily heuristics. For example, for truncated SVD, the calculation of the optimal rank-r factorization is reducible to calculating an optimal rank-one factorization, given the optimal rank-$(r-1)$ factorization. However, this property does not hold if nonnegativity or binary constraints are introduced. Still, the greedy optimization scheme might lead to satisfying solutions if the optimal rank-one factorization is much more easily computed than the factorization of a higher rank.

There are approaches relying on a greedy heuristic for plaid [135, 419, 674], binary [391], and Boolean matrix factorizations [244, 491]. The drawback of the greedy approach is the lack of quality guarantees, where comparable numerical optimization methods at least assure convergence to a local minimum of the objective.

5.4.4 A Novel Proximal Gradient Descent Method to Optimize Matrix Factorizations Subject to Binary Constraints

The objective of matrix factorization is nonconvex. This entails that there are multiple local optima that are typically not all suited to reflect a good clustering structure. Moreover, binary constraints on the matrices make this issue even more evident: every binary matrix induces a local optimum. Indeed, every binary matrix is the only feasible (and, therefore, the best) optimizer within its ϵ-ball neighborhood for small enough ϵ. In addition, if other factorization matrices are allowed to have continuous values, as in k-means or checkerboard biclustering, the optimization of the continuous matrix can lead to a significant decrease in the approximation error, even if the binary cluster assignment matrix is far away from the global optimum. This phenomenon makes it hard to distinguish between local optima and the global optimum by means of the objective function value (i.e., by observing the approximation error). In other words, having a *good* optimizer is generally not enough: we need a *very good* optimizer that simultaneously *i)* handles the existence of many local optima that are almost indistinguishable from the global optimum by observing only the objective function, *ii)* integrates binary constraints, and *iii)* is robust to noise and can handle the presence of outliers.

Proximal Gradient Descent Bolte, Sabach, and Teboulle [60] extend optimization results known for convex optimization to the nonconvex case with the *Proximal Alternating Linearized Minimization* (PALM). This technique focuses on objectives breaking down into a smooth part F and a possibly nonsmooth component ϕ

$$\min_{X,Y} F(X,Y) + \phi_X(X) + \phi_Y(Y) \qquad \text{s.t. } X \in \mathbb{R}^{d\times r}, Y \in \mathbb{R}^{N\times r}. \tag{5.28}$$

We assume for now that $F(X,Y) = \left\|D - YX^\top\right\|^2$ returns the approximation error in the Frobenius norm. The nonsmooth part ϕ may return ∞, which can be used to model restrictions of the search space, e.g., the nonnegativity constraint of NMF. PALM performs alternating *proximal mappings* from the gradient descent update with respect to F. That is, the following steps are repeated for $t \in \{1, \ldots\}$:

$$X_{t+1} = \operatorname{prox}_{\alpha_x \phi_x}(X_t - \alpha_x \nabla_X F(X_t, Y_t)); \tag{5.29}$$

$$Y_{t+1} = \operatorname{prox}_{\alpha_y \phi_y}(Y_t - \alpha_y \nabla_Y F(X_{t+1}, Y_t)). \tag{5.30}$$

The proximal mapping of a function ϕ returns a matrix satisfying the following minimization criterion:

$$\operatorname{prox}_\phi(X) \in \arg\min_{X^\star} \left\{ \frac{1}{2} \|X - X^\star\|^2 + \phi(X^\star) \right\}. \tag{5.31}$$

Loosely speaking, the proximal mapping gives its argument a little push in a direction that minimizes ϕ. For a detailed discussion, see, e.g., [552]. As we can see in Equations 5.29 and 5.30, the evaluation of this operator is a base operation. Finding the minimum of the proximal mapping in every iteration by numerical methods is infeasible in practice. Thus, the trick is to use only simple functions ϕ for which the proximal mapping can be calculated in a closed form.

The PALM optimization scheme provides furthermore a step-size strategy that guarantees convergence to a local minimum [60]. The step-sizes are here given by the inverse of the Lipschitz constants of $F(X, Y)$.

Penalizing Nonbinary Values Binary constraints on matrices are incorporated into a relaxed objective by a penalizing term. We employ here the penalizing function

$$\Lambda(x) = \begin{cases} -|1 - 2x| + 1 & x \in [0, 1] \\ \infty & \text{otherwise.} \end{cases} \tag{5.32}$$

Function Λ is shown in Figure 5.14; it achieves its maximum value 1.0 at 0.5, its minimum value 0.0 at binary values, and returns infinity outside of the interval $[0, 1]$. Further, we define that the function Λ applied to a matrix X returns the matrix $\Lambda(X) = (\Lambda(X_{is}))$ of the same dimensionality.

The function Λ is non-smooth, but feasible for optimization by proximal gradient descent. Hess, Morik, and Piatkowski [310] have shown that the proximal operator for Λ satisfies for $x \in \mathbb{R}$

$$\operatorname{prox}_{\lambda\Lambda}(x) = \begin{cases} \max\{0, x - 2\lambda\} & x \le 0.5, \\ \min\{1, x + 2\lambda\} & x > 0.5. \end{cases} \tag{5.33}$$

The parameter $\lambda > 0$ is the regularization weight. The larger λ is, the more the value x is pushed into the direction of binary values.

5.4.5 PAL-TILING – Optimizing Boolean Matrix Factorizations

A possible reason for the prevailing usage of heuristics in Boolean matrix factorization is the reasonable belief that relaxations to nonnegative or other continuous values are not apt to approximate a product in Boolean arithmetic. Contrary to this belief, we argue for the opposite: a nonnegative relaxation is particularly suited to derive overlapping clusterings and is therefore also suited to approximate Boolean matrix factorizations, whose main characteristic is to allow for overlap between the clusters. Now we need to be a bit careful with the word *approximate*. The Boolean Matrix Factorization (BMF) problem is NP-hard and NP-hard to approximate within a constant factor [491]. Hence, we will not be able to produce an efficient algorithm that comes arbitrarily close to the optimal Boolean solution (unless NP=P). Yet, we are able to find good local Boolean optima in a relaxed space.

First of all, we can compute the Boolean matrix product in elementary algebra with the use of the thresholding function

$$\theta_\rho(x) = \begin{cases} 1 & \text{if } x \le \rho \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \theta(x) = \theta_{0.5}(x).$$

We define the function $\boldsymbol{\theta}(X) = (\theta(X_{is}))_{is}$ to map an input matrix to a binary matrix of the same dimensionality. The property $1 \oplus 1 = 1$ is maintained because the thresholding function θ maps every value larger than one to one. Hence, the Boolean matrix product $Y \oplus X^\top = \boldsymbol{\theta}(YX^\top)$ is computable in elementary algebra with a thresholding operation.

We demonstrate the relationship between relaxed matrix factorizations and the Boolean product in Figure 5.15. The binary data matrix D has two overlapping biclusters, and is approximated by NMF as shown in the top two equations. The matrices D_A and D_B in Figure 5.15 show the resulting binary and Boolean approximations, where θ maps every value larger or equal than one half to one. We find that the reconstruction error is largest when the thresholded NMF factors are multiplied in elementary algebra, corresponding to a binary matrix factorization. In contrast, the fuzzy cluster indication by NMF is suited to indicate a definite clustering with respect to the Boolean algebra.

In conclusion, we propose a two-step procedure. In the first step, the relaxed, but nonbinary penalized objective is optimized by PALM. In the second step, the approximately binary factor matrices are rounded to binary values, such that the Boolean product is minimized.

Algorithm Specification 1 (PAL-Tiling). *Given a data matrix $D \in \{0,1\}^{N\times d}$, and a Boolean optimization problem, such as*

$$\min_{X,Y} \|D - Y \odot X^\top\|^2 \qquad s.t.\ X \in \{0,1\}^{d\times r},\ Y \in \{0,1\}^{N\times r}.$$

$$
\begin{aligned}
D = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} &\approx \begin{pmatrix} 1 & .9 & .9 & .1 \\ .7 & 1.2 & 1.2 & .7 \\ .1 & .9 & .9 & 1 \end{pmatrix} \\
&\approx \begin{pmatrix} 1 & 0 \\ .6 & .6 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & .9 & .9 & .1 \\ .1 & .9 & .9 & 1 \end{pmatrix} \\
D_A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} &= \theta\begin{pmatrix} 1 & 0 \\ .6 & .6 \\ 0 & 1 \end{pmatrix} \cdot \theta\begin{pmatrix} 1 & .9 & .9 & .1 \\ .1 & .9 & .9 & 1 \end{pmatrix} \\
D_B = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} &= \theta\left(\theta\begin{pmatrix} 1 & 0 \\ .6 & .6 \\ 0 & 1 \end{pmatrix} \cdot \theta\begin{pmatrix} 1 & .9 & .9 & .1 \\ .1 & .9 & .9 & 1 \end{pmatrix}\right)
\end{aligned}
$$

Fig. 5.15: Approximation of a binary matrix D with two overlapping biclusters (top) applying NMF (second from above) and the factorizations resulting from thresholding the factor matrices to binary matrices in elementary algebra (second from below) and Boolean algebra (below). Biclusters are highlighted.

1. *Optimize the following objective with proximal gradient descent:*

$$
\begin{aligned}
&\min_{X,Y} \|D - YX^\top\|^2 + \lambda_x\langle \boldsymbol{\Lambda}(X), \mathbf{1}\rangle + \lambda_y\langle \boldsymbol{\Lambda}(Y), \mathbf{1}\rangle \\
&s.t.\ X \in \mathbb{R}^{d\times r},\ Y \in \mathbb{R}^{N\times r}.
\end{aligned}
\tag{5.34}
$$

That is, perform the PALM update steps from Equations 5.29 and 5.30 for $F(X, Y) = \|D - YX^\top\|^2$ *and the regularizing functions*

$$
\phi_x(X) = \lambda_x\langle \boldsymbol{\Lambda}(X), \mathbf{1}\rangle, \qquad \phi_y(Y) = \lambda_y\langle \boldsymbol{\Lambda}(Y), \mathbf{1}\rangle.
$$

2. *Return the binary matrices that result from a suitable thresholding operation of the relaxed result. That is, perform a grid search on the set* $\mathcal{T} = \{0, 0.05, 0.1, \ldots, 1\}$*:*

$$
\begin{aligned}
(\rho_x^\star, \rho_y^\star) &= \arg\min_{\rho_x,\rho_y}\{\|D - \theta_{\rho_y}(Y_t) \odot \theta_{\rho_x}(X_t^\top)\|^2 \mid \rho_x, \rho_y \in \mathcal{T}\} \\
(X, Y) &= (\theta_{\rho_x^\star}(X_t), \theta_{\rho_y^\star}(Y_t))
\end{aligned}
$$

The matrix or vector $\mathbf{1}$ indicates a constant one matrix, whose dimensionality can be inferred from context. The Frobenius inner product $\langle \Lambda(X), \mathbf{1}\rangle = \sum_{i,s} \Lambda(X_{is})$ sums the penalization terms over all matrix entries. As a default value, we employ a natural choice for the regularization weight $\lambda_x = \lambda_y = 1$.

The procedure of PAL-Tiling describes the general framework for the optimization of the Boolean matrix factorization error. Recent contributions in the field of Boolean matrix factorization also incorporate a mechanism to automatically determine the rank of the factorization. To this end, objective functions other than the approximation error in Frobenius norm are commonly optimized in BMF. So far, we have proposed two approaches to facilitate the optimization with PAL-Tiling. The rank determination can be integrated into an outer loop of PAL-Tiling, where the models of various ranks are compared. One approach uses the well-established Minimum Description Length (MDL) principle to determine the rank as the one yielding the minimum description length of the model. Here, the objective in Boolean algebra is the employed description length [310]. The other approach is to optimize the Boolean approximation error, and to conduct statistical tests on the probability that at least one of the clusters is generated by noise [311].

5.4.6 BROCCOLI – Optimizing Tri-Factorization Biclusterings

While the final thresholding step for the optimization of Boolean factorizations is needed to translate the fuzzy clustering structure to the Boolean algebra, the biclustering models of checkerboard and block-diagonal clustering use the elementary algebra matrix product. As a result, the relaxed formulation of the objective with the nonbinary penalization terms in Equation 5.34 will have the same optimizers as the corresponding binary matrix factorization objective (cf. Table 5.9), if the penalizing weights λ_x and λ_y are large enough.

After every gradient descent step of one of the cluster indicator matrices, the prox-operator is applied and pushes the matrix towards binary values. Hence, if the nonbinary penalization weights λ_x and λ_y are large enough, then we will get binary matrices after a couple of iterations. However, in this case, we also risk converging to a local optimum close to the initialization. This would make our method even more sensitive to the initialization than it already is due to the nonconvexity of the objective. In turn, if we choose a value for λ that is too small, then the optimum of the penalized objective might not return binary matrices. We circumvent these issues by gradually increasing the regularization weights throughout the optimization process. In addition, we employ individual regularization weights. To this end, we introduce the regularization weights as optimization parameters that are as large as possible in the optimal solution. We achieve this by subtracting the sum of the regularization parameters $\langle \boldsymbol{\lambda}_x, \mathbf{1}\rangle + \langle \boldsymbol{\lambda}_y, \mathbf{1}\rangle$ from the objective function value (cf. Equation 5.35).

Algorithm Specification 2 (Broccoli). *Given a data matrix $D \in \mathbb{R}^{N\times d}$ and a biclustering optimization problem, such as*

$$\min_{X,Y,C} \|D - YCX^\top\|^2 + \phi_c(C) \qquad s.t.\ Y \in \{0,1\}^{N\times r}, X \in \{0,1\}^{d\times r}, C \in \mathbb{R}^{r\times r}.$$

1. *Optimize the following objective with proximal gradient descent:*

$$\min_{\substack{X,Y,C,\\ \boldsymbol{\lambda}_x,\boldsymbol{\lambda}_y}} \|D - YCX^\top\|^2 + \langle \boldsymbol{\lambda}_x, \Lambda(X) - \mathbf{1}\rangle + \langle \boldsymbol{\lambda}_y, \Lambda(Y) - \mathbf{1}\rangle + \phi_c(C)$$
$$s.t.\ (\boldsymbol{\lambda}_x)_{is}, (\boldsymbol{\lambda}_y)_{js} \le \lambda_+ \textit{ for all } 1 \le i \le d,\ 1 \le j \le N,\ 1 \le s \le r.$$
$$Y \in \mathbb{R}^{N\times r}, X \in \mathbb{R}^{d\times r}, C \in \mathbb{R}^{r\times r} \tag{5.35}$$

That is, perform the PALM update steps in an alternating fashion for X,C, λ_x, Y, C, and at last for λ_y. Where $F(X,Y,C) = \|D - YCX^\top\|^2$, and the regularizing functions

$$\phi_x(X) = \langle \boldsymbol{\lambda}_x, \Lambda(X)\rangle \qquad \phi_y(Y) = \langle \boldsymbol{\lambda}_y, \Lambda(Y)\rangle$$

The parameter λ_+ in Equation 5.35 is employed as a placeholder for the maximally required regularization weights $\boldsymbol{\lambda}_x$ and $\boldsymbol{\lambda}_y$ such that the optimizing factor matrices Y and X of Equation 5.35 are binary. Bounding the regularization weights above by the parameter λ_+ ensures that the objective in Equation 5.35 is well-defined. However, we do not need to determine the parameter λ_+ in practice.

The parameter matrices $\boldsymbol{\lambda}_x$ and $\boldsymbol{\lambda}_y$ are the regularization weights of the non-binary penalization terms $\Lambda(X)$ and $\Lambda(Y)$. The Frobenius inner product

$$\langle \boldsymbol{\lambda}, \Lambda(X)\rangle = \sum_{i,s} \boldsymbol{\lambda}_{is}\Lambda(X_{is})$$

sums the elementwise penalization terms weighted by the parameters $\boldsymbol{\lambda}$.

The implementation details of the Broccoli optimization scheme can be found in Hess, Pio, Hochstenbach, and Ceci [312]. In contrast to the Boolean factorization framework PAL-Tiling, the initialization plays an important role for Broccoli. Instead of the vanilla PALM optimization method, Broccoli employs stochastic proximal gradient descent [187].

5.4.7 Experiments

We highlight here a few results from the applications of the PAL-Tiling instance Primp [310] and the Broccoli implementation using a nonnegative matrix factorization for initialization [312]. More experiments than the ones displayed here can be found in the corresponding literature [309, 310, 311, 312].

We compare the PAL-Tiling instance Primp (henceforth indicated as PAL-Tiling) with the available implementations of the BMF methods Panda+[13], Mdl4bmf[14], and Nassau.[14]

We compare Broccoli with six competitors: two methods based on a nonnegative relaxation (henceforth denoted by N [447] and NN [165]), two methods based on an

[13] http://hpc.isti.cnr.it/~claudio/web/archives/20131113/index.html.

[14] http://people.mpi-inf.mpg.de/~skaraev/.

orthogonal relaxation (henceforth denoted by O [725] and OO [165]) and the biclustering methods FABIA [318] and FLOC [716]. Since N, NN, O and OO return fuzzy membership values for each observation, we binarize the result for comparison purposes. For each sample (observation or feature) we set the top-k fuzzy cluster indicator values to one, where k is the number of ground truth clusters the sample belongs to. Note that in this way we provide our competitors with additional background knowledge, that is not available in real-world scenarios. The goal is to estimate how good the clustering, derived from a relaxed result, could potentially be, if supported by additional knowledge (e.g., from domain experts).

Quality Metrics We quantify how well a computed cluster indicator matrix matches the ground truth by an adaptation of the micro-averaged F_1-measure, known from multi-class classification tasks. We compute a one-to-one matching τ between computed and ground truth clustering and compute the average F_1-measure of the matched clusters. That is,

$$F_Y = \frac{1}{r}\sum_{s=1}^{r} F_1(Y_{\cdot s}, Y^{\star}_{\cdot \tau_Y(s)}), \qquad F_X = \frac{1}{r}\sum_{t=1}^{r} F_1(X_{\cdot t}, X^{\star}_{\cdot \tau_X(t)}).$$

We return the average F_1-score of the feature and observation clusters:

$$F = \tfrac{1}{2}(F_Y + F_X).$$

The F_1-measure has values between zero and one. The closer it approaches one, the more the computed clustering matches the ground truth. The plots that display the F-measure indicate its average value with error bars having the length of twice the standard deviation.

5.4.8 Synthetic Dataset Experiments

PAL-Tiling We generate data matrices according to the scheme established by Karaev, Miettinen, and Vreeken [356], Lucchese, Orlando, and Perego [451], and Miettinen and Vreeken [492]. We generate (1600 × 500) and (1000 × 800) dimensional datasets as outlined by Hess, Morik, and Piatkowski [310]. Given dimensions d, N, and noise parameter p, a factorization of rank $r^{\star} = 25$ is generated by uniformly randomly drawing each tile $(X^{\star}_{\cdot s}, Y^{\star}_{\cdot s})$ from all tiles of size $|X^{\star}_{\cdot s}| \in [0.01d, 0.1d]$ and $|Y^{\star}_{\cdot s}| \in [0.01N, 0.1N]$. Finally, each bit entry $(Y^{\star} \odot X^{\star\top})_{ji}$ is flipped with probability p.

We compare the effects of the matrix dimensions and aggregate results over 10 generated matrices with dimensions 1000 × 800 and 1600 × 500. Figure 5.16 plots the F_1-measure and the rank of the returned BMF against the percentage of noise. NASSAU particularly strongly underestimates the rank for the 1600 × 500 dimensional matrices. Here, NASSAU returns close or equal to zero tiles, even if the noise is low. This effect can

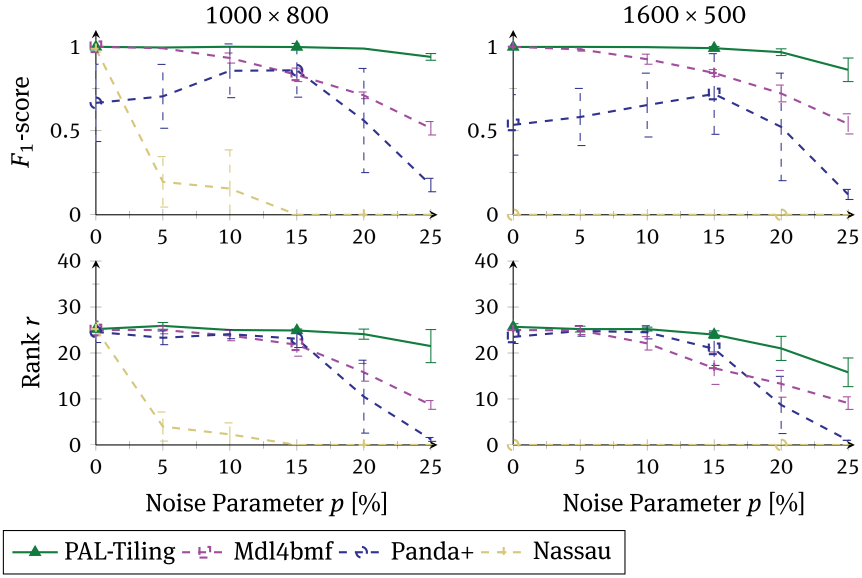


Fig. 5.16: Variation of Bernoulli noise parameter p for 1000 × 800 and 1600 × 500 dimensional data. Comparison of F_1-measures (the higher the better) and the estimated rank of the calculated Boolean matrix factorization (the closer to 25 the better) for varying levels of noise, i.e., p is indicated on the x-axis (best viewed in color).

actually be alleviated if we transpose the matrix, which makes Nassau perform similar to Mdl4BMF, yet with a stronger tendency to underestimate the rank. We observe that all algorithms tend to underestimate the rank the more the noise increases. This culminates in the replication of almost none of the tiles at the highest noise level for the algorithms Panda+ and Nassau. Panda+ yields correct rank estimations up to a noise of 15 %, but its fluctuating F-measure indicates that planted tiles are not correctly recovered after all. Mdl4bmf shows a robust behavior. Its suitable rank estimations up to a noise of 15 % are mirrored in a high F-measure. PAL-Tiling is characterized by overall high values in the F_1-measure. The experiments demonstrate a high robustness of the proposed BMF optimization scheme PAL-Tiling to noise on synthetic data.

Broccoli For the biclusterings generated by a tri-factorization, we create a set of synthetic clusterings with overlap and outliers by sampling every cluster indicator matrix by a Bernoulli distribution. Entry $X^\star_{it}$ and $Y^\star_{js}$ is equal to one with probability $q = 0.2$. Thereby, we ensure that a cluster contains at least 1 % of the data points/features, that are exclusively assigned to this particular cluster. The core matrix is sampled as a sparse matrix containing uniformly distributed values $C_{st} \in [0, 5]$. The probability that a non-diagonal element is not equal to zero is equal to $1/r$. The data matrix is generated

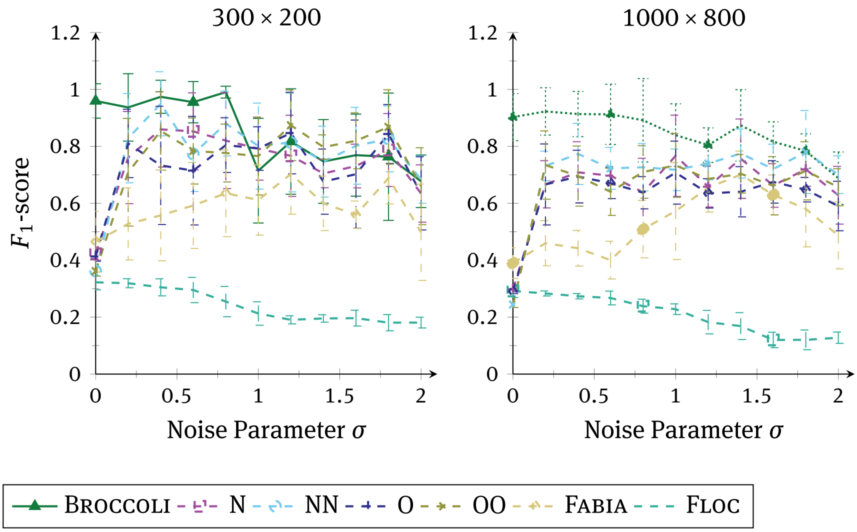


Fig. 5.17: Variation of the Gaussian noise parameter σ, comparison of F-measures (the higher the better) for 300 × 200 data matrices with three row- and column-clusters and 1000 × 800 data matrices with five row- and column-clusters.

by adding random Gaussian noise to the ground truth factorization:

$$D_{ji} = [Y^\star_{j\cdot} C X^{\star\,\top}_{i\cdot} + \epsilon_{ji}]_{\geq 0},$$

where $\epsilon_{ji} \sim \mathcal{N}(0, \sigma)$ and the operator $[\cdot]_{\geq 0}$ projects negative values to zero. We generate for every noise variance $\sigma \in \{0, 0.2, 0.4, \dots, 2\}$ and dimensionality $(N, d) \in \{(300, 200), (1000, 800)\}$ five datasets. For the smaller 300 × 200 dataset, we choose a rank of $r = 3$ and for the larger 1000 × 800 dataset, we choose a rank of $r = 5$.

In Figure 5.17, we plot the F_1-measure against the Gaussian noise parameter σ. The maximum value of σ is 2.0. For $\sigma = 2$, roughly $1/3$ of the noise samples are larger than or equal to 1.0, and about $2/3$ of all noise samples have an absolute value larger than or equal to 1.0 in expectation. We see that throughout the increase of the noise parameter, Broccoli attains a high F-measure, close to 1.0, which slightly drops when the noise variance exceeds 1.0. The methods N, NN, O, and OO, which are based on orthogonal and nonnegative relaxations seem largely unaffected by the noise parameter, and attain on average an F_1-score between 0.7 and 0.8. Fabia and Floc attain the lowest F_1-score of all competitors, where the F-score of Fabia has a tendency to increase with the noise parameter up to 0.7. This is possibly due to the fact that Fabia and Floc do not explicitly handle the possible presence of noise in the data.

5.4.9 Qualitative Experiments

PAL-Tiling In this experiment, we explore how the algorithms relate to their actual cognition of structure and noise, and illustrate what their biclusters look like. Image data allows us to visually inspect the resulting factorizations and to intuitively assess the captured relevant sub-structures.

We employ a standard representation of images: the RGB888 pixel format. Each of the $w \times h$ pixels is represented by 24 bits, using 8 bits per color (red, green and blue). In order to convert an image into a set of observations, we divide it into blocks (patches) of 4×4 pixels, resulting in a total of $\frac{w}{4} \times \frac{h}{4}$ observations per image. We adopt this representation from computer vision, where image patches are a standard preprocessing step for raw pixel data [340]. Within each block, let $(r, g, b)_{l,k}$ denote the pixel at row l and column k, where $r, g, b \in \{0, 1\}^8$ are the 8-bit binary representation of its red, green, and blue color values. We model the concatenation of all 16 pixels within one block as one observation

$$\left[(r,g,b)_{1,1},(r,g,b)_{1,2},(r,g,b)_{1,3},(r,g,b)_{1,4},(r,g,b)_{2,1},\ldots,(r,g,b)_{4,4}\right] \tag{5.36}$$

which has a length of $24 \cdot 16 = 384$ bits.

This way, we process a selection of “aliens” from the classic game *Space Invaders*. Reconstruction results and top-4 patterns of the Space Invaders image are shown in Figure 5.18. All methods reconstruct at least the shape of the aliens. In terms of color, however, the results diverge. Panda+ and Nassau interpret all colors as negative noise effects on the color white; white has a binary representation of 24 ones. PAL-Tiling and Mdl4bmf reconstruct all three colors of the original image, yet the reconstruction of Mdl4bmf exhibits injections of white blocks. Hence, only PAL-Tiling is capable of reconstructing the color information correctly.

Having a look at derived biclusters, the greedy processes of Panda+ and Nassau become particularly apparent: Panda+ and Nassau overload the first factor with all the shape information. The remaining factors reduce the quantitative reconstruction error, but have no deeper interpretation. Mdl4bmf tries to model one type of alien by each bicluster. Although this would result in a reasonable description of the image, the actual extraction of tiles suffers from the greedy implementation. For example, we can see that the first tile captures information about the yellow aliens as well as strayed parts of other aliens. This unfortunate allocation of tiles results in the injection of white blocks in the reconstruction image. PAL-Tiling separates by its tiles the three basic color channels that are actually used to mix the colors that appear in the original image. The results of this qualitative experiment illustrate the benefits of a non-greedy minimization procedure.

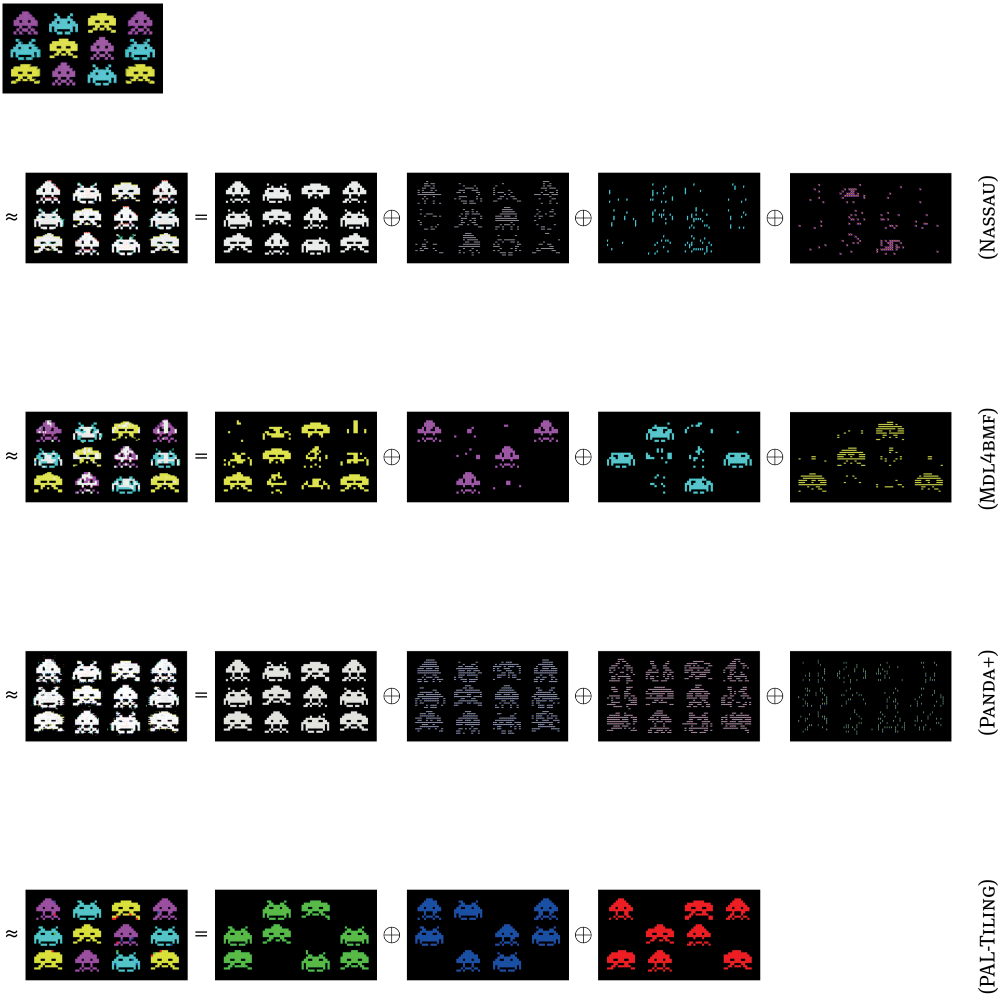


Fig. 5.18: Reconstructions of the Space Invaders image and visualizations of the top-4 outer products. Best viewed in color.

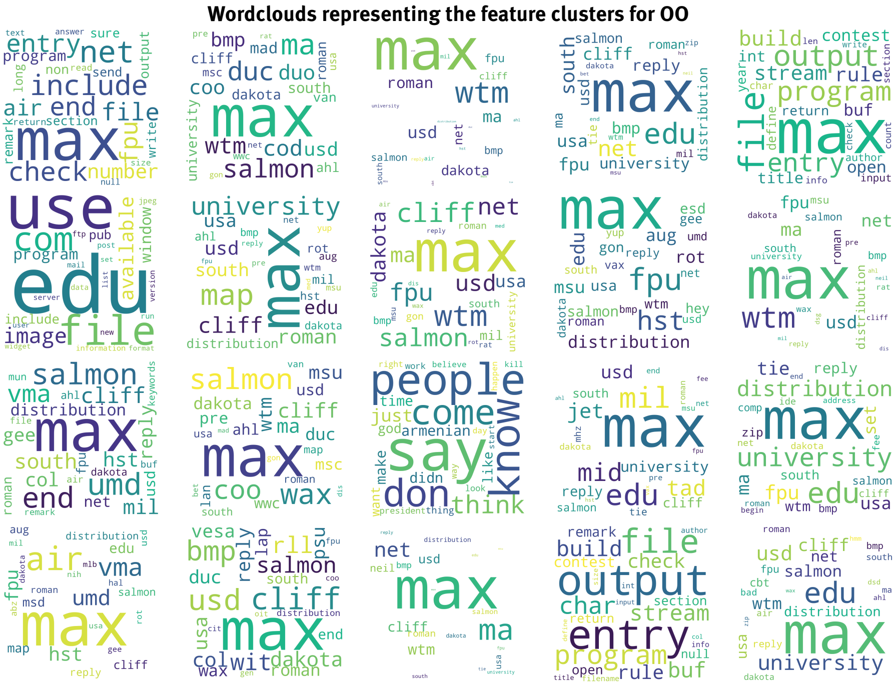


Fig. 5.19: Illustration of derived word-clusters by the method OO on the 20 Newsgroups dataset. The size of a word reflects its weight in the corresponding cluster ($X_{\cdot s}$).

Broccoli We perform a qualitative inspection of the results by means of the *20 Newsgroups* dataset.[15] The 20 Newsgroups dataset is a collection of posts belonging to one of twenty topics that are hierarchically organized. We process the textual data as a data matrix, reflecting for $N = 11\,314$ posts the term-frequency of $d = 6643$ lemmatized words. We apply the methods Broccoli, NN, and OO to derive $r = 20$ row- and column-clusters. Fabia and Floc were not able to successfully process such a large dataset.

The obtained column-clusters (the feature clusters that in this case are clusters of words) are shown in Figures 5.19–5.20. We display here only the fuzzy cluster indicators of the orthogonal relaxation method OO; the results from NN were very similar [312]. The size of word i in the wordcloud of a fuzzy cluster s corresponds to the assigned weight $X_{is} \geq 0$. In turn, the binary word-indicators of Broccoli are visualized such that the size of a word in the cloud is proportional to the inverse of the number of clusters the word is assigned to. That is, those words that are unique to the respective cluster are larger than those words which are assigned to multiple clusters. Looking at the visualizations

[15] http://qwone.com/~jason/20Newsgroups/.

Wordclouds representing the feature clusters for Broccoli

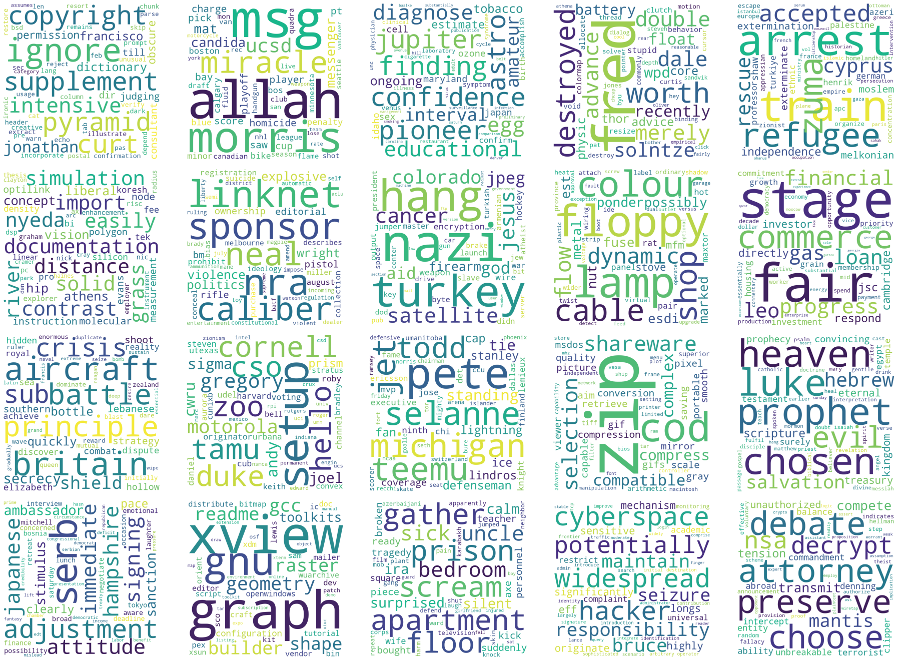


Fig. 5.20: Illustration of derived word-clusters by Broccoli on the 20 Newsgroups dataset. The size of a word reflects its weight in the corresponding cluster ($X_{\cdot s}$).

of clusters, we see that the word *max* pops up prominently in many clusters. The word *max* obtains comparably very high term frequencies. The average term frequency of a word is equal to 1.59, and 99 % of all words have a term frequency smaller than or equal to 8. The word *max* occurs in 149 posts and obtains term frequencies in $[1, 800]$. Hence, the word *max* attains exceptionally high term frequencies in a few posts and exhibits therewith a special role. The unusual high term frequency of the word *max* is handled differently among the clustering methods. While OO gives a high weight to this word in almost all clusters, Broccoli ignores the high frequency of this word. This demonstrates the more modular approach of biclustering with binary cluster-indicator matrices and a robustness to outliers.

We can detect meaningful clusters that address a specific topic for all clustering methods. Comparing the addressed topics among the clustering methods, we see that Broccoli provides a distinctive view on the dataset, identifying, for example, a *religion* cluster that is not featured by the method OO. Hence, although Broccoli's optimization makes use of a relaxed objective, its results still offer another view on the data with respect to that provided by the relaxed counterparts.

5.4.10 Applications, Impact, and Future Work

In this work, we have reviewed the optimization methods for clusterings that have a matrix factorization objective. Our comparison of popular clustering objectives has shown that the majority of methods is designed to find partitioning clusters, adhering to the EXCLUSIVITY CONSTRAINT: every point is assigned to exactly one cluster. Suitable adaptations of Lloyd's algorithm guarantee the convergence to a local optimum of the objective function subject to the partitioning and particularly binary constraints. This offers an undeniable advantage over other practical alternatives that relax the binary constraint or heuristics. The major drawback of relaxing approaches is the discretization step, in which theoretical guarantees, which might be provided for solutions of the relaxed objective, are usually lost. However, the exclusivity constraint, enabling the alternating minimization according to Lloyd, is not feasible in some applications. Overlapping and nonexhaustive clusterings are more likely to represent the *true* model when it comes among other things to the clustering of text or genomic data. In this case, the theoretical foundation regarding the efficient optimization of corresponding objectives is leaky.

We have proposed a general optimization framework for overlapping clusterings by means of proximal alternating minimization. In particular, we have proposed two approaches to optimize biclustering objectives, where the exclusivity assumption is most often inept. The method PAL-TILING is designed for the optimization of a Boolean matrix factorization, which is used to derive overlapping and non-exhaustive clusters of binary data. The method BROCCOLI is designed for a biclustering of real-valued data, based on a tri-factorization.

Our experimental analysis highlights the power of the proposed optimization approach on two instances: the MDL-based BMF method PRIMP (denoted here as PAL-TILING) [310] and the NMF initialization of the BROCCOLI framework [312]. Our experiments on synthetic data indicate in particular the robustness of our proposed optimization approach to noise, amount of overlap, and number of outliers (cf. Figures 5.16 and 5.17). Our qualitative inspection of found clusters indicates the meaningfulness of the found clusters (cf. Figures 5.20 and 5.18).

This makes PAL-TILING and BROCCOLI a theoretically founded, practically well-performing and flexible approach that has the potential to spark further research on the optimization of non-exclusive clusterings in particular, and on the learning of discrete structures in general.

Future Work The proposed optimization approaches are flexible and have the potential to found a standard method for the optimization of clustering structures based on matrix factorization that does not require the exclusivity assumption. Note, that many popular clustering methods are based on (or can be viewed as) a matrix factorization: k-means, spectral clustering, and variants of deep clustering. In addition, techniques to cope with specific data characteristics in matrix factorization can easily be transferred to the optimization scheme adopted in PAL-Tiling and Broccoli.

In addition, nonconvex optimization is an ongoing field of research. There are stochastic [163, 187], accelerated [427], and inertial [581] variants of the PALM optimization scheme. That is, the power of the proposed optimization framework grows with the research on the underlying optimization schemes. An analysis of proposed nonconvex optimization schemes for the optimization subject to binary constraints and clusterings in particular, would be a topic of further research.

6 Hardware-Aware Execution

Efficient learning has been the focus of research for decades. Many studies explore various software/hardware techniques to improve the efficiency of learning process while preserving the accuracy of the derived learning models. Along with the various demands of applications nowadays, from simple like image recognition to advanced like fundamental steering, how to deploy learned models and execute them efficiently has been of key interests in the industry. Considering various resource constraints such as throughput, timeliness, and energy consumption imposed by the targeted scenarios and the adopted hardware platforms, most machine learning techniques, which often rely on high-performance computers and clusters, must be carefully redesigned to fulfill the assigned missions at edge devices while addressing the efficiency of resource usage.

To this end, we summarize in this chapter relevant research conducted in CRC 876, which is oriented to the awareness of hardware execution, and supplement two external contributions to cover a broader spectrum of this research direction. Unlike most existing techniques for executing neural networks on Field-Programmable Gate Arrays (FPGAs), we focus on the of learning, which is actually more computationally demanding (see Section 6.1). In addition, we exploit modern graphics processing units (GPUs) for efficient database query processing (see Section 6.2) and study how parallelization on multicore systems should be deployed for accelerating extreme multi-label classification (see Section 6.3). At the end, we present our RAMBO framework, which can efficiently optimize machine learning models even on heterogeneous distributed systems (see Section 6.4). The mentioned techniques in this chapter tend to reveal different perspectives to achieve efficient learning on various hardware platforms. Although it is not possible to cover all relevant techniques, the introduced insights should clearly reveal that a proper usage of hardware can be very effective, especially for the efficiency of learning process.

https://doi.org/10.1515/9783110785944-006

6.1 FPGA-Based Backpropagation Engine for Feed-Forward Neural Networks

Wayne Luk
Ce Guo

Abstract: Feed-Forward Networks (FFNs), or multilayer perceptrons, are fundamental network structures for deep learning. Although feed-forward networks are structurally uncomplicated, their training procedure is computationally expensive. It is challenging to design customized hardware for training due to the diversity of operations in forward- and backward-propagation processes. In this contribution, we present an approach to train such networks using Field-Programmable Gate Arrays (FPGAs). This approach facilitates the design of reconfigurable architectures by reusing the same set of hardware resources for different operations in backpropagation. In our empirical study, a prototype implementation of the architecture on a Xilinx UltraScale+ VU9P FPGA achieves up to a 5.2 times speedup over the PyTorch platform running on 8 threads on a workstation with two Intel Xeon E5-2643 v4 CPUs.

6.1.1 Introduction

The majority of FPGA-based deep learning architectures are for inference procedures, which makes predictions using pre-trained networks. However, training is a computationally demanding procedure that limits the application of deep neural networks. Backpropagation is the core process in the training procedure of neural networks. This contribution discusses an FPGA-based architecture for backpropagation for Feed-Forward Networks (FFNs). An FFN consists of multiple layers of connected nodes. Each node in a layer takes the weighted sum of the activation signals from the previous layer and generates an activation signal by evaluating a nonlinear activation function.

We study FFNs for two reasons. First, as stand-alone models, they are useful in learning problems with unstructured information such as non-image and non-sequential data. For instance, in the event classification problem for the Higgs boson [3], the correlation between attributes are difficult to capture with other neural networks such as Convolutional Neural Networks (CNNs), while FFNs can provide decent accuracy. Second, as deep network components, FFNs usually appear as the decision-maker in convolutional neural networks; they also serve as generators and discriminators in Generative Adversarial Networks (GANs) [266].

Although FFNs appear less complicated than other neural types of neural networks, training FFNs efficiently on FPGAs is challenging. For instance, in convolutional neural networks, a layer of nodes may share a small set of parameters. This parameter-sharing property naturally reduces on-chip memory usage. However, FFNs do not have similar properties as each connection between a pair of nodes carries a unique scalar parameter, resulting in a high memory bandwidth usage.

Unlike research in general hardware design for neural network training like [431] and [374], we pay attention to the features and limitations of reconfigurable hardware. The following summarizes the challenges that we face.

1. Challenge of diverse arithmetic operations. The hardware resources on an FPGA chip stay unchanged while the arithmetic operations frequently change during backpropagation. To reuse the hardware resources and minimize the overhead for reconfiguration, it is desirable to design multifunctional hardware modules that support multiple operations without runtime reconfiguration.
2. Challenge of hardware adaptability. The size of the network and hardware resources vary greatly in different applications. A proper hardware design should be adaptive to all reasonable network sizes and hardware platforms.
3. Challenge of complex control logic. When a multifunctional hardware module supports more operations, the control logic becomes more complicated. In particular, it is challenging to define a set of commands to control the hardware modules to collaborate at different stages in backpropagation.

It is difficult to find off-the-shelf solutions because most existing systems for gradient computation and training run on CPUs and GPUs. The novel aspects covered in this contribution include the following:

1. Reuse of hardware resources for different operations. We extend the multifunctional multiplication block in [278] to allow different operations to share the same set of hardware resources without runtime reconfiguration, which addresses the challenge of diverse arithmetic operations. In addition, the resource usage of the architecture is independent of the network layout, which addresses the challenge of hardware adaptability.
2. Commands to control the hardware. To use the same set of hardware resources for different operations without significant loss of efficiency, it is necessary to find a proper set of commands to control the hardware. We define a small set of commands in this section, addressing the challenge of complex control logic.
3. Empirical evaluation. We conduct experiments to compare an experimental implementation of the hardware on an FPGA platform with the PyTorch deep learning framework running on CPUs and GPUs.

6.1.2 Background and Related Work

This section provides a short introduction to Feed-forward networks and and their hardware-based training approaches.

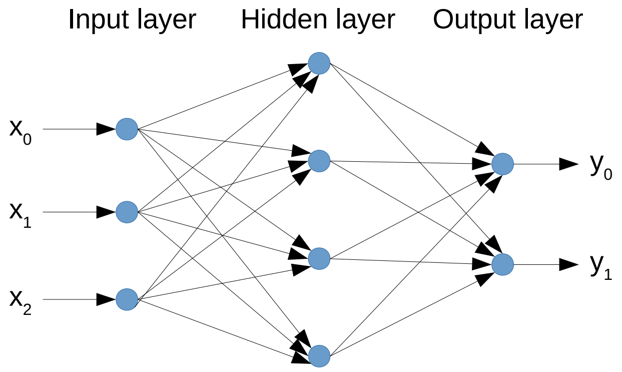


Fig. 6.1: An example feed-forward network.

6.1.2.1 Feed-Forward Networks and Backpropagation

A feed-forward network contains nodes arranged in layers. For instance, Figure 6.1 shows a layout of an FFN with three layers of nodes and two layers of connections. Each node in the input layer feeds a feature of the data point to the network. All nodes in layer l receive signals from all nodes in the previous layer $(l-1)$ and produce an output signal in the form of a vector. The generation of output signal involves two steps. The first step is to calculate the weighted-sum vector:

$$\mathbf{t}_l = W_{l-1}\mathbf{x}_{l-1} \tag{6.1}$$

where $\mathbf{x}_l$ is a vector containing the output of all nodes in layer l; and W_{l-1} is a matrix of real numbers specifying N_l weight vectors. The second step is to produce an output signal x_l with

$$\mathbf{x}_l = f_{\text{act}}(\mathbf{t}_l) \tag{6.2}$$

where $f_{\text{act}}(\cdot)$ is a non-linear function defined on real vectors.

It is necessary to specify the weights $W = \{W_0 \ldots W_{L-1}\}$ before using the network to make predictions. One may find out the weights using data via training. A training algorithm searches for a set of weights that fits a dataset D with respect to an error measure $E(W, D)$. An efficient training algorithm typically updates the parameter set using the gradient

$$\nabla W = \frac{\partial E(W, D)}{\partial W} \tag{6.3}$$

to minimize the error on the training data. The de facto method to compute the gradient ∇W is backpropagation [422].

An episode of backpropagation includes a forward pass and a backward pass of signals. In the forward pass, the network takes a data point and computes a prediction following the direction of the network. In the backward pass, the network propagates an error signal in the opposite direction to compute the gradient. Note that the backpropagation process does not include the optimization algorithm, which uses gradients to update the weights [421].

The backpropagation process is computationally demanding. The sheer amount of network parameters results in problems in the algorithms and hardware. With regard to algorithms, the high dimensionality of the parameter space slows down the convergence of the optimization procedure. As a result, the optimization algorithm needs to invoke a large number of backpropagation episodes before obtaining an accurate network model. With regard to hardware, parameters consume considerable memory space and IO bandwidth during computation because the nodes do not share parameters.

The Graphics Processing Unit (GPU) is arguably the most widely-used hardware platform to implement training algorithms for neural networks. The GPU platform provides high performance with relatively low hardware costs and a short design cycle. However, trends in the development of machine learning suggest that FPGAs may become more promising than GPUs for two reasons. First, more neural networks will be based on customized data types, especially quantized numbers [326]. GPUs can natively support only a limited number of data types. By contrast, FPGAs can support customized data types efficiently. Second, the performance gap between GPUs and FPGAs is narrowing fast [541]. In particular, the size of on-chip memory, clock speed, number of hardware DSP units, memory bandwidth, and the process technology of FPGAs have significantly advanced.

6.1.2.2 Reconfigurable Computing for Neural Network Training

Among the statistical models for classification and regression, neural networks are some of the most popular candidates for reconfigurable acceleration [488]. Because we focus on the training process, we do not cover the hardware engines that only perform inference. Reviews that cover inference engines include [408, 497, 500, 546]. The first type of solutions is the acceleration of the training process of general-purpose neural network architectures. Eldredge and Hutchings [198] divide the backpropagation algorithm into three stages and design hardware for each stage separately. During the training process, the hardware performs a runtime reconfiguration at the beginning of each stage. Paul and Rajopadhye [558] propose a systolic backpropagation engine that avoids the runtime reconfiguration. In their design, all calculations in a complete background procedure are mapped to hardware. Murugan et al. [522] designs a training architecture for a network with five nodes. An implementation on a Xilinx Virtex-E FPGA runs at 5.332 MHz. Li and Pedram [437] propose a coarse-grain architecture mainly to implement the matrix multiplication operations in training. Langhammer and Pasca [417] discuss architectures that evaluate common activation functions with different approximation methods. Kim et al. [374] present the DeepTrain platform to perform energy-efficient training for various types of deep networks. The DeepTrain platform offers tools to generate sequences of operations for the hardware architecture using network descriptions extracted from the TensorFlow deep learning framework. Maeda and Tada [458] propose a training engine for neural networks using the simultaneous perturbation rule [457] to avoid the gradient computation in the training process.

The second type of solutions is the design and optimization of hardware-oriented neural network structures. A popular network structure in this category is the Block-based Neural Network (BbNN). Moon and Kong [502] first propose the BbNN and implement the prediction facility on the FPGA platform. A BbNN connects a collection of neutron blocks. A neutron block carries four numeric I/O ports. Each I/O port may serve as either an input port or an output port. An output port provides an activation signal computed from the input signals on the same neutron block. Jiang et al. [344, 345, 346] study the training process of the BbNN using evolutionary algorithms on the CPU platform. The idea behind their training approach is to encode the topology of the network and the configuration of the neural blocks into a vector so that an evolutionary algorithm may improve the network by manipulating the vector. Merchant and Peterson [488] make it possible to train the block-based neural networks on the FPGA platform. The third type of solutions is the customization of domain-specific or problem-specific neural networks. A representative neural network structure in this category is the convolutional neural network. The convolutional neural network [403] is a feed-forward network structure inspired by the visual cortex of animals. The major application of CNNs is image recognition [403]. The reconfigurable acceleration solutions for CNNs usually take advantages of the unique properties of structured data. Farabet et al. [212] propose an FPGA-based RISC processor that matches basic operations in the CNN. The processor uses a description of a pre-trained CNN in the form of a sequence of instructions to make predictions. A useful observation in this section is that a proper reduction of the precision of image operations results only in a little negative impact on the predictive accuracy. However, the precision reduction may save a considerable amount of hardware resources. Further optimizations [428, 684, 734] have made FPGA devices faster and more energy efficient than CPUs and GPUs. Zhao et al. [744] propose the first stand-alone training engine for CNNs using a streaming data path. The data path contains a collection of parameterized modules. The organization of these modules changes over time with the runtime reconfiguration, which enables the data path to train different layers of a network. In addition to CNNs, FPGA-based reinforcement learning methods have emerged in recent years. Shao and Luk [625] present an architecture trust region policy optimization, which allow robots or agents to efficiently learn policies by interacting with an environment. An implementation on an Intel Stratix-V FPGA achieves up to a 20 times speedup against a 6-thread software reference on an Intel Core i7-5930K CPU running at 3.5 GHz. Gankidi et al. [240] design a Q-learning architecture for a planetary robot. An implementation of the architecture on an Xilinx Virtex-7 FPGA achieves 43 times speedup compared to an Intel 6-gen i5 CPU running at 2.3 GHz.

6.1.3 Architecture for Backpropagation

This section presents the hardware architecture of the backpropagation engine and automated generation of control sequences. During a backpropagation process, two major operations consume the majority of execution time.

1. Linear combination. A node takes the linear combination of the outputs from all nodes in the previous layer. In the forward pass, the operation combines activation signals. In the backward pass, the operation combines error signals and computes gradients.
2. Non-linear function evaluation. A node evaluates a real-valued non-linear function. In the forward pass, the operation evaluates an activation function. In the backward pass, the operation evaluates the derivative of the activation function.

The two major operations are computationally demanding because their time complexity depends on the network layout [421]. Specifically, the time spent on linear combination grows linearly with the number of network parameters. By contrast, the time spent on function evaluation grows linearly with the number of nodes in the network. Other operations in backpropagation are less computationally expensive than the two major operations. For instance, it is necessary to compare the actual label of the training data and the prediction to calculate the initial error signal for the backward pass. However, the comparison runs only once in a backpropagation episode, and the calculation typically has linear complexity regarding the data dimension.

The two major operations are fundamentally different regarding arithmetic operations. A straightforward way to customize hardware architecture is to create separate arithmetic modules for both operations [437]. However, the sequential dependencies between the calculations allow only the execution of one type of operation at the same time. Therefore, when the arithmetic module for one operation is active, the corresponding arithmetic modules work with a full load, but the modules for all other operations are idle. In other words, only a small fraction of logic units work, resulting in wastage of hardware resources. Admittedly, it is possible to prepare separate bitstreams such that each operation takes all hardware resources [198, 744]. However, it is necessary to reconfigure the hardware to switch between operations. Frequent runtime reconfiguration may take a considerable amount of execution time.

We design a hardware block that works for all backpropagation operations. Our objective is to allow different operations to share as many arithmetic facilities as possible. We use a command sequence to dynamically switch between operations by adjusting the behavior of a small subset of arithmetic units without incurring runtime reconfiguration. Figure 6.2 shows the top-level diagram of the architecture, which supports both linear combination and function evaluation. Major components include the buffer crossbar and the arithmetic block.

- The buffer crossbar communicates with two buffers. At any time, the buffer control signal from the command specifies a source buffer and a target buffer. The crossbar

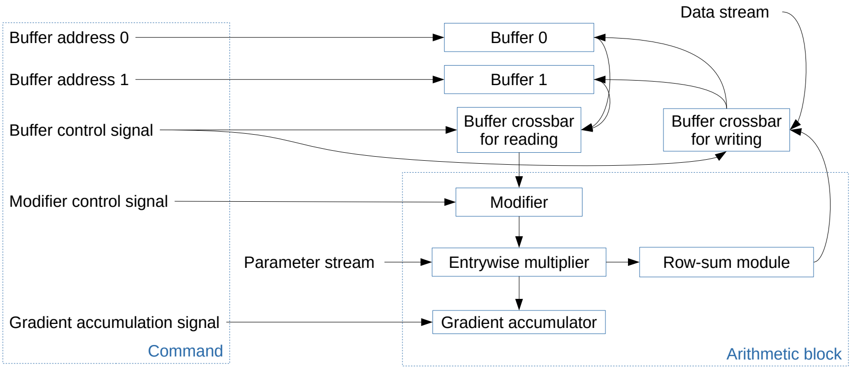


Fig. 6.2: Top-level diagram of the backpropagation engine.

reads a vector from the source vector and passes the vector to the arithmetic block. The crossbar also accumulates the output vector from the arithmetic block to the target buffer.

- The arithmetic block extends the multifunctional multiplication block proposed in [278]. The modifier in the arithmetic block corresponds to the overrider in [278]. This block has two execution modes: the linear mode, which multiplies a matrix by a vector, and the function evaluation mode, which evaluates a non-linear function for all elements in the vector. Our extension includes the gradient accumulator and the row-sum module. A binary signal from the command controls whether the multiplier accumulates the entry-wise products to the gradient. The entrywise multiplier feeds its results to a row-sum module which calculates the sum of each row in the linear mode.

The arithmetic module switches to the linear mode for linear combination. The core calculation for a linear combination operation is to evaluate a vector of weighted sums using a b-dimensional input vector $\mathbf{x}$ and a $b \times b$ weight matrix W. We evaluate an approximate version in the form of a piecewise linear function.

The architecture addresses two challenges in Section 1. First, the components and their connections are independent of the operation. As a result, it is unnecessary to perform a runtime reconfiguration when the operation changes, which addresses the challenge of diverse arithmetic operations. Second, the resource usage is independent of the layout of the network. Therefore, it is possible to scale the architecture for different hardware platforms to control cost or power consumption, which addresses the challenge of hardware adaptability.

One may follow the design flow illustrated in Figure 6.3 to apply the architecture to a backpropagation task. The design flow includes hardware customization and command sequence generation. Hardware customization is the process to set two design parameters. One design parameter is the batch size b, which determines the size of

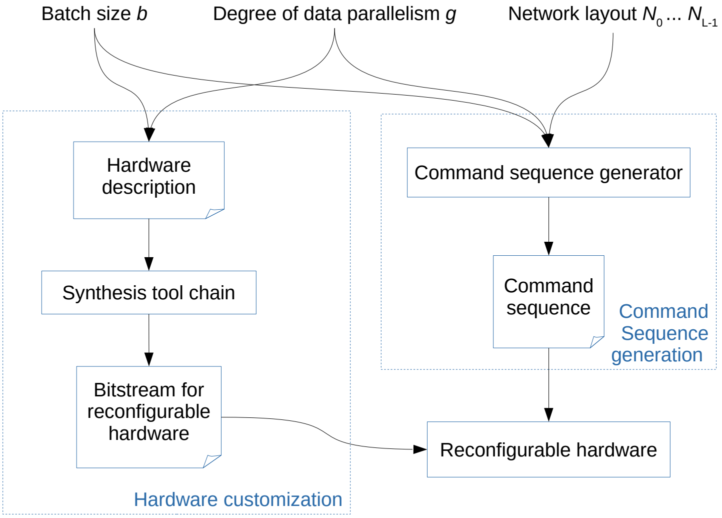

Batch size b
Degree of data parallelism g
Network layout $N_0 \ldots N_{L-1}$
Hardware description
Synthesis tool chain
Bitstream for reconfigurable hardware
Hardware customization
Command sequence generator
Command sequence
Command Sequence generation
Reconfigurable hardware

Fig. 6.3: Design flow.

Tab. 6.1: Memory traffic (number of data entries per command).

Operation	ME	Inward	Outward
Data load	0	bg	0
Linear combination for forward pass	0	b^2	0
Function evaluation for forward pass: batch 0..(U – 2)	1	b^2	0
Function evaluation for forward pass: batch (U – 1)	1	b^2	b
Linear combination for backward pass: node error	0	b^2	0
Linear combination for backward pass: gradient output	0	b	b^2
Function evaluation for backward pass	0	0	0

the entrywise multiplier. A larger b allows the entrywise multiplier to process more multiplications in parallel for a single data point. The other parameter is the degree of data parallelism g, which determines the number of data points processed in parallel. After customization, the architecture has g arithmetic blocks. Each arithmetic block contains a modifier, an entrywise multiplier, and a row-sum module. Each entrywise multiplier includes $b \times b$ scalar multipliers. After filling the design parameters to the hardware description, it is possible to generate a bitstream to program the reconfigurable hardware using the synthesis toolchain. Command sequences generation is the process that produces a sequence of commands from the layout of the network.

The memory bandwidth usage of the hardware depends on the operation and the hardware parameters. For ease of discussion, we assume that all data entries in the feature matrix, network parameters, and gradients have the same width. We may measure the memory traffic by the number of data entries transmitted per command. Table 6.1 summarizes the memory traffic for different operations.

6.1.4 Collaboration of Components

In this section, we explain how the components collaborate to execute different operations in backpropagation. The modifier in the arithmetic block determines whether the system performs linear combination or non-linear function evaluation.

- The modifier switches to the linear mode for linear combination. The core calculation for a linear combination operation is straightforward. In particular, the arithmetic block evaluates a vector of weighted sums using a b-dimensional input vector $\mathbf{x}$ and a $b \times b$ weight.
- The modifier switches to the function evaluation mode for non-linear function evaluation. Rather than evaluating an activation function following its mathematical definition, the arithmetic block evaluates an approximate version in the form of a piecewise linear function.

Before running the hardware, it is necessary to define a list of commands to control the hardware architecture. We briefly discuss a set of commands that can be used to perform backpropagation in a straightforward manner. This command set addresses the challenge of complex control logic discussed in Section 1. We first describe two commands that read and write the same buffer including data load and memory reset. We then present the commands for linear combination and function evaluation where the arithmetic block read and write different buffers.

The architecture supports two commands that operate on a single buffer. The first command sets the addressed location in the target buffer to zero. As the arithmetic block always accumulates to the target buffer, it is necessary to initialize N_l entries in the target buffer to zero to ensure correct calculation. A command to reset a memory location needs to set the source buffer to be the same as the target buffer and point both addresses to the location to reset. The parameter memory provides a $b \times b$ negative identity matrix. With these settings, the output of the row-sum module is the opposite of the original value $-\mathbf{x}_t$. The accumulation of the value back to the target memory location resets the content to zero. The second command loads d dimensions from g data points to the target location. The memory crossbar directly reads a data point from the data stream, ignoring the output of the arithmetic block.

The other two commands operate on two buffers. The first command is for the linear combination operation. In each batch, the arithmetic block takes b signals as the input and begins to propagate the signals to b nodes in the adjacent layer in parallel. The second command is for the function evaluation operation. Each modifier takes a copy of the variable and evaluates the piecewise linear function that approximates the activation function.

Tab. 6.2: Resource usage.

Resource	Total	Used	Percentage
Logic utilization	1 182 240	382 256	32.33 %
DSP blocks	6840	4105	60.01 %
Block memory (BRAM18)	4320	1292	29.91 %
Block memory (URAM)	960	150	15.63 %

6.1.5 Evaluation

We empirically evaluate the architecture in this section by comparing an FPGA implementation of the architecture and the PyTorch machine learning platform on a dual-CPU workstation.

6.1.5.1 Experiment Settings

We compare our architecture running on an FPGA-based acceleration card with a CPU implementation running on a multicore CPU. The architecture runs on a Xilinx UltraScale+ VU9P FPGA with 16 nm technology. We run the FPGA chip at 120 MHz. The architecture executes command sequence to $g = 16$ data instances in parallel with dimensional batch size $b = 8$. Table 6.2 shows the resource usage of the implementation. The software implementation runs on the PyTorch 1.0 machine learning platform running on a workstation with two Intel Xeon E5-2643 v4 CPUs and 128 GB DDR4 memory. The process technology of the CPUs is 14 nm, which is slightly more advanced than the FPGA. The workstation has 12 physical cores supporting 24 threads in total. The base frequency of the CPU cores is 3.4 GHz, and the maximum turbo frequency is 3.8 GHz.

We consider two representative types of network layouts in the experiments. We call them *bucket-shaped* networks and *cone-shaped* networks respectively for ease of discussion. In a "bucket-shaped" network, all layers contain an identical number of nodes. These networks usually appear in stand-alone classifiers, generative models, and reinforcement learning. In a "cone-shaped" network, a hidden layer has no more nodes than its previous layer. These networks learn compressed features and representation as each layer introduces information loss in a controlled manner.

Table 6.3 shows the test cases we designed using network layouts similar to those in real-world applications. We test two activation functions–rectifier linear function (relu) and the hyperbolic tangent function (tanh)–for each layout. Due to the alignment requirement of our hardware platform, we round the size of each layer to the next multiple of 32. We also produce challenging test cases for each application by linearly scaling the size of all layers. Specifically, the design of test cases is as follows.

- The experiments with "bucket-shaped" networks include 12 test cases. Test cases B0 and B1 correspond to the network structure for reinforcement learning in [276]. The network has 2 hidden layers with 200 nodes in each layer. Test cases B2 and B3 correspond to a study of traffic-flow prediction [454]. The network with the largest layer size contains 3 layers of hidden nodes with 400 nodes in each layer. Test cases B4 and B5 correspond to the network in the generative adversarial networks in [23]. The network contains 4 hidden layers with 512 nodes in each layer. Test cases B6–B11 are challenging versions for B0–B5, where the layer size of each case is 8 times that of the original version.

- The experiments with "cone-shaped" networks include 8 test cases. Test cases C0 and C1 correspond to the stacked autoencoder in [713]. The network has two hidden layers containing 400 and 225 hidden nodes, respectively. Test cases C2 and C3 correspond to the denoising autoencoder for speech data recognition in [221]. The network contains two hidden layers, one with 1000 nodes and another with 500. Test cases C4–C7 are challenging versions for C0–C3, where the layer size of each case is 4 times that of the original version.

We use randomly generated data and network parameters to test the efficiency of the system. Assuming that the function evaluation procedure takes the same time for different inputs, the total execution time for each backpropagation process is independent of the data distribution and the network parameters. In other words, given the same data size, the total execution time should stay unchanged regardless of the data source. As a result, using randomly generated data and network parameters facilitates experiments with various data sizes without affecting observations. The number of data instances for each test case is 2^{20}. In each test case, we calculate the gradient with respect to 100 sets of weights.

6.1.5.2 Results and Discussion

Table 6.3 records the experimental results. In this table, the benchmark column records the numbers of data instances; the 'CPU-1T', 'CPU-4T', and 'CPU-8T' columns contain the execution times in seconds for the corresponding implementation. The 'SU' columns give the speedup of the FPGA implementation over the CPU with 1 thread, 4 threads, and 8 threads, respectively.

The architecture discussed in this contribution is faster than the CPU system in all but one test case. In the tests with bucket-shaped networks, the architecture achieves up to a 9.4, 5.4, and 5.2 times speedup compared with the software reference on 1, 4, and 8 threads. In the tests with cone-shaped networks, the architecture achieves up to a 7.8, 4.6, and 4.7 times speedup compared with the software reference on 1, 4, and 8 threads. In addition to the overall speed advantage of the architecture, we have the following additional observations:

1. The software implementation scales poorly with the number of threads. The software running 4 threads achieves only around 2 times speedup against a single thread. The speed advantage on 8 threads over 4 threads is insignificant. In test

Tab. 6.3: Execution time (seconds) and speedup.

ID	Layers	Activation	CPU-1T	CPU-4T	CPU-8T	FPGA	SU-1T	SU-4T	SU-8T
B0	224x2	tanh	0.285	0.219	0.155	0.071	4.0	3.1	2.2
B1	224x2	relu	0.181	0.104	0.090	0.071	2.5	1.5	1.3
B2	416x3	tanh	0.731	0.356	0.274	0.105	7.0	3.4	2.6
B3	416x3	relu	0.521	0.202	0.162	0.104	5.0	1.9	1.6
B4	512x4	tanh	1.020	0.504	0.430	0.138	7.4	3.7	3.1
B5	512x4	relu	0.786	0.320	0.235	0.137	5.7	2.3	1.7
B6	1792x2	tanh	4.102	2.493	2.109	0.615	6.7	4.1	3.4
B7	1792x2	relu	3.715	2.163	1.820	0.615	6.0	3.5	3.0
B8	3328x3	tanh	21.758	13.120	13.176	2.576	8.4	5.1	5.1
B9	3328x3	relu	20.827	13.783	12.400	2.574	8.1	5.4	4.8
B10	4096x4	tanh	45.501	24.847	25.286	4.822	9.4	5.2	5.2
B11	4096x4	relu	44.755	24.036	24.320	4.817	9.3	5.0	5.0
C0	416,256	tanh	0.179	0.125	0.167	0.089	2.0	1.4	1.9
C1	416,256	relu	0.137	0.086	0.102	0.087	1.6	1.0	1.2
C2	1024,512	tanh	0.577	0.339	0.421	0.169	3.4	2.0	2.5
C3	1024,512	relu	0.494	0.266	0.299	0.169	2.9	1.6	1.8
C4	1664,1024	tanh	2.092	1.189	1.259	0.375	5.6	3.2	3.4
C5	1664,1024	relu	2.121	1.104	1.021	0.375	5.7	2.9	2.7
C6	4096,2048	tanh	13.467	7.997	8.147	1.732	7.8	4.6	4.7
C7	4096,2048	relu	13.207	7.643	7.636	1.725	7.7	4.4	4.4

cases B10 and B11, the software running on 8 threads is even slower than when running on 4 threads.

2. The software is more sensitive to the selection of the activation function than the architecture. With the same network layout, the software tends to be faster with the rectifier linear function than with the hyperbolic tangent function. The speed advantage is especially significant when the software runs on 4 and 8 threads. This observation confirms the speed advantage of the rectifier linear function on CPUs [255]. By contrast, the architecture on FPGA evaluates any activation using the same number of commands. Therefore, the execution time of the architecture for a given layout is independent of the activation function.
3. The experimental implementation achieves slightly higher speedup with larger networks that carry more network parameters. For instance, in the experiments with bucket-shaped networks with the ‘tanh’ function, the speedup over 8 threads rises from 2.2 to 5.2 while the layer size expands from 224 to 4096. An exception of the observation is that architecture achieves high speedup against a single CPU thread in test case B0. A possible reason for the exception is that artifacts such as memory initialization for both software and hardware take significant time when the network is small. In this case, the comparison is less reliable than it is in other tests.

Besides the observations above, we have two conjectures based on the hardware design and the experimental results. First, the speed of the architecture will grow if more DSP blocks are available. The arithmetic block contains b^2g scalar multipliers in parallel. Our synthesis tool implements these multipliers mainly with DSP blocks. Therefore, DSP blocks become the critical resource for the design, as shown in Table 6.2. The number of data points processed in parallel grows linearly with the number of multipliers. As a result, when more DSP blocks are available, we may set a larger g to deploy more multipliers to improve the speed. Second, given the same set of hardware resources, our architecture can process networks with more nodes and parameters than some existing solutions such as [558] and [522] for two reasons. One reason is that the number of multipliers is independent of the network layout. The other reason is that the on-chip memory only needs to keep the activation and error signals for two adjacent layers.

6.1.6 Conclusions

We presented a hardware architecture to perform backpropagation for training multilayer perceptrons. The key to acceleration is to reuse the same set of hardware resources to process different operations involved in backpropagation. Our architecture does not incur runtime reconfiguration when switching between operations. The hardware resource usage is independent of the network layout. A prototype implementation of the architecture on a Xilinx UltraScale+ VU9P FPGA achieves up to 5.2 times speedup over PyTorch running on 8 threads on a workstation with two Intel Xeon E5-2643 v4 CPUs.

6.2 Processor-Specific Code Transformation

Henning Funke
Jens Teubner

Abstract: During the last decade, the compilation of database queries to machine code has emerged as a very efficient alternative to classical, interpretation-based query processing modes [529]. Compiled code can better utilize advanced features of modern CPU instruction sets; avoid interpretation overhead; and—most importantly—minimize data I/O (*e.g.*, to main memory).

This success story raises the hope that compilation strategies can be lifted to non-standard architectures, such as GPUs or other accelerators, as well as to support other data-intensive processing tasks. However, as we shall see in this section, the data-parallel nature of the devices is at odds with established techniques in query compilation, resulting in massive resource under-utilization if compilation strategies are applied too naively.

As a remedy, we propose two novel mechanisms that re-establish compute efficiency of compiled code on data-parallel hardware: *Lane Refill* and *Push-Down Parallelism* are "virtual operators" that participate in optimization and code generation just like true query operators (making our approach seamlessly integrate with existing systems). At runtime, they compensate for lurking resource under-utilization by adapting parallelization strategies on-the-go. The outcome is a resource utilization that is close to the hardware's maximum, while causing negligible overhead even in unfavorable situations.

Lane Refill and *Push-Down Parallelism* are part of our compiler platform *DogQC*, which leverages modern graphics processors for efficient database query processing.

6.2.1 Data-Parallel Processing Models

Data-parallel processing models are a particularly promising way to max out the achievable compute performance within the constraints of hardware technology (power and heat dissipation). Instead of dedicating chip resources to control flow management, data-parallel architectures target throughput. For instance, executing an instruction for 32 fields at a time can reduce the control flow management work by a factor of 32, when compared with a scalar execution.

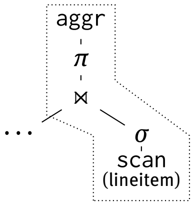


Fig. 6.4: Plan excerpt.

6.2.1.1 Divergence in Data-Parallel Architectures

GPUs are a popular incarnation of this idea, and spectacular performance results have been reported in various application domains. However, actually leveraging the available hardware resources in a beneficial way can be challenging. *Divergence effects*, which may arise whenever data is not perfectly regular, may compromise the benefits.

In this section, we will look at mechanisms to combat performance penalties that may result from divergence effects. To understand the divergence problem, let us consider the execution of a database query, as illustrated here in Figure 6.4 for Query Q10 from the TPC-H benchmark set. A query compiler will attempt to compile the plan region ⬚ into a straight-line sequence of code, *i.e.*, a *pipeline*. The motivation to do so is to propagate tuples within registers rather than spilling data to (slow) memory.

During execution, not all lineitem tuples will actually traverse the full pipeline. Some tuples might instead be *eliminated* by operators such as filter σ or join $\bowtie$. If this happens, a sequential processor will immediately abort the pipeline, continue with the next input item, and hence keep CPU efficiency at peak.

Data-parallel execution back-ends, by contrast, do not have the option of aborting a pipeline early, unless *all* tuples in the same batch of work are eliminated.

Figure 6.5 illustrates this effect for a GPU-based back-end (assuming a batch—or "*warp*"—size of eight for illustration purposes). In some warp iteration, only *warp lanes* 1, 5, and 7 might have passed the filter σ, leaving the five remaining warp lanes *inactive* (indicated as dashed arrows - - ›). The following join de-activates another two warp lanes, bringing GPU efficiency down to 1/8 in this example.

The resulting GPU under-utilization is even worse in real settings. To scan a lineitem table with 150 million rows, actual GPUs will require 5 million *warp iterations*, each consisting of 32 warp lanes. Although σ filters out about 2/3 of all rows, it is extremely unlikely that all lanes within a warp become inactive. Therefore, (almost) all 5 million warp iterations proceed into the join operator $\bowtie$. Only 1 % of the remaining rows find a match during the join. In an actual dataset, 2.9 million rows remain after the join, but they are spread across 1.1 million warp iterations. Ideally, the projection π and aggregation aggr operators could have been processed by only $2.9\,\text{M}/32 = 90\,\text{K}$ warp iterations. In other words, state-of-the-art query compilation techniques will leave 92 % of the GPU's processing capacity unused.

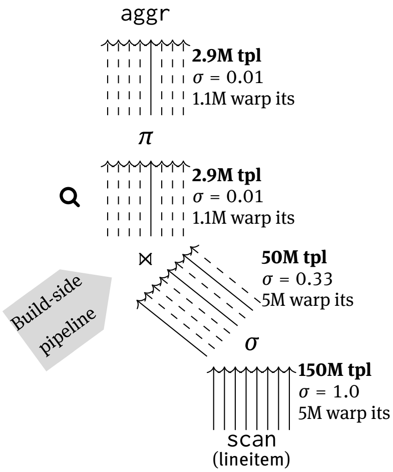


Fig. 6.5: GPU under-utilization due to filter divergence.

6.2.1.2 *DogQC*: A Database Query Compiler for GPUs

GPU code generated by our query compiler *DogQC*[1] leverages *Lane Refill* and *Push-Down Parallelism* techniques to counter divergence effects like the ones illustrated above. In the rest of this section, we will give a high-level idea of the *Lane Refill* and *Push-Down Parallelism* techniques (Sections 6.2.2 and 6.2.3), then report on experimental results for DogQC (Section 6.2.4), and wrap up in Section 6.2.5. More details on the *Lane Refill* and *Push-Down Parallelism* mechanisms can be found in the respective full paper [237].

6.2.2 *Lane Refill* Technique

Divergence effects (here: *filter divergence*) are a consequence of the SIMT ("single instruction, multiple threads") execution paradigm embodied in all modern graphics processors. A number of threads (or *lanes*, typically 32 of them) are grouped into a *warp*. During execution, *all* lanes within a warp execute the *same* GPU instruction.

The SIMT model encounters a problem whenever some lanes or data elements need a different amount or kind of processing than others. In such situations, control flows will *diverge*. Since all lanes within a warp *still* execute the same instruction, lanes will be turned *inactive* and their computation result will be discarded. As illustrated above, this can result in resource under-utilization.

To illustrate the severity of this effect, we instrumented the query plan shown earlier (Figure 6.5) to monitor warp utilization at the plan point marked with a magnifying glass 🔍. Figure 6.6 shows a histogram on the number of warps that have passed this

[1] https://github.com/Henning1/dogqc.

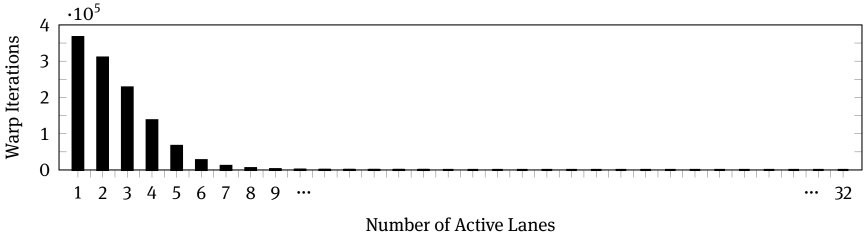


Fig. 6.6: Lane activity profile with filter divergence.

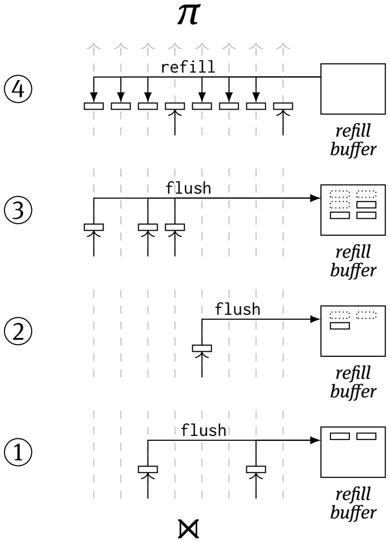


Fig. 6.7: *Lane Refill*: tuples from three low-activity iterations are suspended to the *refill buffer* and resumed for full lane activity in the fourth iteration.

plan stage with a warp utilization of 1, . . . , 32 active lanes. It is easy to see that only a fraction of the available compute capacity is used; in most warps, only one or two out of 32 warp lanes performed actual work.

6.2.2.1 Balance Operators and Refill Buffers

To combat the situation, DogQC injects *balance operators* into the relational query plan. Code generated for these operators detects warp under-utilization at runtime. Whenever utilization drops below a configured threshold, the state of all remaining active lanes is suspended to a *refill buffer* and the pipeline starts over with a fresh set of input tuples.

Figure 6.7 illustrates this for three successive warp iterations ➀ through ➂. Since only 2, 1, and 3 lanes remained active in these iterations (respectively), their state is

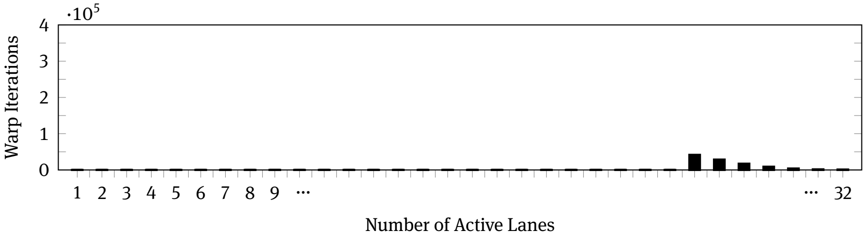


Fig. 6.8: Lane activity profile with lane refill buffer to consolidate filter divergence.

flushed to the refill buffer. After flushing, each of those warp iterations is terminated and processing starts over with the next set of input tuples.

6.2.2.2 Refilling

As soon as a sufficient number of lane states have been stored to the refill buffer, the buffer can be used to *refill* lanes that have become inactive. This time, the under-utilized warp iteration is not terminated but continues processing with full utilization after refilling. This is visualized in Step ④ of Figure 6.7. Here, only two out of eight warp lanes remained active after the downstream join operator. Using the refill buffer, the remaining six warp lanes can be filled with useful work, resulting in full warp utilization upstream.

Implementationwise, flushing and refilling are backed up in DogQC by CUDA's `__ballot_sync`, `__popc` ("population count"), and shuffling primitives. These primitives are highly efficient; balance operators will cause little overhead even when only a few warps go below the utilization threshold.

6.2.2.3 Effect of *Lane Refill*

Lane Refill brings warp utilization back to a high compute efficiency. Following the balancing operator, all executed warps (except for the last warp in each grid block) are *guaranteed* to have a warp utilization above the configured threshold.

In Figure 6.8, this is illustrated with a histogram for the same plan point that we profiled earlier (Figure 6.6), but this time with a balance operator applied. The histogram confirms that *(a)* (almost) no warps exist with a utilization below 26 lanes (the threshold we configured); and *(b)* the total number of executed warps has dropped by a factor of about 10. In terms of overall execution performance, *lane refill* will improve execution times by about 2 – 3× for the example plan shown in Figure 6.5.

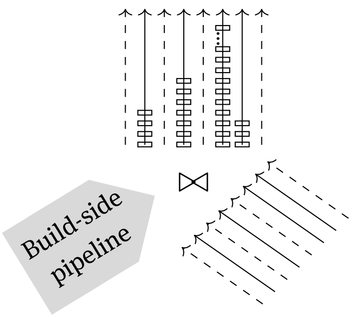


Fig. 6.9: Expansion divergence. Here, some rows in the probe-side relation will find more join partners on the probe side than on the other side.

6.2.3 *Push-Down Parallelism* Technique

DogQC's *Push-Down Parallelism* technique addresses another flavor of divergence that may arise orthogonally to the aforementioned filter divergence. *Expansion divergence* is the effect when a different amount of work is needed to process each of the items within a warp. Database *join operations* are a common situation where this effect arises. Figure 6.9 on the right illustrates the effect. Probe side tuples coming from the right may find a different number of join partners each. Specifically, in the example, lane 6 will have significantly more tuples to process than the remaining warp lanes. In such a situation, existing query compilers will process all matches of a single probe-side tuple *within* the same warp lane. In the example, execution times would be dominated by the sequential processing of all matches for lane 6.

Push-Down Parallelism mitigates the situation by parallelizing the processing of the matches of a single probe-side tuple *across* the available warp lanes. To this end, the execution state of probe-side lanes is *broadcast* over lanes, as illustrated in Figure 6.10. Build-side matches are *partitioned* across. Again, we leverage efficient CUDA primitives, such as `__ballot_sync` and `__shfl_sync` ("shuffle sync"). Please refer to [237] for details.

As illustrated in Figures 6.11 and 6.12, *Push-Down Parallelism* improves lane utilization and reduces the overall number of iterations needed to complete the query. *Lane Refill* and *Push-Down Parallelism* complement one another, and Figure 6.9 shows an example where both flavors of divergence co-exist. Another typical occurrence of expansion divergence is the processing of *variable-length data*, strings in particular. If possible, DogQC will parallelize the processing of strings across warp lanes to improve resource utilization.

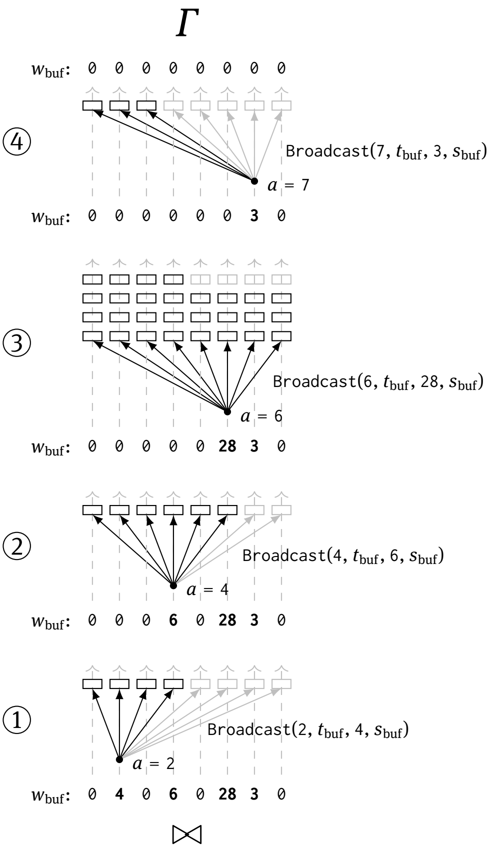

Γ
w_{buf}: 0 0 0 0 0 0 0 0
④
Broadcast(7, t_{buf}, 3, s_{buf})
a = 7
w_{buf}: 0 0 0 0 0 0 3 0
③
Broadcast(6, t_{buf}, 28, s_{buf})
a = 6
w_{buf}: 0 0 0 0 0 28 3 0
②
Broadcast(4, t_{buf}, 6, s_{buf})
a = 4
w_{buf}: 0 0 0 6 0 28 3 0
①
Broadcast(2, t_{buf}, 4, s_{buf})
a = 2
w_{buf}: 0 4 0 6 0 28 3 0
⋈

Fig. 6.10: Illustration of push-down parallelism that expands the join matches of four warp lanes.

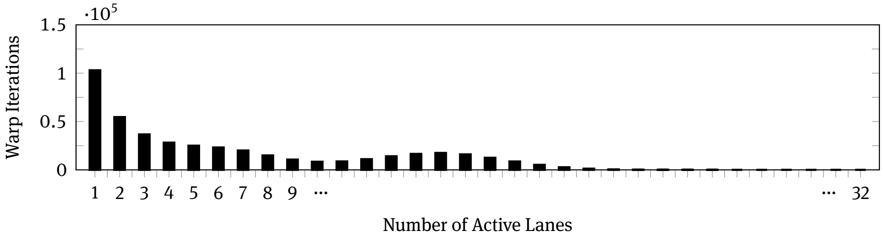


Fig. 6.11: Lane activity with expansion divergence.

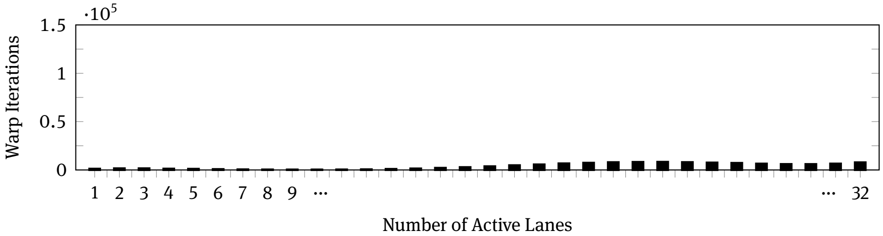


Fig. 6.12: Lane activity profile with push-down parallelism to consolidate expansion divergence.

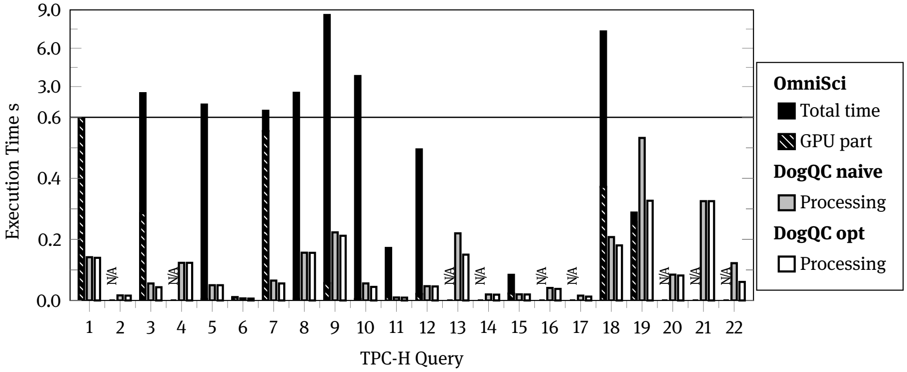


Fig. 6.13: Execution times of DogQC for TPC-H benchmark queries (scale factor 25). The divergence optimizations improve query performance.

6.2.4 Evaluation

With *DogQC*, we provide a query compiler with a wide range of SQL functionality, sufficient to support all queries from the TPC-H benchmark set. Here we use DogQC and the database domain to illustrate the aforementioned anti-divergence mechanisms, which could equally be applied to other data-intensive tasks, including those related to machine learning.

6.2.4.1 TPC-H Performance

To assess the benefits of measures to contain divergence, we performed a series of measurements with the TPC-H benchmark set. Our measurements were based on an NVIDIA RTX2080 GPU with 46 Streaming Multiprocessors and 8 GB GPU memory, installed in a host system with an Intel i7-9800X GPU and 32 GB of main memory. As a reference, we compared DogQC with the hybrid CPU/GPU system *OmniSci* [545].

Our benchmark results are depicted in Figure 6.13. For each of the 22 TPC-H queries, the bars indicate the query execution time assuming that the dataset is resident in GPU memory.

For OmniSci, we report the total wall clock time needed to execute the query as well as the amount of time spent on GPU processing. OmniSci is a hybrid execution engine in which both CPU and GPU will be used to jointly answer the query. As can be seen in Figure 6.13, several queries can, in fact, not benefit much from GPU in OmniSci. Also note that OmniSci could successfully execute only 13 of the 22 TPC-H benchmark queries. DogQC, by contrast, can run all 22 TPC-H queries entirely on the GPU, with execution times that are up to 86× faster than those of OmniSci.

6.2.5 Summary

In this research, we put the processing capabilities of data-parallel co-processors for non-uniform, data-intensive workloads to the test. DogQC introduces techniques that allow us to gracefully align parallel processing units with work items, even when problems are heavily skewed. We observe that *Lane Refill* and *Push-Down Parallelism* are able to increase processing efficiency for these non-uniform workloads, sometimes with dramatic effects on processing throughput.

Existing query coprocessors typically avoid imbalances by working on a uniform surrogate (such as dictionary keys or materialization barriers). This has led to the perception that GPUs have limited capabilities of processing irregular problems. DogQC avoids the overhead of maintaining such additional data structures and instead restores balance during non-uniform processing.

Here we showcase *Lane Refill* and *Push-Down Parallelism* based on an application to database query processing. Compared with state-of-the-art platforms, our prototype DogQC achieves better resource utilization, a bigger functionality range, and better runtime performance on realistic benchmarks. Looking ahead, our anti-divergence measures could be applicable to many machine learning scenarios, especially when the problems involved are heavily skewed and/or depend on non-linear computations.

6.3 Extreme Multicore Classification

Erik Schultheis
Rohit Babbar

Abstract: There are classification problems, such as assigning categories to a Wikipedia article, where the possible set of labels is very large, numbering in the millions. Somewhat surprisingly, these so-called Extreme-Multilabel Classification (XMC) problems can be solved quite successfully by applying a linear classifier to each label individually. This decomposition into binary problems is called a one-vs-rest reduction. As these problems are completely independent, the reduced task is embarrassingly parallel and can be trivially spread across multiple cores and nodes. After training, the model can be sparsified by culling small weights to only require a fraction of the memory and computational power for prediction on new samples.

6.3.1 Introduction to Extreme Multilabel Classification

Extreme Multi-label Classification (XMC) refers to supervised learning with a large target label set where each training/test instance is labeled with a small subset of relevant labels. Machine learning problems consisting of hundreds of thousands of labels are common in various domains such as annotating web-scale encyclopedias [585], hashtag suggestion in social media [171], and image-classification [168]. For instance, all Wikipedia pages are tagged with a small set of relevant labels that are chosen from more than a million possible tags in the collection. It has been demonstrated that, in addition to automatic labelling, the framework of XMC can be leveraged to effectively address learning problems arising in recommendation systems, ranking, and web-advertising [9, 585].

Notation and Setup Let the training data $D := \left\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})\right\}$ consist of input feature vectors $\mathbf{x}^{(i)} \in \mathcal{X} \subseteq \mathbb{R}^d$ and respective output vectors $\mathbf{y}^{(i)} \in \mathcal{Y} := \{0, 1\}^m$ such that $y_l^{(i)} = 1$ iff the l-th label belongs to the training instance $\mathbf{x}^{(i)}$. The feature vectors form the rows of the feature matrix $\mathbf{X}$. In XMC settings, the cardinality m of the set of target labels, the dimension of the input d, and the size of the dataset N can all be of the order of hundreds of thousands or even millions.

For text data, the input can be represented by term-frequency inverse-document-frequency (tf-idf) features. In that case, the dimensionality of the feature space is determined by the size of the vocabulary, and for each text $\mathbf{x} \in \mathcal{X}$ the feature x_j is nonzero only if the corresponding word appears in the text. As a result, the input features are highly sparse. The magnitude of the feature is determined by how often

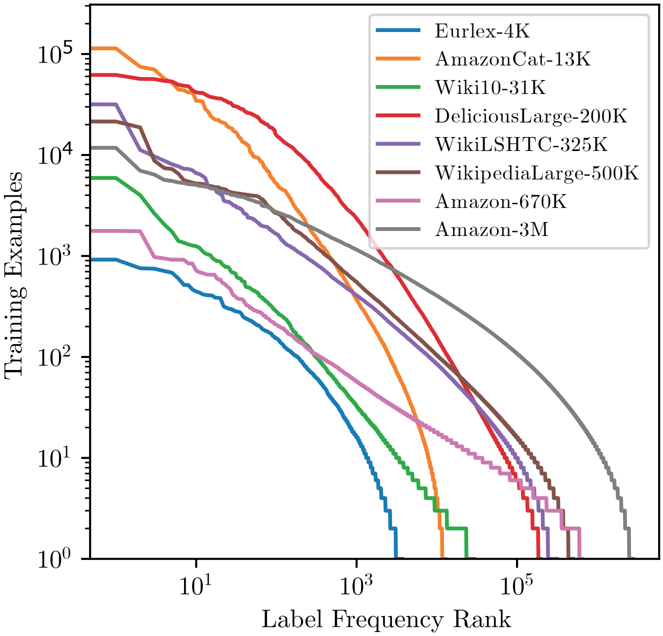


Fig. 6.14: Label frequency in XMLC datasets. X-axis shows the label rank when sorted by the frequency of positive instances and Y-axis gives the number.

the word appears in the document and in the entire corpus. For details on tf-idf, see, e.g., Manning, Raghavan, and Schütze [463].

Similarly, for any given instance $\mathbf{x}^{(i)}$ only a small subset of the labels will be relevant, $\|\mathbf{y}^{(i)}\|_1 \ll m$. Additionally, the number of instances for which a label is relevant is very imbalanced: Few labels will be relevant to many instances, but most labels will apply only to an extremely small fraction. This gives rise to a *long-tailed* label distribution, as shown in Figure 6.14. The labels with very few positives are called *tail labels*. The characteristics of well-known benchmark datasets in XMC are presented in Table 6.4.

In traditional multi-label classification, the goal is to learn a multi-label classifier in the form of a vector-valued output function $h : \mathbb{R}^d \mapsto \{0, 1\}^m$. In XMC, one often wants to restrict the classifier to predict a fixed number of labels because, say, a web interface might have a fixed number of slots in which to suggest related searches. This leads to classification functions $h_k : \mathbb{R}^d \mapsto \mathcal{Y}_k := \{\mathbf{y} \in \mathcal{Y} : \|\mathbf{y}\|_1 = k\}$. Such a function is typically constructed by first learning a score function $r : \mathbb{R}^d \mapsto \mathbb{R}^m$, and then taking the k highest-scoring labels as the prediction.

Evaluation Metrics Due to the extreme sparsity in the label vector, metrics such as accuracy are not informative in the case of XMC. Instead, one typically uses metrics that

Tab. 6.4: Multi-label datasets from XMC repository [50]. APpL and ALpP represent average points per label and average labels per point, respectively.

Dataset	#Training	#Features	#Labels	APpL	ALpP
AmazonCat-13K	1 186 239	203 882	13 330	448.6	5.0
AmazonCat-14K	4 398 050	597 540	14 588	1 330.1	3.5
Amazon-670K	490 449	135 909	670 091	4.0	5.5
Wiki10-31K	14 146	101 938	30 938	8.5	18.6
Delicious-200K	196 606	782 585	205 443	72.3	75.5
WikiLSHTC-325K	1 778 351	1 617 899	325 056	17.5	3.2
Wikipedia-500K	1 813 391	2 381 304	501 070	24.8	4.8

focus on the k predicted labels. Most commonly used are precision at k, denoted P@k, and normalized Discounted Cumulative Gain, denoted nDCG@k [50]. Let $\mathbb{R}^m \ni \hat{\mathbf{y}} = r(\mathbf{x})$ be the predicted scores for an instance with corresponding label vector $\mathbf{y}$. These metrics are defined by

$$\mathrm{P@}k(\mathbf{y}, \hat{\mathbf{y}}) := \frac{1}{k} \sum_{l \in \mathrm{rank}_k(\hat{\mathbf{y}})} y_l \tag{6.4}$$

$$\mathrm{nDCG@}k(\mathbf{y}, \hat{\mathbf{y}}) := \sum_{l \in \mathrm{rank}_k(\hat{\mathbf{y}})} \frac{y_l}{\log(\mathrm{R}_l(\hat{\mathbf{y}}) + 1)} \Big/ \sum_{l=1}^{\min(k, \|\mathbf{y}\|_1)} \frac{1}{\log(l + 1)}, \tag{6.5}$$

where $\mathrm{rank}_k(\mathbf{y})$ returns the k largest indices of $\mathbf{y}$ ranked in descending order, and R_l gives the ordering of the l'th index. Note that unlike P@k, nDCG@k takes into account the ranking of the correctly predicted labels. For instance, if there is only one of the five labels that is correctly predicted, then P@5 gives the same score if the correctly predicted label is at rank 1 or rank 5. By contrast, nDCG@5 gives a higher score if it is predicted at rank 1 and the lowest non-zero score at rank 5.

6.3.2 Parallel Training of Linear One-vs-Rest Models

The P@k (Equation 6.4) and nDCG@k (Equation 6.5) metrics introduced above are non-differentiable and thus not directly usable in typical gradient-based empirical-risk-minimization procedures. However, it can be shown that in order to achieve optimal predictions for precision at k, one only needs to train the scoring function r in such a way that the scores are strictly monotone transformations of the labels' marginals [486]. Therefore, one can train the classifier by independently applying a classification-calibrated[2] loss ℓ_{BC}, such as binary cross entropy or (squared) hinge loss, to each label

[2] See e.g. Bartlett, Jordan, and McAuliffe [44]. Intuitively, this means that a classifier that minimizes ℓ_{BC} also minimizes the binary 0-1 loss.

individually. Therefore, the training objective is given by

$$\min_{r} \sum_{i=1}^{N} \sum_{l=1}^{m} \ell_{\mathrm{BC}}\left(y_l^{(i)}, r_l\left(\mathbf{x}^{(i)}\right)\right). \tag{6.6}$$

Such a decomposition is called the *One-vs-Rest* (or One-vs-All) reduction.

Objective Functions for Linear One-vs-Rest This expression becomes particularly favorable if the scoring function r is linear. In that case, the minimization task decomposes into m completely independent subtasks

$$\forall l \in [m] : \min_{\mathbf{w}^{(l)} \in \mathrm{R}^d} \sum_{i=1}^{N} \ell_{\mathrm{BC}}\left(y_l^{(i)}, \mathbf{x}^{(i)\mathsf{T}}\mathbf{w}^{(l)}\right). \tag{6.7}$$

Due to the embarrassingly parallel nature of the training tasks, the computation can easily scale to use thousands of compute cores. A scalable implementation of this method yielding state-of-the-art prediction performance was demonstrated via the DiSMEC algorithm [30], which is a multi-label wrapper around the Liblinear solver [210]. In DiSMEC, the underlying binary loss is the squared hinge loss with an additional l_2 regularization term. Its objective is

$$\forall l \in [m] : \min_{\mathbf{w}^{(l)} \in \mathbb{R}^d} \left(\|\mathbf{w}^{(l)}\|_2^2 + c \sum_{i=1}^{N} \left(\max(0, 1 - s_l^{(i)}\mathbf{x}^{(i)\mathsf{T}}\mathbf{w}^{(l)})\right)^2 \right), \tag{6.8}$$

where $c \in \mathbb{R}_{>0}$ is the parameter to control the trade-off between empirical error and the model complexity and $s_l^{(i)} := 2y_l^{(i)} - 1$ is the label represented as $\{+1, -1\}$.

A similar method is ProXML [29], which switches the l_2 regularization for l_1 regularization in order to induce robustness to l_∞ perturbations in the input samples. This robustness is particularly helpful for tail labels, which have very few positive training instances. The objective of ProXML thus is

$$\forall l \in [m] : \min_{\mathbf{w}^{(l)} \in \mathbb{R}^d} \left(\|\mathbf{w}^{(l)}\|_1 + c \sum_{i=1}^{N} \left(\max(0, 1 - s_l^{(i)}\mathbf{x}^{(i)\mathsf{T}}\mathbf{w}^{(l)})\right)^2 \right). \tag{6.9}$$

Suppose for now that we have a method $\mathcal{A} : (\mathbf{X}, \mathbf{s}) \mapsto \mathbf{w}_\star^{(l)}$ available to solve these individual problems efficiently in a single thread. (This will be discussed in Section 6.3.3). Then the following framework can be used to scale the training process to multiple cores and nodes:

Two-Level Parallelization The distributed training for the optimization problems defined by equations (6.8) and (6.9) is implemented using a two-layer parallelization architecture. At the top level, labels are separated into batches of, say, $M = 1000$, which can be processed independently in parallel on available compute nodes, or sequentially if the number of batches exceeds the number of nodes. On each node, training of a batch of M labels is parallelized using multiple threads, which forms the second layer of parallelization.

After each $\mathbf{w}_\star^{(l)}$ is trained, weights of small magnitude are pruned to reduce overall model size drastically, often by more than 99 %. Since this can be performed as soon as $\mathbf{w}_\star^{(l)}$ has been computed, there is no need to store the complete dense model, even for a single batch, which reduces the RAM requirements for the algorithm. Unfortunately, in typical sparse matrix formats such as compressed sparse row/column matrices, insertion of new values cannot be done by multiple threads in parallel, because it might require reallocation and the shifting of data in other parts of the matrix. For this reason, we represent the sparse weight matrix as an array of independently allocated sparse vectors that can be written to independently.

The two-layer distributed training framework is summarized in Algorithm 4.

Algorithm 4: Framework for hardware-aware embarrassingly parallel training in DiSMEC and ProXML solvers. The iterations of both loops are independent and can thus be run in parallel.

```
Input: Training data $D = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}) \dots (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})\}$ in sparse representation,
       input dimensionality $d$, label set $\{1 \dots m\}$, batch size $M$
Output: Learnt matrix $\mathbf{W} \in \mathbb{R}^{d \times m}$ in sparse format
// 1st parallelization; independent nodes
for $\{b = 0;\ b < \lfloor \frac{m}{M} \rfloor + 1;\ b{+}{+}\}$ do
    Load single copy of feature matrix $\mathbf{X}$ into main memory
    Prepare array $\mathbf{W}_b$ of $M$ sparse vectors
    // 2nd parallelization; independent threads
    for $\{l = b \times M;\ l \leq (b + 1) \times M;\ l{+}{+}\}$ do
        Generate binary sign vector $\mathbf{s}^{(\ell)} = \{+1, -1\}_{i=1}^{N}$
        train weight vector $\mathbf{w}_\star^{(l)}$ on a single core using $\mathcal{A}(\mathbf{X}, \mathbf{s}^{(l)})$,
        Prune small weights in $\mathbf{w}^{(l)}$
    return $\mathbf{W}_{d,M}$
return $\mathbf{W}$
```

An advantage of the two-level parallelization over just running m instances of an off-the-shelf solver for binary problems is that the feature matrix $\mathbf{X}$ can be shared for all training jobs running on the same node. This allows us to keep the entire dataset, which may be several gigabytes in size, in main memory. However, on modern CPUs with a large number of cores, or on nodes with a two-socket configuration, this might cause problems due to *Non-Uniform Memory Access* (NUMA). In such a system, the overall RAM is partitioned into regions, called *NUMA domains*. Even though all cores in the

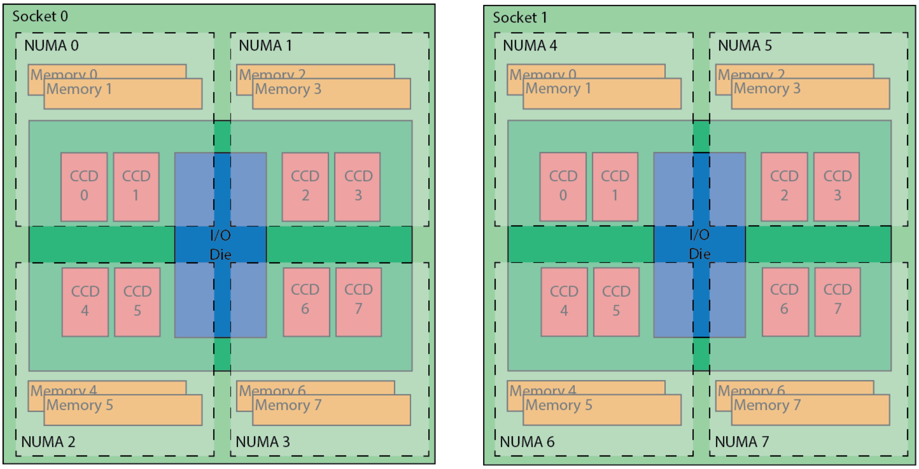


Fig. 6.15: NUMA memory in a dual-socket 64-core AMD Rome 7H12 system. Each CCD contains 8 cores, and each core has fastest access only to the memory within its own NUMA domain, marked with dashed lines. Image by CSC - IT Center for Science under CC-BY-4.0.

system can access the entirety of the memory, access latencies to the different domains vary depending on the distance of the core to the domain. An example of a NUMA setup is given in Figure 6.15.

In such a setup, a single copy of the feature matrix would be accessed by all cores on the system, quickly bottlenecking the memory bus and preventing the program from making efficient use of the available CPU cores. This can be mitigated by pinning threads to their CPU cores and replicating the feature matrix once per NUMA domain. In this way, each thread can read the feature matrix from its local domain, reducing latency, and the memory reads are spread out across different memory modules, improving throughput. In order to achieve this, the outer parallelization layer has to be performed not over physical nodes but over NUMA domains.

6.3.3 Second-Order Optimization Using Conjugate Gradients

The objective Equation 6.8 can be minimized in batch mode using second-order optimization. Compared with the popular (stochastic) gradient descent strategy, second-order optimization can take the curvature of the loss landscape into account and thus converges to the minimum in much fewer iterations. However, the computations for each single iteration are much more involved, as the second-order information is encoded in the potentially very large Hessian matrix. Fortunately, it is possible to implement this procedure without ever actually forming the Hessian, as will be described below.

Dropping the label index, we can write

$$\mathrm{R}_D[\mathbf{w}] = \mathbf{w}^\mathsf{T}\mathbf{w} + c\sum_{i=1}^{N} \ell_{\mathrm{SH}}\left(s_i\mathbf{x}^{(i)\mathsf{T}}\mathbf{w}\right), \tag{6.10}$$

where ℓ_{SH} is the squared hinge loss

$$\ell_{\mathrm{SH}}(r) = \max(0, 1 - r)^2. \tag{6.11}$$

Note that this objective function is convex. As a consequence, the optimizer will converge to the global optimum regardless of the starting point.

Determining the Descent Direction The main idea of second-order optimization is to approximate the objective locally using its quadratic Taylor approximation

$$\mathrm{R}_D[\mathbf{w} + \boldsymbol{\delta}] \approx \mathrm{R}_D[\mathbf{w}] + \nabla\mathrm{R}_D[\mathbf{w}]\boldsymbol{\delta} + 0.5\boldsymbol{\delta}^\mathsf{T}\nabla^2\,\mathrm{R}_D[\mathbf{w}]\boldsymbol{\delta}. \tag{6.12}$$

Therefore, the step $\boldsymbol{\delta}_\star$, which is ideal, i.e. which leads to the minimum, in this approximation can be calculated by solving the linear system

$$\nabla^2\,\mathrm{R}_D[\mathbf{w}]\boldsymbol{\delta}_\star = -\nabla\mathrm{R}_D[\mathbf{w}]. \tag{6.13}$$

For Equation 6.8, the gradient and Hessian have a simple structural form [239, 368]

$$\nabla\mathrm{R}_D[\mathbf{w}] = 2\mathbf{w} + c\sum_{i=1}^{N} \ell'_{\mathrm{SH}}\left(s_i\mathbf{x}^{(i)\mathsf{T}}\mathbf{w}\right) s_i\mathbf{x} \tag{6.14}$$

$$\nabla^2\mathrm{R}_D[\mathbf{w}] = 2\mathbf{I} + c\mathbf{X}^\mathsf{T}\mathbf{D}\mathbf{X}, \tag{6.15}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{X} = [\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}]^\mathsf{T}$ is the data matrix, and $\mathbf{D}$ is diagonal with entries $D_{ii} = \ell''_{\mathrm{SH}}(s_i\mathbf{x}^{(i)\mathsf{T}}\mathbf{w})$.

The Hessian matrix has size $N \times N$, and thus would be far too large to be stored in memory. Fortunately, Equation 6.13 can be solved using a conjugate-gradient procedure, which requires only Hessian-vector products. These can be calculated efficiently through

$$\nabla^2\,\mathrm{R}_D[\mathbf{w}]\boldsymbol{\delta} = 2\boldsymbol{\delta} + c\mathbf{X}^\mathsf{T}\mathbf{D}\mathbf{X}\boldsymbol{\delta}, \tag{6.16}$$

because $\mathbf{X}$ is a very sparse matrix. In practice, Equation 6.13 is only solved approximately, drastically reducing the number of conjugate-gradient iterations, and thus Hessian-vector products, that need to be calculated.

Determining the Step-Size The resulting step vector $\boldsymbol{\delta}$ might be outside the region in which the quadratic approximation (see Equation 6.13) accurately models the true risk landscape $\mathrm{R}_D[\mathbf{w}]$. Therefore, a step-size mechanism is needed usually either by using a trust region or by doing a line search.

Due to the linear nature of the ranking function $r(\mathbf{x};\mathbf{w}) = \mathbf{x}^\mathsf{T}\mathbf{w}$, the line search can be implemented efficiently by using

$$r(\mathbf{x};\mathbf{w}+\lambda\boldsymbol{\delta}) = \mathbf{x}^\mathsf{T}\mathbf{w} + \lambda\mathbf{w}^\mathsf{T}\boldsymbol{\delta} \tag{6.17}$$

$$\|\mathbf{w}+\lambda\boldsymbol{\delta}\|_2^2 = \|\mathbf{w}\|_2^2 + 2\lambda\mathbf{w}^\mathsf{T}\boldsymbol{\delta} + \lambda^2\boldsymbol{\delta}^\mathsf{T}\boldsymbol{\delta}. \tag{6.18}$$

By caching the values of $\|\mathbf{w}\|_2^2$, $\mathbf{w}^\mathsf{T}\boldsymbol{\delta}$, $\boldsymbol{\delta}^\mathsf{T}\boldsymbol{\delta}$, $\mathbf{x}^\mathsf{T}\mathbf{w}$ and $\mathbf{x}^\mathsf{T}\boldsymbol{\delta}$, the cost of evaluating the loss for any value of λ after the first evaluation drops to $O(N)$ evaluations of ℓ_{SH} and the corresponding additions and multiplications of the cached values in Equations 6.17 and 6.18.

Implicit Hard-Instance Mining in Hinge Losses When using the squared hinge loss (see Equation 6.11) for ℓ_{SH}, the loss and all its derivatives become zero once the sample is classified correctly with a sufficient margin[3] $s\mathbf{x}^\mathsf{T}\mathbf{w} > 1$. Consequently, the corresponding entries in the diagonal matrix $\mathbf{D}$ in Equation 6.16 become zero. This means that in the product $\mathbf{X}^\mathsf{T}\mathbf{D}\mathbf{X}$, the feature matrix $\mathbf{X}$ can be replaced with a much smaller matrix $\tilde{\mathbf{X}}$ that contains only the examples that are not classified correctly with a margin. Denote with $\mathcal{E} := \{i \in [N] : s_i\mathbf{x}^{(i)\mathsf{T}}\mathbf{w} \le 1\}$ the set of indices of examples with nonzero loss (the *hard* instances), then $\tilde{\mathbf{X}} = [\mathbf{x}^{(i)} : i \in \mathcal{E}]^\mathsf{T}$.

As a consequence, the full feature matrix $\mathbf{X}$ is only needed once per step to determine the gradient $\nabla \mathrm{R}_{D}[\mathbf{w}]$, $\mathbf{D}$, and the hard examples $\mathcal{E}$. Afterwards, each CG iteration only requires the reduced matrix $\tilde{\mathbf{X}}$. This can be interpreted as an implicit hard instance mining step that is performed at the beginning of each step. As the weight vector $\mathbf{w}$ approaches the optimal weights $\mathbf{w}_\star$, most instances will have sufficient margin, and only few hard instances remain, $|\mathcal{E}| \ll N$ (cf. Figure 6.16). Therefore, later iterations require significantly less computation time than earlier ones. In fact, by using an initial vector $\mathbf{w}_0$ for which the hard-example set $\mathcal{A}$ is already small can speed up the overall computation time tremendously, as discussed below.

6.3.4 Further Performance Improvements

Mean-Separating Initialization A simple attempt to improve the initial weight vector is to chose a hyperplane that separates the means of the positive and negative instances for that label. Denote $\mathcal{P} := \{\mathbf{x}^{(i)} : i \in [N], y^{(i)} = 1\}$ and $\bar{\mathbf{x}} := N^{-1}\sum_{i=1}^{N}\mathbf{x}^{(i)}$, then the means are

$$\bar{\mathbf{p}} := \frac{1}{|\mathcal{P}|}\sum_{\mathbf{x}\in\mathcal{P}}\mathbf{x}, \qquad \bar{\mathbf{n}} := \frac{N\bar{\mathbf{x}} - |\mathcal{P}|\bar{\mathbf{p}}}{N - |\mathcal{P}|}. \tag{6.19}$$

[3] The margin of an instance denotes how far its score is from the classification boundary. An instance with a margin of 0 is classified correctly, but the slightest perturbation of its features or the classifier's weights could change the classification.

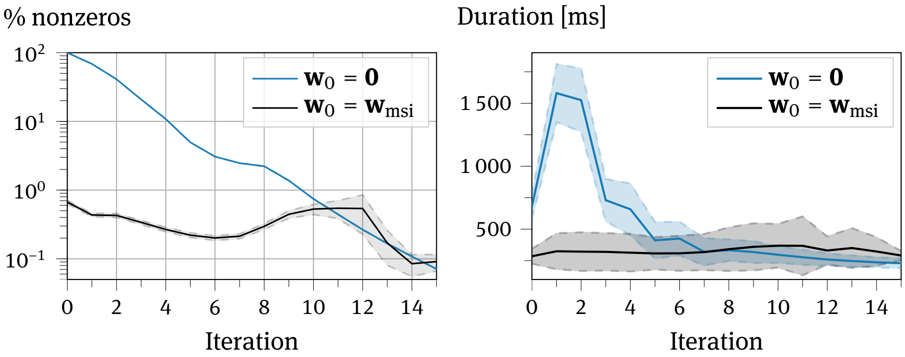


Fig. 6.16: Sparsity of the Hessian calculation $|\mathcal{E}|/N$ (left) and average duration of each optimization iteration (right) over the index of the iteration for zero and mean-separating initialization.

As $\bar{\mathbf{x}}$ only need be calculated once for the dataset, and $\bar{\mathbf{p}}$ can be computed quickly due to the data imbalance $|\mathcal{P}| \ll N$, these values can be computed efficiently.

This procedure can be viewed as an extreme case of data summarization (cf. Chapter 3), in which the entire dataset is reduced to just two instances. The idea is now to solve this very small classification problem, and use its solution as the starting point for solving the full problem. This will be particularly useful if the simple solution already classifies many of the easy negative instances correctly, as the set of hard examples $\mathcal{E}$ will be small in such a case.

For two data points, the classification problem can be solved explicitly, and we call its result the *mean-separating initialization* (`msi`) vector $\mathbf{w}_{\text{msi}}$. This vector is the minimum-norm vector that attains pre-specified margins $\mu_\pm$ for classifying the two data points. As a consequence, it lies in the plane spanned by $\bar{\mathbf{x}}$ and $\bar{\mathbf{p}}$, and is characterized by the following equations:

$$\mathbf{w}_{\text{msi}} = \alpha\bar{\mathbf{x}} + \beta\bar{\mathbf{p}}\,, \tag{6.20}$$

$$\bar{\mathbf{p}}^{\mathsf{T}}\mathbf{w}_{\text{msi}} = \mu_+\,, \qquad \bar{\mathbf{n}}^{\mathsf{T}}\mathbf{w}_{\text{msi}} = \mu_-\,. \tag{6.21}$$

Heuristically, values $\mu_+ = +1$, $\mu_- = -2$ work well, and are based on the rationale that negative samples cover a larger volume in the instance space, and thus the initial decision boundary should be closer to the mean of the positives than to the mean of the negatives.

The efficacy of this method can be seen from Figure 6.16, which evaluates the two initialization strategies on the AmazonCat-13k [478] dataset using an AMD Rome 7H12 CPU.[4] The data shows that

- starting from a zero initial vector, the fraction of nonzeros starts at 100 % and decreases as the training progresses;

[4] Computational resources provided by CSC – IT Center for Science, Finland.

- starting from a mean-separating initial vector, the sparsity is already high in the beginning; and
- increased sparsity translates to significant reductions in computation time and corresponding energy savings.

In terms of wall-clock training duration t_{wc}, the speedup that can be achieved by switching from $\mathbf{0}$ to $\mathbf{w}_{\mathrm{msi}}$, defined as $t_{\mathrm{wc}}(\mathbf{0})/t_{\mathrm{wc}}(\mathbf{w}_{\mathrm{msi}})$, lies between 150 % and 500 % as shown in Table 6.5.

Feature-Sorting A large portion of the computation time is spent on calculating the initial margins $\mathbf{X}^{\mathsf{T}}\mathbf{w}$ at the beginning of each iteration. Because $\mathbf{X}$ is a sparse matrix, this computation has low arithmetic density, and because the feature dimension is typically very large, the vector $\mathbf{w}$ does not fit into the L2-cache. These two properties mean that this operation is severely memory-bound.

The caching behavior of $\mathbf{w}$ can be improved significantly by making use of the dataset characteristic – in particular the fact that typical XMC tf-idf datasets have a long-tailed distribution in the feature vector, meaning that some features have a large amount of non-zero entries, but most features have few non-zeros[29]. By sorting the feature indices according to the frequency of their occurrence, the corresponding entries in $\mathbf{w}$ are brought closer together in the address space, thus improving the caching behaviour.

While this has no effect on the scaling of the performance with the thread count, it does induce an absolute speedup, as indicated by the dashed lines in Figure 6.17.

The Memory-Bottleneck On machines with many cores, the performance of the computations presented here is memory-bound. This can be seen in Figure 6.17, where despite the embarrassingly parallel nature of the computations, the performance scales sublinearly once a certain core count is exceeded.

In the specific case of running the computations on a 2-socket AMD Rome 7H12 (64 cores per CPU, cf. Figure 6.15) machine, Figure 6.17 shows almost perfect scaling from 8 threads, corresponding to 1 thread per NUMA node, up to 32 threads, corresponding to one thread per L3 cache. For higher thread counts, the speedup saturates and in some cases more threads may even be disadvantageous to performance.

In addition to the memory bottleneck, there will be a thermal/power bottleneck involved. The used CPU has a base clock of 2.6 GHz, but if only a few cores are used the clock frequency may be increased up to 3.3 GHz.[5] This indicates that even without the memory bottleneck, the expected performance increase of 128 cores over 16 cores would be less than $16\times$. This shows that the sublinear scaling cannot be explained by

[5] Given that the computations are memory-bound, even when running with 128 cores the execution ports of the CPU will be idle for a significant amount of time. Only moderate downclocking to ≈ 3.18 GHz occurred in our setup.

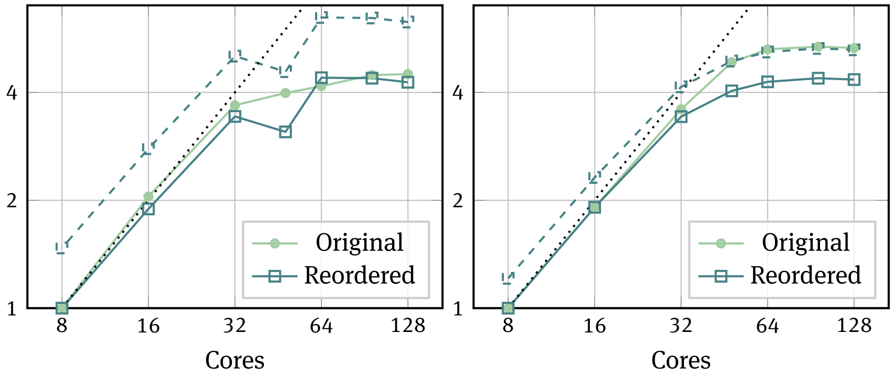


Fig. 6.17: Relative speedup for increasing thread counts using both original and reordered features for the first 10,000 labels of the Wikipedia-500k [50] dataset (left) and for the AmazonCat-13 [478] dataset (right). The dashed line shows the speedup of reordered features, normalized to the computation time with original features. The dotted line indicates perfect scaling. The (non-parallel) portion of the program run-time that is spent parsing the input dataset has been subtracted from the timings presented here.

Tab. 6.5: Training time (in hours) for zero and mean-separating initialization, as well as the number of non-zero weights (NNZ) after pruning (in millions) and their fraction. The experiments were run on a two-socket AMD Rome 7H12 machine.

Dataset	Zero	MSI	Speedup	NNZ	Fraction
Wiki10-31k	0.05	0.03	164 %	114	3.62 %
Amazoncat-13k	0.33	0.09	353 %	75	2.78 %
Amazoncat-14k	1.31	0.39	339 %	110	1.26 %
WikiTitles-500k	5.34	1.19	451 %	83	0.09 %
Amazon-670k	6.82	1.36	503 %	405	0.44 %
Delicious-200k	7.84	4.73	166 %	1 175	0.73 %
WikiLSHTC-350k	18.52	5.17	358 %	434	0.08 %

Tab. 6.6: Results of DiSMEC in comparison with the state-of-the art results as reported in March 2022 in Bhatia, Dahiya, Jain, Prabhu, and Varma [50], for selected XMC datasets.

Dataset/Metric	DiSMEC	SOTA Method	SOTA
Amazon-670K	—	—	—
P@1	44.7	LightXML	49.1
P@3	39.7	LightXML	43.8
P@5	36.1	LightXML	39.9
AmazonCat-13K	—	—	—
P@1	93.4	LightXML	96.8
P@3	79.1	LightXML	84.0
P@5	64.1	LightXML	68.7
Wikipedia-500K	—	—	—
P@1	70.2	AttentionXML	82.7
P@3	50.6	AttentionXML	63.8
P@5	39.7	AttentionXML	50.4
EURLex-4K	—	—	—
P@1	82.4	APLC-XLNet	87.7
P@3	68.5	APLC-XLNet	74.6
P@5	57.7	APLC-XLNet	62.3

reduced clock frequencies alone; rather, another resource, such as memory bandwidth, is also limiting performance.

6.3.4.1 Comparison With Deep Learning Methods

As shown in Table 6.6, the DiSMEC instantiation of the embarrassingly parallelizable one-vs-rest framework described in Algorithm 4 can be a competitive baseline. Its performance is not significantly worse in comparison to the state-of-the-art deep learning methods, which typically employ transformers encoders [174]. Unlike the deep learning models that require careful hyper-parameter tuning, the linear binary classification underlying DiSMEC is more readily interpretable and well-understood from a theoretical viewpoint. For sparse tf-idf data representation, linear XMC classifiers also work at par with tree-based approaches [371, 584] and those involving dense low-dimensional label embeddings [51, 279].

6.3.5 Summary and Outlook

In this section we presented linear classification algorithms for extreme multi-label classification. The linear model makes the training parallelize perfectly across different labels, though in practice the scaling levels out with too many cores in a single node. This is because even though the training itself does not require any communication or synchronization between the threads, the different cores within a machine still compete for resources such as memory access. By placing a copy of the feature matrix in each NUMA domain, it can be ensured that each CPU can read its data from a part of the memory that it has fastest access to, and the load on the memory interface is spread out across the different NUMA domains. Additionally, by reordering the columns in the sparse feature matrix, data locality and, accordingly, cache efficacy can be improved. An implementation that combines techniques can be found at https://doi.org/10.5281/zenodo.6699587. For further discussion on the interaction between machine learning and the memory hierarchy, see Chapter 7.

The amount of work required to train the linear classifier for the highly imbalanced data typical for XMC can be drastically reduced by starting the weights from a good initialization. One way to find such a weight vector is to reduce the dataset to just two training instances, the centers of masses of the positive and negative training points in the original dataset, which can be calculated efficiently. By initializing the full training procedure with a weight vector that separates this summarized data, one can capitalize on the speedup of the conjugate-gradient optimizer due to implicit hard-instance mining of the hinge loss. This procedure can be seen as a variation of the *sketch-and-solve* principle introduced in Section 3.2. The main difference is that here the solution based on the sketch is used to initialize the full training, and thus no compromise in accuracy is made.

The presentation in the book is focused on the computational and implementation challenges of XMC problems. However, the scale of the label space also leads to interesting statistical consequences such as a long-tailed label distribution and corresponding data-scarcity for tail labels, as well as incomplete training data with missing labels. For a discussion of these issues, see the works of Babbar and Schölkopf [29], Jain, Prabhu, and Varma [336], and Qaraei, Schultheis, Gupta, and Babbar [587].

6.4 Optimization of ML on Modern Multicore Systems

Helena Kotthaus
Peter Marwedel

Abstract: This section demonstrates how the integration of knowledge about underlying hardware platforms and learning algorithms can provide results that would not be feasible by using the knowledge of only one type. In particular, this section presents the optimization of ML algorithms on multicore systems, and in this way addresses the same type of architectures as in Section 6.3. The optimization is based on resource-aware scheduling strategies for parallel machine learning algorithms. The focus is on Model-Based Optimization (MBO), also known as Bayesian optimization, which is an ML algorithm with huge resource demands, including a large number of computational jobs. Execution times of these jobs are estimated in order to enable their scheduling on parallel processors. The section demonstrates that this scheduling enables the processing of larger problem sizes within a given time budget and reduces the end-to-end wall-clock time for a constant problem size.

6.4.1 Motivation

The notion of resource-constrained systems is typically associated with small, integrated, and special-purpose devices exhibiting limitations with respect to, say, computational power, size, or battery life in embedded and cyber-physical systems. However, reducing the understanding of resource restriction to systems of this kind is not sensible. In fact, even high-performance computers and clusters can suffer from resource constraints when solving highly challenging problems that require massive amounts of resources [166, 666]. Therefore, it makes sense to consider resource constraints also for applications typically executed on larger systems.

Here, this is shown for the case of *parallel* MBO. MBO is a state-of-the-art global optimization method for black-box functions that are expensive to evaluate. To reduce the number of necessary evaluations of the black-box function, conventional MBO uses an iteratively refined regression model on a set of already evaluated configurations to approximate the objective function. However, such approaches neglect the heterogeneous resource requirements for evaluating different configurations in the model space, which often leads to inefficient resource utilization. This calls for new resource-aware scheduling strategies to efficiently map configurations to the underlying parallel architecture in accordance with their resource demands. In contrast to classical scheduling problems, the scheduling for MBO needs to interact with the configuration proposal mechanism to select configurations with suitable resource demands for parallel evaluation.

The fundamentals and related approaches of parallel MBO are presented in Section 6.4.2. An overview of the RAMBO (Resource-Aware MBO) framework including the resource-aware scheduling strategies, as well as the corresponding evaluation results on homogeneous multiprocessor cluster systems, is given in Section 6.4.3. Section 6.4.4 proposes a concept for resource-aware scheduling strategies on heterogeneous embedded systems. The results are shown in Section 6.4.5.

6.4.2 Fundamentals and State of the Art for Parallel MBO

In machine learning, selecting the best algorithms for a given optimization problem and simultaneously tuning the corresponding hyperparameters of these algorithms can be very computationally intensive. Many strategies for hyperparameter optimization have been developed. (For an overview see, e.g., [55]). Hyperparameter optimization refers to finding the best configuration θ of a model, e.g., for a prediction problem a model with high predictive performance on an independent test set. When the evaluation of a single configuration already requires high resources, e.g., a very long runtime, then very wasteful optimization methods like evolutionary algorithms are not applicable. A popular approach for algorithm selection is F-racing [448], where a population of configurations is racing against each other and underperforming candidates are iteratively eliminated. This approach also requires many evaluations, at least in the early stage of the algorithm.

An established alternative in the situation of expensive time constraints is Model-Based Optimization (MBO), also known as Bayesian optimization, a state-of-the-art technique for expensive black-box optimization. In this optimization process, an unknown function, say, a machine learning algorithm, is evaluated to find the parameter configuration with the highest quality of the output measured by a given performance criterion within a limited time budget. This process is computationally challenging due to the huge parameter space that needs to be contemplated, and can result in extremely long response times. For this reason it is desirable to reduce the optimization time while maintaining the prediction quality, i.e., $\theta^{\star} := argmin_{\theta \in \Theta} f(\theta)$ for a search space Θ and an evaluation $f(\theta)$ of the black-box with input $\theta \in \Theta$ [348]. Aiming to reduce the number of evaluations of f required to find the best configuration $\theta^{\star}$, we used an iteratively refined and updated regression model (surrogate model), which attempts to approximate the black-box function by predicting $f(\theta)$ based on previous evaluations of f. During each iteration, a so-called *infill criterion* (acquisition function) proposes new promising configurations for evaluation.

In its original formulation, the MBO algorithm operates purely sequentially, proposing one configuration to be evaluated after the other [348]. For applications such as hyperparameter tuning for machine learning algorithms or computer simulations, the parallelization of MBO has become of an increasingly interesting approach to reduce the overall execution time [288]. In order to propose multiple points (configurations) simultaneously in a parallel MBO setting, several modifications to the infill criteria or the general technique have been suggested. The modifications result in multiple configurations being proposed in each iteration [54, 254, 328]. The number of simultaneously proposed configurations is typically chosen to match the number of available CPU cores. However, these modifications in general neglect the heterogeneous resource requirements for evaluating different configurations in parallel. Depending on the parameter configuration of the applied machine learning algorithm, resource requirements such as CPU utilization or memory footprint usage can vary heavily [666].

The most important parallel extensions of MBO update the regression model either synchronously or asynchronously. Both variants are based on different infill criteria and have different advantages and drawbacks.

Synchronous Execution To allow for parallelization with a synchronous model update, infill criteria and techniques that propose multiple configurations in each iteration (constant liar, Kriging believer, qEI [254], qLCB [328], MOI-MBO [54]) have been suggested. *Multi-point proposals* are able to derive q configuration proposals $\boldsymbol{x}_1^\star, \dots, \boldsymbol{x}_q^\star$ simultaneously instead of only proposing one configuration $\boldsymbol{x}^\star$ from a *surrogate model*. Here, the model is updated after all evaluations within one iteration are finished. Hutter et al. [328] introduced the qLCB criterion, which is an extension of the single-point LCB criterion using an exponentially distributed random variable to generate q different candidate proposals by drawing random values of $\lambda_j \sim \mathrm{Exp}(\lambda)$ ($j = 1, \dots, q$) from the exponential distribution:

$$\mathrm{qLCB}(\boldsymbol{x}, \lambda_j) = \hat{\mu}(\boldsymbol{x}) - \lambda_j \hat{s}(\boldsymbol{x}) \text{ with } \lambda_j \sim \mathrm{Exp}(\lambda). \tag{6.22}$$

The λ variable guides the exploration-exploitation trade-off. Sampling multiple different λ_j might result in different “good” configurations by varying the impact of the standard deviation term.

Another popular multi-point infill criterion is the qEI criterion [254], which directly optimizes the single-point EI criterion over q points. As the computation of EI uses Monte Carlo sampling, it is quite expensive [136]. Therefore, a less expensive alternative, the *Kriging believer* approach [254], is often chosen. Here, the first configuration is proposed based on the standard single-point EI criterion. Its posterior mean value is treated as a real value of f to refit the surrogate, penalizing the surrounding region with a lower standard deviation for the next point proposal using EI again. This is repeated until q proposals are generated.

The above mentioned multi-point infill criteria can cause inefficient resource utilization when the parallel executed evaluations have heterogeneous execution times. Before new configurations are proposed, the results of all evaluations within one iteration are gathered to update the model. Thus the slowest evaluation becomes the bottleneck, and all other parallel worker processes idle after finishing their evaluation before a new MBO iteration can start. However, performing the model updates only once per MBO iteration also leads to less computation overhead. Varying the execution times of parallel evaluations have already been addressed by Snoek et al. [635], who suggest that these be modelled with an additional surrogate, leading to an "expected improvement per second" favoring less expensive configurations. The resource-aware scheduling strategies for parallel MBO presented in this section also use regression models to estimate resource requirements, but instead of adapting the infill criterion, they use them to guide the scheduling of parallel evaluations. The goal is to guide MBO to interesting regions in a faster and resource-efficient way without directly favoring less expensive configurations.

Asynchronous Execution To avoid CPU idling, asynchronous execution replaces the evaluation of multiple configurations in batches, and the synchronous refitting of the model by refitting the model after each evaluation. Here, the number of worker processes equals the number of available CPU cores, but each worker proposes the next point for evaluation independently, even if configurations $\boldsymbol{x}_{\text{busy}}$ are currently under evaluation on other CPU cores. The main challenge is to avoid evaluations of very similar configurations by modifying the infill criterion to deal with points that are currently under evaluation. The fast Kriging believer approach [254], which is based on EI (also used for multi-point proposals), can be applied to block these regions.

Another approach assessing pending values is the *Expected* EI (EEI) [253, 339, 635]. Here, the unknown value of $f(\boldsymbol{x}_{\text{busy}})$ is integrated out by calculating the expected value of y_{busy} via Monte Carlo sampling, which is, similar to qEI, computationally demanding. For each Monte Carlo iteration, values $y_{1,\text{busy}}, \dots, y_{\mu,\text{busy}}$ are drawn from the posterior distribution of the surrogate regression model at $\boldsymbol{x}_{1,\text{busy}}, \dots, \boldsymbol{x}_{\mu,\text{busy}}$, with μ denoting the number of pending evaluations. These values are combined with the set of already known evaluations and used to fit the surrogate model. The EEI can then simply be calculated by averaging the individual expected improvement values, which are formed by each Monte Carlo sample (n_{sim} denotes the number of Monte Carlo iterations):

$$\widehat{\text{EEI}(\boldsymbol{x})} = \frac{1}{n_{\text{sim}}} \sum_{i=1}^{n_{\text{sim}}} \text{EI}_i(\boldsymbol{x}) \tag{6.23}$$

Besides the advantage of an increased CPU utilization, asynchronous execution can also cause additional runtime overhead due to the higher number of model updates and the computational costs for new point proposals, especially when the number of available CPU cores increases. Furthermore, the heterogeneous execution times of job configurations can lead to very similar point proposals due to model updates that are based on similar histories. Instead of using asynchronous execution to efficiently utilize parallel computer architectures, the new approach presented in this section uses the synchronous execution combined with resource-aware scheduling. The next section includes a comparison of this approach (RAMBO) [385, 389, 390, 666] with the synchronous and asynchronous parallel variants of MBO described above.

6.4.3 Resource-Aware Scheduling Strategies

To enable the interaction between resource-aware scheduling strategies and the general MBO process, the RAMBO framework is proposed. Its foundations are based on the `mlrMBO` library [53]. The framework shown in Figure 6.18 aims to resource-efficiently reduce the end-to-end wall clock time needed by parallel MBO, and thus converge to the optimal configuration more rapidly. RAMBO consists of three main steps:

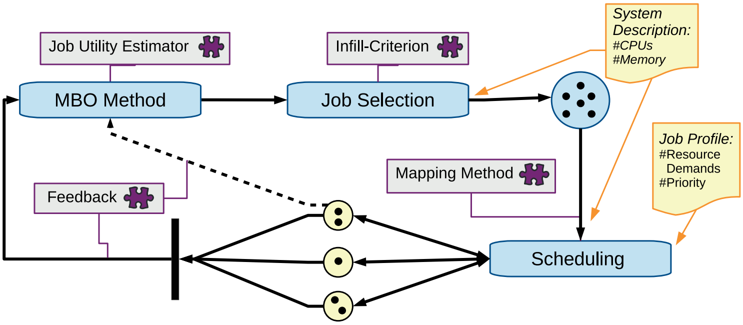


Fig. 6.18: Key steps (shown in blue) in the Resource-Aware Model-Based Optimization Framework [385]: building a regression model, selection of evaluation jobs, and job scheduling. Asynchronous execution (dashed line) and synchronous execution (solid lines) are possible.

First, a previously initialized regression model is built by the *MBO method*. Simultaneously, a *job utility estimator* creates profiles for the evaluations of configurations *(jobs)* by means of an additional regression model. These *job profiles* include runtime estimates, which are used as an input for the respective scheduling strategy later on. In conventional synchronous MBO approaches, such runtime estimates are not available. Hence, the slowest evaluation becomes a bottleneck within one MBO iteration, and the already finished parallel worker processes remain idle. As a consequence, the feedback of all idling processes and hence the model update is delayed.

The second step, i.e., the *job selection*, follows the MBO principles for configuration proposals. Typically, an *infill criterion* such as qLCB in equation (6.22) is used to propose configurations offering a proper compromise between the predicted outputs *(exploit)* and the uncertainty about the search space region *(explore)*, i.e., that have a high potential to optimize the quality of the regression model. To this end, RAMBO provides mechanisms to interact with the job proposal mechanism by postponing or skipping suggested configurations that are deemed to be insufficiently promising or exhibit unsuitable job profiles. As part of this process, a knapsack-based heuristic can be applied to select the most promising and suitable configurations.

Finally, a configurable *scheduling strategy* allocates the jobs to the available resources (system description) according to their particular resource demands. In addition, an execution priority based on the infill criterion is required to ensure that the model is updated i) with the most promising configurations and ii) as soon as possible. This model update follows the synchronous approach, i.e., it is performed when the results of all jobs executed within one MBO iteration are gathered. In a nutshell, the regression model is iteratively updated based on the results of all previous iterations until the runtime budget is exhausted.

Priorities for Job Selection To model the usefulness of a candidate for the objective function, Kriging is used as a surrogate regression model, and qLCB (6.22) is used as a multi-point infill criterion to generate a set of job proposals. Compared with the multi-point proposal qEI [254], the qLCB criterion is more suitable since it is able to propose a set of independent candidates. qLCB can simultaneously generate q candidates by drawing q random values of $\lambda_j \sim \text{Exp}(\lambda)$ $(j = 1, \dots, q)$ from the exponential distribution. Each λ_j results in a different trade-off between exploitation $(\lambda_j \downarrow)$ and exploration $(\lambda_j \uparrow)$, and thus leads to a different optimal configuration $\boldsymbol{x}_j^\star$ after solving

$$\mathbf{x}_j^\star := \operatorname*{argmin}_{\mathbf{x}} \left[\text{LCB}(\mathbf{x}, \lambda_j)\right] = \operatorname*{argmin}_{\mathbf{x}} \left[\hat{y}(\mathbf{x}) - \lambda_j \hat{s}(\mathbf{x})\right] , \tag{6.24}$$

where $\hat{y}(\mathbf{x})$ denotes the posterior mean and $\hat{s}(\mathbf{x})$ denotes the root of the posterior standard deviation of the surrogate model at point $\mathbf{x}$.

Since the set of proposed candidates $\boldsymbol{x}_j^\star$ cannot be directly ordered by how promising a candidate is, an additional order is introduced to guide the search for the best candidate towards more promising areas. Therefore, the highest priority is given to the candidate $\boldsymbol{x}_j$ that was proposed using the smallest value of λ_j and is thus closest to the optimum (exploitation). The priority for each job is defined as $p_j := -\lambda_j$.

However, qLCB does not include a penalty for the proximity of selected configurations, which might become a problem if the number of parallel evaluations is high. Therefore, the Euclidean distance is used to reprioritize p_j to $\tilde{p}_j$, encouraging the selection of configurations that are more scattered in the domain space.

First, a set of $q > m$ configurations is sampled from the qLCB criterion. These configurations are then hierarchically clustered by their distance in the domain space of the objective function using the complete linkage method. The procedure starts with the configuration that has previously been assigned the highest priority and assigns it to the first position in the list of selected jobs $\tilde{J}$. For each following step $i \geq 2$, all candidates are split into i clusters according to the hierarchical clustering. Of these i clusters the $i - 1$ clusters that already contain candidates with assigned positions are discarded, leaving one cluster. The position i in $\tilde{J}$ is assigned to the job with the highest priority within this cluster. This goes on until all q candidates have assigned positions.

Thereby an ordering following the hierarchy induced by the clustering is generated. Finally, new priorities $\tilde{p}_j$ are assigned based on the order of $\tilde{J}$, i.e. the first job in $\tilde{J}$ gets the highest priority q and the last job gets the priority 1.

As a result, the set of candidates contains batches of jobs with similar priority, which are spread in the domain space. The priorities serve as input for the scheduling, which groups the q jobs to m CPU cores using the runtime estimates $\hat{t}$.

Resource Utility Estimation The runtime estimates of the set of jobs proposed in each MBO iteration are needed for the scheduling to avoid the execution of jobs with high runtime variances and thus to reduce idling worker processes. This is accomplished by using an additional regression model. As for the MBO algorithm itself, the runtime of a job is predicted in each iteration based on the runtimes of all previously evaluated jobs to build the runtime model of the black-box function. For the model, Kriging is used for homogeneous CPU systems since the runtime is expected to be a continuous function. For parallel architectures with heterogeneous CPUs, Random Forest is used for the model instead. Here, the runtime of a job is estimated for different CPU types (as described in Section 6.4.4). The accuracy of the runtime estimation also influences the scheduling decision. Therefore the runtime estimation quality is also included in the evaluation results.

Knapsack-Based Scheduling Strategy The goal of the knapsack-based scheduling strategy is also to reduce the CPU idle time on the workers while acquiring the feedback of the workers in the shortest possible time to avoid model update delay. Here the qLCB multi-point infill criterion is used to form a set of jobs $J = \{1, \ldots, q\}$ that should be executed on the available CPU cores $K = \{1, \ldots, m\}$. The estimated runtime is given by $\hat{t}_j$ and the corresponding priority by p_j for each job proposal. The time bound for each MBO iteration (deadline) is defined by the runtime of the highest prioritized job. The goal is to maximize the profit, given by the priorities, of parallel job executions within each MBO iteration. To solve this problem, we apply the $0-1$ multiple knapsack algorithm for global optimization routines [62]. Here, the knapsacks are the available CPU cores and their capacity is the maximally allowed computing time, defined by the runtime of the job with the highest priority. The items are the jobs J, their weights are the estimated runtimes $\hat{t}_j$, and their values are the priorities p_j. Accordingsly, the capacity for each CPU core is $\hat{t}_{j^\star}$, with $j^\star := \operatorname{argmax}_j p_j$. To select the best subset of jobs, the algorithm maximizes the profit Q:

$$Q = \sum_{j \in J} \sum_{k \in K} p_j c_{kj}, \tag{6.25}$$

It is the sum of priorities of the selected jobs, under the restriction of the capacity

$$\hat{t}_{j^\star} \geq \sum_{j \in J} \hat{t}_j c_{kj} \; \forall k \in K \tag{6.26}$$

per CPU. The restriction with the decision variable $c_{kj} \in \{0, 1\}$ s.t.

$$1 \geq \sum_{k \in K} c_{kj} \; \forall j \in J, c_{kj} \in \{0, 1\} \tag{6.27}$$

ensures that a job j is at most mapped to one CPU.

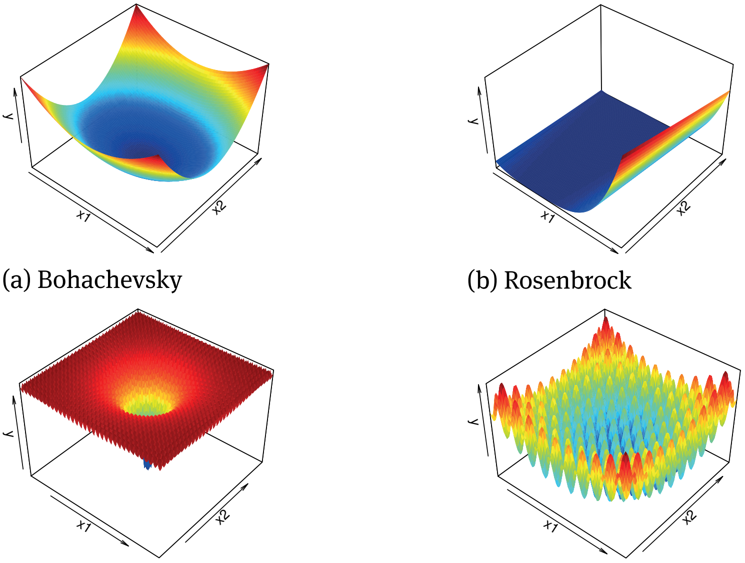
The job with the highest priority defines the time bound (deadline) $\hat{t}_{j^*}$ and is thus mapped to the first CPU core $k = 1$ exclusively, while single jobs with higher execution times are directly discarded (discarded jobs will be proposed again in the next MBO iteration if they are promising enough). Then, the knapsack algorithm is applied to assign the remaining candidates in J to the remaining $m - 1$ CPU cores. This leads to the best subset of J that can be run in parallel, minimizing the delay of the model update. If a CPU core is left without a job, the regression model can be optionally queried for a job with an estimated runtime smaller or equal to $\hat{t}_{j^*}$ to fill the gaps. Jobs having an estimated runtime *shorter* than the deadline, however, can lead to idle times if no other job can be executed within the time remaining until the next model update. The idle time resulting from suboptimal resource usage can be additionally exploited by enabling preemption and migration. More precisely, allowing jobs to be preempted and migrated to other cores provides the opportunity to fill unused time slots within an MBO iteration with high-priority jobs that would be skipped otherwise. Thus a larger set of jobs can be executed. The details of RAMBO's flexible migration mechanisms are described in [389].

Evaluation To evaluate the resource-aware MBO scheduling strategies included in the RAMBO framework, a comparison with different synchronous and asynchronous parallel MBO approaches was performed. The comparison included two asynchronously executed MBO strategies [253, 338] aiming to use all available CPU time to solve the optimization problem in parallel. Both of them used Kriging as a surrogate, with the EEI criterion 6.23 [339] and the Kriging believer [254] criterion. In Kotthaus et al. [390], RAMBO was also compared with a third asynchronous execution strategy, which is included in the SMAC (Sequential Model-based Algorithm Configuration) tool [329], using a random forest surrogate. The results showed that RAMBO and the two other asynchronous execution strategies always converged faster to the optimum compared to SMAC, which is why SMAC is not included in the following presentation. Besides the comparison with the asynchronous strategies, the following presentation also includes two synchronously executed MBO approaches. One of them used the qLCB multi-point infill criterion 6.22 and the other used the qEI criterion [254]. All parallel MBO approaches including the new RAMBO approach were evaluated on a set of established continuous synthetic functions combined with simulated execution times to ensure a fair and disturbance-free environment.

The usage of synthetic functions ruled out technical problems emerging on multi-user systems (swapping, network congestion, CPU cycle stealing, other users occupying fast caches, etc.). Furthermore, synthetic functions eased the evaluation of MBO approaches on different difficulty levels. Two different categories of objective functions (implemented in the R library `smoof` [66]) were considered:

1. Functions with a smooth surface: `rosenbrock`(d) and `bohachevsky`(d) with dimension $d = 2, 5, 10$, which are likely to be fitted well by MBO.
2. Highly multimodal functions: `ackley`(d) and `rastrigin`(d) ($d = 2, 5, 10$), for which MBO is expected to have problems achieving good results.

For each objective function, a 2-, 5- and 10-dimensional version were used, each of which was optimized using 4 and 16 CPU cores in parallel to investigate scalability. Figure 6.19 visualizes the synthetic test functions for $d = 2$ [385].



(a) Bohachevsky

(b) Rosenbrock

(c) Ackley

(d) Rastrigin

Fig. 6.19: Synthetic test functions used for the evaluation for $d = 2$. (a) and (b) show a smooth surface; (c) and (d) are highly multimodal [385].

Since synthetic functions are illustrative test functions, they have no significant runtime. Therefore, these functions were also used to simulate different runtime behaviors. For each benchmark two different synthetic functions were combined. One determines the number of seconds it would take to calculate the objective value of the other function. For example, for the combination `rastrigin`(2)`.rosenbrock`(2) it would require `rosenbrock`(2)($\boldsymbol{x}$) seconds to retrieve the desired objective value `rastrigin`(2)($\boldsymbol{x}$) for an arbitrary proposed configuration $\boldsymbol{x}$. Technically, the benchmark sleeps `rosenbrock`(2)($\boldsymbol{x}$) seconds before returning the objective value. The runtime was simulated with either `rosenbrock`(d) or `rastrigin`(d), and all combinations of the four objective functions were analyzed except where the objective and the time function were identical. For the unification of the input space, values from the input space of the objective function were mapped to the input space of the function that simulated the runtime behavior. The output of the runtime functions were scaled to return values between 5 minutes to 60 minutes.

To examine how fast the parallel approaches converge to the optima of the benchmark functions within a limited time budget, the distance between the best found configuration at time t and a predefined target value (optimal configuration) was measured. This measurement reflects the accuracy of the receptive MBO approach within the given time budget. To make this measurement comparable for all objective functions, the function values were scaled to [0, 1]. Here, 0 is the target value, defined as the best configuration y reached by any optimization approach within the given time budget. The upper bound 1 is the best y found in the initial set of already evaluated configurations, and is identical for all approaches per given benchmark. Both values were averaged over 10 repetitions. If an optimization needs 2 hours to reach an accuracy of 0.5, this means that within 2 hours half of the way to the best configuration 0 has been accomplished, after starting at 1. The differences between the approaches were compared at the three accuracy levels 0.5, 0.1, and 0.01. The optimizations were repeated 10 times and conducted on $m = 4$ and $m = 16$ CPUs to examine scalability. Time budgets were 4 hours for 4 CPU cores and 2 hours for 16 CPU cores in total, including all computational overhead and CPU idling. All experiments were executed on a Docker Swarm cluster using the R library `batchtools` [415]. The initial set was generated by latin hypercube sampling [481] with $n = 4 \cdot d$ configurations, and all of the following optimizations start with the same initial set in all 10 repetitions:

- `rs`: Random search, served as a base-line.
- `qLCB`: Synchronously executed MBO using qLCB where in each MBO iteration $q = m$ configurations were proposed.
- `ei.bel`: Synchronously executed approach using Kriging believer where in each MBO iteration m configuration were proposed.
- `asyn.ei.bel`: Asynchronously executed MBO using Kriging believer.
- `asyn.eei`: Asynchronously executed MBO using EEI (100 Monte Carlo iterations).
- `rambo`: New synchronously executed MBO using qLCB with job priority refinement and the knapsack-based resource-aware scheduling strategy, $q = 8 \cdot m$ candidates proposed in each iteration.

Optimizations `qLCB` and `ei.bel` are implemented in the R library `mlrMBO` [53]. Optimizations `asyn.eei`, `asyn.ei.bel` and `rambo` are also based on `mlrMBO`. For all MBO approaches, a Kriging model was used from the library `DiceKriging` [605] with a Matern $\frac{5}{2}$-kernel [474] and a nugget effect of $10^{-8} \cdot \mathrm{Var}(\boldsymbol{y})$, where $\boldsymbol{y}$ denotes the vector of all observed function outcomes.

The quality of resource-aware scheduling depends on the accuracy of the resource estimation. Without reliable runtime predictions, the scheduler is unable to optimize for efficient utilization. The runtime for all benchmarks was simulated with either `rosenbrock`(d) or `rastrigin`(d). Figure 6.20 shows an example where the runtime estimation for the `rosenbrock`(5) time function works well (left part). Here, the residual values for the runtime estimation of the evaluated configurations decrease over time. However, the runtime prediction for `rastrigin`(5) (right part) is imprecise. For the 2- and 10-dimensional versions, the results are similar.

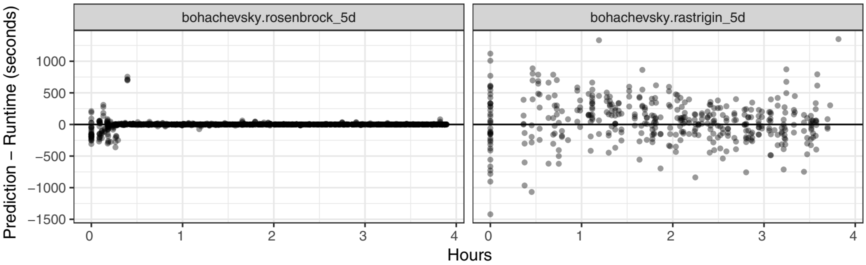


Fig. 6.20: Residuals of the runtime estimation over time for the `rosenbrock`(5) and `rastrigin`(5) time functions on 4 CPU cores combined with `bohachevsky`(5) as the objective function. Positive values indicate an overestimated runtime and negative values indicate an underestimation [385].

This encourages us to consider separate scenarios where runtime estimation has a high quality (`rosenbrock`(·)), and scenarios where runtime estimation is usually error-prone (`rastrigin`(·))s. In the following, we will focus on the scenario with high resource estimation quality. The evaluation results of the scenario with low runtime estimation quality can be found in [385] and are further optimized by a flexible scheduling mechanism [389].

Box plots for the time required to reach the three different accuracy levels in 10 repetitions within a budget of 4 hours on 4 CPU cores are shown in Figure 6.21, and within a budget of 2 hours on 16 CPU cores in Figure 6.22. The faster an approach reaches the desired accuracy level, the lower the box and the better the approach. If an approach was unable to reach an accuracy level within the given time budget, the respective time budget plus a penalty of 1 000 s is entered. Table 6.7 lists the aggregated ranks over all objective functions, grouped by approach, accuracy level, and number of CPU cores. For this computation, the approaches are ranked with regard to their performance for each repetition and benchmark before they are aggregated with the mean. If there are ties in Figures 6.21 and 6.21 (e.g., if an accuracy level was not reached), all values are assigned the worst possible rank. The benchmarks indicate an overall advantage of the new resource-aware MBO algorithm `rambo`. On average, `rambo` is always fastest. `rambo`

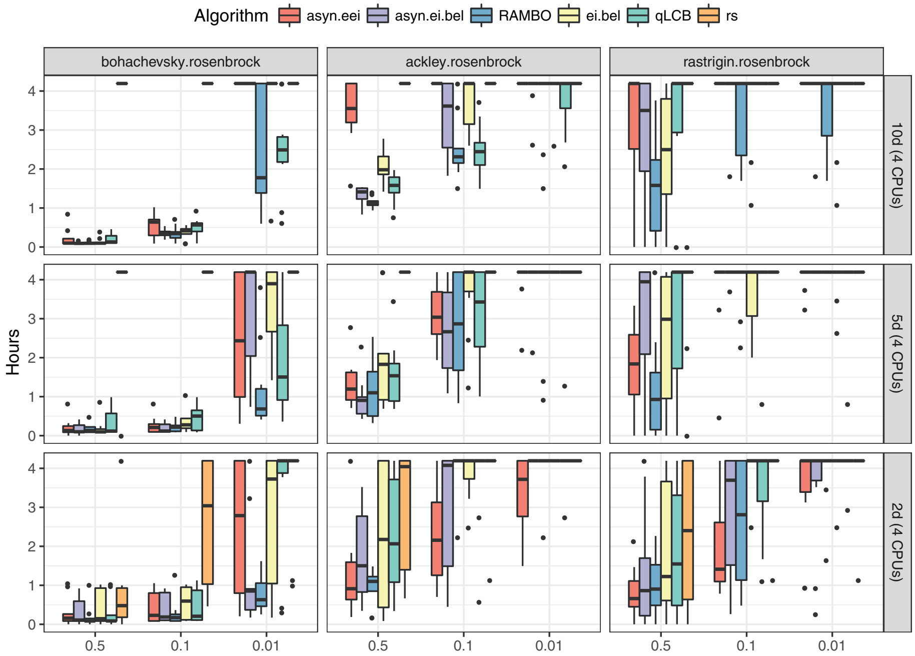


Fig. 6.21: Execution times on 4 cores as a function of the accuracy level for different objective functions using the time function rosenbrock(·) [385]. Execution times are low for moderate accuracy levels and favourable for RAMBO (shown in blue).

is closely followed by the asynchronous MBO variant `asyn.ei.bel` for accuracy levels 0.5 and 0.1 on 4 CPU cores but the lead becomes more clear on 16 CPU cores, especially for the highest accuracy level 0.01.

In comparison with the conventional synchronous MBO approaches `ei.bel` and `qLCB`, `rambo`, `asyn.eei`, and `asyn.ei.bel` reach the given accuracy levels in shorter time on 16 CPU cores. This is especially true for objective functions that are highly multimodal and thus hard to model (`ackley`(·), `rastrigin`(·)) by the surrogate, as seen in Figure 6.22.

Table 6.7 shows that the less expensive `asyn.ei.bel` approach performs better than the computationally demanding `asyn.eei` on 16 CPUs. On 4 CPUs the synchronous `qLCB` approach is faster than the asynchronous approaches for the highest accuracy level 0.01. This result is influenced by the good performance of `qLCB` on functions with a smooth surface, as can be seen in Figure 6.21 in the 5– and 10-dimensional version of the `bohachevsky`(·) benchmark. When comparing the performance of the approaches for the 2-dimensional versus the 10-dimensional versions of the benchmarks, Figure 6.22 shows that the `rambo` approach outperforms all other approaches at higher dimensional problems compared with lower dimensions.

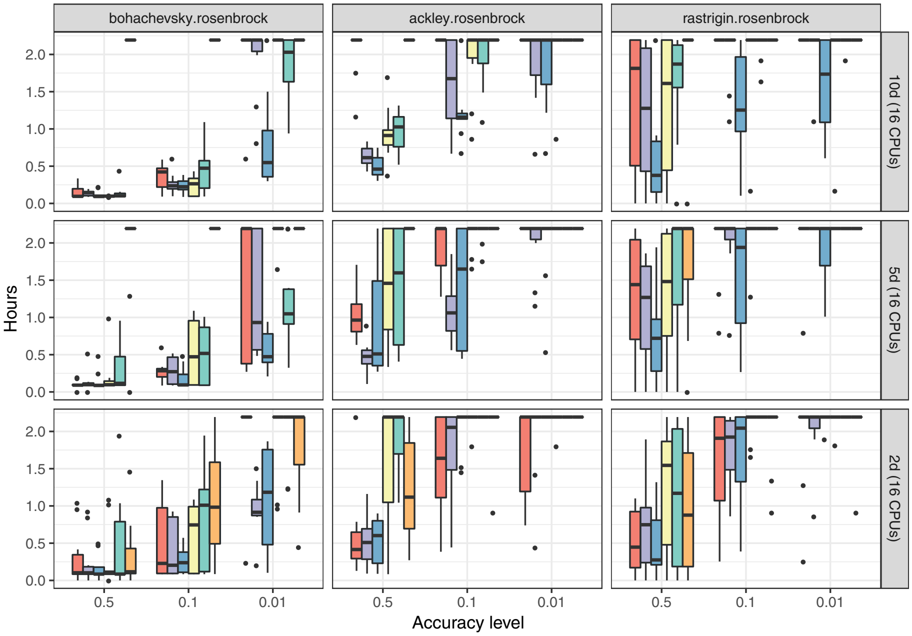


Fig. 6.22: Execution times on 16 cores as a function of the accuracy level for different objective functions using the time function `rosenbrock`(·) [385].

Figure 6.23 exemplarily visualizes the mapping of the parallel configuration evaluations (jobs) for all MBO approaches on 16 CPU cores for the $5d$ versions of the benchmarks. Each gray box represents the execution time of a job on the respective CPU. The gaps represent CPU idle time. For the synchronously executed MBO approaches `rambo`, `qLCB`, and `ei.bel`, the vertical lines represent the end of an MBO iteration. Red boxes indicate that the CPU is busy with a point proposal.

The necessity of a resource estimation for jobs with varying runtimes is obvious because the synchronous variants `qLCB` and `ei.bel` can cause long idle times by queuing jobs together with large runtime differences. The scheduling in `rambo` manages to reduce this idle time. This effect of efficient resource utilization increases with the number of CPUs. `rambo` reaches nearly the same effective resource utilization as the asynchronous approaches and at the same time reaches the accuracy level fastest. The Monte Carlo approach `asyn.eei` generates a high computational overhead as indicated by the red boxes, which reduces the effective number of evaluations. Here, the overhead for a new point proposal sometimes needs the same amount of time as the job evaluation. Idling occurs because the calculation of the EEI is encouraged to wait for ongoing EEI calculations to include their proposals. This overhead also increases with the number of evaluated points. By contrast, `asyn.ei.bel` has comparably low overhead and thus basically no idle time. This seems to be an advantage for `asyn.ei.bel` on 16 CPU cores, Tab. 6.7: Execution times for accuracy levels 0.5, 0.1, 0.01 averaged over all benchmarks with the rosenbrock(·) time function on 4 and 16 CPU cores with a time budget of 4 hours and 2 hours, respectively [385]. Relative ranks within a column are included in parentheses.

	4 CPUs			16 CPUs		
Algorithm	**0.5**	**0.1**	**0.01**	**0.5**	**0.1**	**0.01**
asyn.eei	3.53 (3)	3.91 (3)	4.91 (3)	3.64 (3)	4.30 (3)	5.30 (3)
asyn.ei.bel	3.21 (2)	3.66 (2)	5.04 (4)	2.93 (2)	3.31 (2)	4.48 (2)
rambo	**2.47 (1)**	**3.40 (1)**	**4.23 (1)**	**2.54 (1)**	**2.98 (1)**	**3.72 (1)**
ei.bel	3.64 (4)	4.36 (5)	5.31 (5)	3.81 (4)	4.57 (4)	5.70 (5)
qLCB	4.02 (5)	4.24 (4)	4.83 (2)	4.27 (5)	5.04 (5)	5.40 (4)
rs	5.57 (6)	5.89 (6)	5.89 (6)	5.17 (6)	5.71 (6)	5.82 (6)

where on average it performs better on all accuracy levels than the computationally demanding asyn.eei, especially for higher dimensional problems.

Observations rambo outperforms the conventional synchronous MBO. The resource utilization obtained by the scheduling in rambo leads to faster and better results, especially when it comes to increasing problem dimensions (configurable parameters) and increasing numbers of available CPU cores. On average, rambo converges faster to the optimum than all considered asynchronous approaches. This indicates that the resource utilization obtained by the RAMBO approach improves MBO, especially when the number of available CPU cores increases. Predictable runtimes can be assumed for real applications like hyperparameter optimization for machine learning methods, even if the runtime estimation quality is difficult to determine in advance. The results also suggest that, on some setups, the choice of the infill criterion determines the parallelization strategy for better performance.

6.4.4 Scheduling Strategies for Heterogeneous Architectures

As described in Section 6.4.3, the resource-aware scheduling for MBO uses two inputs: the estimated resource utilization and the priority of the proposed candidates. While the priority of a candidate is computed as described above, the estimation of the resource utilization needs to be enhanced for heterogeneous systems.

Resource Estimation for Heterogeneous Systems The regression model used to estimate the execution times of the candidates was previously based on Kriging; now Random Forest is applied instead. Random Forest is more suitable for heterogeneous systems since the job execution times build up a discontinuous model due to the additional categorical variable that represents the processor type. The regression model now needs to estimate the runtime $\hat{t}_j$ for each candidate in the proposed set of jobs $J =$

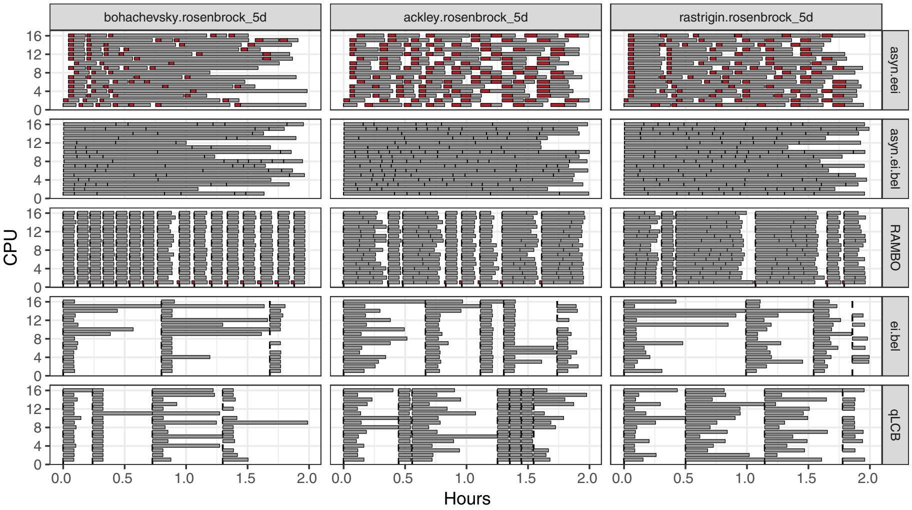


Fig. 6.23: Scheduling of MBO algorithms: Time shown on x-axis and mapping of candidates to $m = 16$ CPU cores on y-axis. Each gray box is a job. Each red box represents the overhead of the point proposal. The gaps represent CPU idle time [385].

$\{1, \dots, q\}$ and for each available CPU core $K = \{1, \dots, m\}$. This is required since the execution of a job is processor-dependent. If the underlying heterogeneous architecture is known, the number of runtime estimates per job can be reduced to the number of different processor types. Thus the runtime of a job $j \in J$ is predicted for each available processor type $k \in K$ in each MBO iteration based on the runtime of all previously evaluated jobs to build the runtime model of the black-box function and is therefore denoted as $\hat{t}_{kj}$.

Knapsack-Based Scheduling To apply the $0-1$ multiple knapsack algorithm for scheduling on heterogeneous architectures, the original formulation from Section 6.4.3 needs to be extended. Now the items representing the jobs J have different weights, represented by the different runtime estimates $\hat{t}_{jk}$ per processor type k. Since the capacity of the CPU cores is now heterogeneous, a reformulation is needed. For this purpose, a ratio variable representing an approximated ratio of the runtime differences produced by the different processor types is introduced.

To minimize the delay of the model update with the results of the most promising candidate, the job with the highest priority $j^\star := \text{argmax}_j\, p_j$ is now always placed on the CPU core $k^\star := \text{argmin}_k\, \hat{t}_{kj^\star}$ leading to the shortest estimated runtime for $j^\star$. The capacity for the remaining CPUs, and thus the time bound for each MBO iteration, is accordingly defined by the shortest estimated runtime of the highest prioritized job $\hat{t}_{k^\star j^\star}$. We introduce the ratio variable $\hat{t}_{k^\star j^\star}/\hat{t}_{kj^\star}$ representing the runtime difference of the highest prioritized job on the remaining k CPU cores.

The assumption that runtimes on different CPU types differ by a constant factor goes back to the uniform processor model described by Pinedo, which is a simplified model of real hardware [579]. For example, one CPU might offer vector instructions that some jobs benefit from, whereas others make no use of them. Instead of relying on statically precomputed ratios (such as those derived from the ratio of CPU frequencies), the selected job $j^\star$ is used as the "benchmark" for comparing CPU speeds in a given MBO iteration, under the assumption that in this iteration the speed on CPU k differs from $k^\star$ by a factor of $\hat{t}_{k^\star j^\star}/\hat{t}_{kj^\star}$. The formulation of the restriction of the capacities for the remaining CPU cores is thus as follows, while the rest of the knapsack algorithm remains as described in Section 6.4.3:

$$\hat{t}_{k^\star j^\star}\frac{\hat{t}_{k^\star j^\star}}{\hat{t}_{kj^\star}} \geq \sum_{j\in J}\hat{t}_{k^\star j}c_{kj}\ \forall k \in K. \tag{6.28}$$

Here, the estimated execution times of the remaining candidates on the fastest CPU core $\hat{t}_{k^\star j}$ on the right-hand side of equation 6.28 is expected to be approximately similar to the estimated runtime of a job on the remaining CPU cores $\hat{t}_{kj}$, multiplied by the ratio variable:

$$\hat{t}_{k^\star j} \approx \hat{t}_{kj}\frac{\hat{t}_{k^\star j^\star}}{\hat{t}_{kj^\star}}\ \forall k \in K, \forall j \in J. \tag{6.29}$$

This formulation is needed to reduce the number of weights (number of runtime estimates per CPU type) per item j to a single weight $\hat{t}_{k^\star j}$ in order to apply the original knapsack algorithm.

Evaluation The effectiveness of the heterogeneous RAMBO approach is evaluated by targeting the ARM big.LITTLE architecture[6] of the Odroid-XU3 platform,[7] which is also commonly found in mobile devices. This platform is equipped with four "big" Cortex A15 CPUs (quad-core) with a frequency that can be scaled up to 2.0 GHz and four "little" Cortex A7 CPUs that have about half the processor speed (1.4 GHz). The Odroid-XU3 platform also includes a Mali-T628 GPU (not considered here) and 2 GB of main memory. For the evaluation of RAMBO on heterogeneous processing architectures, not only the runtime that is needed to find the best possible configuration is examined but also the energy consumption. This is accomplished by reading from the power measurement sensors INA231 offered by the Odroid-XU3 platform, which report energy consumption for both processor types as well as for the RAM and the GPU. To measure the energy and power consumption of the resource-aware scheduling strategy and its competing MBO approaches, a so-called Relay Reader [526] is used to read out the sensor data in regular

[6] ARM big.LITTLE Technology: https://developer.arm.com/technologies/big-little (accessed Feb. 22nd, 2022).

[7] Odroid-XU3: https://developer.arm.com/graphics/development-platforms/odroid-xu3 (accessed Feb. 22nd, 2022).

intervals of approximately one second via threads for both CPU types. These threads are executed on separate CPUs and do not influence the runtime measurements.

The experimental setup consists of a subset of the setup described in Section 6.4.3. RAMBO is compared with the conventional synchronous MBO approach using the qLCB multi-point infill criterion and with the asynchronous MBO approach, which aims to exploit all available CPU time to solve the optimization problem in parallel and using the Kriging believer criterion [254]. All MBO approaches are evaluated on the 2-dimensional versions of the synthetic functions and executed on 4 CPU cores. The runtime of the objective functions was previously simulated by sleeping for a given time, determined via an additional synthetic function that represented the runtime behavior of the respective objective function. For the power consumption measurements, a real computation is needed. This is accomplished by repeatedly executing a function that draws random numbers. The runtime of this real computation is still controlled via an additional synthetic function that defines the number of repetitions and simulates the time that is needed for calculating the objective value. For the synthetic function that simulates the runtime of the objective functions, the `rosenbrock`(d) function is used, since it delivers a more reliable runtime estimation than `rastrigin`(d) (see Figure 6.20). The output of the `rosenbrock`(2) function is scaled to return values from 5 min to 50 min. The MBO approaches run for 2 hours on $m = 4$ CPU cores, and include all computation overhead and CPU idling. The initial set is generated as with the homogeneous experiments by using the latin hypercube sampling [481] with $n = 4 * d$ configurations. All approaches start with the same initial set in all 10 repetitions.

Tab. 6.8: Ranking for accuracy levels 0.5, 0.1, 0.01 averaged over all problems with `rosenbrock`(2) time function on 4 CPU cores with a time budget of 2 hours [385].

Algorithm	0.5	0.1	0.01
rambo	**1.90 (1)**	**1.77 (1)**	**1.90 (1)**
asyn.ei.bel	2.07 (2)	2.43 (2)	2.63 (2)
qLCB	2.67 (3)	2.63 (3)	2.70 (3)

Table 6.8 lists the aggregated ranks over all 2-dimensional objective functions, grouped by accuracy level. As described in Section 6.4.3, the approaches are ranked with regard to their performance for each of the 10 repetitions and for each benchmark before they are aggregated into the mean. Figure 6.24 shows the corresponding box plots for the time required to reach the three different accuracy levels, as described in Section 6.4.3. The faster an approach reaches the desired accuracy level, the lower the displayed box and the better the approach.

The benchmarks indicate an overall advantage of the new knapsack-based algorithm for heterogeneous systems, especially for the highest accuracy level 0.01. On

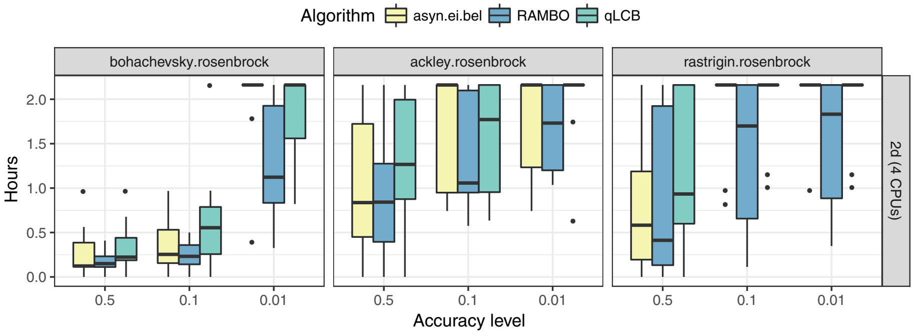


Fig. 6.24: Execution time as a function of the accuracy level for the 2-dimensional objective functions using time function rosenbrock(2) (lower is better) [385].

average, rambo is always fastest in reaching each of the three accuracy levels, and thus converges faster to the optimum in the time budget of 2 hours. In comparison with rambo, the conventional synchronous MBO approach qLCB is unable to reach the accuracy level 0.01 for the rastrigin(2) and ackley(2) functions in all 10 repetitions (see Figure 6.24). The same can be said about the asynchronous MBO approach asy.ei.bel for the bohachevsky(2) and rastrigin(2) functions.

Figure 6.25 shows the box plots for the energy consumption over all 10 repetitions for each benchmark on each CPU type (upper part, Cortex A7, and Cortex A15) and over all CPUs (lower part, combined). Low boxes indicate a small energy consumption. The results indicate that rambo consumes more energy than the default qLCB approach on the “slow” Cortex A7 CPU cores, while it consumes less energy on the “fast” Cortex A15 CPU cores. In comparison with the asy.ei.bel approach, rambo manages to consume less energy on the “slow” Cortex A7 CPU cores. The reason for the higher energy consumption of rambo compared with the synchronous qLCB approach on the “slow” Cortex A7 cores (see upper part of Figure 6.25) lies in the resource-aware scheduling strategy, which is able to utilize the less energy consuming A7 CPU cores more efficiently by mapping jobs to specific cores. Furthermore, only jobs with a runtime smaller or equal to the job with the highest priority are executed within one MBO iteration. Accordingly, longer running jobs with a lower optimization potential are discarded and more MBO iterations can be performed in the given time budget. By contrast, qLCB is not able to map jobs to specific CPU cores; it starts four jobs on the 4 available CPU cores that were proposed by the infill criterion in each MBO iteration, without respect to the heterogeneity of the underlying architecture and the job execution times.

Another contributing factor to the higher energy consumption of the qLCB approach is that it executes more jobs on the more energy-consuming A15 CPU cores due to the OS scheduling. Within one MBO iteration, the OS scheduler migrates jobs from a “slow” A7 CPU to a “fast” A15 CPU for cases where a job on a fast CPU finishes earlier than a job on a slow CPU. This speeds up computation and thus executes more MBO-

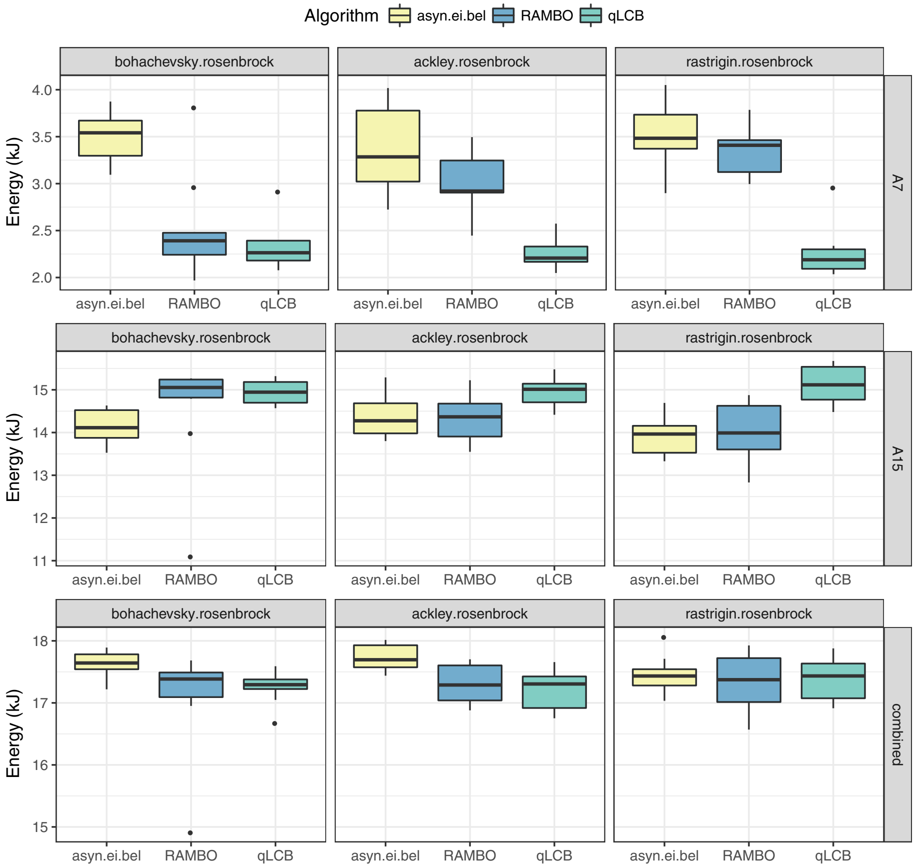


Fig. 6.25: Energy consumption in kJ on the two A15 CPUs (2.0 GHz), the two A7 CPUs (1.4 GHz), and combined consumption on both CPU types across all 10 repetitions for each objective function, with rosenbrock(2) time function and a time budget of 2 hours (lower is better) [385].

iterations. Hence, qLCB has nearly no idle time on the A15 CPU cores. However, the conventional synchronous approach only performs approximately half as many MBO iterations as rambo. In general, rambo executed more job evaluations in the given time than both competing MBO approaches. However, the combined energy consumption on all four CPU cores depicted in the lower part of Figure 6.25 shows that rambo consumes approximately the same amount of energy as qLCB, while it consumes less energy than asy.ei.bel for the bohachevsky(2) and ackley(2) benchmark functions.

The asynchronous asy.ei.bel approach in most cases consumes more energy than rambo since it has nearly no CPU idle time. However, it still converges more slowly to the optimum. The reason for this is that rambo selects more promising candidates with shorter runtimes since it executes only jobs with a runtime shorter than or equal to the most promising candidate, and thus aims to find the cheapest way of evaluations through the model.

Overall, the results show that the resource utilization obtained by the scheduling for heterogeneous architectures in `rambo` enables MBO to converge faster to the optimum without consuming more energy resources than the competing approaches.

6.4.5 Summary: Resource-Aware Scheduling for ML on Multicores

We presented resource-aware scheduling strategies for parallel machine learning algorithms on multicore systems. The resource-aware model-based optimization framework RAMBO was introduced and evaluated. RAMBO can fully use the potential of parallel architectures. This was accomplished with an estimation model for the runtimes of each evaluation of a black-box function to guide the scheduling of configurations to available resources. In addition, an execution priority reflecting the estimated profit of a black-box evaluation was used to guide MBO to interesting regions in a faster, resource-efficient way without directly favoring less expensive configurations. The evaluation results showed that RAMBO converged faster to the optimum than the existing parallel approaches. RAMBO was especially efficient for complex high-dimensional problems, and strongly improved upon the existing approaches in terms of scalability when the number of available CPU cores was increased. Overall, it was shown that the integration of knowledge from the theory of using the underlying hardware (like scheduling) with knowledge about machine learning algorithms achieved results that would not have been feasible without crossing the boundaries of traditional knowledge areas.

6.4.6 Conclusion

The advantage of linking information on underlying hardware platforms with algorithmic knowledge is assumed to exist not only in this particular case. Lowering the walls between disciplines is likely to provide benefits in other cases as well.

7 Memory Awareness

Due to the involvement of massive data and the growing size of trained models, most machine learning techniques are memory intensive. As one of essential components in the von Neumann architectures widely used nowadays, memory is a well-known bottleneck on the execution time, particularly due to the “Memory Wall” problem. That is to say, the access time of memory is way larger than the processor cycle time. In addition, the energy and power consumption required by the memory are known to be significant in the overall system. On embedded systems, which is the focus of this book, such design constraints are amplified and impose great challenges for machine learning techniques. Although the emerging non-volatile memories appear to be promising because of their attractive features, e.g., low leakage power, high density, and low unit costs, they also bring up new design constraints like higher error rates, which might degrade the performance of machine learning techniques. To this end, several optimization and architecture-aware approaches have been proposed to improve the usage of memory and enhance the reliability of learning algorithms.

In this chapter, several techniques are briefly introduced to tackle some of the aforementioned issues related to memory. By leveraging the application-specific knowledge, we demonstrate that the memory footprint can be effectively reduced (see Section 7.1). Given learning models, we can further optimize the memory layout proactively in the model implementation to favor the underlying cache memories with a probabilistic perspective (see Section 7.3). Last but not the least, learning models can be reliable with unreliable memories if we take bit errors into account during the training phase (see Section 7.2). Overall, this chapter tends to suggest that the design constraints of underlying memory can be handled in a post-optimization fashion, within the implementation of learning models, or even earlier at the training phase. The insights presented in this chapter should remain highly relevant in upcoming years, and become more important for future applications along with emerging memory technologies and their new design constraints.

https://doi.org/10.1515/9783110785944-007

7.1 Efficient Memory Footprint Reduction

Helena Kotthaus
Peter Marwedel

Abstract: This section discusses optimization approaches for the efficient memory footprint reduction of machine learning algorithms that are written in the GNU R programming language. The presented optimization strategies target the memory management layer between the R interpreter and the operating system and reduce the memory overhead for large data structures by ensuring that memory will only be allocated for memory pages that are definitely required. The proposed approaches use additional information from the runtime environment, e.g., the short-term usage pattern of a memory block, to guide optimization. The evaluation is based on statistical machine learning algorithms. When the memory consumption hits the point that the OS starts to swap out memory, optimization strategies are able to speed up computation by several orders of magnitude.

7.1.1 Motivation

In order to execute machine learning algorithms on resource-constrained devices, it is important to make efficient use of the available resources. These resources include processors (including runtime), memories, communication bandwidth, and energy. This book includes sample optimization algorithms aiming to achieve resource efficiency. In particular, Chapters 6 to 9 present such sample optimizations. The current section demonstrates the optimization potential memories as resources. Ideally, memories have an infinite capacity, but their size can have a relevant impact on the applicability of certain techniques. This is especially true for resource-constrained embedded systems. The current section focuses on the efficient use of memories for machine learning algorithms written in the R language. The R language is used for many machine learning applications and, therefore, it is considered here. As shown in [387, 503], the R environment has several drawbacks leading to slow and memory-inefficient program execution. In R programs, most data structures are vectors. When the size of a vector is above a certain threshold, the R interpreter allocates a large vector. For each large vector, a dedicated block of memory is allocated, potentially spanning multiple pages. These pages, even when unused, take up memory. When the amount of memory required for computations exceeds the physical memory available to the application, the execution is drastically slowed by frequent page swaps that require I/O, a phenomenon also known as “thrashing”. The performance penalty due to thrashing might render complex computations entirely infeasible.

The current contribution is based on the work of Kotthaus et al. [383, 385, 386]. Section 7.1.2 provides a survey of related work and explains the fundamentals of R's memory management. Section 7.1.3 discusses the page-sharing strategies for efficient memory utilization of R machine learning algorithms. Finally, Section 7.1.4 presents the evaluation results and concludes with a summary.

7.1.2 Related Work and Fundamentals: Memory Footprint Reduction and the R Environment

Related Work - Memory Footprint Reduction The memory optimizations presented in Section 7.1.3 work on a layer between the R interpreter environment and the OS. This enables the optimization of arbitrary R applications, especially memory-hungry machine learning applications, with only small modifications to the R interpreter and without requiring application changes. Thus in the following, the related system-level approaches for reducing memory utilization will be discussed.

In general, related work on utilizing main memory more efficiently can be categorized into two groups: memory compression approaches, often influenced by embedded systems resource constraints, and memory deduplication, which is mostly used in virtualization.

Memory compression tries to reduce the swapping activity of a system by compressing memory contents instead of swapping pages to the secondary storage. Compression approaches share the drawback that a significant amount of processor time is spent on compressing and decompressing memory contents.

By contrast, memory deduplication reduces the memory overhead by mapping virtual pages with identical contents to a single physical page. This is often beneficial in virtualized environments where large amounts of read-only memory, such as shared libraries, are used in multiple virtual machines [626]. However, deduplication can introduce significant computational overhead, since the contents of pages have to be scanned periodically in order to identify pages with identical content. An often used implementation of deduplication that has been the subject of multiple improvements is available in Linux as the *Kernel Samepage Merging* (KSM) [22]. KSM has also been optimized in [133] by introducing a classification scheme based on access characteristics, comparing only pages within the same class to reduce the overhead of page scanning. A memory trace-based evaluation of different deduplication and compression approaches is presented by Deng et al. [169], showing that deduplication yields better results than memory compression.

Sharing memory pages within a single process appears to be a rarely-used concept: on Linux, it is automatically used to map a set of newly allocated virtual pages to a single physical page filled with null bytes. This can cause performance issues in high-performance environments since each write to any newly allocated page will trigger a page fault. Here, an enhancement by Valat et al. [678] was proposed that avoids unnecessary page removal when the application knows that it will overwrite a page in the near future. A language-level version of this *copy-on-write* technique, operating on objects instead of memory pages, is sometimes implemented using reference counters [665]. The R language also implements a copy-on-write scheme. Here, the complete object (potentially spanning multiple pages) is copied when it is modified, resulting in page duplications for partial modifications.

OS level optimizations lack knowledge about the specific memory behavior of the runtime environment. Although some information can be used to improve the time needed to detect duplicates, the application-specific knowledge of why the data was copied in the first place is ignored. By contrast, the memory optimization presented in Section 7.1.3 employs specific knowledge about the interpreter state to reduce the number of pages that need to be scanned for identical content and *proactively* avoids the main sources of identical-content pages from object allocation and duplication by optimizing the copy-on-write mechanism for partial object modification.

Fundamentals – The R Environment The *lifecycle of an object*, (e.g., a vector data structure) in the R runtime environment starts with its allocation. In R, vectors are assumed to consist of a contiguous block of (virtual) memory. Depending on the size of the object, the R interpreter uses a system of multiple memory pools for vector objects with a data size of up to 128 B. For larger vectors, memory is allocated directly via the malloc C library function instead of pooling the allocations. This reduces the memory fragmentation when many small objects are created and some of them are released. The R language does not require the programmer to explicitly manage memory; a garbage collection is needed to automatically free memory. The garbage collector in R is a mark-and-sweep, non-moving, generational collector. It can be manually triggered, but it also starts automatically when the interpreter is in danger of running out of heap space.

The R interpreter ensures that an allocated object is always initialized—either by explicit initialization or implicitly by writing the results of a computation to it. After the object is allocated and initialized, it can be used as input for various R functions such as the plus operator. The fact that functions can modify an object, in conjunction with R implementing *call-by-value* semantics, means that objects need to be copied when being passed to a function. However, at this point a copy-on-write optimization is triggered: copying an object is done by merely sharing the reference; the actual copy is delayed until the object is modified (if at all). The interpreter now has two references to the same object, which may be modified later. When this modification happens, the copy process is triggered and a full copy of the affected object, potentially spanning multiple pages, is created using the interpreter-internal *duplicate* function. This is illustrated in Figure 7.1.

On the left-hand side, a large R vector object consisting of a header *H* and four pages *A* to *D* is shown both in *virtual memory* on the top (marked with dotted lines) and its corresponding allocated *physical memory* on the bottom (solid lines). On the right-hand

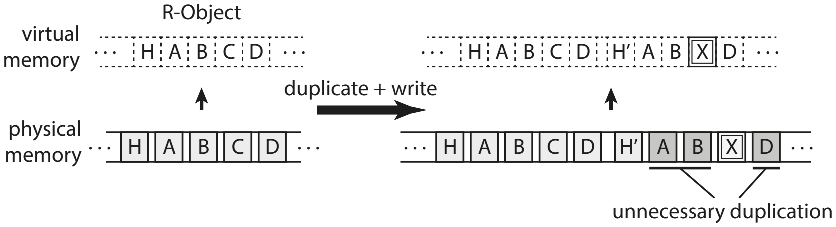


Fig. 7.1: Example of the copy-on-write mechanism in the R interpreter. R copies (duplicates) at object level instead of page level granularity [385].

side, the situation after a duplication that was triggered by a write access is shown. Now there are two R objects, shown in the virtual memory on top and their corresponding physical memory on the bottom. In one of the copies, page *C* was modified and is now marked as *X*, and the copy has its own header *H'*. Although unmodified, the R interpreter needs to use additional memory to create duplicates of pages *A*, *B*, and *D* (marked in gray) since it assumes that objects are organized as contiguous blocks of memory and thus it has to duplicate at *object-level granularity*.

The memory optimization presented in this contribution has the goal of reducing this memory overhead by copying only parts of the object, sharing the same memory pages between multiple objects as long as they are not modified. This scheme is transparent to the interpreter's memory management including the garbage collection, requiring only small changes in memory allocation and freeing, as well as in the duplicate function. This optimization will be presented in the next section.

7.1.3 Memory Footprint Reduction via Page Sharing Strategies

Different optimization strategies are combined for the efficient memory footprint reduction of machine learning algorithms implemented in the R language. The first strategy that proactively *avoids the duplication* of memory pages is based on optimizing the allocation and duplication mechanisms of the R interpreter. This approach is further refined by a second strategy using *static annotations* to reduce the optimization overhead and by *dynamic refinement* using a page content analysis for page deduplication to increase the amount of shared memory.

Page Duplication Avoidance As shown in the previous section, the R interpreter can only allocate complete objects that potentially span multiple pages. The first part of the memory optimization is based on the object allocation mechanism of R. To enable the allocation and thus the sharing of memory at *page-level granularity* instead of object granularity, a custom memory allocator is employed when a large vector has to be allocated, as shown in Figure 7.2. When the internal function of the R interpreter *allocVector* is called to allocate a large vector, the optimized interpreter selects between the *custom allocator* to share memory on page granularity or the *default malloc* function if this is not required. In both cases, the allocated memory is accessible within the address space of the R interpreter. The custom allocator uses a memory management scheme similar to standard virtual memory schemes commonly used in Operating System (OS) kernels. For ease of implementation, it is completely implemented in the user space. The downside of such a user-space implementation is that it needs to replicate certain data structures that are already present in the OS (e.g., for mapping virtual to physical memory) because those OS kernel data structures are not sufficiently exposed to user space. This replication could be avoided by implementing the optimization in the Operating system kernel (cf. [383]), but this is significantly more invasive and not applicable in many environments where the user has no control over the Operating system kernel. Since the user space has no direct access to physical memory, a single file located on a RAM disk (see *custom heap* in Figure 7.2) is used.

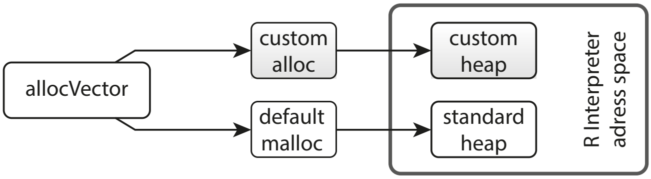


Fig. 7.2: Memory allocation scheme for dynamic page sharing [385].

The allocation of physical memory from this file is realized via a simple free-bitmap based allocator. The file can be dynamically enlarged if needed. Mapping physical pages into the virtual address space of the R interpreter can be accomplished by utilizing the mmap Unix system call. For changing the access permissions of these physical pages, the mprotect system call that modifies the settings of the memory management unit of the processor is employed. Besides these system calls, an additional page table is needed to enable the mapping from a virtual address to a physical address. For simplicity reasons a hierarchical page table with the same four-level structure as used by the processor is implemented. To enable the sharing of pages, the user space memory management system needs to map the same physical page to multiple locations in virtual memory. Therefore, a reference counter is required for each physical page. A reference counter greater than 1 indicates that the page is shared between multiple objects or multiple times within one object.

To avoid the zero-initialization of allocated large vector objects, a *global shared zeroed page* is utilized. This also ensures that memory is only allocated for pages that are actually written to at a later time. Figure 7.3 illustrates an example for this optimized R object allocation. Here, the custom memory allocator was asked to allocate an object with a total size of five pages. While the object has the requested size of five pages in

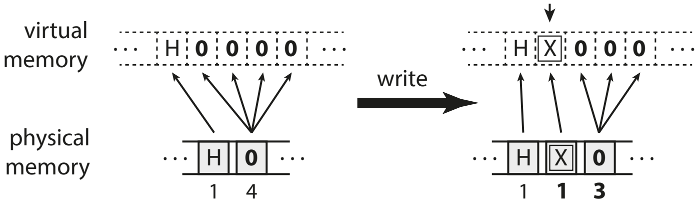


Fig. 7.3: Optimized object allocation via sharing a global zeroed page [385].

virtual memory (dotted, left upper part), physically it only consists of two pages (left lower part). Those two pages are a single non-shared page, marked with *H* for header in the beginning, followed by a shared page, marked with ***0***, called the global zeroed page. The numbers in small print below the physical pages are the *reference counters*. The zeroed page has a reference counter of 4 since it is shared four times within the allocated object (mapped four times into virtual memory). The shared zeroed page is filled with zero-bytes. The concept of prepared zeroed pages is already implemented in OS kernels. However, the standard R interpreter does not benefit from this concept since it immediately writes to all memory that it allocates for initialization. The non-shared initial page *H* is required as it will contain not just data but also the object header. The R interpreter writes this object header to the front of the allocation area. Since it will be updated frequently (e.g., during garbage collection), it is not shared between multiple objects. Since the header page *H* is mapped only once, its reference count is 1.

The R interpreter now has the illusion that it has allocated five pages of memory, even though only two pages are allocated physically. To sustain this illusion, the optimized allocation mechanism has to ensure that any write access to a virtual page that points to a shared physical page can be detected and handled. If such a write access is not handled correctly, it affects not only the intended virtual page but also all virtual addresses where the same physical page is shared. Therefore, all pages with a reference counter greater than 1 are marked as *read-only*, ensuring that a write access triggers a segmentation fault. This fault is caught by a *signal handler* that performs the unsharing of the affected page. To determine the affected physical page the handler uses the virtual address of the write access. It then allocates a new page, copies the contents of the original page to it, and replaces the page that caused the segmentation fault with the new one. The resulting situation is shown on the right side of Figure 7.3: one of the instances of the zeroed page that was written to was replaced with a new page marked with *X*. This updates the reference count of both the zeroed page and the newly allocated page. Since the new page is only mapped once, it can now be marked as read-write so that further access no longer requires special handling.

As noted, the R interpreter can only copy on the object level. Thus, if an object consists of multiple pages, parts of the copy may end up with the same content as the original (see Figure 7.1). To avoid this, the *duplicate mechanism* of the interpreter is optimized by improving the granularity of the copy from object level to page level.

While the allocation optimization avoids the immediate allocation of pages by using the global zeroed page, the duplicate optimization allows the reuse of already-allocated pages of the original object instead of allocating new pages. An example of the duplicate optimization is shown in Figure 7.4.

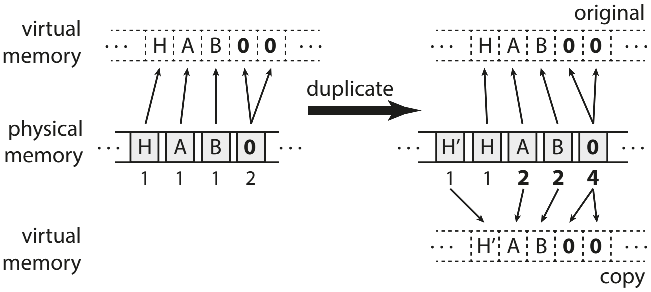


Fig. 7.4: Optimized copy mechanism on page-level instead of object-level granularity via page sharing [385].

The left side shows the situation before the duplication is shown: an object occupies five virtual pages, two of which reference the global zeroed page. Unlike the original R interpreter that would need to allocate five new pages for the copy of this object, the optimized version reduces this to a single allocated physical page. This is shown on the right side with the original object at the top and its copy at the bottom. Here, a *virtual-only copy* of the first page that contains the object header is not created, since the header of the copy is updated immediately by the R interpreter after the duplication. This would otherwise trigger an unsharing of this page. To avoid the overhead of this event, the optimized duplication immediately creates a physical copy of the header page. Most of the pages of the original object are now mapped twice in virtual memory and their reference counters are updated. Both the original and copy are marked as read-only to allow for unsharing on write access.

Overall, the finer copy granularity of the optimization enables storing both the original and copied objects from the example in just five pages of memory. By contrast, the original R interpreter would need ten pages of memory to store the same objects. Although the mechanisms of sharing pages during allocation and duplication described above always result in a valid view on memory for the interpreter, there are cases where additional overhead is caused that can be avoided by further refinements described in the next subsection.

Static Refinement via Annotations To reduce the runtime overhead caused by proactively avoiding page duplications, a static refinement consisting of two kinds of annotations is applied. The first annotation is based on the expected utilization of an object immediately after allocation and the second annotation is based on the size of the allocated object.

The optimized memory allocation (see Figure 7.3) reduces the memory footprint by using a global zeroed page, assuming that not all pages of the allocated object will be written to immediately. However, this assumption is not always valid. For instance, (built-in) vector arithmetic functions in the R interpreter allocate a new object and immediately write to all pages of it to store their results. This would cause a segmentation fault for the first write of every page, triggering the memory allocation for all pages of the object. These segmentation faults cause runtime overhead that would not occur when allocating an object with non-shared pages.

To avoid this overhead, *annotations* are placed in the C source code of the R interpreter built-in functions where newly allocated memory is completely overwritten directly after allocation. Here, the custom allocator returns an object where every virtual page references a new physical page, so no segmentation faults will be triggered by write accesses. Although these R objects do not save memory on allocation, they still have the opportunity for later optimizations, e.g., when they are duplicated. Currently, the annotations for these *"full-overwrite"* functions need to be manually placed in the R interpreter's C source code by locating calls to *allocVector*, followed by loop structures that write to every element of the newly-allocated object. Those manually placed annotations could also be automated by a static code analysis checking for allocation calls followed by loops writing to the newly-allocated object.

The second annotation for reducing the runtime overhead incurred by the optimization relates to the size of the allocated object. The R interpreter can allocate objects with a variety of sizes, not all of which span multiple pages. The optimized custom allocator is therefore enabled only for object sizes that indicate a potential for page sharing. Here, the potential is limited for smaller objects. The first page of an object stores not just data but also the frequently modified object header that is therefore never shared. Thus R objects smaller than two pages of memory are passed to the standard, non-sharing memory allocator. This size limit could also be used as a tunable parameter to select a trade-off between memory savings and runtime overhead.

Dynamic Refinement via Page Contents In addition to the above-described static refinements, an additional dynamic refinement for increasing the number of shared pages is applied. During the execution of an R program, allocated objects are updated with the results of calculations. Those updates can result in multiple distinct pages with the same contents, which opens up the opportunity for sharing those pages. The general idea of locating identical objects in a system and saving memory footprint by reducing them to a single object is known as deduplication.

The memory optimization employs a restricted version of locating identical contents. Here, the content scan only checks for pages identical to the already existing global zeroed page. The *deduplication of zeroed pages* is illustrated in Figure 7.5. On the

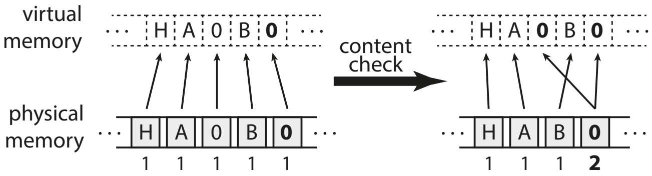


Fig. 7.5: Deduplication optimization for zeroed pages [385].

left side, the situation before the page content scan is shown where an object contains two identical zero pages. One of those pages is already mapped to the global zeroed page (shown in bold), while the other uses a separate physical page. On the right side, the situation after deduplication is shown. Here, the *content check* has detected the separate copy and mapped its virtual page to the global zeroed page, freeing the memory used for the unnecessary duplicate.

Although a general scan that is able to detect duplicated pages with arbitrary content could be applied, such a scan would incur a significant runtime overhead (e.g., due to the calculation of hash values) compared to scanning just for zeroed pages. While a scan for zeroed pages can use an early abort condition as soon as a non-zero element is found, a scan for arbitrary content would need to check the full content of all pages in the system. The overhead incurred by deduplication of zeroed pages is influenced by the frequency of the content check and by the number of pages that need to be scanned. To reduce this overhead, the scan is only activated after the completion of a garbage collection in the interpreter. This avoids scanning the pages that would soon be discarded and also provides a natural regulation mechanism for the frequency of content checks, as the frequency of garbage collection depends on the memory requirements of the executed program.

With the deduplication optimization, pages that were previously excluded from sharing the global zeroed page, in arithmetic vector operations, say, can now be dynamically shared. Thus, both the static and the dynamic refinements of the memory optimization complement each other. Details on the interaction of the refinement strategies and the general page duplication avoidance strategy can be found in a separate publication [384].

7.1.4 Evaluation: Memory Footprint Reduction Strategies

The results obtained by applying the proposed memory optimization strategies for R to real-world machine learning benchmarks are presented in this section. Both, the evaluation results related to the memory consumption and the runtime effects of the page sharing optimization strategies will be discussed.

Experimental Setup For the following experiments, a computer equipped with a 2.67 GHz Intel Core i5 M480 CPU and 6 GB of RAM, using a 64-bit version of Debian Linux 7.0 as the operating system is used. On this system, memory pages have a size of 4096 bytes. Although a larger page size than the system page size could be used for the memory optimization, the same size was chosen as it is expected to maximize the amount of memory that can be shared (using a smaller page size than the system size is inefficient since the optimization relies on the hardware Memory Management Unit (MMU) for efficient page access protection). To evaluate the proposed memory optimization approach, the memory usage and runtime of the R interpreter including the described optimizations is compared to the standard GNU R interpreter. Both the standard as well as the optimized interpreter are compiled using GCC version 4.7.2 with the default flags (-O2) selected by the build system of R version 3.1.0.

The standard memory measurement functions for user space functions in Linux measure only the virtual memory of a process. Since the optimization approach maps the same physical page multiple times into virtual memory, these functions are not sufficient for the evaluation. They are not able to measure the amount of physical memory saved since they only count every virtual instance of a shared physical page. Therefore, a separate memory measurement function was created. To measure the amount of memory allocated by the default allocator, the standard allocation functions such as malloc are overwritten with versions that track the current total amount of memory allocated and the original functions are called afterwards. For the optimized custom allocator, the number of physical pages that need to be reserved is directly tracked along with the size of the memory management data structures. With these mechanisms, the allocated physical memory can be measured accurately.

For the evaluation of the optimization, two different benchmark sets are applied. The first set is a shorter-running set of benchmarks, selected from the R benchmark 2.5 suite [274], which was originally developed to measure the performance of various configurations of the R interpreter (in the following denoted by GU) plus one additional benchmark, as listed in Table 7.1. The R benchmark 2.5 suite was also applied in other optimization approaches that focus on dynamic compilation for R [353]. To analyze if the memory optimization is also beneficial for algorithms that already try to reduce the memory footprint by using application-specific knowledge, the additional benchmark *glmnet* is included. This benchmark utilizes an existing sparse matrix optimization implemented as an R package. For accurate measurements, the iteration counts for the outer loop of each benchmark were scaled to result in a runtime of approximately 1 minute with the standard R interpreter.

The second set of benchmarks is based on a set of publicly available long-running real-world machine learning benchmarks [384], listed in Table 7.2. The choice of these classification algorithms is based on the method's popularity and the availability of its implementation. The default parameters or, if available, the implementation's internal auto-tuning process was used to configure the algorithm parameters. The input dataset is a 2-class classification problem and has a sufficiently large number of observations

Tab. 7.1: Misc Benchmark Set.

Benchmark	Description
GU/08a-1	Linear regression over a 3000 × 3000 matrix
GU/08a-2	FFT of 2 400 000 random values
GU/08a-3	Inverse of a 1600 × 1600 random matrix
GU/08a-4	Greatest common divisors of 400 000 pairs (recursive)
glmnet	Regression using glmnet on a sparse 20 000 × 1000 matrix

Tab. 7.2: Machine learning benchmark set.

Benchmark	Description
ada	Boosting of classification trees
gbm	Gradient boosting machine
kknn	*k*-nearest neighbour classification
lda	Linear discriminant analysis
logreg	Logistic regression (binary classification decision derived using a probability cutpoint of 0.5)
lssvm	Least-squares support vector machine
naiveBayes	Naive Bayes classification
randomForest	Random classification forest
rda	Regularized discriminant analysis
rpart	Recursive partitioning for classification trees

to achieve accurate results. The machine learning benchmarks were configured to use a 20-fold cross-validation. The size of the input dataset (15 000 samples with 200 numeric features) was chosen to ensure that the runtime of the fastest algorithms is approximately one minute on the standard interpreter. To allow for a better comparison of the memory requirements, the same dataset was applied to all machine learning algorithms.

Each benchmark was executed 10 times with the standard and the optimized version of the R interpreter. The results are given as the arithmetic mean of these 10 executions. To make the results reproducible, the random number seed is selected as a fixed value placed as the first statement in each of the benchmarks. Each repetition was started in a fresh interpreter process; hence initialization costs are included in the measurements (an expected overhead on the order of one second or less). The R interpreter does not use adaptive optimizations. All system services that might interfere with the measurements were disabled. Both runtime and memory usage were measured simultaneously. For these measurements, we calculated a 95 % confidence interval and the ratio of the means using the percentile bootstrap method. We use geometric means here to reduce the influence of outliers.

Memory Consumption To analyze the benefits of the page sharing optimization techniques with regard to the memory consumption we evaluate the global peak memory usage and the average memory usage of each benchmarks. The *Peak usage* represents the maximum amount of memory that was consumed during execution of a benchmark. However, the peak memory consumption does not represent information about changing memory usage over time, since the peak memory usage may occur only for an instant of time depending on the benchmark. To gain a complete view of the memory consumption the short-term peak usage is measured in intervals of 1 second, resulting in a memory-over-time profile. The *Average usage* of memory is calculated as the arithmetic mean of these 1 second measurements and used as a second indicator to allow easier comparisons of the memory behavior.

Figure 7.6 shows the peak (*Peak usage*) and average (*Average usage*) memory consumption of each benchmark running with the page-sharing optimization. The 100 % baseline represents the standard R interpreter without optimizations. Values below this baseline indicate relative memory savings realized by the page sharing strategies. Error bars have been omitted as the confidence intervals were smaller than 0.5 % for all values. The detailed values are presented in Table 7.3, including the number of pages identified as shareable by the content check. They indicate the optimization potential of the dynamic refinement (deduplication of zero pages).

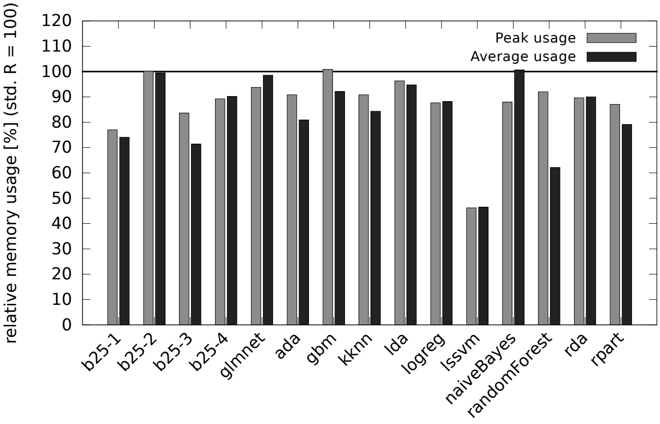


Fig. 7.6: Relative memory usage with page-sharing optimization compared with standard R (lower is better). The 100 % baseline represents the standard R interpreter without optimizations. Geometric means for the memory savings are 13.6 % for peak and 18.0 % for average memory usage [385].

The gain for reducing the peak memory usage (*GainP*) of the standard R interpreter (*StdPeak*) ranges from −0.9 % for gbm to 53.8 % for lssvm. However, the negative values in the columns *GainP* and *GainA* of Table 7.3 indicate that the page-sharing optimizations do not gain memory savings for three of the benchmarks. Here, the peak memory

Tab. 7.3: Memory Optimization Results: StdPeak – peak memory usage by the standard R interpreter; OptPeak – peak memory usage by optimized interpreter; GainP – relative peak memory reduction achieved by optimization; StdAvg – average memory usage by the standard interpreter; OptAvg – average memory usage by optimized interpreter; GainA – relative average memory reduction achieved by optimization; ZPG – number of zero pages found by the content check [385].

Benchmark	StdPeak [MB]	OptPeak [MB]	GainP [%]	StdAvg [MB]	OptAvg [MB]	GainA [%]	ZPG [#]
GU/08a-1	296.2	228.1	23.0	259.6	192.2	25.9	13
GU/08a-2	131.1	131.4	-0.2	128.8	128.0	0.6	13
GU/08a-3	197.2	164.8	16.4	157.7	112.6	28.6	37 919
GU/08a-4	134.2	119.7	10.8	127.2	114.6	9.9	194 892
glmnet	354.9	332.8	6.2	249.5	246.0	1.4	46 877
ada	187.2	170.1	9.1	156.0	126.2	19.1	2 031 992
gbm	191.5	193.2	-0.9	147.7	136.0	7.9	464
kknn	316.5	287.6	9.1	274.0	231.0	15.7	421
lda	216.2	208.2	3.7	184.8	175.1	5.3	20 447
logreg	213.0	186.7	12.3	184.7	162.8	11.9	955
lssvm	1365.1	631.0	53.8	820.2	381.1	53.5	3 972 699
naiveBayes	143.6	126.2	12.1	80.8	81.3	-0.6	78
randomForest	565.5	520.4	8.0	390.8	242.7	37.9	1 130 650
rda	254.1	227.7	10.4	197.0	177.3	10.0	707
rpart	144.5	125.8	12.9	130.7	103.3	20.9	56 214

consumption for two of the benchmarks (gbm, GU/08a-2) and the average memory consumption for one benchmark (naiveBayes) increase slightly. This is caused by the additional data structures that are needed for the internal handling of memory pages.

For gbm, a reduction of the average memory usage by 7.9 % (*GainA*) is achieved. For naiveBayes the situation is reversed: the optimization saves 12.1 % of its peak memory usage while its average memory usage (–0.6 %) is slightly increased. Since the number of pages recovered by deduplication (see column *ZPG*) is low (78), the savings of the peak memory usage are assumed to be induced by the proactive avoidance of page duplicates via the optimized allocation and duplication strategies. For GU/08a-2, the optimization was not able to save memory for peak memory usage and no meaningful amount for the average memory usage was saved (*GainA*). The reason why GU/08a-2 does not gain from the optimization is that even though it uses large vectors with 2.4 million elements, it allocates a vector that is immediately filled with random numbers. Thus, it does not profit from the optimized allocation and the content check can only recover a low number of zero pages as shown column *ZPG* (13). GU/08a-2 does not use any object duplication. Therefore, the optimized duplication has no potential for saving memory.

Even though the page-sharing optimization results in a slight increase of peak or average memory usage for the three benchmarks described above, all of the twelve other benchmarks benefit from the optimization with savings in both peak and average memory usage. We compute the geometric mean over all 15 benchmarks, thereby avoiding the impact of outliers on the geometric mean. The result is a reduction of peak memory usage by 13.6 % and a reduction of average memory usage by 18.0 %. Here, the highest amount of memory that could be saved occurs in the `lssvm` benchmark with 53.8 % for peak usage and in `randomForest` with 37.9 % for the average memory usage. Both of these benchmarks have a high number of zero pages recovered by the content check. Thus for those benchmarks, the reduction of the memory footprint is not just triggered by the allocation and duplication optimization but also by the dynamic refinement that deduplicates zero pages.

Table 7.3 shows summarized values for the memory consumption over the complete runtimes of all benchmarks. To gain additional insights into the memory consumption behavior, the complete profile of the memory usage over runtime will be also analyzed. The four most interesting memory consumption profiles for the benchmarks (`glmnet`, `gbm`, `randomForest`, and `naiveBayes`) are shown in Figure 7.7. For each benchmark, the run with the execution time closest to the average of its 10 executions is selected. The confidence intervals over all 10 runs of each benchmark are less than 1 %, thus the figure shows only the data from a single run. The x-axis represents the runtime in seconds while the y-axis represents the corresponding memory consumption of the benchmark. Both the profile for the standard R interpreter (yellow curves) and the interpreter including the page-sharing optimizations (green curves) are presented. The straight lines at the top indicate the peak memory usage, while the dotted lines mark the average memory usage.

glmnet As mentioned, the `glmnet` benchmark utilizes an already-existing memory optimization for sparse matrices. It is included in the evaluation to determine if the page-sharing optimization can offer additional memory savings in the presence of an optimization that applies specialized application knowledge. In the top left of Figure 7.7 the memory-over-time behavior of this benchmark is illustrated. While there is only a small improvement for the average memory usage (see dotted green line), 6.2 % of the peak memory consumption is saved (see lines on the top). The memory consumption curves show that at all local memory peaks the optimized version of the R interpreter saves a small amount of memory while the memory consumption during the remaining parts of the execution is largely unaffected. This results in only a minor reduction of the average memory consumption. Still, even in the presence of a very specific optimization for sparse matrices the page sharing optimization can offer additional memory savings. As can be seen from column *ZPG* (Table 7.3), savings are triggered by a large number of pages recovered by deduplication (46 877).

gbm Not all benchmarks benefit from the content checks, though. For example, Table 7.3 shows that in `gbm` only 464 zero pages are recovered. This benchmark benefits more from the optimizations in allocation and duplication. The corresponding memory-over-time behavior is shown in the top right of Figure 7.7. Here, the optimization does not reduce the peaks of the memory consumption, but there is a

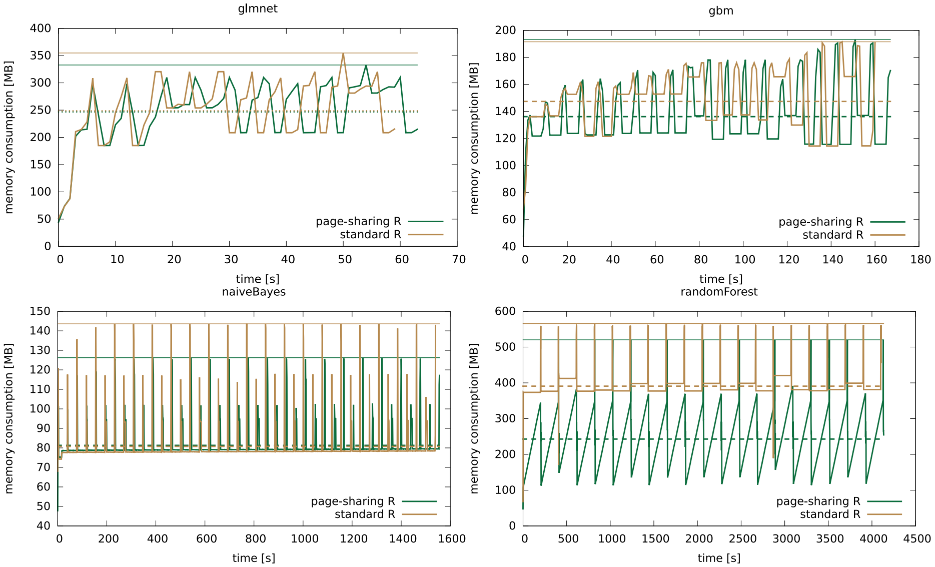


Fig. 7.7: Memory consumption over time profiles for benchmarks with different memory behavior for the standard R interpreter vs. the interpreter with the page-sharing optimization. Lines at the top indicate the peak memory usage; dotted lines mark the average memory usage [385].

marked reduction of memory usage in the valleys between the peaks, reducing the average memory consumption by 7.9 %.

naiveBayes Another benchmark that does not benefit from the content checks is `naiveBayes` with just 78 zeroed pages recovered. Its memory-over-time profile is illustrated in the bottom left of Figure 7.7. In `naiveBayes` only the peak memory usage is reduced by the optimization (large distance between the straight lines at the top), but the average memory usage (small distance between the dotted lines) is not affected. The profile also shows that `naiveBayes` has much smaller peaks compared with `gbm`. Thus, the large reduction of memory usage at those peaks results only in a small effect on the average memory consumption.

randomForest Finally, `randomForest` in the bottom right of Figure 7.7 represents one of the benchmarks where the recovery of zeroed pages triggers high memory savings. Here, the content checking reclaims 1 130 650 pages, which corresponds to slightly more than 4 GB of memory. The `randomForest` profile shows a saw-tooth curve for the optimized interpreter (see green curve). This indicates that the benchmark uses large blocks of memory that are slowly written to. For the page sharing optimizations, this represents an ideal memory usage pattern, as the allocation of memory is delayed until the benchmark writes data to it. This results in a 37.9 % improvement of the average memory consumption (large distance between dotted lines)—the average time during which the benchmark has a high memory consumption is thus reduced.

Looking back at the profile of `glmnet` (top left), the green curve that shows the profile for the optimized interpreter is longer than the yellow curve for the standard interpreter and there is an increasing shift between the peaks of both curves over time. The reason for this lies in the additional CPU time needed to provide the page-sharing optimizations. The runtime overhead induced by the memory optimization will be referred to in the next paragraph.

Runtime Overhead There are multiple reasons for the runtime overhead caused by the optimizations. For the 15 benchmarks shown so far, 4 have a runtime overhead of ≤ 1 %, an additional 6 have an overhead ≤ 5 %, an additional 2 have an overhead around 8 %, and the remaining 3 have an overhead between 13 % and 17 %. More details on the overhead are available in a separate publication [385].

Runtime Reduction In all previous measurements, the RAM available in the system was sufficient to hold all data used by the benchmark. If this is not the case, runtime overhead can become insignificant. This will be illustrated in the following. When the amount of RAM in the system is too small to hold all data required, there are situations where the proposed memory optimization is also able to reduce the runtime of the benchmark instead of adding overhead. This is due to frequent page swaps requiring I/O when the total capacity of RAM is exceeded, also known as “thrashing”. To analyze this situation, two benchmarks are considered. The first one is the lssvm benchmark where the optimization provides a large reduction in memory consumption. The second benchmark is an instance of logreg where the optimization provides only smaller memory gains.

For the analysis, the memory requirements of the benchmarks need to be increased beyond the capacity of the RAM in the system. Instead of increasing the dataset size of both benchmarks, the system is limited to just 1 GB of RAM, since the runtimes of the benchmarks do not scale linearly with the dataset size, leading to excessively high execution times. However, since the logreg benchmark has a much smaller memory consumption than 1 GB, the dataset size for logreg is increased to 70 000 samples with 300 numeric features. This increases the memory requirements of this benchmark to approximately the same level as lssvm. This still results in acceptable execution times for logreg.

Table 7.4 shows the results for the previous 6 GB system configuration and the limited 1 GB RAM configuration for both benchmarks. The logreg benchmark is now shown as logreg-2 because it was executed with the previously described larger dataset. In the 1 GB configuration, the system had to swap for both the standard and optimized interpreters, resulting in a large increase in runtime compared with the 6 GB configuration. The peak memory usage for the interpreters is identical in both configurations while the average memory usage differs because this value is time-dependent and thus influenced by swapping. This swapping also increases the variability in the runtime

Tab. 7.4: Evaluation results with two configurations of RAM; Std – standard R interpreter; Opt – optimized R interpreter; Gain – relative gain; Speedup: runtime speedup factor (Std / Opt). Confidence intervals (C) for runtime are shown; others are ≤ 0.8 % [385].

Benchmark	Std Peak [MB]	Opt Peak [MB]	Gain Peak [%]	Std Avg [MB]	Opt Avg [MB]	Gain Avg [%]
logreg-2, 1 GB	1228.2	1094.8	10.9	965.7	789.6	18.2
logreg-2, 6 GB	1228.2	1094.8	10.9	967.8	823.2	14.9
lssvm, 1 GB	1365.1	631.1	53.8	970.0	381.3	60.7
lssvm, 6 GB	1365.1	631.0	53.8	820.2	381.1	53.5

Benchmark	Std [s]	Opt [s]	Speedup (CI)
logreg-2 1 GB	6395.5	5785.6	$1.105^{1.144}_{1.071}$
logreg-2, 6 GB	579.8	598.5	$0.969^{0.971}_{0.967}$
lssvm, 1 GB	3080.3	593.8	$5.188^{5.350}_{5.029}$
lssvm, 6 GB	530.5	601.2	$0.882^{0.885}_{0.880}$

measurements, thus the confidence intervals for the speedup factors are also included (see lower part of Table 7.4).

Reducing the available memory from 6 GB to 1 GB drastically increases the runtime for both versions, the standard R interpreter (*Std*) and the interpreter including the memory optimization (*Opt*). Still, the reduction in memory consumption for logreg-2 has turned the slowdown (factor 0.969) in its 6 GB configuration into a small speedup (factor 1.105) when the RAM is limited to 1 GB. Depending on the benchmark and its memory usage pattern, a different situation could also happen. In the worst case, the content check of the optimized interpreter touches a large number of pages, forcing them to be swapped in. This additional swap activity can increase the runtime so that the gains from a reduced memory footprint may become irrelevant. The second benchmark `lssvm` shows something closer to the best case for the optimization: Here, the page-sharing optimization manages to save enough memory to avoid swapping. In this case, significant speedups are gained, as shown in the lower part of Table 7.4 for the 1 GB configuration of `lssvm`.

Similar to `logreg-2`, memory usage does not vary much between both configurations (see upper part of Table 7.4). Considering the runtime results, the optimized interpreter *Opt* only needs 593.8 seconds to run the `lssvm` benchmark. This is almost unchanged from the 6 GB configuration (601.2 seconds). By contrast, the standard interpreter *Std* has now increased its runtime to 3080.3 seconds (51.3 min.) when limited to 1 GB of RAM. This makes the overhead of the memory optimization irrelevant because the time gained by avoiding page I/Os is much larger. The page-sharing optimization enables a speed up by a factor of 5.2 for llsvm by reducing the peak memory consumption by 53.8 %. This speed up is also illustrated in Figure 7.8. It shows the

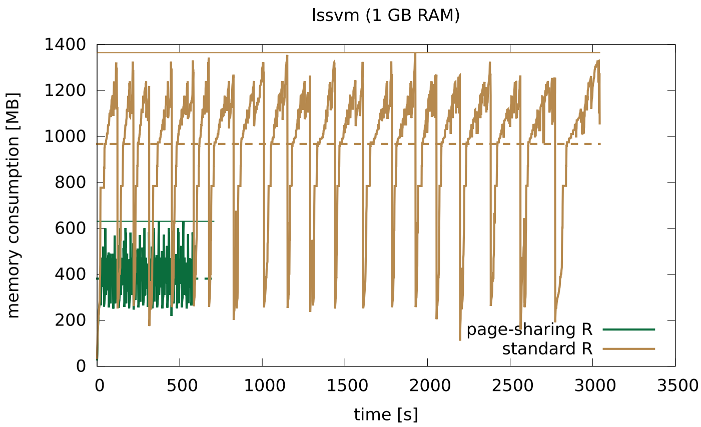


Fig. 7.8: Memory consumption over time profile for the lssvm benchmark. Speed-up reaches a factor of 5.2 on a system with 1GB of RAM. Solid lines indicate the peak memory and dotted lines mark the average memory usage [385].

memory consumption profile for one exemplary execution of the `lssvm` benchmark. This demonstrates that reducing the memory consumption with the page-sharing optimization can significantly improve the runtime for memory-hungry benchmarks if the available RAM is constrained. In turn, this can enable the processing of larger datasets.

7.1.5 Summary

The R interpreter induces a large memory overhead in the machine learning applications, due to wasteful memory allocation [387]. The goal of the presented memory optimizations was to enable efficient memory utilization, especially for memory-hungry R applications like machine learning algorithms. To accomplish this goal, this contribution presented an application-transparent memory optimization employing page sharing at a memory management layer between the R interpreter and the operating system's memory management. The optimization benefits a large number of applications since it preserves compatibility with the available software libraries that most R programs are based on, and covers one of the most important resource bottlenecks of machine learning algorithms. By concentrating on the most rewarding optimizations—the sharing of zero-filled pages and deduplicating at the page level instead of the object level—the overhead of more general OS level memory optimization approaches such as deduplication and compression is avoided. With the proposed optimization, considerable reductions of the memory consumption for a large number of typical real-world benchmarks have been achieved. This is an important step towards processing larger input sizes. It also significantly speeds up the computation in cases where previously pages had to be swapped out due to insufficient main memory.

7.1.6 Conclusion

Designers of machine learning applications should be allowed to focus on the functionality of their algorithms. In order to execute these on resource-constrained embedded systems, possible optimizations of the implementation should be performed. The presented work demonstrates the benefits of such optimizations for the case of memory resources. In addition to the other optimizations in this contribution, we conjecture that more memory-oriented optimizations exist and propose that they should be exploited in order to execute machine learning algorithms in particular on hardware with limited amounts of memory.

7.2 Machine Learning Based on Emerging Memories

Mikail Yayla
Sebastian Buschjäger
Hussam Amrouch

Abstract: Due to the exceptional recent developments in deep learning, many fields have benefited from the application of Artificial Neural Networks (ANNs). One of the biggest challenges in ANNs, however, is the resource demand. To achieve high accuracy, ANNs rely on deep architectures and a massive amount of parameters. Due to this, the memory sub-system is one of the most significant bottlenecks in ANNs.

To overcome the memory bottleneck, recent studies have proposed using approximate memory in which the supply voltage and access latency parameters are tuned for lower energy consumption and for faster access times. However, these approximate memories frequently exhibit bit errors during the read process. Typical software solutions that monitor and correct these errors require a large processing overhead that can negate the performance gains of executing ANNs on these devices. Hence, error-tolerant ANNs that work well under uncorrected errors are required to prevent performance degradation in terms of accuracy and processing speed.

In this contribution, we review the available and emerging memories that can be used with ANNs, with a focus on approximate memories, and then present methods to optimize ANNs for error tolerance. For memories, we survey existing memory technologies such as Static Random-Access Memory (SRAM) and Dynamic Random Access Memory (DRAM), but also present emerging memory technologies such as Ferroelectric FET (FeFET), and explain how the modeling on the device level needs to be performed for error tolerance evaluations with ANNs. Since most approximate memories have similar error models, we assume a general error model and use it for the optimization and evaluation of the error tolerance in ANNs. We use a novel hinge loss based on margins in ANNs for error tolerance optimization and compare it with the traditional flip regularization. We focus on Binarized Neural Networks (BNNs), which are one of the most resource-efficient variants of ANNs.

7.2.1 Introduction

Artificial neural networks have been applied successfully in numerous fields, and are being executed on a variety of systems ranging from large computing clusters to small, battery-driven embedded systems. In most cases, state-of-the-art neural network models rely on a large number of parameters to achieve high performance. This leads to an expensive, slow, and energy-consuming *memory bottleneck*. On neural network acceleratorswith SRAM, the energy consumption of the memory makes up the largest fraction of system energy, while advances in memory bandwidth are significantly slower than processing speed. Hence, improving the memory consumption of ANNs and improving the memory sub-systems is imperative to further push the applications of ANNs. One design paradigm to improve the memory sub-system is to use approximate memory in which resource efficiency is achieved by allowing for bit errors during the read and/or write process. Likewise, reducing the memory consumption of ANNs is an established part of deep learning research. Here, arguably, the most extreme form is to use Binarized Neural Networks (BNNs) that only use binary weights $\{0, 1\}$ leading to a potential 32 times memory reduction as high as 32 times that of their floating-point siblings. Interestingly, it has been shown that BNNs can be trained to tolerate bit errors by bit flip injections during training. However, this method has a large overhead and does not scale well with model size and higher bit error rates .

In this contribution, we first summarize the currently available and emerging memories that are possible to be used with neural network inference systems. Here, we focus on approximate memories, which are unreliable due to bit errors and for which countermeasures are necessary. One of the most promising emerging memory components is the FeFET, which has high speed, and low energy consumption, but faces reliability issues. We explain how FeFET can be used as approximate memory for neural networks despite the bit errors caused by temperature and read voltage. Finally, we present results on how bit error tolerance in ANNs is achieved without bit flip injections based on margin-maximization and compare it to the traditional methods for bit error tolerance optimization of ANNs. This contribution was previously published as a conference paper in [113].

7.2.2 Emerging Memories

Recent studies on efficient ANN-based inference systems have explored the use of approximate memory, which has been realized by reducing the memory supply voltage and tuning latency parameters with the goal of lower power consumption and faster access. If these methods are pushed to the limit, high Bit Error Rates (BERs) can occur. Before discussing bit errors and how to deal with them in more detail we will quickly survey volatile memories (SRAM, DRAM) and other emerging non-volatile memories (FeFET, Resistive Random Access Memory (RRAM), Spin Transfer Torque Random Access Memory (STT-RAM) or Magnetoresistive Random Access Memory (MRAM)) here.

SRAM For ANN inference systems using on-chip SRAM, the works in the literature mainly employ scaling of various device parameters. To reduce energy consumption, the SRAM voltage is scaled in [306, 652]. Yang et al. [717] separately tune the weight and activation values of BNNs to achieve fine-grained control over the energy consumption. Sun et al. [652] propose similar techniques for ternary ANNs. A similar approach is employed by Henwood et al. [306], in which layer-wise the best energy-accuracy trade-off for SRAM is chosen.

DRAM For DRAM, the study by Koppula et al. [381] provides an overview of studies related to ANNs that use different DRAM technologies and proposes a framework for evaluating ANN accuracy when using approximate DRAM in various different settings and inference systems. Specifically, the study shows that DRAM parameters can be tuned such that energy and performance are optimized to achieve significant improvements, whereas the ANN accuracy drop stays negligible due to the ANNs' adaptations in retraining. Other studies, e.g. [532, 672], also optimize the refresh rate of DRAM to achieve energy savings.

RRAM Hirtzlin et al. [316] propose computing BNN operations with RRAM that features in-memory processing capabilities. They set the write energy of RRAM low and show that BNNs can tolerate the resulting errors by error tolerance training. This low-energy setting also increases the RRAM cell lifetime since low-energy writes stress the cells less. The work by Yu et al. [727] also uses RRAM to implement on-chip BNNs. They show that under limited bit yield, BNNs can still operate with satisfying accuracy. Sun et al. [651] propose an RRAM synaptic array to deploy BNNs. They investigate the accuracy impact of errors from sense amplifiers that have offsets due to process variation.

MRAM or STT-RAM Another branch in the literature is about ANNs on STT-RAM or MRAM. Hirtzlin et al. [315] propose deploying BNNs on MRAM with a low-energy programming setting that causes relatively low error rates, and no significant accuracy drop, but decreases write energy by a factor of two. Tzoufras et al. [675] also propose operating BNNs on MRAM with reduced voltage with similar results. They test a wide range of error rates and discuss the implications of BNN bit error tolerance on the lifetime, performance, and density of MRAM. Pan et al. [549] take a different approach for energy reduction and investigate the benefits of multi-level cell MRAM for the in-memory acceleration of BNNs. For more general ANN models, Vincent et al. [686] propose tunable STT-RAM to save resources.

FeFET FeFET is considered to be one of the most promising memory technologies. The reason why FeFET store logic '0' and logic '1' lies in the available dipoles inside the FE. The directions of these dipoles can switch if a sufficiently strong electric field is applied. This state is non-volatile because the dipoles retain their direction when the field is turned off. The logic '0' and logic '1' can be read out from the FeFET based on the intensity of the current returned (e.g. high or low), which can be converted into the digital domain with sensing circuits.
The three main advantages of FeFET over other non-volatile memories are as follows:

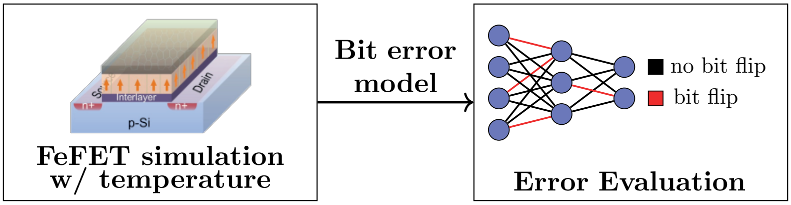


Fig. 7.9: Errors due to temperature, stemming from underlying FeFET devices, are modeled and then injected during the ANN inference [720].

1. FeFET is fully CMOS-compatible, which means that it can be fabricated using current manufacturing processes. This has been demonstrated by GlobalFoundries [668].
2. FeFET-based memories can perform read operations within 1ns latency. This reduces the differences to traditional SRAM technology, while the energy usage of FeFET is significantly lower [668].
3. FeFET memory has the potential to enable extremely low-density memory since the core cell consists merely of a single transistor.

One of the major disadvantages of FeFETs is error susceptibility. Manufacturing variability (i.e. process variation during production) and temperature fluctuations at run-time can cause variations in the FeFET properties. This shrinks available noise margins and may cause errors. To still employ FeFETs despite the errors in, say, on-chip memory for Binarized Neural Networks (BNNs) inference systems, it is necessary to extract the error models for the stored bits. With the error model, the impact of the temperature-induced bit errors on the inference accuracy of BNNs can be evaluated.

In Figure 7.9, the steps for extracting the temperature-dependent error model of FeFET transistors are shown. The entire FeFET device has been implemented and modeled in the Technology CAD (TCAD) framework (Synopsys Sentaurus [656]). The variation in the underlying transistor and the added ferroelectric layer are considered. After incorporating the temperature and variation effects in the calibrated TCAD models, Monte-Carlo simulations for the entire FeFET device are performed. Then the probability of error is extracted for a certain read voltage, i.e. the probability that logic ‘0’ is read as logic ‘1’ and a logic ‘1’ is read as logic ‘0’. Details on device physics modeling and reliability analysis for FeFET under the effects of temperature variability (runtime) and manufacturing (design-time) variability can be found in [280] and [534], respectively.

7.2.3 Binarized Neural Networks

Traditional neural networks use floating-point (e.g., 32 bits) or integer values (e.g., 8 bits) to represent the ANN parameters (i.e., weights, activations, inputs, etc.) . In such a case, the position of the occurred bit error (i.e., the bit flip in the value) does matter. Specifically, in floating-point ANNs, a one-bit error in one weight can cause the prediction of the ANN to become useless (see e.g. [381]). This typically occurs when a bit flip in the exponent of the floating-point representation occurs leading to an error with an unacceptable magnitude. As mentioned before, BNNs are resource-efficient neural networks that are ideally suited for small devices. Additionally, they can be trained to be resilient against bit errors, which makes them ideal candidates for approximate memories. In BNNs each weight (and possibly each activation) is stored in a single bit $\{0, 1\}$. Hence, a bit error in a binary weight or binary input causes a change of the computation result by merely 1, reducing its overall impact. In addition to the reduced impact of bit errors and reduced memory footprint due to smaller weights the execution of BNNs also becomes simpler. Consider, for example, the output of the fully connected l-layer with activation σ and weights W^l

$$f^l(X) = \sigma(W^l X) \tag{7.1}$$

In regular floating-point neural networks, the execution of this layer requires the repeated computation of matrix-vector products $W^l X$ as well as the application of σ. In a BNN this operation becomes

$$2\texttt{popcount}(\texttt{XNOR}(W^l, X)) - B > T \tag{7.2}$$

where popcount counts the number of 1s in the XNOR-result, B is the number of bits in the XNOR operands, and T is a learnable threshold parameter if batch normalization layers are used, whose comparison produces binary values (representing a shifted binarization function) [325, 609].

A common method of training ANNs is to apply stochastic gradient descent (SGD) with mini-batches. Let $\mathcal{D} = \{(x_1, y_1), \dots, (x_I, y_I)\}$ be the training data with $x_i \in \mathcal{X}$ as the inputs, $y_i \in \mathcal{Y}$ as the labels, and $\ell \colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ as the loss function. $W = (W^1, \dots, W^L)$ are the weight tensors of layer $1 \dots L$ and $f_W(x)$ is the output of the ANN. The goal is to find a solution for the optimization problem

$$\arg\min_W \frac{1}{I} \sum_{(x,y)\in\mathcal{D}} \ell(f_W(x), y) \tag{7.3}$$

with a mini-batch SGD strategy that computes gradients using backpropagation.

To train BNNs, Hubara et al. [325] proposes to deterministically binarize the weights and activations during the forward pass. For backpropagation, the floating-point numbers are used for parameter updates. This leads to training times similar regular ANNs but assumes binary values during the forward pass. More formally, let $b \colon \mathbb{R} \to \{-1, +1\}$ be a binarization function with

$$b(x) = \begin{cases} 1 & x > 0 \\ -1 & \text{else} \end{cases} \tag{7.4}$$

and let $B(W)$ denote the element-wise application of b to a tensor W. Now we simply apply B during the forward pass to each weight tensor. During the backward pass, the authors propose using full floating point precision, whereas during the backward-pass they replace the gradient of b with the straight-through estimator. Consider the forward computation $Y = B(X)$. Let $\nabla_Y \ell$ denote the gradient with respect to Y. The straight-through estimator approximates

$$\nabla_X \ell := \nabla_Y \ell, \tag{7.5}$$

essentially pretending that B is the identity function. Algorithm 5 summarizes this approach.

```
Algorithm 5: Binarized forward pass for a network with L layers, each with
weight tensors $W^l$ performing a generic operation $\circ^l$ (e.g. a convolution).
1 for $l \in \{1, \dots, L\}$ do
2     $x \leftarrow B(B(W^l) \circ^l x)$
```

7.2.3.1 Flip Regularization

To make BNNs bit error-tolerant, the state-of-the-art method is bit flip injections in the binarized values during the forward pass, as proposed by Hirtzlin et al. [316]. The idea is simple: To make BNNs robust against bit errors, we simulate the errors already during training time. During each forward pass computation, we generate a random bit-flip mask and apply it to the binary weights.

Let M denote a random bit-flip mask with entries ±1 of the same size as W that we multiply component-wise to the binarized weights. We first consider computing the bit-flip operation as $H = (B(W) \cdot M) \circ X$ where $\circ$ denotes the application of the ANN to the input X. Standard backpropagation on a loss ℓ that is a function of H yields the following gradient of ℓ with respect to $B(W)$

$$\nabla_{B(W)} \ell = M \cdot \nabla_{B(W) \cdot M} \ell \tag{7.6}$$

which for fully connected layers amounts to a gradient update

$$\nabla_{B(W)} \ell = M \cdot (\nabla_H \ell \, X^T). \tag{7.7}$$

We see that an update computed this way accounts for the bit-flips that were performed. We propose instead using a special flip-operator with straight-through gradient approximation. We denote by e_p the bit error function that flips its input with probability p and let E_p denote its component-wise version. During training, we change the forward pass such that it computes

$$X^{l+1} := B(E_p(B(W^l)) \circ X^l). \tag{7.8}$$

We replace the gradient of E_p with a straight-through approximation. This way, in the example above we now have $H = E_p(B(W)) \circ X$ with gradient updates $\nabla_{B(W)}\ell = \nabla_{E_p(B(W))}\ell$ which for fully connected layers yields the update

$$\nabla_{B(W)}\ell = \nabla_H \ell \, X^T \tag{7.9}$$

which is unaware of bit flips and just uses the corrupted outputs H.

The original bit-flip regularization proposed in [316] reports extreme overfitting to the flip probability used during training. As we will see later in the experiments, we do not report such an overfitting. We believe that the approach using straight-through gradient approximation is superior and that the extreme overfitting is attributable to the use of the naive gradient.

7.2.3.2 Margin-Maximization for Bit Error Tolerance Optimization

Bit-flip regularization improves the error tolerance of the network by simulating bit errors during the forward pass. This introduces two objectives to the training: Given a set of labeled input data, train a BNN for high accuracy and for high bit error tolerance. Hence, another approach is to combine high accuracy and high bit error tolerance into a single loss function directly so that both objectives are jointly optimized during training. To do so, we now introduce a margin-based neuron-level bit error tolerance metric for BNNs that is then extended to formulate a bit error tolerance metric for the output layer.

In the following, we use a notation describing the properties of neurons in convolutional layers, but our considerations also apply to neurons in fully connected layers. Let n be the index of one neuron in a ANN, and $x \in \mathcal{X}$ an input to the ANN. The output of a neuron in a convolutional layer is a feature map with height U and width V. Let $h_{x,n,u,v} \in \mathbb{Z}$ be the pre-activation value of neuron n at place $(u, v) \in \{0, \dots, U\} \times \{0, \dots, V\}$, *before* applying the activation function. For BNNs, the pre-activation values of a neuron are computed by a weighted sum of inputs and weights that are ±1. Therefore, one bit flip in one weight changes the pre-activation value by 2.

Theorem 25. *Let $n \in \{0, \dots, N\}$ be the index of one neuron. Furthermore, let q be the number of bit flips induced in the weights of neuron n. The pre-activation of neuron n at place (u, v) after induction of these bit flips is in the interval $[h_{x,n,u,v} - 2q, h_{x,n,u,v} + 2q]$.*

The proof can be found in [113].

A detailed analysis of the error tolerance for hidden-layer neurons has been conducted in [114], but the use of Theorem 25 for optimizing bit error tolerance on the neuron-level has been reported to be unsuccessful. We hypothesize that bit flips of neuron outputs can only affect the BNN prediction if the effect of bit flips reaches the output layer and leads to a change in the predicted class. Therefore, we now shift our focus on applying the notion of margin to the output layer, i.e., to neurons with index in N_O.

Each neuron in the output layer has only one output value $h_{x,n,1,1}$ which is one entry in the vector of predictions $\hat{y}$. No activation function is applied to the output value of these neurons. There are as many values in $\hat{y}$ as there are neurons in the last layer. The index of the entry with the maximum value in $\hat{y}$ determines the class prediction, where we assume that ties are broken arbitrarily.

If bit errors modify the output values in the output layer such that another neuron provides the highest output value, then the class prediction changes. Let $h_{x,n',1,1}$ and $h_{x,n'',1,1}$ with $n^{'}, n^{''} \in N_O$ be the highest and the second-highest output value of neurons in the output layer. The following corollary shows that the margin

$$m := h_{x,n',1,1} - h_{x,n'',1,1} \tag{7.10}$$

serves as a bit error tolerance metric for the output layer.

Corollary 26. *If* $m > 0$, *then the output layer of the BNN tolerates* $\max(0, \lfloor \frac{m}{2} \rfloor - 1)$ *bit flips.*

The proof can be found in [113].

We now focus on constructing a loss function based on Corollary 26 and the hinge loss known from Support Vector Machines (SVMs). The hinge loss [602] for maximum margin classification is defined as

$$\ell(y, f) = \max(0, 1 - y \cdot f), \tag{7.11}$$

with the ground truth prediction $y = \pm 1$ and the prediction $f \in \mathbb{R}$. This loss becomes small if the predictions have the same sign as the predicted class and are close to 1 in magnitude. For predicted values larger than 1, the loss becomes 0. The “1” in the loss forces the classifier to maximize the margin between two class predictions.

For BER tolerance of the last layer, the margin m as introduced in Equation 7.10 needs to be large so that the maximum number of bit flips the output layer can tolerate is high. The margin can be directly computed by subtracting the second-highest entry $\hat{y}_{c''}$ of the output vector $\hat{y}$ from the highest entry $\hat{y}_{c'}$, i.e., $m = \hat{y}_{c'} - \hat{y}_{c''}$. However, optimizing with respect to m without considering the other entries $\hat{y}_c$ of $\hat{y}$ may not exhaust the full potential of the margin between $\hat{y}_{c'}$ and the output of the other classes $\hat{y}_c$. The larger the margin between $\hat{y}_{c'}$ and $\hat{y}_c$ of other classes c, i.e. $m_c = \hat{y}_{c'} - \hat{y}_c$, the more bit errors can be tolerated in the neuron that calculates $\hat{y}_c$ without a change in the prediction. To put it concisely, for a bit error tolerant output layer, $\hat{y}_{c'}$ needs to be as large as possible, while the other $\hat{y}_c$ need to be as small as possible.

In the case of BNNs for multi-class problems, however, the version of the hinge loss in Equation 7.11 cannot be directly used. To extend the hinge loss to multiple classes, we define y_{enc} as a one-hot vector with elements in $\{-1, 1\}$, which has a +1 at the index with the ground truth, else -1. y_{enc} has the same number of elements as $\hat{y}$. Then the element-wise product $y_{enc} \cdot \hat{y}$ is computed. In this product, in case of correct predictions, positive predictions in the correct class will stay positive, and negative predictions that

Tab. 7.5: Datasets used for experiments.

Name	# Train	# Test	# Dim	# classes
FashionMNIST	60000	10000	(1,28,28)	10
CIFAR10	50000	10000	(3,32,32)	10

Tab. 7.6: Parameters used for experiments.

Parameter	Range
Fashion FCNN	In → FC 2048 → FC 2048 → 10
Fashion CNN	In → C64 → MP 2 → C64 → MP 2 → FC2048 → 10
CIFAR10 CNN	In → C128 → C128 → MP 2 → C256 → C256 → MP 2 → C256 → C256 → MP 2 → FC 2048 → 10

should be as negative as possible become positive. In case of wrong predictions, i.e. high negative values for the correct class and high positive values for the wrong class, the values become negative. For a high penalty in the wrong case and a small penalty for the correct case, we subtract the product $y_{enc} \cdot \hat{y}$ from a parameter b, and get $(b - y_{enc} \cdot \hat{y})$. Since we do not demand higher prediction values than b, we set negative values to zero with the max function, and denote the Modified Hinge Loss (MHL):

$$\ell_{MHL}(\hat{y}, y_{enc}) = max\{0, (b - y_{enc} \cdot \hat{y})\}. \tag{7.12}$$

7.2.4 Experiments

We evaluate fully connected binarized neural networks (FCBNNs) and convolutional binarized neural networks (CBNNs) in the configurations shown in Table 7.6 for the datasets FashionMNIST and CIFAR10 (see Table 7.5). In all experiments, we run the Adam optimizer for 100 epochs for FashionMNIST and 250 epochs for CIFAR10. We use a batch size of 128 and an initial learning rate of 10^{-3}. To stabilize training, we exponentially decrease the learning rate every 25 epochs by 50 %. In the following, we compare the margin-based methods (MHL) to Flip Regularization (FR). FR uses the Cross-Entropy Loss (CEL) by default. We first compare MHL without FR to FR. In a second step, we compare MHL without FR to MHL in combination with FR.

7.2.4.1 MHL Only vs. FR

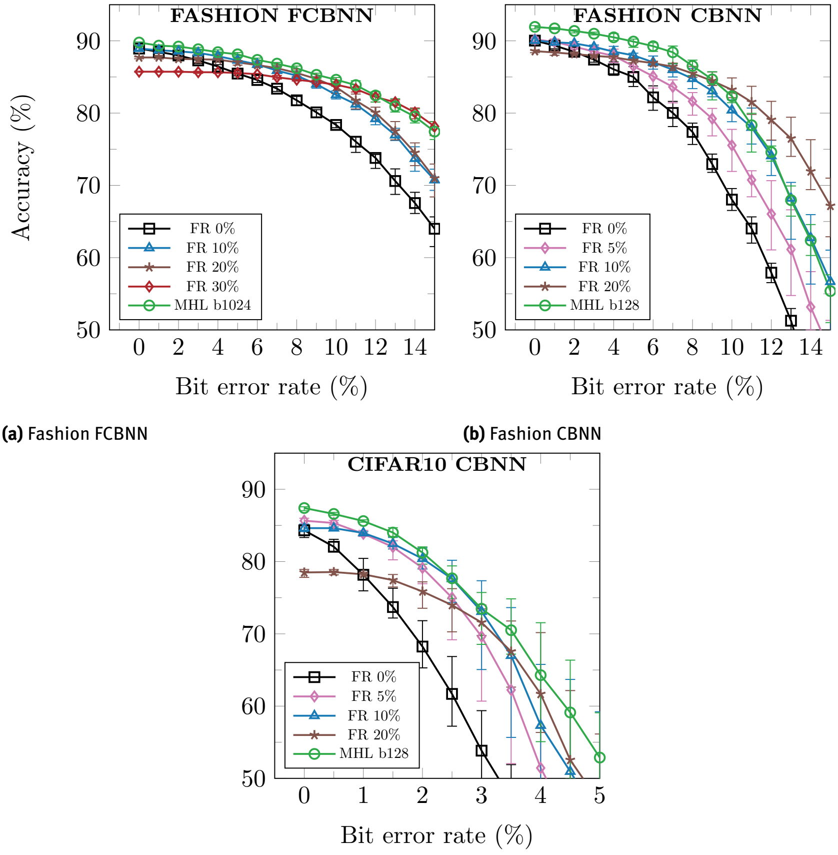
Figure 7.10 presents the experimental results of different BNNs with respect to the accuracy over BER (from 0 % to up to 15 % in Figure 7.10(a) and (b), and from 0 % to up to 5 % in Figure(c)). For each dataset, five BNNs were conducted using MHL without any FR and FR with different BERs for bit-flip injections. Moreover, for all BNNs trained with MHL, we employed a parameter search for b, testing powers of two, up to two times the maximum value the neurons in the output layer can compute (maximum output value of a neuron in the output layer is the number of neurons in the layer before the output layer). Among these configurations of b, the best one was chosen. We observe that BNNs trained with the MHL without FR have better accuracy over BER than the BNNs trained with FR, i.e., in Figure(a) and (b) up to 10 %, and in Figure(c) up to 5 %. The BNNs trained with FR suffer from a significant accuracy drop for lower BERs, when the BER during training is high, e.g., CEL 20 % and/or CEL 30 % at low BER. The BNNs trained with MHL, however, do not suffer from this accuracy drop. Although the BNNs trained with FR 20 % and bit-flip injections have better accuracy for Fashion CBNN in Figure 7.10(b) when the error rate is higher than 10 %, the accuracy of the BNNs drops by a significant amount, which may be unacceptable. Below, we thus present further investigations.

7.2.4.2 MHL Combined With FR

We evaluate BNNs trained with the MHL and FR under different BERs. In addition, the BNNs trained with the MHL without FR (i.e., those BNNs generated using the MHL in Figure 7.11 under 0 % BER) are included here as the baseline in this subsection. For all configurations, we employed the same parameter search for b as in the previous section. Figure 7.11 presents the experimental results of different BNNs with respect to the accuracy over BER (from 0 % to up to 30 % in Figure 7.11(a) and (b) and from 0 % to up to 6 % in (c)). In all experiments, we observe that the accuracy over the BER of the BNNs trained under MHL and FR is significantly higher than that of the baseline trained by only MHL. For example, for Fashion in Figure 7.11, the BER at which the accuracy degrades significantly is extended from 5 % (baseline, green curve) to 20 % and 15 %, respectively, with a small trade-off in the accuracy at 0 % BER. If more accuracy at low error bit rates is traded, the BER at which accuracy degrades steeply can be shifted even further. For CIFAR10 in Figure 7.11, this breaking point can also be increased. However, more accuracy has to be traded compared with previous cases. If b is higher than the ones shown, the accuracy for lower BERs suffers similarly to how it would using CEL with high BERs. If b is lower, there will be no significant change compared with CEL with 0 % BER. We only show the results with the best b.

7.2.5 Conclusion

Deep learning is notoriously memory hungry and hence new memory sub-systems must be developed to push the application of ANNs to small devices. Likewise, new ANN architectures can help to reduce memory consumption and offer a more resource-friendly execution of deep networks. Non-volatile memories such as Ferroelectric FET (FeFET) are a promising technology for new memory sub-systems. FeFET enables faster and more energy-efficient read/write operations but it introduces bit errors into the execution. While standard software solutions can monitor and correct bit errors, they negate the advantages of non-volatile memories by introducing further processing overhead. Neural networks that are resilient to random bit errors by design, on the other hand, can retain the advantages of non-volatile memories leading to potentially faster and more energy-efficient solutions. BNNs are a novel class of small, resource-efficient neural nets that are ideally suited for such a setting. In BNNs each weight consists of weights $\{0, 1\}$ so that they require 32 times less memory than their floating-point counterpart while being more resilient to random bit flips. In this contribution, we provided an in-depth discussion of the bit errors in BNNs and derived a novel max-margin optimization from it. Our approach offers a better accuracy across most error rates while preventing the overfitting of the BNN to a specific error rate. Hence, our approach allows the deployment of BNNs on a variety of different devices with unknown and varying error rates.

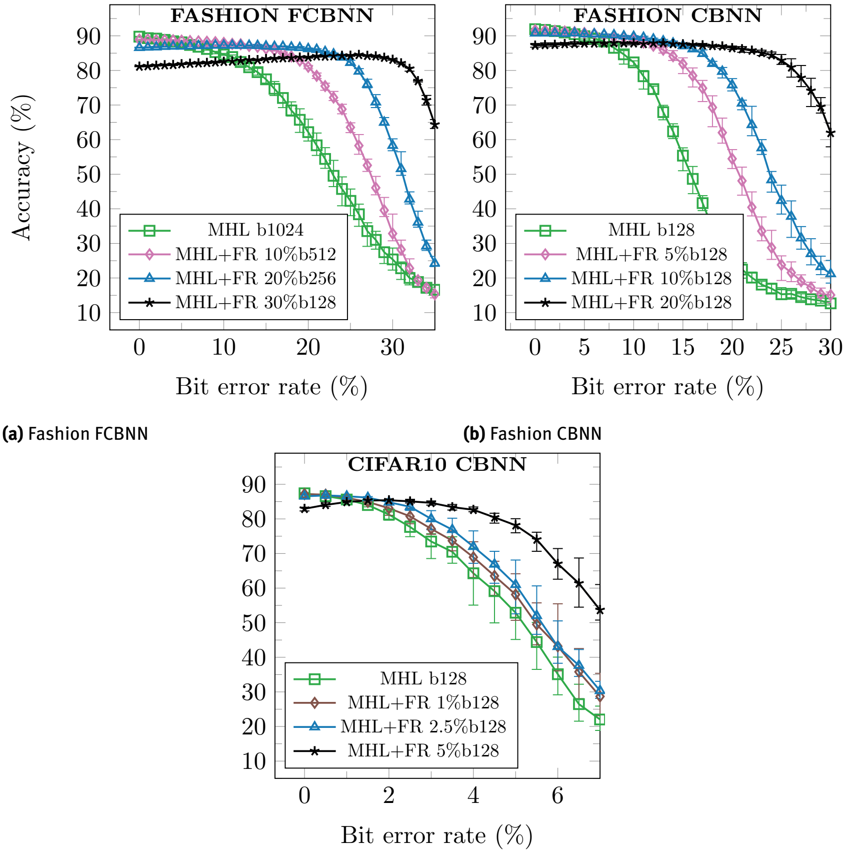


(a) Fashion FCBNN

(b) Fashion CBNN

(c) CIFAR10 CBNN

Fig. 7.10: Accuracy over bit error rate for BNNs trained with FR under a given bit flip injection rate (specified in the legend, 0 %, 5 %, 10 %, etc.) and BNNs trained with MHL without FR for a specified *b* in Equation 7.12.



(a) Fashion FCBNN

(b) Fashion CBNN

(c) CIFAR10 CBNN

Fig. 7.11: Accuracy over bit error rate for BNNs trained with MHL and FR (denoted as FR 0 %, 1 %, etc). The number after the *b* is the value to which the parameter *b* in the MHL is set during training (see Equation (7.12)).

7.3 Cache-Friendly Execution of Tree Ensembles

Sebastian Buschjäger
Kuan-Hsun Chen

Abstract: Ensembles of decision trees are among the most used classifiers in machine learning and regularly achieve state-of-the-art performance in many real-world applications, e.g., in the classification of celestial objects in astrophysics, pedestrian detection, etc. Machine learning practitioners are often concerned with model training, re-training different models again and again to achieve the best performance. Nevertheless, once a learned model is trained and validated, the executing cost of its continuous application might become the major concern.

Applying decision trees for inferences is very efficient in run-time, but it requires many memory accesses to retrieve nodes. For example, it is common to train several thousand trees, e.g., each with depth 15 leading to $2^{15} = 32\,768$ nodes per tree. This leads to millions of decision nodes that must be stored in memory and processed. Cache memory is commonly adopted to hide the long latency between the main memory and the processor. However, an improper memory layout might bring up additional cache misses, leading to performance degradation. Thus, designing a suitable memory layout of tree ensembles is of key importance to achieve efficient inference over tree ensembles.

In this contribution, we discuss the deployment of tree ensembles on different hardware architectures. Given a pre-trained decision tree ensemble, we first present different realization techniques commonly used in the literature. Afterwards, we study different layout strategies to optimize the node placement in the memory, focusing on the caches available on different hardware architectures. Finally, we present the evaluation results over different configurations and combine all approaches into a single framework that automatically generates the optimized realization for a target hardware architecture.

7.3.1 Introduction

Efficient learning has always been the focus of research, but the demand for the *efficient application* of learned models has emerged only recently. Consider, for example, self-driving cars. Current prototypes use machine learning (ML) for image recognition and fundamental steering.[1] Thus, the ML model must not only be applied continuously, but it also must react on time. As a second example, consider search engines that utilize ML

[1] ,https://towardsdatascience.com/teslas-deep-learning-at-scale-7eed85b235d3.

models such as Gradient Boosted Trees[2] to rank search results. These engines routinely process roughly 12 billion search queries a month worldwide.[3] The 4 480 287 queries per second they process demand fast model application.

While deep learning is excellent for unstructured image data, tree ensembles are often referred to as one of the best black-box methods available for structured data. They offer high accuracy with only a few parameters to tune [120, 223] and frequently place among the top methods in data science competitions.[4] For real-time application, tree ensembles have become important in many domains, e.g., the real-time classification of celestial objects in astrophysics [115], real-time pedestrian detection [466], real-time 3D face analysis [211]), the real-time classification of noise signals [608], nano-particle sensors [439].

However, these trees are usually stored in the main memory and processed directly out of the memory. The runtime of such a memory-intensive application is mainly determined by the use of the various caches of the CPU. Surprisingly, as the line between realizational details and algorithmic contributions becomes blurry on modern computing systems, caching behavior determines the performance of implemented algorithms even more than algorithmic differences [615]. For tree ensembles, we can foresee that an analytical approach to an efficient layout of the memory is desirable. Given a pre-trained tree ensemble, we present several cache-aware approaches to optimize the memory layout (so-called tree-framing), while preserving the original ensembles' accuracy. The proposed approaches are wrapped in a code generator that automatically adapts to underlying architectures to produce optimized code segments. Overall, we present the following contributions:

Cache-aware tree-framing approaches We analyze the source of cache misses on two common tree realizations, i.e., *native* and *if-else* trees, and discuss several approaches at the application level to optimize the memory layout by artificially creating instruction/data locality .

Architecture-aware code generator We present a code generator that exploits the analytical insights for generating optimized realizations of a given tree ensemble. The code generator is publicly available at https://github.com/sbuschjaeger/fastinference.

Empirical evaluation We perform 1800 experiments across three different computer architectures and show that our approaches offer a speed-up factor at 2 – 4 on average without changing the prediction accuracy of the given trained model.

This contribution was previously published as a conference paper in [108] and was later expanded in a dissertation in [107].

[2] https://www.seroundtable.com/bing-core-ranking-algorithm-machine-learning-27040.html.

[3] Numbers are for 2019, see https://www.statista.com/topics/4294/bing/.

[4] https://www.kdnuggets.com/2016/01/anthony-goldbloom-secret-winning-kaggle-competitions.html.

7.3.2 Related Work

Tree ensembles are some of the most used machine learning algorithms and, as such, have been studied extensively in the literature. In the context of model application and fast inference, there are two principled approaches. The first set of methods changes the training procedure for Decision tree (DT) ensembles to produce more resource-friendly models. This can be beneficial to achieve the highest accuracy given the computational resources provided, but often result in longer training times and more evolved training procedures. Common examples for this approach are pre- and post-pruning rules for trees (see, e.g. [43]) or the pruning of entire ensemble members [347, 449, 589, 739].

The second set of methods studies the realization of a given DT ensemble and its execution. This approach uses the ensemble as-is and, as such, does not affect the training. We will focus on this methodology in this contribution. Note that both methods can also be combined. For example, Van Essen et al. present in [679] a comprehensive study of different architectures for implementing Random Forests (RFs) on CPUs, FPGAs, and GPUs. Based on the CATE algorithm [586], the authors train an RF with DTs constrained by a fixed height. By fixing the tree-depth, the authors show a practical pipelining approach for executing DTs on CPUs, FPGAs, and GPUs.

Asadi et al. introduce different realization schemes of tree-based models in the context of learning-to-rank tasks [26]. They introduce two different realization schemes, which will be discussed in more detail later: the first one uses a while-loop to iterate over individual nodes of the tree, whereas the second approach decomposes each tree into its if-else structure. For the first realization, the authors also consider a continuous data layout (i.e., an array of *structs*) to increase data locality but do not directly optimize each realization. Also note that the authors mainly consider gradient-boosted trees. There, the individual trees are usually “weak” in a sense, that they are comparably small, as opposed to larger trees in RFs.

Also in the context of ranking models, Lucchese et al. present the QuickScorer algorithm for gradient boosted trees [162, 450]. In this approach, the authors discard the tree structure and decompose each tree into its comparisons. Then, they sort the comparisons of the entire ensemble according to the feature value and perform them one after another instead of traversing trees in a classical sense. To do so, they introduce a 2^{Δ}-dimensional bit vector, where Δ is the height of a tree in which the most significant bit (MSB) signifies the prediction leaf node of that tree. This way, the algorithm can reuse comparisons across all ensemble members while minimizing cache misses. In [452] the authors further enhance their method by adding vectorization over multiple examples for more efficient batch-processing. To mitigate the limitations of a fixed height, Ye et al. propose in [721] using an encoding scheme called epitome that decodes the bitvectors on the fly while preserving vectorization. We note that, while these methods usually offer a tremendous speed-up, they execute *all* possible comparisons in the entire ensemble in the worst case. Thus, they are especially effective for large ensembles of smaller trees commonly produced by gradient boosting algorithms.

Kim et al. present in [373] a realization for binary search trees using vectorization units on Intel CPUs and compare their realization against a GPU realization. The authors provide insight on how to tailor the realization to Intel CPUs by taking into account register sizes, cache sizes, and page sizes. Their work is specialized for Intel CPUs, and thus, it is not directly applicable for different CPU architectures. Lucchese and colleagues have already noticed, that many nodes are seldom visited [450]. Buschjäger and Morik formalize this observation in [110] by estimating the probabilities of specific paths during tree traversal. Based on this probabilistic view of model execution or inference, the authors consider different realization schemes for tree traversal and theoretically analyze their runtime. Note, however, that this model of computation remains at the software level and does not include the memory layout. Buschjäger et al. enhance this model in [108] by including the memory layout in their model. They show how to minimize cache misses and how different realizations affect the instruction and data cache differently for executing ensembles of large trees commonly found in RFs. We will now discuss this paper in more detail.

7.3.3 A Probabilistic View of DT Execution

We consider supervised learning problems, in which we infer a model $f : \mathbb{R}^d \to \mathcal{Y}$ from labeled training data $\{(\mathbf{x}_i, y_i) | i = 1, \dots, N\}$ to predict the value $f(\mathbf{x})$ of new, unseen observations. For $\mathcal{Y} = \mathbb{R}$, we have a regression problem, for $\mathcal{Y} = \{0, 1, \dots\}$, we have a classification problem.

Tree ensembles train a set of individual trees and combine their predictions to establish a joint model. In the classical Random Forest (RF) approach by Breiman [72], K DTs are trained using different samples of input features. Other RFs variations have been explored, such as those that train trees on samples of data (bagging) [71] or those that randomly generate splits for training [250]. Boosting [610] also frequently uses decision trees as their weak base learners, but trains them sequentially to correct each other.

A decision tree is a simple, tree-structured model that consists of inner nodes with two children and leaf nodes. Each inner node compares the feature value x_f of the current sample $\mathbf{x}$ against a threshold t where f and t are computed during tree training. Depending on the outcome of this comparison, either the left or the right child of this node is used until a leaf node is found. The leaf node stores a constant prediction value (e.g. the estimated class probabilities that fall into the leaf) which is then returned.

Our goal is to analyze the probability of performing a certain comparison while traversing a DT. Based on this analysis, we can decide for each tree, which realization and which data layout is best. Our notation is the following: each node receives a unique identifier (e.g., in breath-first order) i. We denote the left child of i with $l(i)$ and the right child with $r(i)$. Note that every observation takes exactly one path $\pi(\mathbf{x})$ from the root node to one leaf. To lighten the notation, we drop the argument $\mathbf{x}$, if we are not

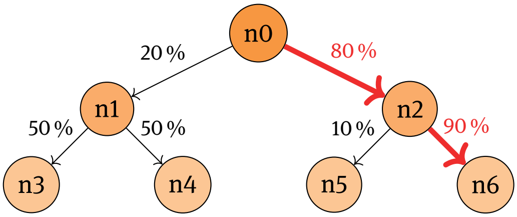


Fig. 7.12: Decision tree with probabilities of the path.

interested in the path of a specific observation. As established in [110], we model each comparison at node i as a Bernoulli experiment in which we take the path towards the left child with probability $p(i \to l(i))$ and towards the right child with $p(i \to r(i))$. It holds that $p(i \to l(i)) = 1 - p(i \to r(i))$. An example can be found in Figure 7.12.

The probabilities $p(i \to l(i))$ and $p(i \to r(i))$ can be estimated with the training data by counting the number of samples at each node i taking the left and right path. Assume a path of length L with $\pi = (i_1, i_2, \dots, i_L)$, where i_{j+1} is either the left or the right child of the j^{th} node on the path. Following this path consists of a series of Bernoulli experiments, each with probability $p(i_j \to i_{j+1})$. Let $\mathcal{P}$ denote the set of all paths in the tree. The probability of taking path $\pi \in \mathcal{P}$ is given by

$$p(\pi) = p(i_0 \to i_1) \cdot \ldots \cdot p(i_{L-1} \to i_L) = \prod_{j=0}^{L} p(i_j \to i_{j+1}) \tag{7.13}$$

Again, let i be a node, there is exactly one path $\pi = (0, \dots, i)$ ending in node i. We call the probability of the path leading to node i the probability of that node, that is $p(i) = p((0, \dots, i))$. Let $\mathcal{T}$ be the set of all nodes in the tree. We define the probability for every subset of nodes $T \subseteq \mathcal{T}$ as:

$$p(T) = \sum_{i \in T} p(i) \tag{7.14}$$

7.3.4 Memory Locality and Tree Realization

As mentioned, tree ensembles can consist of millions of nodes that must be stored and managed in the main memory. Hence, the memory layout of tree ensembles is one of the most crucial aspects of efficient tree traversal. In order to mitigate the performance gap between the main memory and the processor, smaller and faster memory subsystems are often introduced in modern computer architectures to hide the long read/write latency, in the forms of cache and scratchpad memories. Here we focus on the cache memory, which is commonly equipped in modern computing systems.

The cache memory basically acts as a buffer between the main memory and the CPU and stores the data and instructions that the CPU uses more frequently. This way, frequently accessed parts of the memory can be loaded from the smaller, but much faster cache memory to reduce the latency of memory accesses. However, any misuse of cache memory might be even worse than no cache in the memory hierarchy because one cache miss triggers two loading instructions, one from the main memory to the cache and one from the cache to the processor. There are three types of cache misses [183]:

Compulsory misses are due to the first access to a memory block that the cache did not yet have a chance to buffer.

Capacity misses occur when some memory blocks are discarded from the cache memory due to the limited capacity, i.e., the program is working on more data than the cache capacity.

Conflict misses occur in set-associative or direct-mapped caches when several blocks are mapped to the same cache set.

The basic assumption of a cache is that of *memory localities*:

Temporal locality Recent data will be accessed in the near future, say, in small program loops.

Spatial locality Data at addresses close to the addresses of recently accessed data will be accessed in the near future, say, in sequential accesses to elements of an array.

These are the general assumptions for cache design, but please note that knowing how the caches exactly behave is difficult or even impossible. Caches are manufactured as parts of the closed IP of CPU manufacturers and hence the exact design of caches is unknown. Additionally, due to the fact that there are often competing processes running on a single CPU it is difficult to predict the cache behavior deterministically. In this contribution we suppose that the design of cache behaviors cannot be changed. The question we address is this: **How to realize a cache-friendly execution while preserving the functional behaviors of a given DT?**

First, we analyze the memory usage of two common realizations of DT, i.e., native Tree and If-else Tree that do not exploit the memory locality during the execution over the structure of DT. Then we discuss how we can make these two realizations more cache-friendly.

Native Tree The native tree implementation uses a while-loop to iterate over the individual tree nodes that are stored within a continuous data structure, say, in a one-dimensional array. An example code can be found in Listing 7.1. Although the usage of the simple loop with a few lines of codes preserves the temporal locality, the accesses over the nodes of a DT do not have spatial locality. The nodes are often allocated sequentially according to the indexes, whereas such indexes might not take the execution of the DT into consideration, e.g., the nodes on one path might not be allocated sequentially. In addition, if the distance between each node of the path is greater than the number of nodes that can be hosted into a cache set, some nodes will be loaded into caches but not used at all, leading to much *capacity and conflict cache misses*.

Listing 7.1: Example for native tree structure in C++.

```
struct Node {
  bool isLeaf;
  unsigned int prediction; // Predicted label
  unsigned char feature; // Targeted feature
  float split; // Threshold
  unsigned short leftChild, rightChild;
};
Node tree[] = {{0,0,0,8191,1,2},{0,0,1,2048,3,4},..]}
bool predict(short const x[3]){
  unsigned int i = 0;
  While(!tree[i].isLeaf) {
    if (x[tree[i].feature] <= tree[i].split) {
      i = tree[i].leftChild;
    } else {
      i = tree[i].rightChild;
    }
  }
  return tree[i].prediction;
}
```

If-Else Tree An alternative is the if-else tree, which statically encodes the split values of nodes in the instructions. This realization essentially avoids the indirect memory accesses required by the native tree and usually improves the runtime efficiency significantly. An example code can be found in Listing 7.2. However, the advantage of the temporal locality in the instruction cache might be completely abandoned. Since DTs are naturally composed of many branches, some encoded instructions might be prefetched into the instruction cache but not used. Additionally, if the size of the instructions for one DT is greater than the size of the instruction cache, the cached instructions may be evicted out by loading other instructions due to the *capacity and conflict cache misses*.

Listing 7.2: Example for if-else trees in C++.

```
bool predict(short const x[3]){
  if(x[0] <= 8191){
    if(x[1] <= 2048){
      return true;
    } else {
      return false;
    }
  } else {
    if(x[2] <= 512){
      return true;
    } else {
      return false;
    }
  }
}
```

7.3.5 Memory Layout Optimization

In the following, we analyze the caching behaviors of the two different realizations and present our tree-framing algorithms to optimize the memory layout at the application layer accordingly.

Native Tree As shown in Listing 7.1, a DT can be realized by allocating the tree nodes sequentially in an 1-D array, and a simple loop can access them according to the comparison between the feature and the split value. We first observe that, in fact, half of the nodes in a tree are leaf nodes storing a prediction value. This naive realization, however, assumes the same data type for each node, incurring unnecessary memory usage. Second, the access pattern of a DT forms a unique path from the root to a leaf for each input data, but the nodes are typically sequentially allocated in the array according to Breadth-First Search (BFS).[5] The distance between each accessed node becomes larger when the accessed nodes are placed deeper in the DT. The proposed optimization is twofold: 1) reducing compulsory cache misses by encoding the predicted label into the field of children, and 2) reducing capacity and conflict cache misses by allocating as many nodes as possible from the same path into the same cache set.

When a node is loaded, the following nodes in the array are prefetched into the data cache sequentially. If the size of memory for each node can be reduced, more nodes can be loaded into the cache at once so that overall compulsory cache misses can be reduced. To reduce memory consumption we can completely remove the `isLeaf`

[5] Please note that the problem is not limited to BFS. Here we point out the demand of considering the access pattern when allocating nodes to memory.

and `prediction` fields, and store the predicted labels of the children directly in the respective fields by encoding the node type with an indicator field, i.e., removing one Boolean variable and two unsigned shorts by adding one unsigned short.

As mentioned earlier, the sequence of stored nodes is not consistent with the access pattern over the execution of the tree, so the benefit of caching cannot be utilized properly. A sensible solution is to leverage the probabilistic view on DT execution to identify nodes that were likely executed consecutively and place them in memory accordingly. Let τ be the cache set size and `A` be the array in which we place all nodes of $\mathcal{T}$. Furthermore, let $\mathcal{C}$ be the candidate list of nodes in $\mathcal{T}$ that have not been placed in `A` yet and let $\mathcal{S}$ denote the nodes that should be placed in the same cache set. For each node, we greedily choose a child that has the highest probability on the current path and place it in $\mathcal{S}$. Once $\mathcal{S}$ contains $\tau - 1$ elements (and hence is full), we append all nodes from $\mathcal{S}$ to the array `A`, clean up $\mathcal{S}$, and repeat the above procedure for the next set. The details are summarized in Algorithm 6.

Algorithm 6: Optimized path layout

```
  Data: Tree-nodes $\mathcal{T}$, maximum nodes per set $\tau$
  Result: A data array A with the path-oriented layout
1 A = [ ]
2 $\mathcal{C} \leftarrow \{0\}$
3 while $\mathcal{C} \neq \emptyset$ do
4     $i \leftarrow \arg\max_{j \in \mathcal{C}}\{p(\pi(j))\}$
5     $\mathcal{C} \leftarrow \mathcal{C} \setminus \{i\}$
6     $\mathcal{S} \leftarrow \{i\}$
7     while $|\mathcal{S}| \neq \tau$ do
8         if $i$ is leaf-node and $\mathcal{C} \neq \emptyset$ then
9             $i \leftarrow \arg\max_{j \in \mathcal{C}}\{p(\pi(j))\}$
10            $\mathcal{C} \leftarrow \mathcal{C} \setminus \{i\}$
11        else
12            $\mathcal{C} \leftarrow \mathcal{C} \cup \arg\min\{p(i \to l(i)), p(i \to r(i))\}$
13            $i \leftarrow \arg\max\{p(i \to l(i)), p(i \to r(i))\}$
14            if $|S| = \tau - 1$ then
15                $\mathcal{C} \leftarrow \mathcal{C} \cup \{l(i), r(i)\}$
16        $\mathcal{S} \leftarrow \mathcal{S} \cup \{i\}$
17    A.append(S)
```

Please note that adding a new node to $\mathcal{S}$ (Line 7) has two possible actions for the encoding procedure:

- The current node is a split node. The algorithm picks the next node based on the children's probabilities and puts a more probable child in $\mathcal{S}$ and the other children into the candidate list $\mathcal{C}$.
- The checked node is a leaf node, i.e., the end of the path. The algorithm picks a sub-root with the highest probability from the candidate list $\mathcal{C}$ as long as it is not empty. The traversal starts again until $\mathcal{S}$ is full.

If the current $\mathcal{S}$ is full, but a path is not finished yet (Line 14), two children of the current node are returned to the candidate list $\mathcal{C}$ (Line 16). A sub-root that has the highest probability is picked from $\mathcal{C}$ for the next new set $\mathcal{S}$. The algorithm outputs the optimized memory layout over nodes in which path-oriented sets are sequentially allocated to the array.

If-Else Tree As shown in Listing 7.2, a DT can be realized by unrolling the comparisons of a DT into conditional statements with the if-else blocks. This version avoids the indirect memory accesses and does not consider the execution pattern of a DT. The proposed optimization is also twofold: 1) reducing compulsory cache misses by reducing the branch executions, and 2) reducing capacity and conflict cache misses by grouping those nodes used most of the time, e.g., the root node.

When a compulsory cache miss takes place, several consecutive instructions are fetched into the instruction cache, even though some of them might not be executed due to branches. An analysis of the corresponding assembly code reveals that only the branches for else statements are generated in general. In order to increase the chance of using prefetched instructions, the possibility of branch executions should be reduced. Towards this, we propose traversing all paths in the DT and swapping the children of every node i when $p(i \to l(i)) \geq p(i \to r(i))$.

Furthermore, unlike the native tree, the positions of unrolled nodes cannot be freely allocated. The size of nodes from a DT is likely greater than the size of the instruction cache. Because of the capacity and conflict cache misses the cached instructions may be evicted by fetching other instructions. We propose partitioning nodes into different computation kernel functions, and leveraging `goto` statements to break the tie between if-else blocks so that we can put probable nodes together.

Let $\mathcal{K}$ denote the kernel function and let $s(i)$ be a mapping function returning the instruction size of node i. We formulate the following optimization problem:

$$\mathcal{K} = \arg\max \left\{ p(T) \middle| T \subseteq \mathcal{T} \text{s.t.} \sum_{i \in T} s(i) \leq \beta \right\}, \tag{7.15}$$

where β is a given budget related to the size of the instruction cache on the targeted architecture. Given $\mathcal{K}$, these nodes likely remain in the cache once they are fetched, whereas the remaining nodes $\mathcal{L} = \mathcal{P} \setminus \mathcal{K}$ may be evicted more often. In order to avoid iterating over all possible subsets of $\mathcal{T}$, which might be computationally inefficient, we propose a greedy algorithm to partition nodes in a path-wise manner, summarized in

Algorithm 7. At first, the algorithm swaps the children according to their probabilities,

Algorithm 7: Optimized *if-else* tree

```
Data: Tree $\mathcal{T}$, Paths $\mathcal{P} = \{\pi_1, \dots, \pi_M\}$
Result: Kernel $\mathcal{K}$, Label $\mathcal{L}$
swapChildren($\mathcal{T}$)
$\mathcal{P} \leftarrow$ sortByProbabilities($\mathcal{P}$)
$b \leftarrow 0$
for $\pi \in \mathcal{P}$ do
    for $i \in \pi$ do
        if $b + s(i) > \mathcal{B}$ then
            Add $i$ to $\mathcal{L}$
        else
            Add $i$ to $\mathcal{K}$
            $b \leftarrow b + s(i)$
```

and sorts all paths in the tree by their probabilities. Afterwards, the approach greedily appends nodes one by one into $\mathcal{K}$ until the accumulated size of the added nodes b is greater than the given budget $\mathcal{B}$. The rest of the nodes are all added to $\mathcal{L}$. Once the nodes are grouped into $\mathcal{K}$ and $\mathcal{L}$, we can use `goto` statements to break the sequential generation of if-else blocks. First, we generate if-else blocks for all nodes in $\mathcal{K}$. Once the left/right child of one of those nodes is in $\mathcal{L}$, a `goto` statement is generated at the same position to replace the original if-else statement. Then, the corresponding if-else statements of this node and its children are all generated into a label block at the end, which is branched from the goto statement. Listing 7.3 shows an example based on Listing 7.2 by applying Algorithm 7.

Listing 7.3: If-else structure in C++ with goto statements.

```
bool predict(short const x[3]){
  if(x[0] > 8191){
    if(x[2] <= 512){
      return true;
    } else {
      return false;
    }
  } else {
    goto Label0;
  }
Label0:
  {
    if(x[1] <= 2048){
      return true;
    } else {
      return false;
    }
  }
}
```

The remaining question is how to estimate the instruction size $s(\cdot)$ of each node. In general, the instruction set size differs for two different types of nodes:

Split nodes require three types of instructions. First, the values of the target feature and the corresponding threshold are loaded into registers. Second, the values inside the registers are compared against constant values. Last, a jump out of the current block is performed based on the comparison.

Leaf nodes need two types of instructions. First, the return value of the prediction is stored in a register, and second, a jump back to the caller of the if-else tree is performed.

Therefore, we can estimate $s(\cdot)$ by counting the number of generated instructions for a tree node. Table 7.7 summarizes the expected size of instructions for ARM, X86 (Intel), and PPC in an isolated example.[6] Please note that in a real application, the actual number of instructions depends on the adopted compilation tool-chains and the actual realization. An advanced automation can be further explored by exploiting compiler features, e.g., annotations on the source code, to enforce the executing patterns. By doing so, the number of generated instructions can be firmed in the proposal algorithm as for example done in ongoing research such as [132].

[6] We adopted GNU C++ (g++) compiler version 4.8.3 for ARM, version 4.9.2 for PPC, and version 5.4.0 for Intel with `-O0` option.

Tab. 7.7: The expected size of instructions for a split node and a leaf node in a decision tree on ARM (Raspberry PI 2), PPC (NXP T4240 processors) and Intel (Intel Core i7-6700) processors.

	ARM [Bytes]		PPC [Bytes]		Intel [Bytes]	
Type	Int	Float	Int	Float	Int	Float
Split	20	32	20	48	28	17
Leaf	8	8	8	8	10	10

7.3.6 Architecture-Aware Code Generator

As noted earlier, for each combination of tree ensembles and target hardware architecture a different implementation might offer the best inferencing solution. Hence, we implement the discussed tree-framing methods in a single code-generator framework that generates the optimized realizations for a given forest and target platform. Figure 7.13 gives an overview of the whole workflow. First, the pre-trained forest (in a JSON format) is loaded. Afterwards, the corresponding intermediate representation of the ML model is generated, and the proposed optimizations are performed, e.g., branch swapping, node re-indexing, etc. Finally, we provide a set of C-style templates that represent the specific implementation types (e.g. *native* or *if-else*). Several auxiliary scripts scripts are provided to automate the above procedures, e.g., selecting corresponding cross-compilers. Per default sci-kit learn models are targets [561] but other model definitions, in, say, the ONNX format are also supported. More details can be found at https://github.com/sbuschjaeger/fastinference.

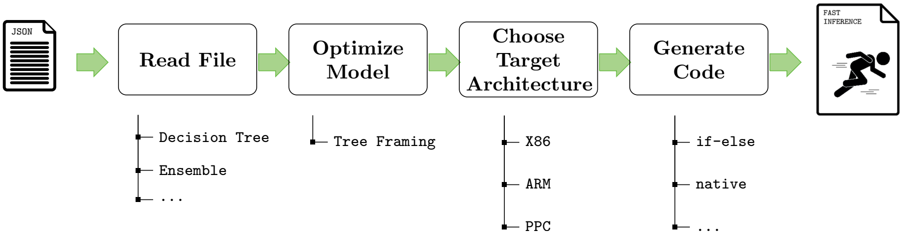


Fig. 7.13: Workflow of our code generator. The model configuration is loaded into an internal representation. If selected, optimizations are performed on the model before code generation. Afterwards, the target architecture and the appropriate templates are selected for final code generation.

7.3.7 Experimental Evaluation

We have performed 1800 different experiments by training Decision Trees (DT) [73], Random Forests (RF), [72] and Extremely randomized Trees (ET) [250] on 12 different datasets with varying tree-depths to generate the aforementioned realizations for different architectures, i.e., X86, PPC, and ARM CPUs. Table 7.8 shows the datasets we used during the experiments. All datasets are available in the UCI Machine Learning Repository [31] except for MNIST [420], IMDB [456], and FACT [17]. In addition to the number of features and the number of examples during test time, we also report the range of accuracy for the three different models DT, RF, and ET. In all experiments we used the CART algorithm with the Gini score criterion for node-splitting and trained models using the `sklearn` package[561]. For RF and ET, we used 25 trees. If the respective dataset comes with a pre-computed train/test split, we use this. Otherwise, we use 75 % of the data for training and 25 % of the data for testing. DTs often do not achieve high accuracy, whereas RF and ET perform best with large trees. We did not perform any hyperparameter optimization with respect to the classification accuracy and report the accuracy here to validate our code generator.

Since `sklearn` is arguably one of the most-used machine learning libraries we also compared its performance against our implementation. We found that, our realization is on average 500 – 1500 times faster than `sklearn`. However, we admit that this comparison is biased, because large parts of `sklearn` are written in Python and optimized for batch execution. Thus, we excluded these comparisons in the following discussion. For space reasons, we focus our evaluation on RF models, but found that DT and ET result in similar behaviors across all systems. We use the *naive native* realization as the baseline for all experiments, and measure the average speed-up for each dataset of each optimization against this realization. To minimize unfairness due to caching, we classify all samples in the test set twice, but only report the runtime of the second run. We repeat the whole process 50 times and report the average speed-up across these 50 repetitions.

For *native* optimizations, we choose $\tau = 25$ on X86, $\tau = 8$ on ARM, and $\tau = 8$ for the PPC CPU. For *if-else* optimizations, we use an instruction-cache size $\beta = 128\,000$ bytes on X86, $\beta = 32\,000$ bytes on ARM, and $\beta = 32\,000$ bytes on the PPC CPU. The experiments were performed on an `Intel Core i7-6700` desktop machine with 16 GB RAM for `X86`. For `PPC`, we use a `NXP Reference Design Board` with `T4240 processors` and 6 GB RAM. For `ARM`, we use an `Raspberry PI 2` with an `ARMv7` CPU and 1 GB RAM.

Experiments on the X86 CPU Architecture Figure 7.14 depicts the average speed-up of the four different optimizations on Intel. First we note, that the *if-else* trees are the fastest on Intel and offer a speed-up of around three across all tree depths. For smaller tree depths from 1 – 10, we see that optimizing *if-else* trees only offers marginal speed-up. However, for larger tree depths of around 15 and 20, we can see that optimized

Tab. 7.8: Summary of datasets for our experiments based on UCI datasets [31], IMDB [456], MNIST [420], FACT [17].

Dataset	# Examples	# Features	Accuracy
adult	8141	64	0.76 - 0.86
bank	10 297	59	0.86 - 0.90
covertype	145 253	54	0.51 - 0.88
fact	369 450	16	0.81 - 0.87
imdb	25 000	10 000	0.54 - 0.80
letter	5000	16	0.06 - 0.95
magic	4755	10	0.64 - 0.87
mnist	10 000	784	0.17 - 0.96
satlog	2000	36	0.40 - 0.90
sensorless	14 628	48	0.10 - 0.99
wearable	41 409	17	0.57 - 0.99
wine-quality	1625	11	0.49 - 0.68

if-else trees can retain their speed-up and outperform unoptimized *if-else* trees with a speed-up factor larger than 3.

Native trees do not perform as well as *if-else* trees on Intel CPUs. Overall, the speed-up compared with *naive native* trees is only marginal for smaller trees below depth 15. Here, both versions, i.e., the *StandardNativeTree* and the *OptimizedNativeTree*, offer a speed-up of 1.5 at most. Interestingly, for larger trees around depth 15 and more, we again notice that our optimizations improve performance.

Experiments on the PPC CPU architecture Figure 7.15 depicts the average speed-up of the four different optimizations on PPC. We can observe that the results here are similar to Figure 7.14, in which *if-else* trees always outperform *native* trees with a speed-up in the range from 2 – 5. Along with the increment of tree depth, the speed-up from both *if-else* tree versions drops. Un-optimized *if-else* trees suffer especially from degraded performance, dropping to almost 2, whereas the optimized version can retain a speed-up of around 3.5.

Similar to X86 CPU, the *native* realization does not seem to be the best choice as it provides a speed-up under 2 in all cases. However, with increasing tree depths, optimizations are more important. It is worth noting, that we can observe cases where the *native* trees outperform *if-else* trees when tree depth is larger than 15.

Experiments on the ARM CPU Architecture Figure 7.16 depicts the average speed-up of the four different optimizations on ARM. We observe that the situation on ARM is more fragmented than that of X86 and PPC. In general, we are able to achieve a speed-up of around 4 for small trees, which drops to around 2 – 3 for larger trees. Both realizations roughly start with the same speed-up factor for small trees, but then quickly diverge for tree depth from around 5 – 15. In this range of tree depth, we see that *if-else* trees are the

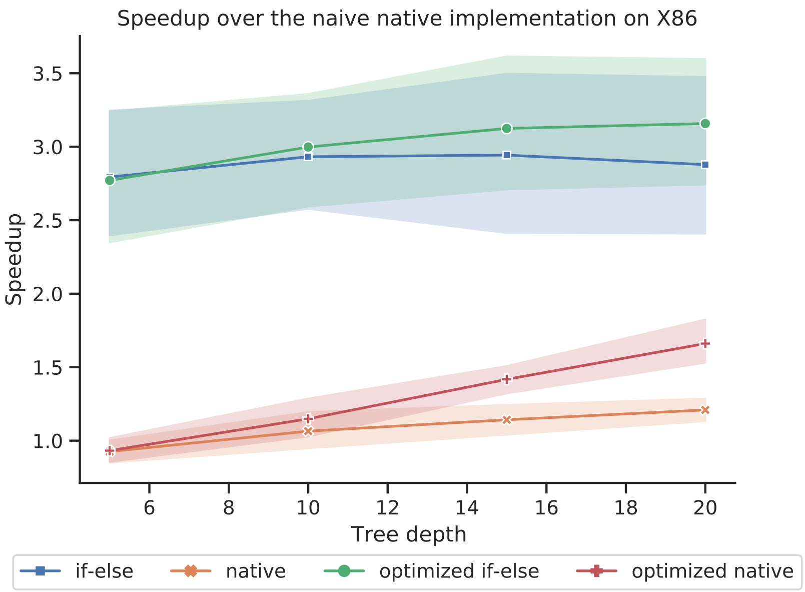


Fig. 7.14: Average speed-up factor for real-time execution compared with the naive native realization on Intel for tree depths from 1 – 20.

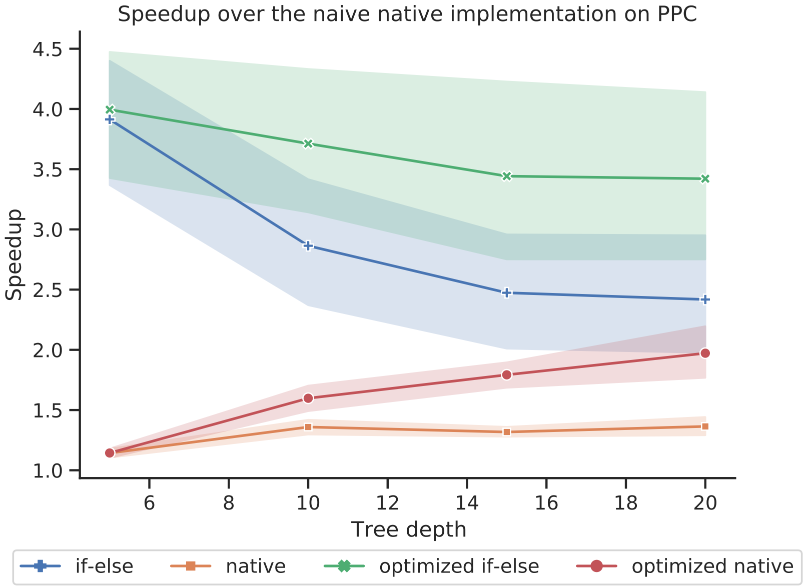


Fig. 7.15: Average speed-up factor for real-time execution compared to the naive native realization on PPC for tree depths from 1 – 20.

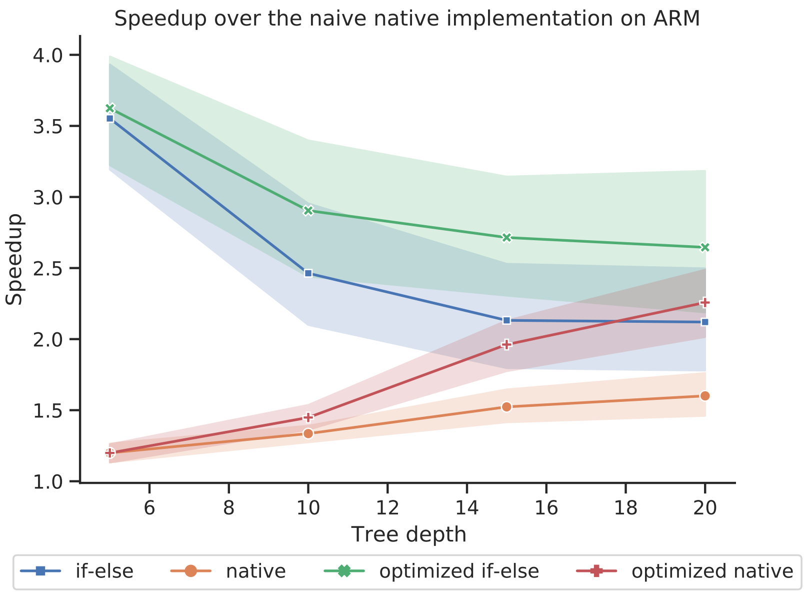


Fig. 7.16: Average speed-up factor for real-time execution compared with the naive native realization on ARM for tree depths from 1 – 20.

fastest choice on ARM. Additionally, we notice that with increasing tree depths cache optimizations become more important and consistently outperform their un-optimized counterpart. Once trees are sufficiently large, we see that the *native* trees match again the performance of *if-else* trees and even outperform them for tree depths of 15 and 20 in some cases. In this sense, the results are similar to what we have seen on the PPC architecture.

7.3.8 Discussion of the Experiments

The experiments show differences and similarities across the three architectures. Here, we want to discuss these phenomena in terms of the properties of the specific architectures, as well as the particular CPU models used for experiments. We note that one of the main architectural differences between X86, ARM, and PPC are the available instructions. Since *native* trees only use a small amount of hot-code, the differences between CPU architectures will likely not matter much here. However, while looking at *if-else* trees, we can expect a larger difference. To further investigate the interplay between

CPU architectures and code size, we consider Table 7.9, [7] which depicts the instruction size of a tree kernel function for varying tree depths over the FACT dataset (containing floating-point features) and the covertype dataset (containing integer features) under the standard *if-else* tree realization. For Intel CPUs, as shown in Figure 7.14, we notice that *if-else* trees are the best choice. There are mainly two reasons. First, X86 CPUs are Complex Instruction Set Computers (CISC) offering a very rich set of instructions that include all sorts of specialized operations. Since *if-else* trees unroll the complete tree structure into instructions, they give the compiler the opportunity to utilize this multitude of instructions to the fullest by encoding larger parts of the tree in single instructions. From Table 7.9 we can also see that the Intel CPU almost always requires the fewest instructions per decision tree. Second, in our experimental setting, the Intel Core i7-6700 CPU has a comparably large instruction cache of 256 KiB combined with two larger shared caches of 1 MiB (L2 Cache) and 8 MiB (L3 Cache). Thus, by encoding a single tree in only a few instructions, it is likely to fit it into the larger instruction cache. By contrast, *native* trees do not benefit from the CISC architecture and require additional space in the data cache by encoding the tree nodes as data instead of instructions.

Tab. 7.9: The actual size of instructions for *if-else* tree executing kernel functions on different architectures with the O3 option.

(a) Kernel size with integer features for covertype dataset

DateType	DT-1	DT-5	DT-10	DT-15	DT-20
Intel	224	575	8185	51005	167644
PPC	232	604	7732	51840	170772
ARM	204	604	9040	55012	180628

(b) Kernel size with floating point features for fact dataset

DateType	DT-1	DT-5	DT-10	DT-15	DT-20
Intel	96	415	17023	127330	404722
PPC	96	556	20996	169696	577952
ARM	88	428	18436	154992	542020

As with the X86 architecture, we have seen that *if-else* trees perform very well on the PPC architecture, but to a lesser extent. The PPC CPU architecture is a Reduced Instruction Set Computer (RISC) with performance enhancement for high performance computing. RISC does not offer instructions for specialized operations as CISC does.

[7] Although the instructions generated by the compiler may differ due to aggressive compiler optimization (O3) compared with the presented node sizes (O0 optimization) in Table 7.7, the code generator at the end selects the O3 option to accelerate the realizations as much as possible.

Thus, the compiler must largely rely on the combination of (comparably) simple instructions to implement *if-else* trees. This, in turn, results in larger code that is less likely to fit into the instruction cache. Comparing the instruction size of PPC with X86 in Table 7.9 we see that the PPC architecture indeed requires more instructions than with X86. Interestingly, this case is less severe for integer features, due to the enhancements in this instruction set architecture. Considering the cache sizes of the T4240 processors used in the experiments, we see that it only has a 32 KiB instruction cache, but also comes with a 2 MiB shared L2 cache, which is even larger than the Intel Core i7-6700 CPU. For smaller trees of around 5 – 10, the cache sizes are still large enough to hold all trees, and thus *if-else* trees are still the fastest choice. If trees become large (depth 10 or more), the instruction cache is not enough to hold all trees and we must rely on the larger L2 cache. However, this cache is slower, which in combination with the larger code size explains the performance drop for larger trees.

Finally, we discuss the fragmented behavior of the ARM architecture. Much like its PPC counterpart, ARM also uses a RISC architecture. However, ARM's RISC does not come with specialized instructions for high-performance computing, and thus the compiler has to completely rely on the combination of simple instructions for *if-else* realization. This in turn results in even larger code for integer features, which is less likely to fit into the instruction cache as shown in Table 7.9. Interestingly, for floating-point features, we see that the ARM CPU uses fewer instructions than the PPC CPU, which is attributable to the specific CPU model used during experiments. The T4240 processors are optimized for high-performance computing in a low-power embedded computing setting, such as networking applications, and thus are optimized for integer operations. By contrast, the ARMv7 CPU of the Raspberry PI 2 is a general-purpose CPU aimed at the needs of the average user, and thus it places a larger emphasis on floating-point operations compared with the T4240 processors. It has a 32 KiB instruction cache in combination with a significantly smaller 512 KiB L2 shared cache. Compared with the other CPUs, this means that the ARM CPU has 2 – 16 times less L2/L3 cache available. For smaller trees around a depth of 5 – 10, the cache sizes are still enough to hold all trees, and thus *if-else* trees are still the fastest choice. For larger tree depths, however, the instruction cache is not enough and *native* structures using the data cache become faster. However, since the data cache is also small, both caches are filled quickly to their maximum. Interestingly, if we optimize both *if-else* and *native* trees, we end up with roughly the same performance.

7.3.9 Conclusion

DTs form one of the building blocks of modern machine learning and ensembles of decision trees are one of the most successful classifiers regularly achieving state-of-the-art performance in real-world applications. DTs are generally regarded as 'simple' classifiers that can be executed even on the tiniest of hardware. However, a tree easily contains up to millions of decision nodes that must be stored and managed which can be a challenge even for large server hardware. Cache memory is commonly adopted in today’s von-Neumann computing architecture to hide the long latency between the main memory and the processor. Hence, an efficient realization of a given tree ensemble must respect this memory hierarchy and provide a suitable memory layout of the decision nodes for optimal performance. In every modern programming language there are at least two ways to implement a DT: either one decomposes the tree into its if-else structure or one uses a while-loop to iterate over a continuous array of nodes. Both approaches offer different caching behaviours that can be further enhanced by the tree-framing methods discussed in this contribution. At the core of these methods lies the fact that DTs do not have a deterministic runtime, but its execution time may vary depending on the current sample. Hence, a probabilistic view of DT execution estimates the most probable paths of the tree and frames the tree so that these paths are likely to remain in the cache. The experimental evaluation shows a speed-up around 2 – 4 across three different hardware architectures on a variety of datasets without any loss in accuracy occurs.

8 Communication Awareness

The ubiquity of connected devices and parallel computing platforms challenges efficient and reliable execution of machine learning algorithms. If machine learning workloads are executed merely locally, a system does not always have sufficient resources at its disposal to perform the necessary operations fast enough. Furthermore, at a smaller scale, multiple hardware components these days are interconnected via on-chip or off-chip networks to create many-core systems. Communication, synchronization, and offloading have thus become essential in designing embedded systems under communication and resource constraints.

This chapter presents (1) the timing predictability of embedded systems and (2) the communication architecture in heterogeneous CPU/GPU environments. Synchronization with resource sharing, communication with potential failures, and probabilistic timing information are presented in Section 8.1. Bandwidth limitations of different execution models and coprocessor-accelerated optimization are presented in Section 8.2.

https://doi.org/10.1515/9783110785944-008

8.1 Timing-Predictable Learning and Multiprocessor Synchronization

Kuan-Hsun Chen
Junjie Shi

Abstract: With the increasing demand for time-predictable machine learning applications, e.g., object detection in autonomous driving systems, such a trend poses several new challenges for resource synchronization in real-time systems, especially when hardware accelerators like Graphics Processing Units (GPUs) are considered as shared resources. When the shared resources have relatively high utilization, conventional synchronization mechanisms might result in performance downgrade.

We thus propose the emerging Dependency Graph Approach (DGA), where the precedence constraints of all the computation segments are pre-proceeded. Such a non-work-conserving approach can schedule long critical sections, which may be even longer than the period of another task. This is not the case in all the other work-conserving protocols typically in use. Throughout numerical experiments, we show that DGA outperforms all the other conventional protocols in all the evaluated configurations when shared resources are highly utilized.

Additionally, a system does not always have sufficient resources at its disposal to perform the necessary operations fast enough if machine learning workloads are executed merely locally. One sound approach is to offload heavy workload to powerful remote servers and expect the inference outcome can be received in time. However, since this approach highly depends on network connectivity and responsiveness, typically only non-critical tasks are offloaded, whose timing requirements are less strict than those of critical tasks. Against such a pessimistic design, we present two novel offloading protocols that offload both critical and non-critical tasks. They handle uncertain connections while providing certain timing guarantees.

To achieve a timing-predictable design, typical timing analyses always consider the worst-case execution pattern to derive timing guarantees. But this approach is often too restrictive for some machine learning applications with soft timing constraints. To mitigate the pessimism, we develop several timing analyses of the probability of deadline misses and the deadline miss rate, two important metrics considered in the literature to quantify timeliness.

8.1.1 Introduction

Under the von Neumann programming model, shared resources that require mutually exclusive accesses, e.g., shared files, data structures, etc., have to be protected by applying synchronization (e.g., *binary semaphores*) or locking (e.g., *mutex locks*) mechanisms. Protected code segments that have to access shared resource(s) mutually exclusively are called *critical sections*. For uni-processor real-time systems, longstanding protocols developed in the 90s, are the current state of the art. These are namely the Priority Inheritance Protocol (PIP) and the Priority Ceiling Protocol (PCP) by Sha et al. [623], as well as the Stack Resource Policy (SRP) by Baker [37].

Along with the development of multiprocessor platforms, multiprocessor resource synchronization and locking protocols have been proposed and extensively studied. These include Distributed-PCP (DPCP) [592], Multiprocessor PCP (MPCP) [591], Multiprocessor SRP (MSRP) [238], Flexible Multiprocessor Locking Protocol (FMLP) [58], $O(m)$ Locking Protocol (OMLP) [69], and Multiprocessor resource sharing Protocol (MrsP) [101].

However, the performance of aforementioned protocols highly depends on 1) how the tasks are partitioned and prioritized, 2) how the resources are shared locally and globally, and 3) whether a job/task being blocked should spin or suspend itself. In the literature, conventional synchronization mechanisms might result in performance downgrade, since most of them are designed for sporadic tasks with relatively low utilization for critical sections, which are often not able to represent emerging heavy-loaded machine learning applications. We thus propose a novel concept called DGA, which can serve high utilization of critical sections well.

Moreover, when the workload of a critical section, e.g., a machine learning workload on GPU, is extremely high, a system does not always have sufficient resources at its disposal to perform the necessary operations fast enough. A sound solution is to offload heavy workload to powerful remote servers and wait for the outcome of the inference processes. However, the performance and stability of this approach highly depends on the quality of the network. To improve flexibility, we propose several adaptive protocols to ensure that the timing requirements of safety- and mission-critical tasks are not violated even in the case of connectivity issues while obtaining the benefits of offloading computation shares.

Last but not least, to achieve a timing-predictable design, conventional timing analyses always focus on the worst-case execution pattern to derive hard timing guarantees. However, such analyses are sometimes too pessimistic when systems can accept rare deadline misses, e.g., for soft real-time systems. Limited deadline misses on many machine learning applications might only lead to performance degradation, e.g., for image and voice recognition on smart edge devices. Some end users might only feel inconvenienced without further serious consequences. However, people might still wonder how resilient the considered system is with respect to deadline misses in a probabilistic argument. To obtain the probability of deadline misses, we innovate a well-known convolution-based approach based on multinomial distribution, and adopt several concentration inequalities to derive analytical upper bounds to further improve the efficiency of calculation.

Overall, the presented contributions in this chapter are as follows:

Dependency Graph Approach (DGA) We propose a novel method for periodic real-time task systems, which predestines the sequence where tasks can access resources. The conducted numerical simulations show that DGA outperforms all state-of-the-art approaches in most evaluated configurations, especially when utilization of critical sections is relatively high.

Offloading protocols for unreliable connection We present two offloading protocols that offload both critical and non-critical tasks. They deal with uncertain connections while providing certain timing guarantees. A case study on a robotic system demonstrates the applicability of the proposed protocols under various configurations.

Deadline-miss analyses A novel approach based on the multinomial distribution is proposed that calculates the deadline miss probability with drastically better analysis runtime without any precision loss. Furthermore, we propose several analytical bounds based on various concentration inequalities. The evaluation shows that our approaches are applicable for significantly larger task sets while preserving the quality of derived results, compared with conventional convolutional-based approaches.

8.1.2 Related Work

For multiprocessor systems, many resource synchronization and locking protocols are extensions of these aforementioned well-known uni-processor protocols. For example, Rajkummar et al. [592] proposed DPCP, where each resource is assigned on a processor statically, and critical sections are executed on the corresponding processor where the requested resource is assigned on. The extension MPCP [591] enables tasks to execute their critical section locally. In order to minimize the usage of stack memory in real-time systems, Gai et al. [238] proposed MSRP. Block et al. [58] introduced FMLP, where resources are divided into two groups, i.e., long and short. For short resources, critical sections are executed in a non-preemptable manner and tasks spin on their processors while waiting for resources. For long resources, tasks suspend themselves into a First In First Out (FIFO) queue while waiting. Brandenburg and Anderson [69] proposed OMLP, which ensures $O(m)$ maximum pi-blocking for any task set. Burns et al. [101] proposed MrsP, which allows tasks to progress other tasks that have occupied the same requested resource, in order to reduce the blocking time. A comprehensive survey of multiprocessor real-time locking protocols can be found in [68].

Besides fully relying on local computational power, offloading computation to remote servers is a reasonable solution to ease the pressure of resource constraints on embedded systems. In 2012, a cloud-assisted system for autonomous driving was firstly studied by Kumar et al. [406]. In 2015, Esen et al. [203] presented a software architecture named *Control as a Service* in which all control functions are completely moved to the cloud. In 2018, Adiththan et al. [4] proposed an adaptive offloading technique for control applications that makes all offloading decisions online based on a network performance monitor. Recently, Al Maruf and Azim [469] proposed a strategy for task offloading in multiprocessor mixed-criticality systems with dynamic scheduling policies under overload conditions. For real-time systems that allow offloading, one concept for modeling this particular local system view is *self-suspension* [127]. One of the state-of-the-art models can be applied such as the dynamic self-suspension model (e.g. [324], [442]), the segmented self-suspension model (e.g. [611]), or a hybrid model, e.g. [84]. For a detailed overview, see [127, 128].

To safely derive probabilistic timing guarantees, which enable a tradeoff between system safety and hardware costs, several techniques have been developed in the literature. Diaz et al. [177] developed a framework for calculating the deadline miss probability based on convolution for periodic task systems. In addition, Tanasa et al. [657] used the Weierstrass Approximation to approximate any arbitrary execution time distributions and applied a customized decomposition procedure to search all the possible combinations. However, the two approaches can derive only the probability of deadline misses with 7 and 25 jobs in the hyper-period, respectively. For sporadic real-time task systems, in which two consecutive jobs of a task do not have to be released periodically, Axer et al. [27] proposed evaluating the response-time distribution and iterating over the activations of job releases for non-preemptive fixed-priority scheduling. Maxim et al. [476] provided a probabilistic response time analysis by assuming a probabilistic minimum inter-arrival as well as probabilistic worst-case execution times (WCETs) for the fixed-priority scheduling policy. Ben-Amor et al. [46] extended the probabilistic response time analysis in [476] by considering precedence-constrained tasks. These convolution-based approaches are in general not scalable due to the huge number of jobs in the interval of interest.

8.1.3 Dependency Graph Approach

In this subsection, the dependency graph approach is presented in detail, including the primary design of DGA, the extension for supporting periodic task systems, and the corresponding scheduling algorithms.

8.1.3.1 Primary Design of DGA

We consider a set of n *frame-based* real-time tasks $\mathbf{T} = \{\tau_1, \ldots, \tau_n\}$ that is scheduled on M identical (homogeneous) processors. Each task is described by $\tau_i = ((C_{i,1}, A_{i,1}, C_{i,2}), T_i, D_i)$. The given tasks release their jobs at the same time and have the same period and relative deadline. Specifically, each task τ_i releases a job (at time 0 for notational brevity) with the following properties:

- $C_{i,1}$ is the execution time of the first non-critical section of the job.
- $A_{i,1}$ is the execution time of the (first) non-nested critical section of the job, in which a binary semaphore or a mutex $\sigma(\tau_{i,1})$ is used to control the access to the critical section.
- $C_{i,2}$ is the execution time of the second non-critical section of the job.

A sub-job is a critical section or a non-critical section. Therefore, each job of task τ_i has three sub-jobs. We assume the task set $\mathbf{T}$ is given and a constrained deadline is considered, i.e., $D_i \leq T_i$. We also make the following assumptions:

- For each task τ_i in $\mathbf{T}$, $C_{i,1} \geq 0$, $C_{i,2} \geq 0$, and $A_{i,1} \geq 0$.
- The execution of the critical sections guarded by one binary semaphore s must be sequentially executed under a total order. That is, if two tasks share the same semaphore, their critical sections must be executed one after another without any interleaving.
- The execution of a job cannot be parallelized, i.e., a job must be sequentially executed in the order of $C_{i,1}, A_{i,1}, C_{i,2}$.
- There are in total Z binary semaphores.

The dependency graph approach consists of the following two steps:

- In the first step, a directed graph $G = (V, E)$ is constructed. A subjob (i.e., a critical or a non-critical section) is a vertex in V and the edges in E describe the precedence constraints of these jobs. The subjob $C_{i,1}$ is a predecessor of the subjob $A_{i,1}$, and $A_{i,1}$ is a predecessor of the subjob $C_{i,2}$. If two jobs of τ_i and τ_j share the same binary semaphore, i.e., $\sigma(\tau_{i,1}) = \sigma(\tau_{j,1})$, then either the subjob $A_{i,1}$ is the predecessor of $A_{j,1}$ or the subjob $A_{j,1}$ is the predecessor of $A_{i,1}$. All the critical sections guarded by a binary semaphore form a chain in G, i.e., the critical sections of the same binary semaphore follow a total order. Therefore, we have the following properties in set E:
  - The two directed edges $(C_{i,1}, A_{i,1})$ and $(A_{i,1}, C_{i,2})$ are in E.
  - Suppose that $\mathbf{T}_k$ is the set of tasks that require the same binary semaphore s_k. Then, the $|\mathbf{T}_k|$ tasks in $\mathbf{T}_k$ follow a certain total order π such that $(A_{i,1}, A_{j,1})$ is a directed edge in E when $\pi(\tau_i) = \pi(\tau_j) - 1$.

  Figure 8.1 provides an example of a task dependency graph with one binary semaphore. Since there are Z binary semaphores in the task set, the task dependency graph G has in total Z connected subgraphs, denoted as $G_1, G_2, \ldots, G_z$. In

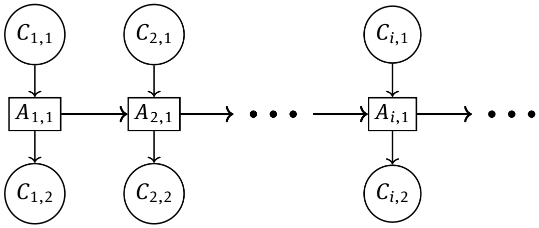


Fig. 8.1: A task dependency graph for a task set with one binary semaphore.

each connected subgraph G_ℓ, the corresponding critical sections of the tasks that request critical sections guarded by the same semaphore form a chain and have to be executed sequentially. For example, in Figure 8.1, the dependency graph forces the scheduler to execute the critical section $A_{1,1}$ prior to any of the other three critical sections.

- In the second step, a corresponding schedule of G on M processors is generated. The schedule can be based on system restrictions or user preferences, i.e., it can be based on either preemptive or non-preemptive schedules, or on either global, semi-partitioned, or partitioned schedules.

Algorithms to Construct G The objective of constructing dependency graph, i.e., G, is to minimize the makespan, i.e., the latest finishing time of all tasks, with the assumption that the number of *virtual processors* is the same as the number of tasks, based on uni-processor non-preemptive scheduling. For each task, $C_{i,1}$ is considered as release time r_i, and $C_{i,2}$ is considered as delivery time. There are several existing algorithms to derive good approximations of $G^\star$, where $G^\star$ is the graph with the optimal makespan: 1) the **extended Jackson's rule** [289], which is a polynomial-time algorithm with 2-approximation [377]; 2) the **Potts** [583], which is a polynomial-time 1.5-approximation algorithm [289]; 3) and the improvement of the approximation ratio to $4/3$ by Hall and Shmoys [289].

8.1.3.2 Extension to Periodic Task Systems

To increase the applicability, we extend the DGA to handle multiprocessor synchronization for *periodic* real-time task systems. That is, we unroll the jobs of all the tasks in one hyper-period and then construct a dependency graph of these jobs. Suppose that the hyper-period H of a task set is the least common multiple (LCM) of the periods of all the tasks in this set. For each task τ_i that requests (at least) one resource, we create H/T_i jobs of task τ_i. For the ℓ-th job of task τ_i, we set its release time to $(\ell-1)T_i$ and its absolute deadline must be no later than $(\ell-1)T_i+D_i$. Since the jobs for one task should not have any execution overlap with each other, we only need one virtual processor or dedicated shop for them, but the release time constraint is added for each job. The three methods in Section 8.1.3.1 can still be applied by adding the release time constraint for each job. Afterward, a dependency graph for all the jobs in one hyper-period is generated. In the end, the schedules are generated offline. And the generated schedules will be repeated in the upcoming hyper-periods.

Please note that such an extension can be applied to any periodic real-time task system but that it comes at the cost of space and computation, due to the increasing number of jobs in one hyper-period.

8.1.3.3 Scheduling Algorithms

In the following, we show three scheduling algorithms for the same dependency graph(s) under different system specifications.

List-EDF Here, we show how to schedule the unrolled dependency graphs over the hyper-period. For the ℓ-th job of τ_i, J_i^ℓ has three subjobs $J_{i,1}^\ell, J_{i,2}^\ell, J_{i,3}^\ell$ that represent the related subjobs $C_{i,1}, A_{i,1}, C_{i,2}$, respectively. The release time of the first subjob is $J_{i,1}^\ell$ is $(\ell - 1)T_i$, and the absolute deadline of the last subjob $J_{i,3}^\ell$ is $(\ell - 1)T_i + D_i$. Regarding the release times of the second and third subjob, we initially set the earliest possible time the job may be released based on the WCETs of the other subjobs. Meanwhile, regarding the deadline of the first and second subjob, we initially assign the latest possible time the subjob can finish while still allowing schedulability. To be precise, the release time of $J_{i,2}^\ell$ is set to $(\ell - 1)T_i + C_{i,1}$, the release time of $J_{i,3}^\ell$ is set to $(\ell - 1)T_i + C_{i,1} + A_{i,1}$, the absolute deadline of $J_{i,2}^\ell$ is set to $(\ell - 1)T_i + D_i - C_{i,2}$, and the absolute time of $J_{i,1}^\ell$ is set to $(\ell - 1)T_i + D_i - C_{i,2} - A_{i,1}$.

We assume that each dependency graph $\mathbf{G}_s$ for a binary semaphore s is constructed for the corresponding jobs released (strictly) within one hyper-period H. If $H_s < H$, then $\frac{H}{H_s}$ copies of $\mathbf{G}_s$ are applied in a consecutive order to represent the precedence constraints of the critical sections. For notational brevity, we denote $r_{i,j}^\ell$ as the release time of the subjob $J_{i,j}^\ell$ and $d_{i,j}^\ell$ as the absolute deadline of $J_{i,j}^\ell$. If the absolute deadline of an immediate predecessor of $J_{i,j}^\ell$, denoted as $IPre(J_{i,j}^\ell)$, is larger than $d_{i,j}^\ell$, the absolute deadline of the immediate predecessor should be reassigned to $d_{i,j}^\ell$ minus the WCET of $J_{i,j}^\ell$. This is a standard procedure for scheduling jobs subject to release dates and precedence constraints. Details can be found in [36].

We assume that the absolute deadline assignment is adjusted accordingly so that $d_{i,j}^\ell$ for the subjob $J_{i,j}^\ell$ is always greater than the absolute deadline of $IPre(J_{i,j}^\ell)$. Scheduling $\mathbf{G}_1, \mathbf{G}_2, \dots, \mathbf{G}_z$ on M homogeneous (identical) processors is a special case of the classical scheduling problem $P|prec; r_j|L_{\max}$, i.e., scheduling a set of jobs with specified release times and precedence constraints on M identical processors, minimizing the maximum lateness. One possible scheduling strategy is to use the List scheduling developed by Graham [269] in combination with Earliest Deadline First scheduling (EDF). A List schedule works as follows: Whenever a processor idles and there are subjobs eligible to be executed (i.e., all of their predecessors in the dependency graph have finished), one of the eligible subjobs is executed on the processor. If more subjobs than processors are available, we prioritize the subjobs that have the earlier absolute deadlines. If two subjobs have the same absolute deadline, the one with the larger remaining workload has a higher priority. We call this scheduling algorithm List-EDF.

Federated-Based Partitioning Algorithm Federated scheduling was proposed by Li et al. [430] in order to schedule parallel real-time task systems with internal precedence constraints that can be modeled as a Directed-Acyclic Graph (DAG). The foremost intention of this scheduling algorithm is to provide provably good approximations with respect to an optimal scheduling algorithm while considering implementation constraints, e.g., cache hit-rates and memory accesses during runtime. The idea of federated scheduling is to assign DAGs (in our case the DAGs resulting from the dependency graph construction) that need to utilize more than one processor (so-called *heavy* graphs) to those processors exclusively. Analogously, the graphs that can be feasibly scheduled on a single processor are denoted as *light* graphs and are scheduled jointly on the remaining processors, i.e., non-exclusively allocated processors. After this initial partition, the actual scheduling is done by a work-conserving scheduler on the assigned processors. If the graphs in both the *heavy* group and the *light* group can be scheduled feasibly, the corresponding partition is returned. Otherwise, there is no feasible partition.

Worst Fit-Based Heuristic In addition, a worst-fit heuristic is proposed in which the tasks are partitioned one by one. The tasks are first sorted according to a sorting strategy. After that, they are partitioned to the available processors using a worst-fit strategy, i.e., each task is assigned to the processor with the currently lowest utilization. Again, Partitioned-EDF (P-EDF) scheduling is applied to verify whether the resulting partition on M processors is feasible.

We proposed two sorting strategies: 1) sort all the tasks decreasingly with regard to the tasks' utilization, no matter which resources they request; 2) sort the graphs decreasingly with regard to the graph utilization and then sort the tasks inside each graph decreasingly with regard to the task utilization. In our proposed heuristic, both sorting strategies are applied. If the partition P generated by the first sorting strategy is not applicable, i.e., if the task set is not schedulable on M processors based on the current partition P using P-EDF, the second sorting strategy and the resulting partition $P^{'}$ are considered, and P-EDF is applied to verify the new partition $P^{'}$ once again. The algorithm only returns infeasible when both aforementioned sorting strategies cannot generate a schedulable partition. Otherwise, the task set is schedulable and the partition is returned. Again, if a time-driven schedule is created, the schedule can be returned as well.

8.1.3.4 Evaluation

We randomly generated task sets based on the number of processors M, shared resources Z, and relative utilization of the critical sections H as parameters. In our evaluation, we considered $M \in \{4, 8, 16\}$, $Z \in \{4, 8, 16\}$, and $H \in \{[5\,\% - 10\,\%], [10\,\% - 40\,\%], [40\,\% - 50\,\%]\}$.

For a given configuration of M, Z, and H, we generated task sets with $10 \times M$ tasks for each total utilization value $\sum_{\tau_i \in \mathbf{T}} U_{\tau_i} \in [0, M]$ with a step 5 %, applying the RandFixedSum method [199]. We enforced the total utilization $U_{\tau_i} \leq 0.5$ for each task τ_i. To determine the subtask utilization of one task, i.e., $U_{C_{i,1}}$, $U_{C_{i,2}}$, and $U_{A_{i,1}}$, we first decided the utilization of the critical section $U_{A_{i,1}}$ by randomly drawing a percentage of the task's total utilization U_{τ_i} based on the parameter H. Next, the remaining utilization U_{C_i} was split by drawing $U_{C_{i,1}}$ randomly uniform from $[0, U_{C_i}]$ and setting $U_{C_{i,2}}$ to $U_{C_i} - U_{C_{i,1}}$. The resource that each critical section of a task requests was selected randomly from all the available resources. In addition, we generated two kinds of task sets according to their settings of available periods:

Periodic task sets with semi-harmonic periods The task periods T_i are selected randomly from a set of semi-harmonic periods, i.e., $T_i \in \{1, 2, 5, 10\}$, which is a subset of the periods used in automotive systems [86, 290, 392, 606, 662].

Frame-based task sets As a special case of periodic task sets, all the tasks have the common period 1. Hence, i.e., $C_{i,1} = U_{C_{i,1}}$, $A_{i,1} = U_{A_{i,1}}$, and $C_{i,2} = U_{C_{i,2}}$.

For each of these setting of periods, 54 configurations are considered in total. For each of the utilization step values, 1000 task sets were randomly generated.

Evaluated Approaches To construct the dependency graphs, POTTS [583] is applied. Other evaluated methods to schedule the tasks sets were: 1) FED-P-EDF: the algorithm based on federated scheduling; 2) WF-P-EDF: the algorithm based on global worst-fit partitioning; 3) LIST-EDF: the List schedule based approach; 4) ROP-FP: Resource-Oriented Partitioned under Fixed-Priority [82]; 5) ROP-EDF: ROP under Dynamic-priority; 6) LP-GFP-FMLP [58]; 7) LP-GFP-PIP [194]; and 8) GS-MSRP [704].

Evaluation Results Only a subset of the results is presented, as the other results show similar trends. The evaluation results for periodic task systems are shown in Figure 8.2. If the workload of the critical sections is increased (Figure 8.2-(a) to (c)), the performance of all methods is reduced, and the difference between methods is decreased as well. The reason is that, when $\beta = [40\,\% - 50\,\%]$, the execution time of the critical section for tasks with period 10 time units can be large, i.e., longer than 2 time units. Therefore, tasks with period 1 time unit directly miss the deadline by default for all other approaches, no matter what kind of partitioning algorithm is applied. The performance drops down quickly when the utilization is increased and the critical section workload is large, as shown in Figure 8.2 (c).

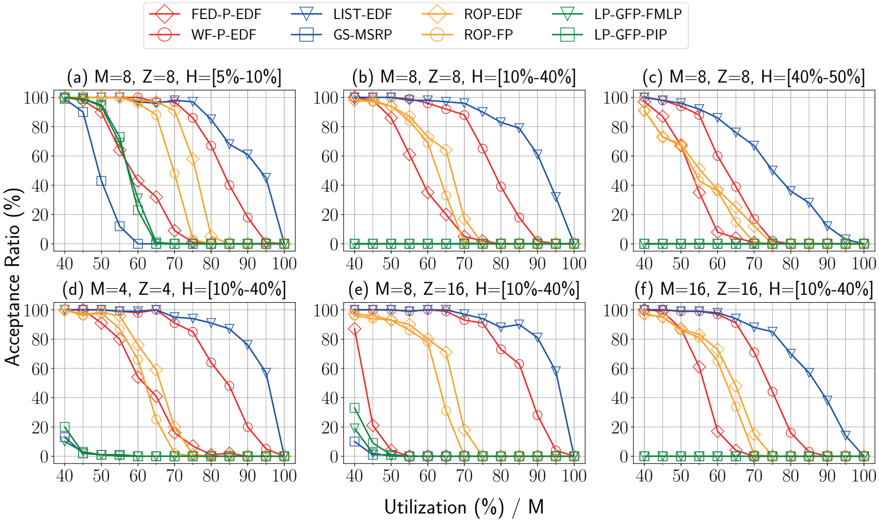


Fig. 8.2: Schedulability of different approaches for periodic task sets.

The evaluation results for frame-based task systems are shown in Figure 8.3. The proposed worst-fit heuristic WF-P-EDF outperforms ROP-EDF and other partitioned scheduling methods significantly. Furthermore, Figure 8.3 shows that WF-P-EDF has a good performance compared with LIST-EDF. In most cases, both LIST-EDF and WF-P-EDF can reach a 100 % acceptance ratio even with a 95 % utilization per processor.

8.1.4 Offloading Protocols for Unreliable Connection

In this subsection, two offloading protocols are presented in detail, addressing two system requirements: 1) the *service protocol*, which provides as much service for non-critical tasks as possible at any point in time, and 2) the *return protocol*, which allows a fast return to normal system behavior in the case of an unsuccessful offloading operation.

8.1.4.1 System Model

We consider a cyber-physical system comprising a set of tasks $\mathcal{T}$ that can be divided into two subsets with different requirements, namely, the set of *critical* tasks $\mathcal{T}_{crit}$, and the set of *non-critical* tasks $\mathcal{T}_{non}$, such that $\mathcal{T} = \mathcal{T}_{crit} \cup \mathcal{T}_{non}$ and $\mathcal{T}_{crit} \cap \mathcal{T}_{non} = \emptyset$. While for each $\tau_k \in \mathcal{T}_{crit}$ timing constraints must be satisfied at any point in time, for each $\tau_k \in \mathcal{T}_{non}$ timing violations may be unpleasant but not hazardous. According to the classification of tasks into two subsets, we specify two different system execution behaviors, i.e., *normal* and *local* execution behavior. When the system exhibits normal

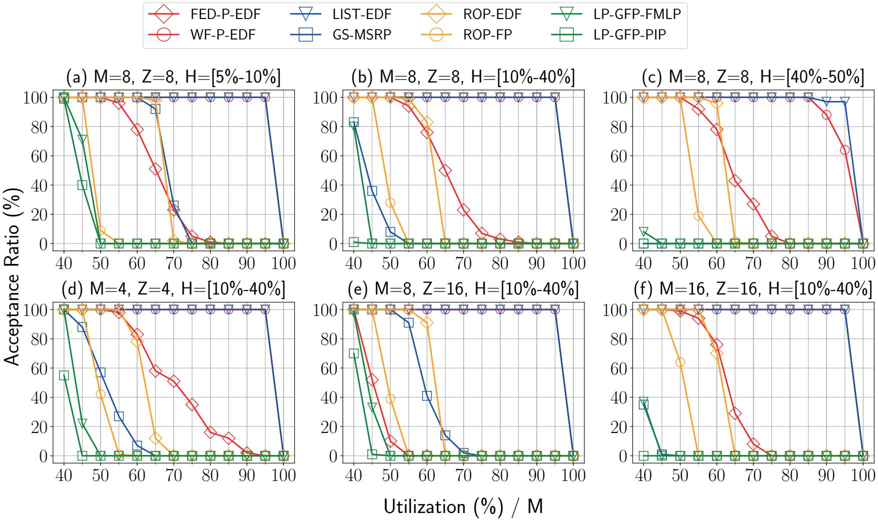


Fig. 8.3: Schedulability of different approaches for frame-based task sets(1).

execution behavior, all timing requirements of all tasks are satisfied at any point in time, whereas, if the system exhibits local execution behavior, timing guarantees can only be given for all critical tasks $\tau_k \in \mathcal{T}_{crit}$.

Each recurrent real-time task $\tau_k \in \mathcal{T}$ is assumed to have a sporadic arrival pattern and is characterized by a tuple $(C_{k,1}, C_{k,s}, C_{k,2}, S_k, p_k, q_k, D_k, T_k)$:

- Each τ_k releases an infinite number of task instances denoted as *jobs*. T_k indicates the minimum inter-arrival time of τ_k.
- D_k describes the relative deadline of τ_k. A constrained-deadline task system is considered, in which $D_k \leq T_k$ for each task τ_k.
- $C_{k,1}$ and $C_{k,2}$ denote the WCETs of the first and second *computation segments*, respectively.
- $C_{k,s}$ is the WCET of the typically offloaded task share if executed on the *local* system.
- p_k and q_k are the WCETs of the pre- and post-processing routines, which are executed before and after the offloading operation of a job of task τ_k, respectively.
- S_k is the offloading or *suspension* time of τ_k.

We assume that $T_k \geq D_k > 0$ and $C_{k,1}, C_{k,s}, C_{k,2}, S_k, p_k, q_k \geq 0$. Moreover, we assume that WCET of pre- and post-processing routines are less than or equal to the WCET of local execution, i.e., $p_k + q_k \leq C_{k,s}$. Furthermore, the WCET of a job of task τ_k under any possible execution scenario is greater than 0, i.e., $C_{k,1} + C_{k,s} + C_{k,2} > 0$ and $C_{k,1} + p_k + q_k + C_{k,2} > 0$. For notational brevity, we denote $C_k^\sharp = C_{k,1} + C_{k,s} + C_{k,2}$ and $C_k^\flat = C_{k,1} + p_k + q_k + C_{k,2}$.
In addition, we assume that the local cyber-physical real-time system, termed *local system*, is a uniprocessor system, in which tasks are scheduled according to a preemptive

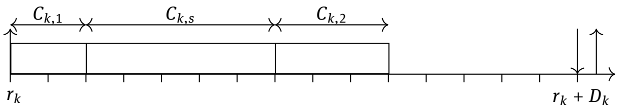


Fig. 8.4: A job of task τ_k is executed locally (local execution behavior).

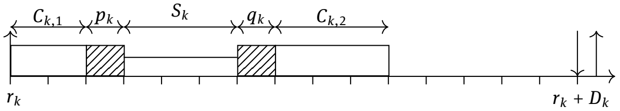


Fig. 8.5: An offloading operation of a job of task τ_k is performed successfully (normal execution behavior).

fixed-priority policy. More precisely, each task is assigned a unique priority, i.e., all jobs of task τ_k have the same priority. If at any point in time multiple jobs are ready, i.e., eligible for being executed on the local system, the job having the highest priority is executed. For each task τ_k, the unique set of the higher-priority tasks is denoted as $hp(\tau_k)$.

For a job of task τ_k arriving at time r_k the following execution scenarios are possible:

- The job is *executed locally* (Figure 8.4). In this case, the WCET of the job released at time r_k is $C_{k,1} + C_{k,s} + C_{k,2}$, i.e., $C_k^\sharp$.
- The job is *offloaded*. In this case, the job is first executed locally for up to $C_{k,1}$ execution time units and thereon enters the pre-processing routine for up to p_k execution time units. Suppose that the first computation segment as well as the pre-processing routine are finished at time ρ. Then, the considered job is offloaded to the remote system at time ρ. The actual offloading operation can be either *successful* or *unsuccessful*:
  - *Offloading is successful* if the computation result or *offloading response* is returned to the local system until time $\rho + S_k$. In this case, the offloading response is post-processed for up to q_k time units and the second computation segment is executed for up to $C_{k,2}$ time units (Figure 8.5). Accordingly, the execution time of the job of τ_k on the local system is at most $C_k^\flat$.
  - *Offloading is unsuccessful* otherwise. In this case, at time $\rho + S_k$, a local re-execution of the offloaded task share is performed for up to $C_{k,s}$ time units followed by the execution of the second computation segment for up to $C_{k,2}$ time units. In this case, the execution time of the job of τ_k on the local system is at most $C_k^\sharp + p_k$.

8.1.4.2 Recovery Protocols

Cyber-physical systems are applied throughout a broad range of areas, each exhibiting individual requirements and thus a need for situationally appropriate system behavior. For safety-critical cyber-physical systems, the timeliness of critical tasks must be guaranteed under any circumstances - even in the event of an unsuccessful offloading operation. Since in this case a larger amount of local resources is required, less resources remain to serve the non-critical tasks, as we explained in Section 8.1.4.1. However, depending on the actual system characteristics, timing constraints for non-critical tasks tend to be less strict. For instance, it is possible that a non-critical task misses its deadline, but that the results are still useful up to a certain degree [83, 87]. Nevertheless, it may be desirable to return to the normal execution behavior and to re-establish timing guarantees for both critical and non-critical tasks as soon as possible, especially since a non-critical task is not necessarily unimportant and thus should provide functionally and temporally correct results most of the time. Further discussion on the relation between criticality and importance can be found in [204].

Against this backdrop, we propose two recovery protocols allowing the system to satisfy its requirements under local execution behavior and to return to normal execution behavior:

- The *service protocol* aims to provide as much service as possible for non-critical tasks, even under local execution behavior.
- The *return protocol* aims to minimize the amount of time, in which the system exhibits local execution behavior after an unsuccessful offloading operation.

Independent of the actual protocol, we assume that the local system exhibits normal execution behavior at time 0, such that offloading is enabled for all tasks in $\mathcal{T}$. The schedule considers the execution of all tasks until the first moment $\gamma_{1,\searrow}$ in which the offloading operation of a certain task τ_k is unsuccessful. That is, a job of task τ_k, which has offloaded its computation at time $\gamma_{1,\searrow} - S_k$, does not receive the offloading response until time $\gamma_{1,\searrow}$ (Figure 8.6). Immediately after $\gamma_{1,\searrow}$, the local system exhibits local execution behavior. Until time $\gamma_{1,\searrow}$, three scenarios are possible for each incomplete job of all critical tasks τ_i in $\mathcal{T}_{crit}$:

- *The job of τ_i has not been offloaded*: In this case, no offloading operation will be performed for this job, but it is executed locally instead. Since it is possible that the pre-processing routine for offloading is already active at time $\gamma_{1,\searrow}$, the WCET of this job is upper-bounded by $C_{i,1} + p_i + C_{i,s} + C_{i,2}$, i.e., $C_i^\sharp + p_i$.
- *The job of τ_i is already offloaded, but no offloading response was received until time $\gamma_{1,\searrow}$*: In this case, the offloading process is aborted and the job is executed locally as of time $\gamma_{1,\searrow}$. Therefore, the WCET of this job is upper-bounded by $C_{i,1}+p_i+C_{i,s}+C_{i,2}$, i.e., $C_i^\sharp + p_i$.
- *The job of τ_i is already offloaded and the offloading response has been received prior to time $\gamma_{1,\searrow}$*: In this case, the job continues its final processing. Therefore, the WCET of this job is upper-bounded by $C_{i,1} + p_i + q_i + C_{i,2}$, i.e., $C_i^\flat$.

After $\gamma_{1,\searrow}$, timing guarantees are provided only for $\mathcal{T}_{crit}$. Moreover, offloading is inhibited for all critical tasks in the near future of $\gamma_{1,\searrow}$ due to the currently unreliable

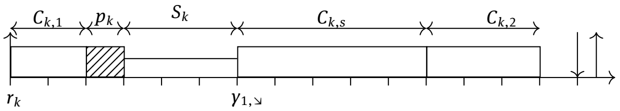


Fig. 8.6: An unsuccessful offloading operation of τ_k resulting in the transition to the local system behavior at time $\gamma_{1,\searrow}$.

connection leading to the missing offloading response. The offloading decision for non-critical tasks, however, depends on the applied recovery protocol:

Service Protocol Under the *service protocol*, offloading is inhibited for all instances of all tasks that are active as long as the system exhibits local execution behavior. The task share of each $\tau_i \in \mathcal{T}$ that is offloaded under normal execution behavior is executed locally within $C_{i,s}$ units of execution time. Since this leads to a higher workload on the local system, timeliness cannot be guaranteed for any non-critical task. Nevertheless, no non-critical task is aborted.

Return Protocol The *return protocol* does not inhibit offloading for *all* tasks, only for critical ones under local execution behavior. Non-critical tasks, by contrast, are offloaded regardless, but neither a re-execution nor a re-transmission is performed if an offloading response is not received in time. More precisely, the second subtask of τ_i is executed only if an offloading response is received, and aborted otherwise. Moreover, a job of τ_i in $\mathcal{T}_{non}$ is aborted whenever it misses its deadline.

As of time $\gamma_{1,\searrow}$, the local system exhibits local execution behavior until the point in time $\gamma_{1,\nearrow}$, in which timing guarantees can be given again for all tasks in $\mathcal{T}$. In the proposed protocols, two options are considered for the transit from local to normal execution behavior. They should be chosen based on the actual system requirements:

Abort-Transit This option aims to re-establish the normal system execution behavior as quickly as possible. Suppose that $\gamma_{1,\nearrow}$ is the earliest moment (after $\gamma_{1,\searrow}$) in which there is no incomplete job from $\mathcal{T}_{crit}$ at $\gamma_{1,\nearrow}$. All released but not yet finished instances of non-critical tasks are discarded.

Idle-Transit This option re-establishes the normal system execution behavior at the earliest moment $\gamma_{1,\nearrow}$ (after $\gamma_{1,\searrow}$) in which there is no incomplete job from $\mathcal{T}$ at $\gamma_{1,\nearrow}$.

We note that the above transitions are well-defined and the local system exhibits normal and local execution behavior in an interleaving manner.

8.1.4.3 Evaluation

In this subsection, we perform a case study on a robotic system to compare the acceptance ratio of schedulability over different protocols. More comprehensive numerical simulations can be found in the original paper [612].

Tab. 8.1: Periodic, implicit-deadline tasks; measurements of a Robotnik RB-1 Base robot platform. Note that the frequency of task τ_{laser} is 15.5 Hz.

Task	WCET [ms]	Period [ms]
τ_{laser}	6.732	64.516
τ_{odom}	1.046	60.0
τ_{tf}	0.333	60.0

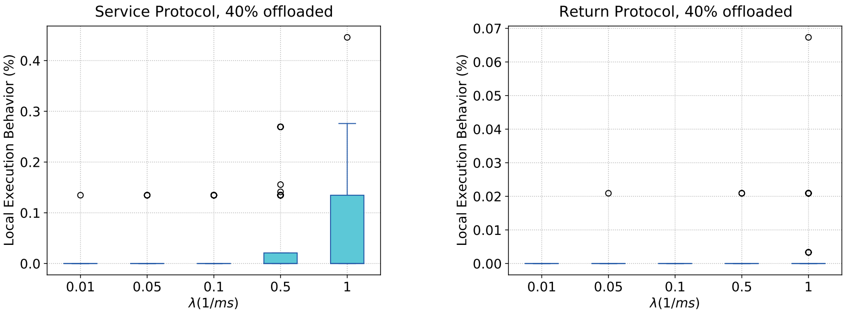


Fig. 8.7: The percentage of time the robot exhibits local execution behavior during the simulation for different probabilities of unsuccessful offloading operations and different percentages of offloaded workload under the service and the return protocol with 40 % offloaded workload per task.

Case Study on a Robotic System We adopt a Robotnik RB-1 Base robot platform [598], which uses the well-known Robot Operating System (ROS) [601]. We simulated the navigation of the robot in a virtual map and measured the timing data of the `move_base` node during a time frame of 60 seconds by using the Real-Time Scheduling Framework for ROS (ROSCH) [607] and RESCH [362]. We obtained three periodic, implicit-deadline tasks, as shown in Table 8.1, which are transformed into self-suspending tasks analogously to the tasks in experiment 1), and we considered the cases that 40 %, and 60 % of the task workload are offloaded. Moreover, we assume that $\mathcal{T}_{crit} = \{\tau_{odom}\}$ and $\mathcal{T}_{non} = \{\tau_{laser}, \tau_{tf}\}$. We simulate the system behavior using the event-based miss rate simulator from experiments 1) with $\lambda = 0.1 \cdot \frac{1}{ms}$. For each offloading case, the simulation was repeated 100 times.

Under the return protocol, Figure 8.8 shows that the amount of offloaded workload has no significant impact on the time that the system exhibits local execution behavior. Under the service protocol, we can observe that the time that the system exhibits local execution behavior is increased along with the increasing amount of offloaded workload. Overall, the derived results suggest that the amount of offloaded workload per task has a strong impact on the system execution behavior under the service protocol and thus should be taken into consideration at system design time.

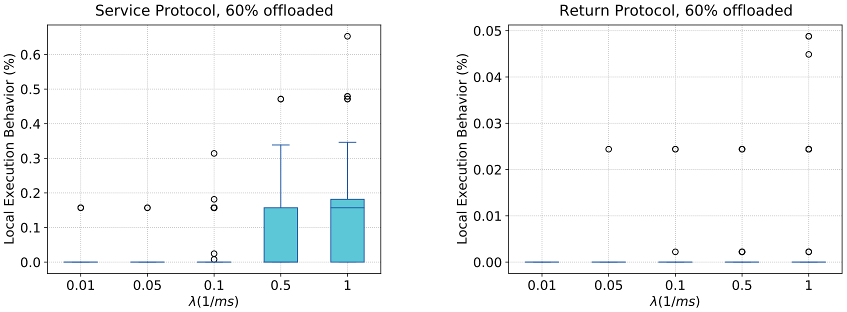


Fig. 8.8: The percentage of time the robot exhibits local execution behavior during the simulation for different probabilities of unsuccessful offloading operations and different percentages of offloaded workload under the service and the return protocol with 40 % and 60 % offloaded workload per task.

8.1.5 Probability-Based Timing Analysis

In this subsection, we present a multinomial-based approach to efficiently calculate the deadline miss probability. Additionally, three analytical approaches are presented, i.e., Chernoff bound, Hoeffding's inequality, and Bernstein's inequality.

8.1.5.1 System Model and Notation

We consider a given set of n independent periodic (or sporadic) tasks $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$ in a uniprocessor system. Each task τ_i releases an infinite number of task instances, called jobs, and is defined by a tuple $((C_{i,1}, \ldots, C_{i,h}), D_i, T_i)$, where D_i is the relative deadline of τ_i and T_i is its minimum interarrival time. In addition, each task has a set of h distinct execution modes $\mathcal{M}$ and each mode j with $j \in \{1, \ldots, h\}$ is associated with a different WCET $C_{i,j}$. We assume those execution modes to be ordered increasingly according to their WCETs, i.e., $C_{i,m} \leq C_{i,m+1}\ \forall m \in \{1, \ldots, h-1\}$. Furthermore, we assume that each job of τ_i is executed in one of those distinct execution modes. To fulfill its timing requirements in the j^{th} execution mode, a job of τ_i that is released at time t_a must be able to execute $C_{i,j}$ units of time before $t_a + D_i$. The next job of τ_i must be released at $t_a + T_i$ for a periodic task and for a sporadic task the next job is released at or after $t_a + T_i$. In this work, we focus on *implicit-deadline* task sets, i.e., $D_i = T_i$ for all tasks, and *constrained-deadline* task sets, i.e., $D_i \leq T_i$ for all tasks. We assume that a job execution is aborted as soon as the absolute deadline is reached, to ensure that there is no 'domino effect' to jeopardize the execution of the other jobs.

We assume a preemptive fixed-priority scheduling policy is used in the considered system. The tasks are indexed according to their priority, i.e., τ_1 has the highest and τ_n has the lowest priority. In addition, $hp(\tau_k)$ denotes the set of tasks with higher priority than τ_k and $hep(\tau_k)$ is $hp(\tau_k) \cup \{\tau_k\}$. $\mathbb{P}_i(j)$ denotes the probability that a job of task τ_i is executed in mode j with related WCET $C_{i,j}$ and we assume that each job is executed in exactly one of these distinct execution modes, i.e., $\sum_{j=1}^{h} \mathbb{P}_i(j) = 1$. In addition, we assume that these probabilities are independent from each other according to the following definition:

Definition 27 (Independent Random Variables). *Two random variables are (probabilistically) independent if the realization of one does not have any impact on the probability of the other.*

Particularly, for a newly arriving job the probability of the execution modes is independent from the execution mode of the jobs of previous jobs.

8.1.5.2 Definition of Deadline Miss Probability

To derive the probability of deadline misses, we look for the probability that the accumulated workload S_t over an interval of length t is at most t, where S_t can be calculated by the sum of random variables, i.e., the sum of probabilistic WCETs from all tasks $\tau_i \in hep(\tau_k)$ over. That is, the situation where S_t is larger than t for an interval of length t and hence $\mathbb{P}(S_t > t)$ is the overload probability at time t. To upper bound the probability that this test fails, the minimum probability among all time points at which the test could fail should be derived. Hence, the probability of a deadline miss Φ_k can be upper bounded by

$$\Phi_k = \min_{0<t\leq D_k} \mathbb{P}(S_t > t) \tag{8.1}$$

When analytical bounds are in use, we seek $\mathbb{P}(S_t \geq t)$ instead of $\mathbb{P}(S_t > t)$. By definition $\mathbb{P}(S_t \geq t) \geq \mathbb{P}(S_t > t)$, so these values can be used directly when looking for an upper bound of $\mathbb{P}(S_t > t)$.

8.1.5.3 A Multinomial-Based Approach

Conventionally, the probability of deadline misses can be derived from convolution-based approaches [476]. In such approaches, the underlying random variable represents the execution mode of each single job. This state space in fact can be transformed into an equivalent space that describes the states on a task-based level by proving the invariance when considering equivalence classes for each task. As a result, we introduce a novel approach that is based on the multinomial distribution. For the simplicity of presentation, we only highlight the insight of the aforementioned transformation.
The traditional convolution-based approach determines the *overload probability* by successively calculating the probability for all other points of interest in the analysis interval. However, the probability for t is evaluated based on the resulting states after all jobs in the analysis interval are convoluted. With respect to t, the intermediate states are not considered. By utilizing this insight, we can merge the states to efficiently calculate the vector representing the possible states at time t. If the number of jobs for a task is known, all possible combinations and the related probabilities can be calculated

directly using the multinomial distribution. The rationale is to construct a tree based on the tasks, which means that the number of children on each level depends on the number of jobs the related task releases.

8.1.5.4 Analytical Upper Bounds

In the following, we demonstrate how common concentration inequalities used in machine learning, statistics, and discrete-mathematics can be used to derive analytical bounds on $\mathbb{P}(S_t \geq t)$.

Chernoff Bound can be exploited to over-approximate the probability that a random variable exceeds a given value. This statement is summarized in the following lemma:

Lemma 28 (Lemma 1 from Chen and Chen [131]). *Suppose that S_t is the sum of the execution times of the $\rho_{k,t} + \sum_{\tau_i \in hp(\tau_k)} \rho_{i,t}$ jobs in $hep(\tau_k)$ at time t. In this case*

$$\mathbb{P}(S_t \geq t) \leq min_{s>0} \left(\frac{\prod_{\tau_i \in hep(\tau_k)} (mgf_i(s))^{\rho_{i,t}}}{\exp(s \cdot t)} \right) \tag{8.2}$$

It is in general pessimistic and there is no guarantee for the quality of the approximation. To find the optimal value of s to minimize the right-hand side in Equation 8.2, it has been proven as a log-convex optimization problem [129].

Hoeffding's Inequality derives the targeted probability that the sum of independent random variables exceeds a given value. For completeness, we present the original theorem here:

Theorem 29 (Theorem 2 from [319]). *Suppose that we are given M independent random variables, i.e., $X_1, X_2, \ldots, X_M$. Let $S = \sum_{i=1}^{M} X_i$, $\bar{X} = S/M$ and $\mu = \mathbb{E}[\bar{X}] = \mathbb{E}[S/M]$. If $a_i \leq X_i \leq b_i$, $i = 1, 2, \ldots, M$, then for $s > 0$,*

$$\mathbb{P}(\bar{X} - \mu \geq s) \leq \exp\left(-\frac{2M^2 s^2}{\sum_{i=1}^{M} (b_i - a_i)^2} \right) \tag{8.3}$$

Let $s^{'} = sM$, i.e, $s = s^{'}/M$. Hoeffding's Inequality can also be stated with respect to S:

$$\mathbb{P}(S - \mathbb{E}[S] \geq s^{'}) \leq \exp\left(-\frac{2s^{'2}}{\sum_{i=1}^{M} (b_i - a_i)^2} \right) \tag{8.4}$$

By adopting Theorem 29, we can derive the probability that the sum of the execution times of the jobs in $hep(\tau_k)$ from time 0 to time t is no less than t. The detailed proof can be found in [85].

Bernstein's Inequality generalizes the Chernoff bound and the related inequality by Hoeffding and Azuma. The original corollary is also stated here:

Theorem 30 (Corollary 7.31 from [232]). *Suppose that we are given L independent random variables, i.e., $X_1, X_2, \ldots, X_L$, each with zero mean, such that $|X_i| \leq K$ almost surely for*
$i = 1, 2, \ldots, L$ and some constant $K > 0$. Let $S = \sum_{i=1}^{L} X_i$. Furthermore, assume that $\mathbb{E}[X_i^2] \leq \theta_i^2$ for a constant $\theta_i > 0$. Then for $s > 0$,

$$\mathbb{P}(S \geq s) \leq \exp\left(-\frac{s^2/2}{\sum_{i=1}^{L} \theta_i^2 + Ks/3}\right) \tag{8.5}$$

The proof can be found in [232]. Note, however, that the result in [232] is stated for the two-sided inequality, i.e., as upper bound on $\mathbb{P}(|S| \geq s)$. Here, the one-sided result, which is a direct consequence of the proof in [232] (page 198), is tighter. Similarly, it can also be used to derive the probability of deadline misses. The detailed proof can also be found in [85].

Final Remark Considering the required runtime and the accuracy of different approaches, when a given task set needs to be analyzed, we suggest first running *Chernoff's*, *Hoeffding's*, and *Bernstein's* bounds. If a sufficiently low deadline miss probability cannot be guaranteed from these analytical bounds, we propose running the multinomial-based approach with equivalence class union in parallel on multiple machines by partitioning the time points equally.

8.1.6 Summary

In this section, we showed a novel resource-sharing protocol for multiprocessors, named DGA, that can serve a high utilization of critical sections while guaranteeing the given hard real-time constraints. In addition, we presented adaptive protocols for computation offloading that are able to countermeasure the unreliable connection. Unlike conventional analyses for hard real-time systems, our innovated convolution-based approach is able to efficiently derive safe upper bounds for the probability of deadline misses under soft real-time constraints.

8.2 Communication Architecture for Heterogeneous Hardware

Henning Funke
Jens Teubner

Abstract: In this section, we look at distributed processing on a smaller scale. Even a single-machine system today internally looks—and behaves—like a distributed system: multiple *processing modules* of different flavors (*e.g.*, CPUs, GPUs, FPGAs) interact with *memory modules*, which are scattered over the system, through an *interconnect* that is comprised of, say, QPI, PCIe, or "real" network links. In such environments, *communication* quickly becomes the limiting factor—not only for observable performance, but also for other system aspects, such as energy consumption.

We specifically look at communication patterns in heterogeneous CPU/GPU environments, and we illustrate how novel processing models can minimize communication overhead in such systems, which in turn results in significant performance improvements for real-world settings.

In GPU-accelerated, data-intensive systems, the PCIe link is often perceived as the limiting factor. Oftentimes, successions of fine-granular GPU kernel invocations amplify the problem, since they tend to cause multiple round-trips over the bandwidth-limited link. As we see in this section, unnecessary round-trips can be avoided by fusing fine-granular kernels into larger work units that can hide costly PCIe transfer times (query compilation can be a device to implement kernel fusion).

Eliminating the PCIe bottleneck, however, only exposes the GPU's *on-chip communication* as the new bottleneck to GPU-assisted data processing. Here, the data-parallel processing modes of graphics processors and the synchronization between parallel units are the cause for redundant round-trips over the precious on-chip communication interfaces. These bottlenecks can be avoided when processing models are deliberately designed to be *communication aware*. A specific example is a novel combination of pipelining/streaming processing models with the data-parallel nature of GPUs, which aligns particularly well with the semantics of database query execution. For real-world settings, this results in a reduction of memory access volumes by factors of up to 7.5× and shorter GPU kernel execution times by factors of up to 9.5×.

8.2.1 Introduction

Graphics Processing Units (GPUs) are frequently used as powerful accelerators for database query processing. As the arithmetic throughput of the coprocessor peaks in the teraflop range, it becomes a challenge to provision enough data. For this reason,

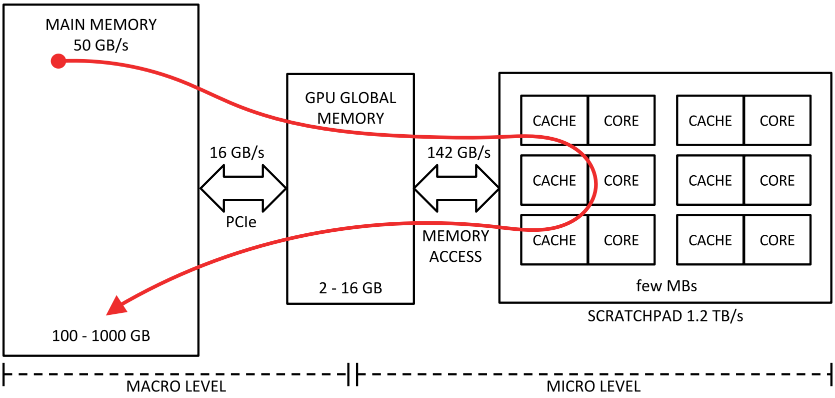


Fig. 8.9: The path of a tuple through the memory levels of a coprocessor environment.

hardware vendors equip graphics cards with high-bandwidth memory that has read and write rates of hundreds of GB/s. Still, memory intensive applications such as query processing fall behind regarding the cost of data movement for different reasons. Figure 8.9 shows the path of relational data through the hierarchical memory levels in a typical coprocessor system. Along the path, several bandwidth and capacity constraints need to be considered to achieve scalability and performance:

PCIe / OpenCAPI / NVLink A widely-acknowledged problem is the data transfer bottleneck between the host system and the coprocessor [270], typically via PCIe. Due to the coprocessor's limited memory capacity, data transfers are necessary *during* computations. With an order of magnitude between internal and external memory bandwidth, database developers are challenged with data locality-aware algorithms that efficiently use inter-processor communication. Recent technologies, i.e., OpenCAPI and NVLink, increase the bandwidth over PCIe, shifting the bottleneck toward GPU global memory.

GPU Global Memory The fine-grained data parallelism of a GPU typically requires that kernels perform additional passes over the data. Performing multiple passes, however, can significantly inflate memory loads and can cause a bandwidth bottleneck especially for random memory accesses.

Main Memory Integrated GPU-style coprocessors are a recent development to directly access the memory of the host CPU. Such an *Accelerated Processing Unit (APU)* allows the use of massively parallel processing without additional data transfers. However, the available memory bandwidth is lower than that of a dedicated GPU (30 GB/s vs. hundreds of GB/s).

Scratchpad Memory[1] Scratchpad memory is located on-chip and placed next to each compute unit of a GPU. It can be controlled as an explicit cache for low-level computations and offers a very high bandwidth. However, the capacity is limited to 16 kB–96 kB per core, which makes it challenging to use it for large-scale computations.

8.2.2 Contributions

With *HorseQC*, we developed a database query compiler that accounts for the hierarchical memory structure of modern coprocessor environments and for their inherent bandwidth limitations. In this section, we elaborate on the key building blocks of *HorseQC*, which can serve as a poster child in bandwidth-aware systems for other application contexts as well.

Specifically, we *(a) analyze the bandwidth limitations* in different database execution models; *(b)* demonstrate a *query compiler* for a coprocessor-accelerated database engine; *(c)* show how database sub-tasks can be realized in a *single pass over the data* (thus avoiding expensive memory round-trips); and *(d)* integrate these contributions in a *fully working system* that we use to evaluate our work.

Coprocessor-enabled database engines are typically classified by the *macro execution model* that they use to orchestrate the processing of query plans. Orthogonally, we devise a *micro execution model* that can be paired with different existing macro execution models, enhancing their communication- and resource-awareness.

8.2.3 Macro Execution Model

We begin by looking at macro execution models that have been employed in the past. To evaluate a relational query operator, state-of-the-art systems will select a number of primitives and execute the corresponding kernel sequence on the GPU. To feed the kernels with data, the macro execution model defines how data transfers will be interleaved with kernel executions. Here, the data movement from kernel to kernel may result in additional bandwidth demand compared with conventional systems. To understand the effect, we study the implications that existing macro execution models have on the use of bandwidth at multiple levels (PCIe, GPU global memory, etc.). We profiled the execution of Query 3.1 as a poster child from the star schema benchmark (SSB) [543]. The query was executed at scale factor 10 with CoGaDB [74] on a NVIDIA

[1] We use the term *scratchpad memory* to disambiguate *shared memory* for CUDA and *local memory* for OpenCL.

GTX970 GPU.[2] In the following, we discuss three macro execution models: *run-to-finish*, *kernel-at-a-time*, and *batch processing*.

Algorithm 8: Run-to-finish execution of two successive kernels.

```
RUN-TO-FINISH – input: R, output: P
move R Host → GPU
tmp ← op1(R)                          /* invoke first GPU kernel */
P ← op2(tmp)                         /* invoke second GPU kernel */
move P GPU → Host
```

8.2.3.1 Run-To-Finish (Not Scalable)

A straightforward way to execute a sequence of kernels is to first transfer all input, execute the kernels, and finally transfer all output. The approach, illustrated in Algorithm 8, has the advantage that intermediate data remains in GPU global memory in-between kernel executions and no significant PCIe transfers are necessary. However, run-to-finish has the disadvantage that it works only if *all* input, output, and intermediate data is small enough to fit in GPU memory. Run-to-finish macro execution models are used, e.g., by Ocelot [302], CoGaDB [74], and others. The *lack of scalability* leads us to evaluate the following execution models.

Algorithm 9: Kernel-at-a-time achieves scalability by transferring I/O for each kernel through PCIe.

```
KERNEL-AT-A-TIME – input: R, output: P
foreach $r_i$ in R=$r_1 \cup \cdots \cup r_m$ do
    move $r_i$ Host → GPU
    $m_i$ ← op1($r_i$)                      /* invoke first GPU kernel */
    move $m_i$ GPU → Host (assemble into M)
foreach $m_j$ in M=$m_1 \cup \cdots \cup$ m$_n$ do
    move $m_j$ Host → GPU
    $p_j$ ← op2($m_j$)                     /* invoke second GPU kernel */
    move $p_j$ GPU → Host (assemble into P)
```

8.2.3.2 Kernel-At-A-Time

To process large data on coprocessors, we can execute each kernel on blocks of data. The pseudocode of this approach is shown in Algorithm 9. Processing blocks of data requires algorithm choices that can deal with partitioned inputs. Joins or aggregations, for instance, can be processed in this mode only if their internal state (e.g. a hash table) can fit in GPU global memory.

[2] We measured 146.1 GB/s GPU global memory bandwidth in a host system with 16 GB/s bidirectional PCIe bandwidth.

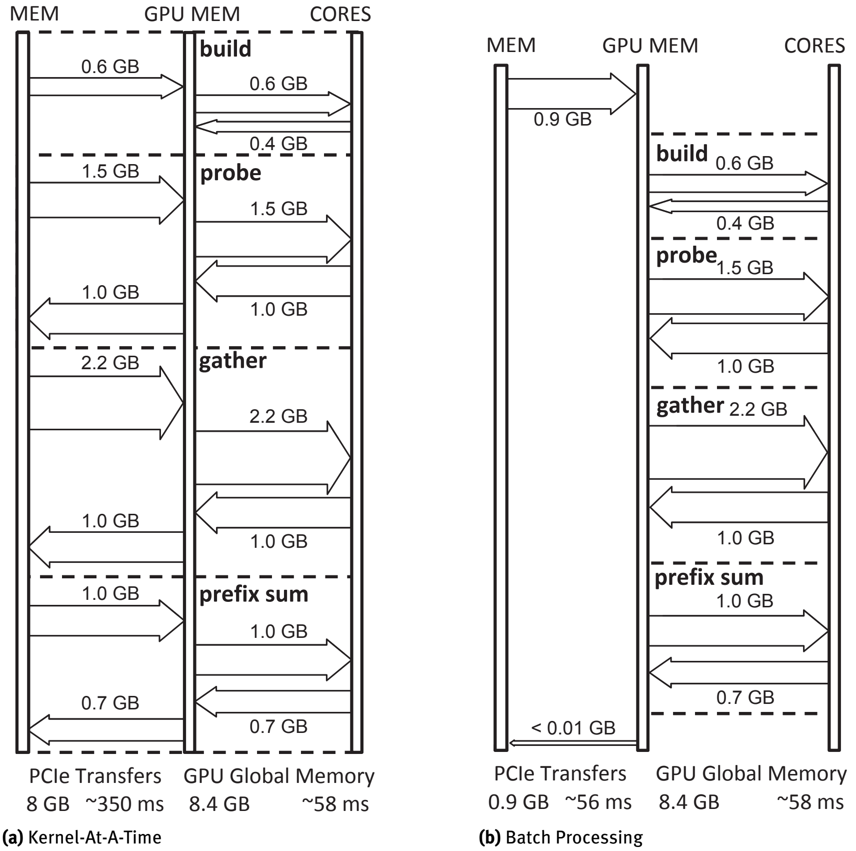


Fig. 8.10: Data movement for processing SSB Query 3.1. While the throughput of a is limited by PCIe transfers, b exposes GPU global memory access as the next bottleneck.

We analyze the data movement of kernel-at-a-time for SSB Query 3.1. Blocks are first moved via PCIe from the host to the coprocessor and then read by the kernel from GPU global memory (output passes both levels vice-versa). In this way, the data volumes for GPU global memory accesses equal the data volume transferred via PCIe, plus the cost to build up the hash tables in GPU global memory (0.4 GB here). Figure 8.10a shows the resulting data movement.

In the figure, the arrows annotated with data volumes represent PCIe transfers and GPU global memory accesses. We aggregated the data volumes by kernel types (e.g. scan, gather) and show only the most important kernels responsible for 88.2 % of the memory traffic. Given a PCIe bandwidth of 16 GB/s, all PCIe transfers together require at least 350 ms to complete. This exceeds the aggregate time for GPU global memory access by a factor of 5.8×. For kernel-at-a-time processing *the PCIe link is clearly the bottleneck*.

Kernel-at-a-time processing is used to scale out individual operators [358]. Unified Virtual Addressing (UVA) produces the same low-level access pattern, though it is transparent to the system developer.

8.2.3.3 Batch Processing

We can alleviate PCIe bandwidth limitations by rearranging the operations of kernel-at-a-time. Instead of running kernels until a column is processed, we can short-circuit the transfer of intermediate results to the host. Batch processing achieves this by reusing the output of the previous operation (op1) as input for the next operation (op2) instead of transferring to the host. This is applicable whenever intermediate batch results can be kept within GPU global memory. The corresponding pseudocode is shown in Algorithm 10.

Algorithm 10: Batch processing executes multiple kernels for each block that is transferred via PCIe.

```
BATCH PROCESSING – input: R, output: P
foreach $r_i$ in R=$r_1 \cup \cdots \cup r_m$ do
    move $r_i$ Host → GPU
    tmp$_i$ ← op1($r_i$)                 /* invoke first GPU kernel */
    $p_i$ ← op2(tmp$_i$)                 /* invoke second GPU kernel */
    move $p_i$ GPU → Host (assemble into P)
```

We analyze the data movement cost with the example of SSB Query 3.1. The GPU global memory load is the same as for kernel-at-a-time processing, because each kernel reads and writes I/O to GPU global memory. We obtain the PCIe transfer cost using the transfer volumes of the input columns of the query and the output of the final result. Figure 8.10b shows the resulting data movement cost. Batch processing reduces the amount of PCIe transfers by a factor of 8.8×. This shows that transferring data in blocks *and* performing multiple operators per block allows scalability and increases the efficiency compared to kernel-at-a-time.

Batch processing macro execution models have been used for coprocessing by GPUDB [728] and Hetero-DB [735]. Wu et al. [711] described the concept as *kernel fission* and identify opportunities to omit PCIe transfers automatically.

Limitations The lower amount of PCIe traffic can expose GPU global memory bandwidth as the next limitation. Batch processing reduces the PCIe transfer cost, but the amount of GPU global memory accesses remains unaffected. The memory access volume inside the device is now an order of magnitude larger. Despite the high bandwidth, this takes longer to process than the PCIe bus transfers (Figure 8.10b). For this reason, batch processing SSB Query 3.1 is *not* limited by PCIe transfers, but by accesses to the (high-speed) GPU global memory. Since in typical query plans, I/O and hashing operations both address the same GPU global memory, the situation may arise frequently in real-world workloads.

Tab. 8.2: Number of passes over the input data for benchmark queries. Out of 25 queries, 9 are definitely limited by GPU global memory.

Query	Passes	Query	Passes	Query	Passes
ssb11	7.5	ssb34	2.2	tpch5	7.2
ssb12	6.9	ssb41	7.4	tpch6	6.2
ssb13	6.7	ssb42	3.9	tpch7	**9.0**
ssb21	**9.6**	ssb43	3.5	tpch9	**9.0**
ssb22	**9.2**	tpch1	**15.5**	tpch10	5.8
ssb23	**9.1**	tpch2	**14.5**	tpch15	6.3
ssb31	**11.0**	tpch3	5.2	tpch18	**38.5**
ssb32	7.9	tpch4	6.6	tpch20	**10.5**
ssb33	7.5				

8.2.4 Micro Execution Model

Tuning the macro level helps to remove the main bottleneck for scalability: data transfers over PCIe. However, the macro level change exposes a new bottleneck: the memory bandwidth of GPU global memory. To utilize the GPU global memory bandwidth more efficiently, we need to apply additional micro-level optimizations using *micro execution models* and combine them with the macro execution model (batch processing) to achieve scalability *and* performance.

Existing micro-level optimizations such as *vector-at-a-time* processing [749] and *query compilation* [529] utilize memory bandwidth more efficiently by leveraging pipelining in on-chip processor caches. Therefore, both techniques are promising candidates for opening up the bottleneck of limited GPU global memory bandwidth. However, vector-at-a-time processing and query compilation are designed in the context of CPUs. While it is highly desirable to apply both techniques in the context of GPUs, mapping the techniques from CPU to GPU is challenging, as we discuss below.

Vector-At-A-Time To mediate the interpretation overhead of Volcano and the materialization overhead of operator-at-a-time, vector-at-a-time uses batches that fit in the processor caches. First, this reduces the number of `getNext()` calls from one per tuple to one per batch. Second, this makes materialization cheap because operators pick up the cached results of previous operators. On CPUs, vector-at-a-time benefits from batch sizes that are large enough to limit the function call overhead and small enough to fit in the CPU caches.

On GPUs, the compromise between tuple-at-a-time and full materialization strategies is not a sweet spot, however. Kernel invocations are an order of magnitude more expensive than CPU function calls. Furthermore, GPUs need much larger batch sizes to facilitate over-subscription and out-of-order execution. This leads to the problem that batches, which fit in the GPU caches, are too small to be processed efficiently. Alternatively, more recent GPUs support *pipes* to move a local execution context from one kernel to another. This has been used by GPL [557] for query processing. However, this technique still introduces an overhead for switching the execution context. In addition, it is limited to a depth of 2–32 kernels depending on the microarchitecture.

Query Compilation Query compilation is a common tool for avoiding excessive memory transfers during query processing. Compiling code for incoming queries becomes feasible with low-level code generation and achieves performance close to hand-written code. The compilation strategy of Neumann [529] keeps intermediate results in CPU registers and passes data between operators without accessing memory at all. The generated code processes full relations or blocks of tuples using a sequential tight loop.

To use query compilation on GPUs, we must integrate fine-grained data parallelism into compiled queries. The parallelization strategy of HyPer [425], however, uses a coarse-grained approach, so that it does not break with the concept of tight loops. In fact, HyPer does not use SIMD instructions [529] and thus omits fine-grained data parallelism. Even on CPUs with a moderate degree of parallelism in SIMD instructions, database researches are challenged by integrating query compilation and SIMD instructions [487, 639].

In summary, using a micro-level technique for efficient on-chip pipelining on GPUs remains a challenge. Applying any of the commonplace techniques makes it necessary to combine at least three things that are hardly compatible: fine-grained data-parallel processing, extensive out-of-order execution, and deep operator pipelines. To achieve our goal of mitigating the GPU global memory bottleneck, we need to develop a new micro execution model.

8.2.5 Data-Parallel Query Compilation

In the following, we show a micro-level execution strategy that reduces GPU global memory access volumes by means of pipelining in on-chip memory. To this end, we show the approach of our query compiler *HorseQC* and its integration with the operator-at-a-time execution engine of CoGaDB [74].

8.2.5.1 Fusion Operators

HorseQC extends the operator-at-a-time approach with the concept of *fusion operators*, operators that embrace multiple relational operations. A fusion operator replaces a

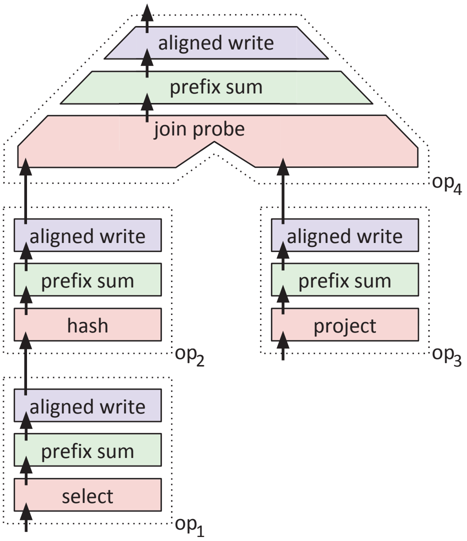


Fig. 8.11: Operator-at-a-time.

sequence of conventional operators in the physical execution plan with a micro-level-optimized pipeline. The data movement within a fusion operator can be improved by applying different micro level execution models.

8.2.5.2 Micro-Level Pipeline Layout

To keep matters simple, we first apply query compilation with the operator-at-a-time primitives described by He et al. [300]. This choice is not limiting as other data-parallel primitives may be used instead. However, a commonality of different primitive sets is that they use *relational primitives* with relational functionality (e.g. select) and *threading primitives* with thread coordination functionality (e.g., map, prefix sum, gather).

State Of The Art We look at a query with two input tables and a total of four relational operators $\mathtt{op}_1, \cdots, \mathtt{op}_4$. Operator-at-a-time runs three primitives per operator (cf. Figure 8.11 on the right): The first pass executes the relational primitive (e.g., select, project) and counts the number of outputs of each thread. The second pass computes a *prefix sum* to obtain unique per-thread write positions. The third pass performs an *aligned write*. This means that the output values are written into a dense array and may include executing the relational primitive for a second time to produce the output values. Thus, the query is processed in twelve operations with separate GPU global memory I/O.

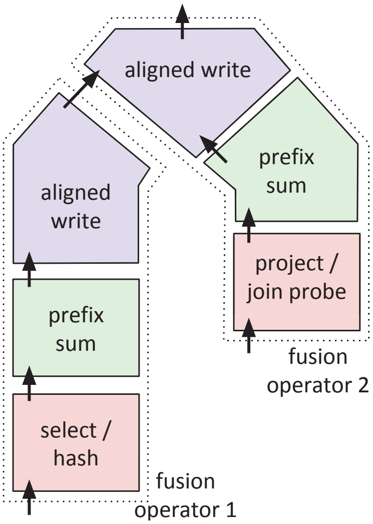


Fig. 8.12: Multi-pass QC.

Multi-Pass Query Compilation By grouping operations that are applied to the same input table, the query may be processed with two fusion operators. Within each fusion operator, we apply the following query compilation strategy (cf. Figure 8.12): We extract the prefix sum from the operators and execute it only once between all relational primitives and all aligned writes. The relational primitives are then compiled into one kernel called `count`, which is executed before the prefix sum. The aligned writes are compiled into one kernel called `write`, which is executed after the prefix sum. In this way, we apply *kernel fusion* [689] to the four relational primitives and to the four aligned writes. The same query is processed with six operations and the operations in compiled kernels communicate through on-chip memory instead of GPU global memory.

8.2.6 Memory Access and Limitations

In Figure 8.13, we illustrate the bandwidth characteristics of our example query when using code generation with three phases. The figure shows the behavior of the three-phase micro execution model described above with the batch processing macro execution model. To analyze the implications of forwarding intermediate results in the generated kernels through registers and scratchpad memory, we extended the illustration with an additional GPU-internal layer of memory.

GPU global memory access has previously been the bottleneck for query execution. Here the `count` kernel accesses 1.7 GB in GPU global memory, the prefix sum computation accesses 0.8 GB in GPU global memory, and the `write` kernel accesses 1.9 GB in GPU global memory. This is a reduction by a factor of 1.9× compared with batch

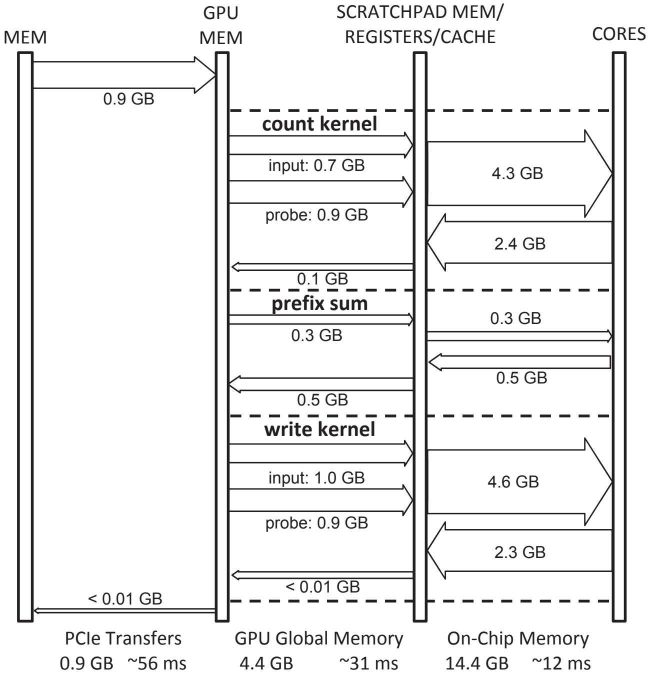


Fig. 8.13: Data movement for data-parallel query compilation with three phases.

processing. In the generated kernels, a substantial amount of memory traffic has moved to on-chip memory. In on-chip memory, the access volume of 14.4 GB is not a limiting factor due to the extremely high bandwidth of 1.2 TB/s of scratchpad memory.

Although the reduced GPU global memory traffic may suggest that the approach eliminates the bottleneck, real-world queries still experience limitations. In fact, Section 8.2.10.6 shows that compilation with three phases can still not saturate PCIe for 9 out of 12 SSB queries. This is because the query complexity prevents the strategy from utilizing the full GPU global memory bandwidth. Therefore, we investigate ways to further increase the processing efficiency in the next section.

8.2.7 Processing Pipelines in One Pass

The previous execution model relied on a typical programming concept of GPUs that executes operations with multiple kernels. The kernels that execute the actual work for the operations are interleaved with kernels that execute prefix sum computations. To further improve the processing efficiency, we have to break with this concept. With a new micro execution model, we avoid round trips to GPU global memory, which are caused by multi-pass implementations. This enables us to radically reduce GPU global memory traffic and lift the bandwidth bottleneck.

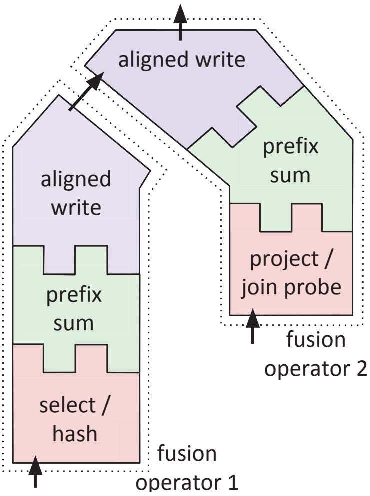


Fig. 8.14: Compound kernel.

Compound Kernel Kernel fusion brought reduction operations (e.g. prefix sum) as boundaries into the spotlight. Previously, we computed the prefix sum *between* two generated kernels to obtain write positions. Instead of two separate kernels, we now generate only one *compound kernel* that integrates the prefix sum computation (cf. Figure 8.14), which eliminates multiple passes. Computing write positions *within* a generated kernel makes it possible to process pipelines in one pass without intermediate materialization. In this way, each fusion operator is executed by a single compound kernel. In the following, we look at implementation strategies for reduction operations that enable fully pipelined processing.

Atomic Prefix Sum The separation into multiple reduction kernels with intermediate materialization impedes pipelining. To introduce a pipelined implementation, let us first look at a very simple sequential prefix sum:

```
for(i=0; i<n; i++)
  if(flags[i]) prefix_sum[i] = sum++;
```

The sequential prefix sum loops through the array `flags` while writing *and* incrementing `sum` for every valid entry. Figure 8.15a illustrates the use of the prefix sum for a dense write of selected input elements. When parallelizing the `for`-loop, this implementation runs into the issue of many threads trying to increment `sum` at the same time. To resolve this parallel dependency, atomic operations can be used to isolate parallel modifications of the same memory address. Atomic operations ensure a consistent state, yet are executed in an undefined order. The following code executes an *atomic prefix sum* to compute unordered, dense write positions:

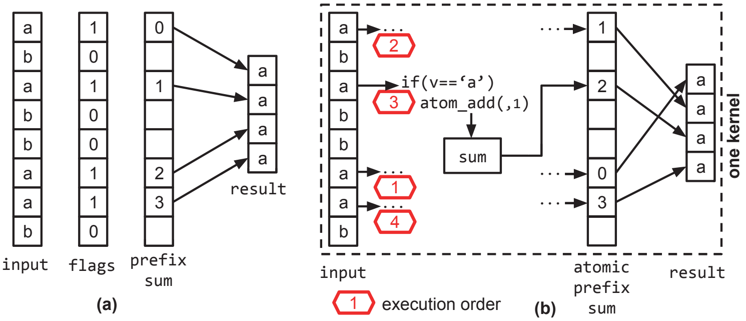


Fig. 8.15: The computation of a prefix sum for writing selected elements to a dense array (a) can be parallelized using atomic operations (b).

```
if(is_selected) wp = atom_add(&sum, 1);
```

Threads contribute an offset of 1 to the sum at address &sum by executing the expression conditionally. Each atomic_add(..) returns the previous state of sum. Thus, threads immediately obtain a unique global write offset as wp in register. This is illustrated in Figure 8.15b.

The use of atomic operations causes a break with the semantic of the prefix sum because the result has *no defined order*. For the relational semantic, however, only the *uniqueness* of output positions is critical. Output permutations lead to non-aligned GPU global memory access where adjacent threads do not write to adjacent memory addresses. The impact on write throughput, however, is limited, because the filter semantics lead to non-aligned access for separate prefix sums as well.

8.2.7.1 Memory Access and Limitations

The compound kernel micro execution model further reduces GPU global memory access by a factor of 2.4× to 1.8 GB (see Figures 8.13 and 8.16). Compared with operator-at-a-time, this is a reduction by a factor of 4.7×. Pipelining the prefix sum avoids round trips to GPU global memory that are necessary in the three-phase micro execution model. The compound kernel has only a minimal GPU global memory access volume for input, output, and hash-table access. Now the on-chip traffic is balanced with the GPU global memory traffic when relating each memory volume to the available bandwidth.

The described approach heavily relies on atomic operations. This has the disadvantage of causing limitations for parallelism. Although the execution order is undefined, the operations *are* sequentialized and reducing n values takes $O(n)$ parallel steps. However, Egielski et al. [195] showed that recent hardware support makes atomic operations competitive with parallel algorithms. Still, the integrated prefix sum puts significant pressure on the atomic functional units, which prevents pipeline kernels from utilizing

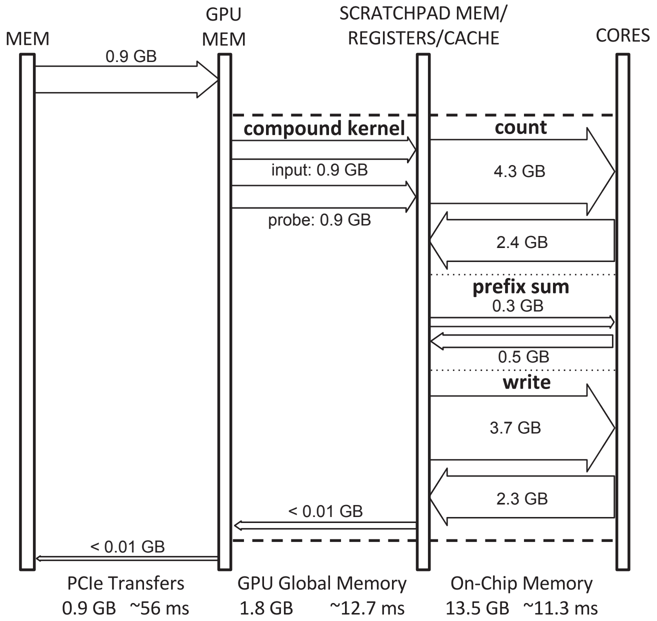


Fig. 8.16: Data movement for query compilation with one pass. The compound kernel reduces data movement by 4.7×.

full GPU global memory bandwidth. In the following, we address this issue and show how the efficiency of parallel reductions in compound kernels can be increased.

8.2.8 Efficient Pipelined Reductions

We have showed a way to pipeline reductions in generated kernels using atomic operations. This benefits the memory efficiency, but also reveals the atomic functional units of a GPU to be a bottleneck. This is especially critical because several operations that are combined in the compound kernel rely on atomic isolation as well. Specifically, the state-of-the-art implementations of hash joins and hash aggregations [358] use atomic operations to isolate hash table inserts.

This section addresses performance bottlenecks that occur when utilizing atomic reductions to pipeline relational operators. We show a new technique called *local resolution, global propagation*, that is used by *HorseQC* to pipeline prefix sums, single tuple aggregation, and grouped aggregation efficiently. The approach reduces the pressure on atomic functional units and offers tunability regarding hardware and thread-group granularity. We describe the approach in the following.

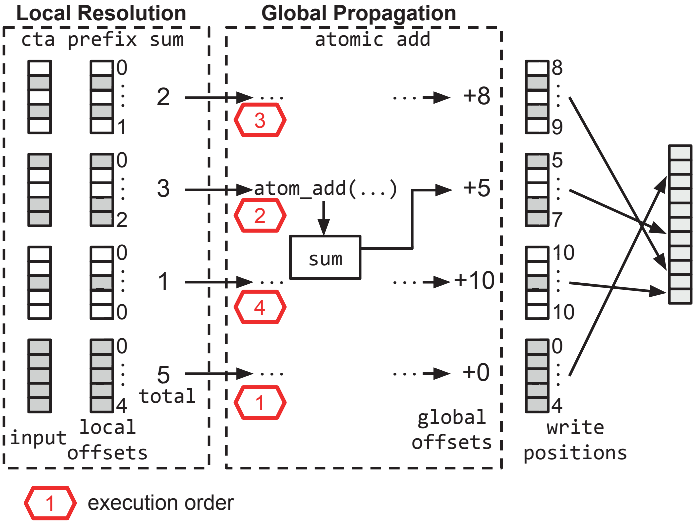


Fig. 8.17: Computing write positions with local resolution (local offset), global propagation (global offset).

8.2.8.1 Local Resolution, Global Propagation

Like other efficient GPU implementations such as in CUB [489], local resolution with global propagation consists of two levels of reductions. In contrast to other techniques, however, it always uses pipelined techniques on *both* levels. Local resolution is an additional pre-reduction step, computed by a local thread group, whereas global propagation is the same atomic reduction as described in Section 8.2.7. We use the term *Collaborative Thread Array* (CTA) for the thread groups in local resolution. CTAs can either match the workgroup (AMD) or thread-block (NVIDIA) size of the GPU kernel or work on a finer granularity.

The following code, illustrated in Figure 8.17, executes an atomic prefix sum using local resolution, global propagation:

```
l_os = cta_prfx,(flags, &cta_total);  //local res.
if,(cta_thread_idx == 0)
  g_os = atom_add,(&sum, cta_total);  //global prop.
wp = l_os + g_os;
```

First, each CTA executes `cta_prfx` to compute a local prefix sum on `flags`. This is the local resolution step. We implement `cta_prfx` with SIMD reductions (cf. Intra-Warp Scan Algorithm by Sengupta et al. [622]). The function returns the local offset `l_os` and the sum of all flags assigned to the CTA `cta_total`. Second, one thread of each CTA adds `cta_total` atomically to a global counter `sum`. This is the global propagation step.

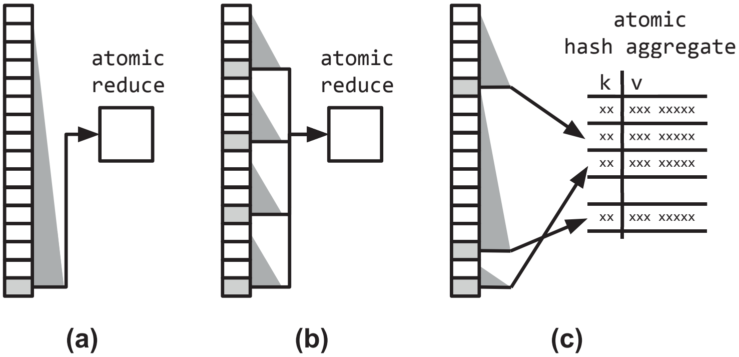


Fig. 8.18: Local resolution mechanisms: (a) Work-efficient reduction (b) SIMD reduction (c) segmented reduction.

The call to `atom_add` returns the global offsets `g_os`. Finally, the write position `wp` is the sum of `l_os` and `g_os`.

Compared with the simple atomic prefix sum, we now add pre-aggregates instead of 1/0 flags to `sum`. Accordingly, each atomic add obtains ranges of output indices instead of a single index. The process is analogous to *allocating* segments of output memory to CTAs. The order of the allocations is undefined, however. (See the execution order in Figure 8.17.) This leads to an output that is ordered *within segments* and permuted *between segments*. Further investigation reveals that, due to the GPUs stream processing engine, the permutations exhibit locality, leading to semi-ordered output data.

Local Resolution Mechanisms The mechanisms used for local resolution are interchangeable. This makes it possible to tune pipelined reductions and to apply them in different operations. Figure 8.18a and 8.18b show the integration of work-efficient reductions [56] and SIMD reductions [622]. Both techniques have different thread group granularities and we can choose between them to adapt to the hardware parallelism of different processors. Figure 8.18c shows the use of pipelined segmented reductions for grouping. First, segmented reductions compute pre-aggregates in scratchpad memory. Second, global propagation inserts the pre-aggregates into a hash table with an atomic operation. The ability to control scratchpad memory opens up a new design space for grouping algorithms in pipelined computations (e.g. handling frequent items). A similar approach PLAT [722] aggregates frequent grouping keys in a table local to each CPU core.

8.2.9 DBMS Integration

We integrated our query compiler *HorseQC* into the open source DBMS CoGaDB, leveraging the built-in code generator *Hawk* [75]. The DBMS uses a columnar data layout and processes full columns operator-at-a-time on GPUs and CPUs. We use the front-end and the storage layer of CoGaDB; *HorseQC* adds a compiler-based execution engine.

We added two components to the DBMS: 1. a query compiler that compiles fusion operators to GPU code (cf. Section 8.2.4); and 2. a *translation layer* that identifies fusion operators and drives the query compiler. Currently, there are two different workflows for the translation layer:

1. CoGaDB parses the SQL code for a query and generates a query plan. The translation layer applies the produce/consume model [529] to the query plan to determine fusion operators. We use this approach for the SSB queries and TPC-H Q6.
2. The translation layer parses a JSON file that describes the query plan including the fusion operators. This enables us to process queries when (1) cannot handle the queries via SQL (e.g. correlated subqueries or automatic unnesting). This is used for the other TPC-H queries.

When the fusion operators are defined, the translation layer drives the query compiler to compile and execute. Finally, the decompression of dictionary compressed columns and sorting are executed by CoGaDB's original execution engine.

8.2.10 Evaluation

Section 8.2.3.1 showed that query coprocessing in existing macro execution models is sensitive to memory bandwidth bottlenecks on various hierarchical levels. We proposed several micro execution models that allow to remove memory indirections to achieve a more efficient use of bandwidth. In this section, we evaluate our approaches and carefully assess bandwidth and throughput in identifying several benefits.

The experimental study is structured as follows. First, we evaluate the *micro execution models* and we execute specific queries to analyze the *reduction performance* of the proposed techniques in Experiments 1 and 2. Then, we evaluate the micro execution models for the SSB and TPC-H benchmarks in Experiments 3 and 4. Next, we analyze the *integration* of our micro execution model with the batch processing *macro execution model*. In doing so, we analyze the *real-world benefits* of our approach with a comparison of end-to-end performance in Experiment 5 and a scalability analysis in Experiment 6. Note that all experiments, except for Experiment 6, were executed with scale factor 10.

8.2.10.1 Processing Techniques

This section describes three micro execution models built into *HorseQC*. The goal is to use them within macro execution models to improve performance. Therefore, it is crucial to achieve a higher throughput than PCIe when executing queries. We show the benefit of our approaches by comparing them with an operator-at-a-time micro

Tab. 8.3: Coprocessors used in the evaluation.

Model	Type	Architecture	Cores	Scratch pad (KB)	B/W (GB/s)
GTX970 (NV)	GPU	Maxwell	13	96	146.1
GTX770 (NV)	GPU	Kepler	8	48	167.6
RX480 (AMD)	GPU	Ellesmere	32	32	104.9
A10 (AMD)	APU	Godavari	8	32	18.7

execution model. In this way, we analyze the benefit of moving data transfers between relational operators to the on-chip level.

Multi-pass The first approach separates reductions from the generated kernels, which leads to an execution in multiple passes (Section 8.2.5). Each reduction is executed on materialized data using the `boost::compute` library.

Pipelined The second approach integrates reductions into a fully pipelined kernel using atomic operations (Section 8.2.7). By using atomic operations for each reduction input, the approach is an instance of local resolution, global propagation that has no local resolution step.

Resolution The third approach increases the efficiency of pipelined reductions with local resolution methods such as pre-aggregation (Section 8.2.8). We differentiate between local resolution implementations using Resolution:SIMD for SIMD reductions and Resolution:WE for work-efficient reductions.

Operator-at-a-time We use CoGaDB 0.4.1, which processes full columns of data in each operator with CUDA kernels. It features a run-to-finish macro execution model and an operator-at-a-time micro execution model.

8.2.10.2 Baselines

PCIE transfer The *PCIe transfer time* is the time it takes to transfer input and output data between the host's main-memory and GPU global memory. It is the target time used by micro execution models for balancing throughput and PCIe bandwidth. The PCIe transfer time is shown in each graph with a dashed line (- - -).

Memory bound The *GPU global memory bound execution time* is the time it takes to access the data. As each approach has to read the input columns and write the output columns, the baseline is a lower bound on the kernel execution time. We indicate it with a solid line (——) in each graph.

Listing 8.1: Query 1 is a simple selection and projection query inspired by the star schema benchmark.

```
SELECT lo_extprice * lo_discount + lo_tax AS revenue
FROM   lineorder
WHERE  lo_quantity BETWEEN 25 - x AND 25 + x
```

8.2.10.3 System Configuration

For the experiments, we use three dedicated GPUs with PCIe gen 3.0 links and one APU that accesses main-memory directly. Table 8.3 specifies the GPU models and shows hardware properties. The amount of scratchpad is available *per core*. The reported bandwidth refers to GPU global memory for the GPUs and to main-memory for the APU. It was measured using on-GPU `memcpy` of 1 GB data. We measured bidirectional PCIe transfers between CPU and GPU as 12.1 GB/s.
Both NVIDIA GPUs GTX770 and GTX970 run in a system with an Intel Xeon E5-1607 CPU. We use the NVIDIA 364.19 driver and CUDA Toolkit 7.5 with OpenCL drivers. The AMD RX480 GPU is placed in a separate system with the A10-7890K APU. We use the AMDGPU-Pro 16.40 driver for the GPU and the fglrx 15.201 driver for the APU. Each system is running Ubuntu 14.04 and uses the boost library 1.61.

We used the profiling tools `nvprof 2.0.28` for NVIDIA hardware and `CodeXLGpuProfiler V4.0.511` for AMD hardware to measure kernel execution times, PCIe transfers, and GPU global memory access. For the measurements of kernel execution times, we used both tools to profile individual kernels and sum up the kernel execution times when multiple kernels were involved.

8.2.10.4 Experiment 1: Pipelined Prefix Sum

We compare several pipelined prefix sum techniques with one non-pipelined technique for a query that filters and projects one table. This allows us to analyze the benefit of integrating prefix sum computations into single-pass kernels. We execute Query 1, shown in Listing 8.1, and vary the selectivity in the range [0, 1] using `x`. By running the experiment on four GPUs, we aim to assess the best local resolution mechanisms for a given hardware. Figure 8.19 shows the results.

Observations Pipelined techniques perform better than Multi-pass in most cases. Integrating the prefix sum computation into single-pass kernels reduces the kernel execution times by a factor of up to 6.3×. While processing with Multi-pass takes up to 328.6 % of the PCIe time, Resolution:SIMD uses only 101.3 % of the PCIe time in the worst case (selectivity 1.0, RX480). This shows that the approach can saturate the bus bandwidth for a variety of configurations. On the A10 there are no PCIe transfers and Resolution:SIMD increases the overall throughput by factors of up to 1.6× over Multi-pass.

The results show that the local resolution step reduces the performance impact of atomic operations. This becomes visible for higher selectivity factors. Pipelined has higher executions times because the strategy executes one atomic addition per output. However, Resolution:SIMD and Resolution:WE show good performance across all selectivities due to local resolution.

Resolution:SIMD achieves the shortest kernel execution times in most cases and allows memory bound processing on the GTX970. On the GTX770, lowering the output

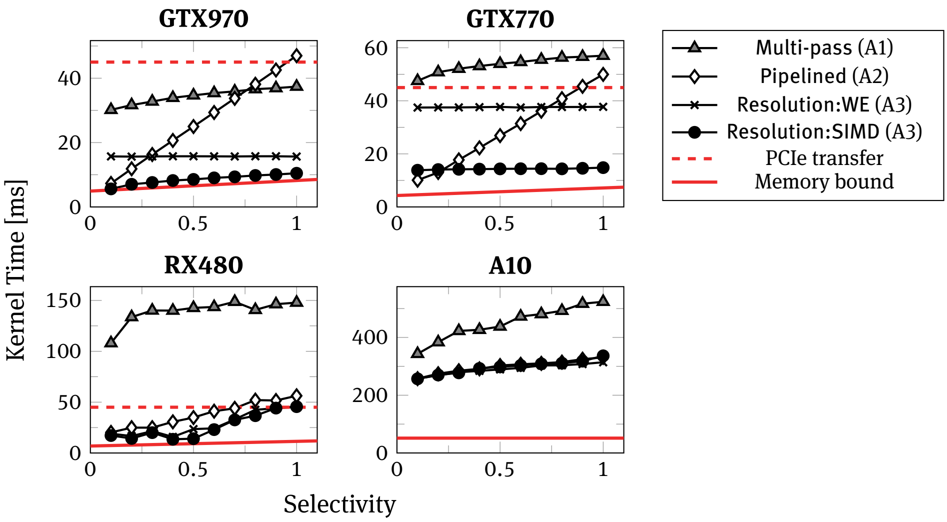


Fig. 8.19: Projection query executed with different approaches. Integrating prefix sums into kernels allows fastest execution.

size down to 0 does not affect the execution time. We conclude that the GTX770 is compute-bound earlier than the GTX970. The higher memory bandwidth of the GTX770 leads to an increased throughput for atomic operations and Pipelined can outperform Resolution:SIMD for selectivities below 10 %. On the RX480 and on the A10 there is no definite advantage for one of the reduction techniques. In the following, we use only Resolution:SIMD and skip the other techniques for a clear presentation.

8.2.10.5 Experiment 2: Pipelined GROUP BY

We evaluate the effect of pipelined GROUP BY aggregations using Operator-at-a-time, Pipelined, and Resolution. The query groups all tuples of `lineorder` according to the computed attribute `lo_orderkey%x` into sums. We vary the number of groups by increasing x from 2 to 16 384. We show the results of the experiment on a GTX970 GPU in Figure 8.20.

Observations The execution times of Operator-at-a-time do not depend on the group size. The main cost factor is sorting the input columns. Pipelined shows up to 11.1× lower execution times but only for larger group sizes. For group sizes below 64, we observe high execution times. This is caused by the heavy contention of parallel aggregation hash-table inserts.

The bottleneck is resolved by Resolution which uses pre-aggregations to reduce the contention. The results show that execution times reduce by a factor of up to 126×. However, the local pre-aggregations have a limited effect on larger group numbers. This

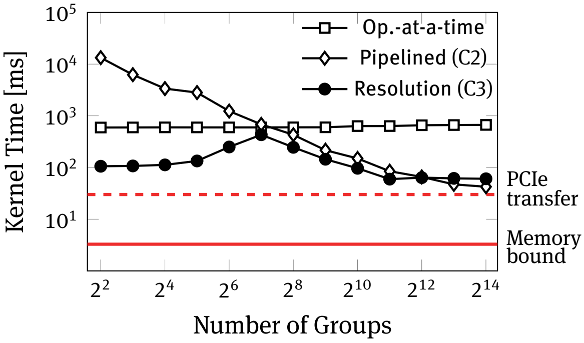


Fig. 8.20: Performance of grouped aggregations.

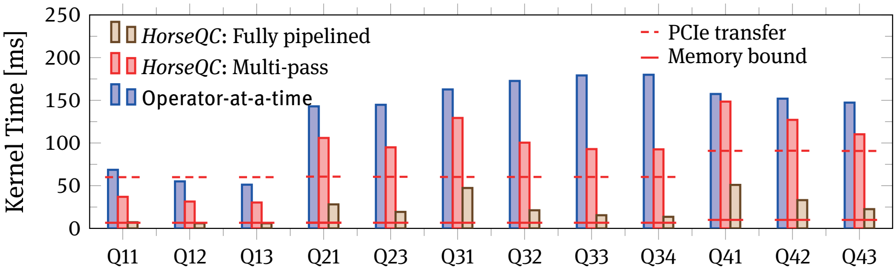


Fig. 8.21: Performance of SSB queries.

explains the spike at 128 groups, where both pre-aggregation and contention have an effect. While the approaches cannot saturate PCIe when aggregating a full table, filters reduce the cost of grouping for real-world queries.

8.2.10.6 Experiment 3: Star Schema Benchmark

The previous experiments showed that pipelining specific reduction operations helps to increase the throughput of query processing. In this experiment, we analyze whether this behavior carries over to real-world situations. To this end, we execute the SSB Queries[3] on the GTX970 GPU.

We use Operator-at-a-time and two variants of our query compiler. *HorseQC*: Multi-pass uses pipeline breaking implementations for reductions (A1, B1 and C1). *HorseQC*: Fully pipelined integrates all pipeline operations in one kernel (using A3, B3 and C2). We show the results of the experiment in Figure 8.21.

[3] We could not process SSB Query 2.2 as we do not yet support range predicates on dictionary compressed columns.

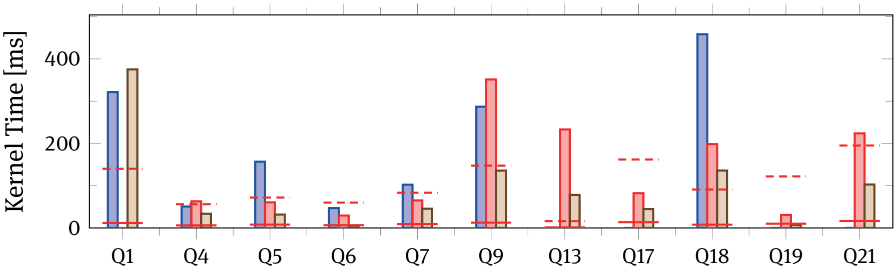


Fig. 8.22: Performance of TPC-H queries.

Observations The bandwidth analysis in Section 8.2.3.1 showed that 4 out of 12 queries are limited by GPU global memory access in operator-at-a-time processing.

- The kernel execution times of Operator-at-a-time show that compute and latencies make the problem worse. While PCIe would allow execution times between 60.6 ms to 90.9 ms, the kernel execution times take longer for 10 out 12 queries with up to 295.5 %.
- *HorseQC*: Multi-pass improves over Operator-at-a-time and uses only 50.5 % of the PCIe bandwidth transfer time in the best case and 215.5 % in the worst case. This shows that without efficient pipelining of reduction operations, the benefit of query compilation is limited.
- *HorseQC*: Fully pipelined lowers all kernel execution times to a level that is consistently lower than PCIe transfer times. This shows that compiling pipelines into one kernel with local resolution, global propagation provides an execution approach with sufficient throughput. Processing takes 9.7 % of the PCIe transfer time in the best case and 78.1 % in the worst case. For Queries 1.1, 1.2, and 1.3, kernel execution is memory bound by GPU global memory access.

8.2.10.7 Experiment 4: TPC-H Queries

We execute and profile queries from the TPC-H benchmark to show the effect when relaxing the specific assumptions of the star schema benchmark (e.g. using one centralized table). We select a subset of queries based on the work by Boncz et al. [61] to capture challenging aspects of the TPC-H benchmark: Q1, Q4, Q13, and Q21 contain heavy aggregation; Q9 and Q18 contain heavy joins; and Q4, Q19, and Q21 contain parallelism bottlenecks. We modified 4 queries, because *HorseQC* currently does not support all operations, e.g., `like` expressions. The results of the experiment are shown in Figure 8.22. For Q1, there is no result for *HorseQC*: Multi-pass, because the strategy ran out of GPU memory. The results shown for Operator-at-a-time are for all TPC-H queries supported by the DBMS.

Observations The PCIe and memory-bound baselines show larger variations than for the SSB benchmark. This is mainly caused by the join structure, e.g., Q13 joins three small tables, while Q17, Q18, and Q21 join multiple instances of the largest `lineitem` table.
The kernel execution times show that *HorseQC* can improve over operator-at-a-time by factors of up to 8.6×. For Q1, Q4, and Q9, there are cases where Operator-at-a-time has shorter kernel execution times than compiled strategies. Further investigation showed that in these cases Operator-at-a-time moves some operators to the CPU, which means that the measurements cover a limited amount of operations.

Comparing the variants of the query compiler, we observe that *HorseQC*: Fully pipelined consistently improves over *HorseQC*: Multi-pass by a factor of up to 5.4×. *HorseQC*: Fully pipelined achieves lower execution times than PCIe transfer times for 8 out of 11 queries. For Q1, Q13, and Q18, the PCIe bandwidth cannot be fully saturated. This is because the queries contain grouped aggregations of unfiltered columns (cf. Experiment 2). The execution times of *HorseQC*: Fully pipelined take 5.6 % of the PCIe transfer time in the best case and 268.1 % in the worst case.

8.2.10.8 Experiment 5: Scalability

Due to the deeply integrated storage layer implementations of the host DBMS CoGaDB, we were unable to build a fully scalable version of *HorseQC*. For this reason, we perform a separate experiment that integrates the Resolution micro execution model with the batch processing macro execution model for the star join from SSB Query 3.1. Decoupling this experiment allows us to apply the rules for coprocessor data management by Yuan et al. [728] and to measure end-to-end performance for larger datasets.

The star join recombines three dimension tables and one fact table with an overall selectivity of 3.4 %. We build hash tables for the dimension tables in GPU global memory. The fact table resides in pinned host memory and each column is partitioned into blocks of 0.5 MB, 2 MB, or 8 MB. The blocks are transferred asynchronously via PCIe into an inner kernel that computes the star join by probing each dimension hash table.

Figure 8.23 shows the end-to-end execution times for each block size when executing the experiment. We observe that execution times grow linearly with increasing scale factors and that block sizes larger than 2 MB can saturate the PCIe bandwidth. The computation does not become a bottleneck for the examined scale factors. With a block size of 4 MB and scale factor 300, the size of intermediate data in GPU global memory is only 473 MB. Therefore, we expect the approach to scale to even larger databases with linear performance.

8.2.10.9 Experiment 6: End-to-End Performance

To make a comparison with other database systems, we execute the TPC-H queries with different database systems and measure end-to-end performance. We compare MonetDB5 Dec2016-SP3 executed on CPUs, and CoGaDB 0.41 and *HorseQC* executed

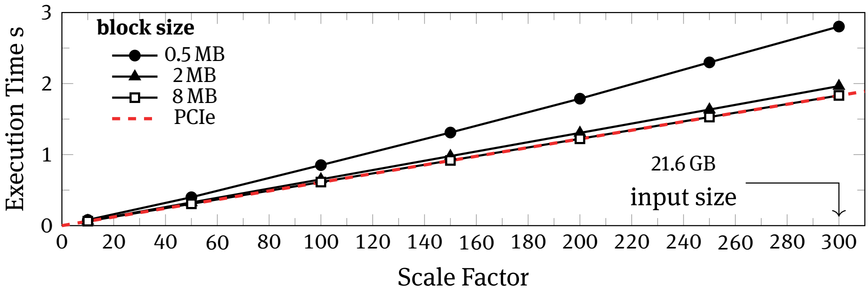


Fig. 8.23: End-to-end performance of star join computation for different scale factors.

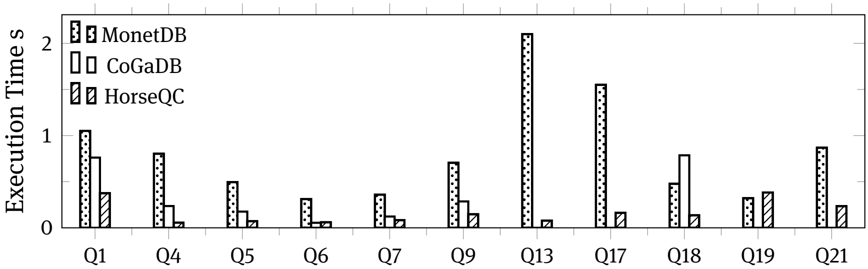


Fig. 8.24: End-to-end performance of TPC-H queries.

on GPUs. Both competitors feature an operator-at-a-time approach. We perform the measurements with warm caches. MonetDB runs on a workstation-class system with an Intel Xeon E5-1607 CPU and 32 GB RAM. CoGaDB and *HorseQC* run on the GTX970. The results are shown in Figure 8.24.

Observations For the supported queries, *HorseQC* is up to 5.8× faster than CoGaDB. While CoGaDB uses GPU global memory as a cache for frequently used columns, *HorseQC* does not cache data between queries. This shows that *HorseQC* uses memory and interconnects more efficiently. For Q6 there is no improvement, because query execution is PCIe bound.

HorseQC has lower execution times than MonetDB by a factor of up to 26.9×. Despite moving data through the PCIe bottleneck, the additional bandwidth resources of GPU global memory offer an acceleration. For Q19, MonetDB has a lower execution time than *HorseQC*. This shows that for queries with a low complexity, it is more effective to process data directly than moving it over PCIe.

8.2.11 Discussion

In the previous experiments, we evaluated our new approaches for querying compilation on coprocessors. Across all experiments, we were able to show improvements of query compilation over operator-at-a-time processing. Operator-at-a-time has a low memory efficiency due to large materialization volumes and repetitive operations. Therefore, the approach cannot efficiently utilize the memory systems surrounding the coprocessor.

While naive compilation techniques increase the memory efficiency, reductions and prefix sums split operator pipelines into multiple passes. In this way, the approach inherits the drawbacks of operator-at-a-time. This becomes visible because kernel execution times frequently exceed PCIe transfer times.

We demonstrated a query compilation technique that merges the operators of a pipeline into one compound kernel. When combined with efficient reduction techniques, the compound kernel achieves substantial advantages over other processing approaches. With upcoming OpenCAPI and NVLink interconnects, these improvements to GPU-local processing are essential in order to take advantage of the increased bandwidth of the new hardware. In the evaluation setting, the PCIe bandwidth can be saturated for all SSB queries. For the TPC-H benchmark, the approach is an improvement over operator-at-a-time and naive compilation, but saturates PCIe in only 8 out of 11 queries. We conclude that the compound kernel works particularly well with star join queries.

8.2.12 Summary

In this section, we showed query processing techniques that help to balance the data movement cost and compute throughput on GPU-style coprocessors. We measure the data transfer volumes in different scalable processing approaches to assess bandwidth bottlenecks. While naive scalable execution techniques are limited by PCIe bandwidth, batch processing is limited by GPU-local throughput. To address the bottleneck, we propose micro execution models that benefit from on-chip pipelining. Naive query compilation techniques allow simple code generation but inherit the memory-intensity of operator-at-a-time. We introduce compound kernels that merge several pipeline phases into one efficient kernel.

9 Energy Awareness

Energy is a fundamental resource constraint that is present almost everywhere in life. Many of the previous chapters indirectly discuss the energy demand of cyber-physical systems and machine learning. Taking a broader view of cyber-physical systems, we see that the total amount of energy required to solve a specific problem is (for constant P)

$$W = P \cdot t$$

where P is the power to run the hardware and t the amount of time this hardware needs to execute a given software. Hence the energy consumption is typically determined by

1. the hardware used (that is P)
2. the algorithm that is run on the hardware (this can influence P and t)
3. the time the hardware executes the implemented algorithm (that is t)

In practice there is often a trade-off between these quantities. Hardware that has great processing power can execute software very quickly, but often also requires much more energy. Similarly, less powerful hardware may take longer to execute a software pipeline while the overall energy consumption is smaller since it requires less power. Last, certain implementations might use specific hardware features (e.g. a GPU) that influences both the energy and time required for execution. With the ongoing integration of Machine Learning (ML) into cyber-physical systems, two research directions must be explored.

First, in order to apply and train Machine Learning models on small devices, the energy consumption of the ML algorithm itself must be reduced. Needed is a holistic approach that takes all steps of the ML pipeline into account, starting from its theoretical model down to its specific implementation on a specific hardware platform.

Second, the application of ML models to reduce the energy consumption of *other* parts of the cyber-physical system must be explored. Here, a cross-domain approach that combines domain-specific knowledge with ML for the right problems is necessary.

This chapter performs an exemplary discussion of both approaches. Section 9.1 discusses how probabilistic undirected models can be rephrased with integer-only operations such that floating-point co-processors are no longer required. It introduces Bit-Length Propagation (BL-Prop) and combines it with a novel IntGD algorithm for numerical optimization with integrality constraints. The resulting algorithm enables the training and inference of Markov random fields on small devices using integer-only arithmetics.

Section 9.2 discusses how ML models can be integrated into the wireless communication of cyber-physical systems. More specifically, methods for modeling power consumption for different communication technologies are discussed including LTE, LTE-A, and NB-IoT. The integration of ML models in the User Equipment (UE) for estimating transmission uplink power under external influences (e.g., signal strength or signal quality) is further explored and discussed in a real-world context.

https://doi.org/10.1515/9783110785944-009

9.1 Integer Exponential Families

Nico Piatkowski

Abstract: In this contribution, we study how knowledge about the underlying compute architecture can be incorporated directly into the learning problem. More precisely, we consider the arithmetic limitations of Ultra-Low Power (ULP) Micro-Controller Units (MCU). Such systems do not contain arithmetic co-processors, which implies that most arithmetic computations must be emulated via integer logic. However, this creates a large performance penalty for any machine learning method that relies heavily on floating-point arithmetic. To mitigate this issue, we show how the model itself can be rephrased with integer-only operations such that floating point co-processors are no longer required. We exemplify this procedure with probabilistic undirected models, so-called Markov random fields. All steps of learning and inference are discussed. An approximate but integer-only probabilistic inference procedure called bit-length propagation (BL-Prop) is presented. BL-Prop is based on belief propagation, where instead of the full messages, only their bit-length is propagated along the models' conditional independence structure. We analyze the algorithm and show what factors have the largest influence on the approximation quality.

Furthermore, we derive IntGD—a numerical optimization method for convex objective functions with integrality constraints. The method is based on an accelerated proximal algorithm for non-smooth and non-convex penalty terms. For integer gradients computed via BL-Prop, IntGD is guaranteed to deliver an integer learning procedure in which the final parameter vector as well as all intermediate results are integers. Numerical experiments on benchmark data show that integer models allow us to achieve a competitive prediction quality on low-end hardware while maintaining a large speedup compared with its double precision counterpart—thus, completely mitigating the performance penalty that arose from the missing floating point unit.

9.1.1 Introduction

Big data analytics for streaming sensor data challenges the resource efficiency of algorithms in several ways. Running data mining methods in resource-constrained computational environments generates challenges in terms of execution time and energy consumption. Fortunately, optimizations that reduce the number of cycles in which the CPU is busy also save energy consumption. When reviewing the specifications of processing units, one finds that integer arithmetic is usually cheaper in terms of instruction latency, i.e., it needs a smaller number of clock cycles until the result of an arithmetic instruction is ready. Table 9.1 shows the latencies of arithmetic instructions measured in terms of clock cycles for Intel CPUs and ARM CPUs and for Nvidia GPUs. Note that transcendental functions are composed of multiple instructions and therefore may take substantially more cycles than the ones reported in Table 9.1. This motivates reducing the number of cycles in which code is executed when designing new, resource-aware learning algorithms.

Nowadays, big data arises in social media, industry, and basically all scientific research areas. Data sets grow in size because they are increasingly being gathered by ubiquitous information-sensing mobile devices. The joint prediction of various unknowns based on multiple observed inputs is a ubiquitous subtask in real-world problems from various domains, including computational biology, computer vision, and natural language processing. Probabilistic graphical models are well-suited for such tasks, but they suffer from the high complexity of probabilistic inference. Many approximate approaches to probabilistic inference based on Belief Propagation (BP) [404, 560] were proposed in the last decade, e.g., Counting BP [369], Lifted BP [10], Stochastic BP [539], Tree-reweighted BP [691], Tree Block Coordinate Descent [642], and Particle BP [331]. Quadrature-based methods [572, 573] deliver promising results, but are not well-suited for embedded or resource-constrained environments. In contrast to these approaches, the underlying model class here is restricted to the integers, which results in a reduced runtime and energy savings, while maintaining good performance. Asymptotically, the new approach has the same complexity as the vanilla BP, but it uses cheaper operations.

This new approach should not be confused with models that are designed for integer state spaces, in which case the state space $\mathcal{X}$ is a subset of the natural numbers or, more generally, is a metric space. Here, the state space may be a random discrete space without any additional constraints.

Estimation in discrete parameter models was recently investigated by Chaorat and Seri [140]. They discuss consistency, asymptotic distribution theory, information inequalities, and their relations with efficiency and super-efficiency for a general class of m-estimators. Unfortunately, we do not consider the case when the true estimator is not included in the search space and therefore, their analysis cannot be used to estimate the error when the optimizer has to be approximated.

Bayesian network classifiers with reduced precision parameters were presented by Tschiatschek et al. [671]. They evaluate empirically the classification performance when reducing the precision of Bayesian networks probability parameters. After learning the parameters as usual in $\mathbb{R}$ (represented as 64-bit double-precision floating-point numbers), they varied the bit-width of mantissa and exponent, and reported the prediction accuracy in terms of the normalized number of correctly classified test instances. They found that after learning, the parameters may be multiplied by a sufficiently large integer constant (10^9) to convert the probabilities into integer numbers. However, Tschiatschek et al. missed an important point, namely that real value probability parameters are necessary only for Bayesian networks.

Tab. 9.1: Instruction latencies (in clock cycles) of Floating-Point (FP) and integer (INT) scalar arithmetic operations for three processing architectures [24, 334, 542]. x/y means that latency is x for 32-bit and y for 64-bit operands. A single value indicates that both latencies are the same or, in case of ARM and GPU, that 64-bit integer arithmetic is not supported. For GPU, the values are based on the operation throughput. Cycles of Intel Sandy Bridge integer division and ARM11 integer multiplication depend on the lengths of their operands.

	Sandy Bridge		ARM		GPU	
	FP	INT	FP	INT_{32}	FP	INT_{32}
Addition	3	1	8	1	3/7	3
Multiplication	5	3	8/9	4-5	3/7	14
Division	14/22	13-25	19/33	-	7/-	-
Bit shift	-	3	-	2	-	7
Square root	14/22	-	19/33	-	14/-	-

For undirected graphical models, this is not the case. As a result, the general framework of undirected graphical models [692] may be mapped to the integer domain. A new optimization scheme is proposed, that allows the resource-constrained learning of integer parameters without the need for floating-point computation. This opens up the opportunity of running data mining tasks on resource-constrained devices. To be more precise, based only on integers, it is possible to compute approximations to the

- the marginal probabilities,
- the Maximum-A-Posteriori (MAP) assignment of the model,
- the Maximum Likelihood Estimate (MLE) either via an approximate closed form solution or an integer variant of stochastic gradient descent.

In this contribution, algorithms for integer models are derived. It turns out that the integer approximations do deliver a reasonable quality and are around twice as fast as their floating-point counterparts. This contribution is based on [574] and [567], and is organized as follows. A short introduction to probabilistic graphical models is given in Section 9.1.2. In Section 9.1.3, the intuition behind integer undirected graphical models is explained, and the corresponding algorithms are derived. Furthermore, a bound on the training error is presented. Two instances of the integer framework, *Integer Markov random fields* and *Integer Conditional Random Fields*, are evaluated in Section 9.1.4 for synthetic and real world data.

9.1.2 Probabilistic Graphical Models

In the following, the basic notation and concepts of probabilistic graphical models are introduced. Let $G = (V, E)$ be a graph with $|V| = n$ and $\mathcal{N}_v := \{w \in V : (v, w) \in E\}$ the neighbors of vertex $v \in V$. Each vertex $v \in V$ corresponds to a random variable (RV) X_v with realization x_v and domain $\mathcal{X}_v$. Consider the n-dimensional RV $\boldsymbol{X} = (X_v)_{v \in V}$ with realization $\boldsymbol{x} \in \mathcal{X} = \otimes_{v \in V} \mathcal{X}_v$. The probability of the event $\{\boldsymbol{X} = \boldsymbol{x}\}$ is denoted by $p(\boldsymbol{X} = \boldsymbol{x})$. $p(\boldsymbol{x})$ is used as a shortcut for $p(\boldsymbol{X} = \boldsymbol{x})$ in the remainder of this report. For a set of vertices $A \subseteq V$, $\boldsymbol{X}_A$ addresses the components of $\boldsymbol{X}$ that correspond to the vertices in A. For ease of notation, $\boldsymbol{X}_v$ and $\boldsymbol{X}_{\{v\}}$ are regarded as the same. For undirected graphical models, the joint probability mass function of $\boldsymbol{X}$ is given by

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{C \in \mathcal{C}(G)} \psi_C(\boldsymbol{x}_C) \tag{9.1}$$

$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x} \in \mathcal{X}} \prod_{C \in \mathcal{C}(G)} \psi_C(\boldsymbol{x}_C) \tag{9.2}$$

where $\mathcal{C}(G)$ is the set of all cliques[1] in G and $Z(\boldsymbol{\theta})$ is the normalization constant (since it does not depend on $\boldsymbol{x}$). Let C be a clique of G and $\mathcal{X}_C$ the corresponding joint domain of all vertices in C. The parameter vector $\boldsymbol{\theta} \in \Theta = \Omega^d$ contains $|\mathcal{X}_C|$ weights for each clique $C \in \mathcal{C}(G)$, i.e., $\boldsymbol{\theta} = (\boldsymbol{\theta}_C)_{C \in \mathcal{C}(G)}$, which results in $d = \sum_{C \in \mathcal{C}(G)} |\mathcal{X}_C|$. The *compatibility functions* ψ_C (also known as *factors*) are typically chosen to be

$$\psi_C(\boldsymbol{x}_C) = \exp(\langle \boldsymbol{\theta}_C, \phi_C(\boldsymbol{x}) \rangle)$$

since this ensures the positivity of $p_{\boldsymbol{\theta}}$ and leads to a canonical form of the corresponding exponential family member.

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \exp(\langle \boldsymbol{\theta}, \phi(\boldsymbol{x}) \rangle - A(\boldsymbol{\theta}))$$

The function ϕ is a *sufficient statistic* for $\boldsymbol{x}$ and may be understood as transformation of $\boldsymbol{x}$ into a binary valued feature space $\phi : \mathcal{X} \to \{0, 1\}^d$. For convenience, the components of $\boldsymbol{\theta}$ and ϕ are indexed by C to denote the subvector of weights or features that corresponds to a clique C. To address a certain component of $\boldsymbol{\theta}$ or ϕ, the corresponding event $\{\boldsymbol{X}_C = \boldsymbol{x}_C\}$ is used as an index, i.e., $\boldsymbol{\theta}_{\boldsymbol{X}_C = \boldsymbol{x}_C}$ or even $\boldsymbol{\theta}_{C = \boldsymbol{x}_C}$. If the parameters $\boldsymbol{\theta}$ are known, the maximum a posteriori prediction of the most likely joint state of all vertices can be computed by

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x} \in \mathcal{X}} p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \arg\max_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{\theta}, \phi(\boldsymbol{x}) \rangle . \tag{9.3}$$

Parameter Estimation A common choice for learning the parameters $\boldsymbol{\theta}$ of an undirected model is the maximum likelihood estimation, where the likelihood

$$\mathcal{L}(\boldsymbol{\theta} \mid \mathcal{D}) = \prod_{\boldsymbol{x} \in \mathcal{D}} p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{9.4}$$

[1] A clique corresponds to a fully connected subgraph.

of the parameters $\boldsymbol{\theta}$ for given i.i.d. data[2] $\mathcal{D}$ is maximized. The MLE $\boldsymbol{\theta}^{\star}$, i.e., the solution that maximizes $\mathcal{L}$, has a closed form, if and only if the underlying graphical structure is a tree or a triangulated graph. In this case, $\boldsymbol{\theta}^{\star}$ is induced by the empirical expectation of the sufficient statistics

$$\begin{aligned}\boldsymbol{\theta}^{\star}_{v=x} &= \log \mathbb{E}_{\mathcal{D}}\left[\phi_{v=x}(\boldsymbol{x})\right] , \\ \boldsymbol{\theta}^{\star}_{vu=xy} &= \log \frac{\mathbb{E}_{\mathcal{D}}\left[\phi_{vu=xy}(\boldsymbol{x})\right]}{\mathbb{E}_{\mathcal{D}}\left[\phi_{v=x}(\boldsymbol{x})\right] \mathbb{E}_{\mathcal{D}}\left[\phi_{u=y}(\boldsymbol{x})\right]} .\end{aligned} \tag{9.5}$$

The MLE $\boldsymbol{\theta}^{\star}$ for partially observed data and certain classes of graphical models like Conditional Random Fields (CRF) [655] can be found with gradient-based methods. Taking the logarithm of Equation 9.4, dividing by $|\mathcal{D}|$, and substituting Equation 9.1 for $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ yields the average log-likelihood (see Equation 9.6). Since the logarithm is monotonic, maximizing ℓ will reveal the same optimizer as $\mathcal{L}$. Since $\mathbb{E}_{\mathcal{D}}\left[\phi(\boldsymbol{x})\right] = \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}} \phi(\boldsymbol{x})$, ℓ is given by

$$\ell(\boldsymbol{\theta} \mid \mathcal{D}) = \left\langle \boldsymbol{\theta}, \mathbb{E}_{\mathcal{D}}\left[\phi(\boldsymbol{x})\right]\right\rangle - \ln Z(\boldsymbol{\theta}). \tag{9.6}$$

Taking the natural logarithm to form the log-likelihood is a random choice that may be replaced with any other $\log_b$ if desired. Since the second term is the cumulant generating function of $p_{\boldsymbol{\theta}}$, its partial derivative is the expected sufficient statistic for a given $\boldsymbol{\theta}$. This is plugged into the partial derivative of ℓ with regard to $\boldsymbol{\theta}_{\boldsymbol{x}_C=\boldsymbol{x}_C}$ (Equation 9.6) to obtain

$$\frac{\partial \ell(\boldsymbol{\theta} \mid \mathcal{D})}{\partial \boldsymbol{\theta}_{\boldsymbol{x}_C=\boldsymbol{x}_C}} = \mathbb{E}_{\mathcal{D}}[\phi_{\boldsymbol{x}_C=\boldsymbol{x}_C}(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{\theta}}[\phi_{\boldsymbol{x}_C=\boldsymbol{x}_C}(\boldsymbol{x})] . \tag{9.7}$$

Here, $\mathbb{E}_{\mathcal{D}}[\phi_{\boldsymbol{x}_C=\boldsymbol{x}_C}(\boldsymbol{x})]$ denotes the empirical expectation of $\phi_{\boldsymbol{x}_C=\boldsymbol{x}_C}(\boldsymbol{x})$, i.e., its average value in $\mathcal{D}$. By using Equation 9.7, the model parameters $\boldsymbol{\theta}$ can be estimated by any first-order optimization technique.

Inference: In the following, it is explained shortly how $\mathbb{E}_{\boldsymbol{\theta}}[\phi_{\boldsymbol{x}_C=\boldsymbol{x}_C}(\boldsymbol{x})]$ is computed with *Belief Propagation* (BP). From now on, assume that the underlying graphical structure is a tree. The maximum clique size is thus 2. The message update rule is

$$m_{v\to u}(x_u) = \sum_{x_v \in \mathcal{X}_v} \psi_{v,u}(x_v, x_u)\psi_v(x_v) \prod_{w \in \mathcal{N}_v \setminus \{u\}} m_{w\to v}(x_v). \tag{9.8}$$

The messages $m_{v\to u}(x_u)$ are computed for all pairs of vertex $v \in V$ and neighbor $u \in \mathcal{N}_v$ until convergence. Converged messages are denoted by $m^{\star}_{v\to u}(x_u)$. The product of all incoming messages of a vertex is given by $M_v(x) := \prod_{u\in\mathcal{N}_v} m_{u\to v}(x)$. After convergence, the vertex marginal probabilities $p_v(x_v)$ that are implied by $\boldsymbol{\theta}$ can be computed with

$$p_v(x_v) = \frac{\psi_v(x_v)M^{\star}_v(x_v)}{\sum_{x\in\mathcal{X}_v} \psi_v(x_v)M^{\star}_v(x)} , \tag{9.9}$$

[2] It is assumed that every training instance in $\mathcal{D}$ is fully observed.

whereas $M^{\star}_v(x)$ is the product of converged messages $m^{\star}_{v \to u}(x_u)$. In case of non-tree-structured graphs, BP performs multiple passes over all vertices until the convergence of messages is reached. The convergence depends on the dynamic range of the potentials. For trees and triangulated graphs, efficient orderings of message computations (schedulings) are known that have polynomial runtime $\mathcal{O}(m\deg(G)|\mathcal{X}|^2)$ and result in the exact marginal probabilities. We refer to [404] for discussions on belief-propagation and related algorithms.

9.1.3 The Integer Approximation

In their fundamental book on graphical models, Wainwright and Jordan [692] write: "It is important to understand that for a general undirected graph the compatibility functions ψ_C need not have any obvious or direct relation to marginal or conditional distributions defined over the graph cliques. This property should be contrasted with the directed factorization, where the factors correspond to conditional probabilities over the child-parent sets." This explains why it might be possible to have an undirected graphical model that is parametrized by integers. But the identification of integer parameters is not enough for excluding every floating-point computation. Moreover, the computations that are required for training and prediction have to be based on integer arithmetic. Last, the integer approximation should still deliver a reasonable quality in terms of training error and test error.

The first step is directly related to the above statement. The potential function

$$\overline{\psi}_C(\boldsymbol{x}_C) := 2^{\langle \boldsymbol{\theta}_C, \phi_C(\boldsymbol{x}) \rangle} = \exp\left(\ln(2)\langle \boldsymbol{\theta}_C, \phi_C(\boldsymbol{x}) \rangle\right) \tag{9.10}$$

is defined in a way, that yields only integer values as long as the parameters are positive integers. It is easy to see that replacing $\psi_C(\boldsymbol{x}_C)$ with $\overline{\psi}_C(\boldsymbol{x}_C)$ does not alter the marginal probabilities as long as the parameters are scaled by $1/\ln 2$. It is possible to convert parameters that are estimated with $\psi_C(\boldsymbol{x}_C)$ to $\overline{\psi}_C(\boldsymbol{x}_C)$ and vice versa without altering the resulting probabilities. Notice that $\overline{\psi}_C(\boldsymbol{x}_C)$ can be computed by a logical bit shift to the left, which consumes less clock cycles than the corresponding transcendental function. As already mentioned above, it requires that $\boldsymbol{\theta} \in \mathbb{N}^d$ for $\overline{\psi}_C(\boldsymbol{x}_C)$ is an integer and the product of compatibility functions and the normalization constant (see Equation 9.2) are computable by means of non-negative integer arithmetic. This restricts $p(\boldsymbol{x})$ and its marginals to $[0, 1] \cap \mathbb{Q}$. Although the computation of a probability would require floating-point division, its actual value is not required for estimating the integer model parameters.

Inference Recalling the message update Equation 9.8, one sees that all messages are integer valued, if $\psi_C(\boldsymbol{x}_C)$ is replaced by $\overline{\psi}_C(\boldsymbol{x}_C)$ and the initial messages are set to 1. Thus, the whole message computation and propagation procedure is already stated without floating-point computation. Nevertheless, recall that a CPU's integer width is constrained by its wordsize ω. $m_{v\to u}(x)$ may exceed the machines' integer precision 2^{ω} quite easily. Thus, many overflows could occur during message computations, which destroy the semantics of the messages and the resulting beliefs are no longer usable.

Initial attempts to make the computation more robust against overflows relied on the fact that messages $m_{v\to u}(x)$ may be scaled arbitrarily without changing the resulting marginal probabilities as long as the same scale is used for all x. Nevertheless, the messages cannot be simply divided by their sum as with floating-point arithmetic, since integer division will pin all messages down to 0. Numerous attempts to scale the integer messages by bit-shift operations have only worked on relatively small graphical structures, but all those approaches suffered from the loss of information that occurred whenever too many bits had to be shifted out in order to prevent overflows.

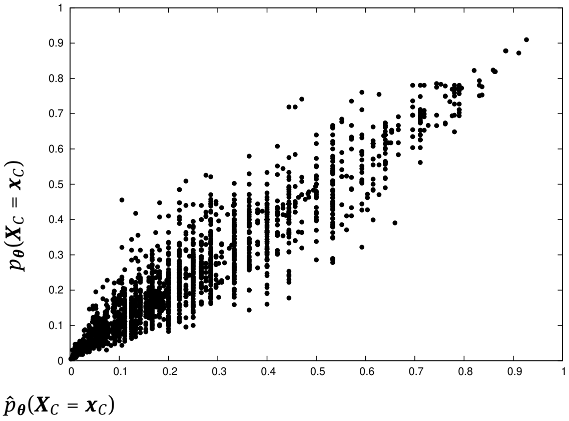


Fig. 9.1: Estimates of edge marginal probabilities for 50 random trees with 50 nodes and 2 states per node. Marginals are computed by the bit-length approximation ($\hat{p}$) and vanilla BP (p) on the same parameter vector $\boldsymbol{\theta}$.

As a solution to this problem, new messages are defined. Instead of computing the original sum-product messages, we propose computing an approximation to the integer message bit length. The approximate bit length $\beta_{vu}(y)$ and the corresponding message $\hat{m}_{vu}(y)$ are given by

$$\beta_{vu}(y) := \max_x \boldsymbol{\theta}_{vu=xy} + \boldsymbol{\theta}_{v=x} + \boldsymbol{\theta}_{u=y} \tag{9.11}$$

$$+ \textstyle\sum_{w\in\mathcal{N}_v\setminus\{u\}} \beta_{wv}(x)\,, \tag{9.12}$$

$$\hat{m}_{vu}(y) := 2^{\beta_{vu}(y)} \tag{9.13}$$

$$= \max_x \psi_{vu}(x,y) \textstyle\prod_{w\in\mathcal{N}_v\setminus\{u\}} \hat{m}_{wu}(x)\,. \tag{9.14}$$

How m and $\hat{m}$ are related to each other is a natural question. The messages $\hat{m}$ that result from the bit-length approximation resemble max-product messages [404]. Their magnitude is related to the original messages m through the following lemma.

Lemma 1. *Let* $(v, u) \in E$ *be an edge of* $G = (V, E)$, $h_v := |\mathcal{X}_v|$ *the size of vs state space, and* $n_v := |\mathcal{N}_v|$ *the number of its neighbors. If* $h_v \geq 2 \wedge \forall y \in \mathcal{X}_u : \exists x \in \mathcal{X}_v : \boldsymbol{\theta}_{vu=xy} + \boldsymbol{\theta}_{v=x} + \boldsymbol{\theta}_{u=y} > 0$, *then*

$$\hat{m}_{vu}(x) < m_{vu}(x) \leq \hat{m}_{vu}^{h_v}(x) .$$

This statement can be proven by induction over the vertex degree. Note, that this implies $M_v(y) = \prod_{w \in n_v} m_{wv}(x) \leq \prod_{w \in n_v} \hat{m}_{wv}^{h_v}(x) = \hat{M}_v^{h_v}(y)$. When it comes to the point-wise estimates of the marginal probabilities, one finds that due to the approximate messages some marginals probabilities simply cannot be present. Figure 9.1 shows edge marginal probabilities for random parameters that are generated with m and $\hat{m}$, respectively. One clearly sees how the probability space is discretized by the approximate messages. One can also see that there is an error in the approximate marginal probabilities computed with $\hat{m}$, since in case of zero error all points would lie on the diagonal.

The previous lemma helps to derive an estimate of the distance between the true outcome of the inference and the one that results from the message update Equation 9.14.

Theorem 1. *Let* $\beta_v^\star := \max_y \max_u \beta_{uv}(y)$ *be the maximum incoming bit length at v and assume that the preconditions of Lemma 1 hold, then*

$$D\left(p_v \| \hat{p}_v\right) \quad \in \quad \mathcal{O}\left(n_v h_v \beta_v^\star\right) ,$$

where $D\left(p_v \| \hat{p}_v\right)$ *denotes the Kullback-Leibler (KL) divergence between the marginal probability mass function of* p_v, *computed with the message update* $m_{vu}(y)$, *and* $\hat{p}_v$, *computed with* $\hat{m}_{vu}(y)$.

This result can be derived by plugging the BP marginals (Equation 9.9) into the definition of the KL divergence and applying Lemma 1 two times. The KL is still unbounded, since there is no bound on $\beta_v^\star$. Nevertheless, it indicates a dependence of the KL of p_v and $\hat{p}_v$ on the state space size $|\mathcal{X}_v|$ and the neighborhood size $|\mathcal{N}_v|$. This relation can also be observed in the numerical experiments in Section 9.1.4. A comprehensive discussion of how message errors generally affect the result of belief propagation can be found in [332].

9.1.3.1 Parameter Estimation

In the following, an integer parameter estimation method based on the closed form solution to the MLE is derived. Recall that $\mathbb{E}_{\mathcal{D}}\left[\phi(\boldsymbol{x})\right] = \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{x} \in \mathcal{D}} \phi(\boldsymbol{x})$ and let $\boldsymbol{f} := \sum_{\boldsymbol{x} \in \mathcal{D}} \phi(\boldsymbol{x})$ and $\mathrm{bl}(a) = \lfloor \log_2 a \rfloor + 1$ the bit length of a. With this, an integer upper bound on the optimal parameters can be found.

$$\log_2 \mathbb{E}_{\mathcal{D}} \left[\phi_{v=x}(\boldsymbol{x})\right] = \log_2 \boldsymbol{f}_{v=x} - \log_2 |\mathcal{D}| \tag{9.15}$$

$$\leq \mathrm{bl}\, \boldsymbol{f}_{v=x} - \mathrm{bl}\,|\mathcal{D}| =: \tilde{\boldsymbol{\theta}}_{v=x} \tag{9.16}$$

$$\log_2 \mathbb{E}_{\mathcal{D}} \left[\phi_{vu=xy}(\boldsymbol{x})\right] \leq \mathrm{bl}\, \boldsymbol{f}_{vu=xy} \tag{9.17}$$

$$- \mathrm{bl}\, \boldsymbol{f}_{v=x} - \mathrm{bl}\, \boldsymbol{f}_{vu=xy} + \mathrm{bl}\,|\mathcal{D}| \tag{9.18}$$

$$=: \tilde{\boldsymbol{\theta}}_{vu=xy}$$

Unfortunately, most of those estimates are negative which is not allowed due to the integer restriction. Let $s := \max_{1 \leq i \leq d} -\tilde{\boldsymbol{\theta}}_i$ be the negative component of $\tilde{\boldsymbol{\theta}}$ with the largest magnitude. Now, consider the weights

$$\tilde{\boldsymbol{\theta}}^+_{v=x} := \boldsymbol{s} + \tilde{\boldsymbol{\theta}}_{v=x}, \qquad \tilde{\boldsymbol{\theta}}^+_{vu=xy} := \boldsymbol{s} + \tilde{\boldsymbol{\theta}}_{vu=xy}$$

with $\boldsymbol{s} := (s, s, \ldots, s)^\top \in \mathbb{R}^d$. Clearly, an error is induced into $\tilde{\boldsymbol{\theta}}$ by replacing $\log_2$ with bl. The following lemma shows that shifting $\tilde{\boldsymbol{\theta}}$ by $\boldsymbol{s}$ introduces no new error.

Lemma 2. *Let $\boldsymbol{s} := (s, s, \ldots, s)^\top \in \mathbb{R}^d$ and ϕ be an overcomplete sufficient statistic, then $\ell(\boldsymbol{\theta} + \boldsymbol{s}) = \ell(\boldsymbol{\theta})$.*

Proof: Since ϕ is overcomplete, it holds that $\langle \boldsymbol{s}, \phi(\boldsymbol{x}) \rangle = \text{const}, \forall \boldsymbol{x}$ and hence:

$$\ell(\boldsymbol{\theta}) - \ell(\boldsymbol{\theta} + \boldsymbol{s}) \tag{9.19}$$

$$= \frac{1}{D} \sum_{\boldsymbol{x} \in \mathcal{D}} \log \frac{\sum_{\boldsymbol{y} \in \mathcal{X}} \exp \langle \boldsymbol{\theta} + \boldsymbol{s}, \phi(\boldsymbol{y}) \rangle}{\exp \langle \boldsymbol{s}, \phi(\boldsymbol{x}) \rangle \sum_{\boldsymbol{y}' \in \mathcal{X}} \exp \langle \boldsymbol{\theta}, \phi(\boldsymbol{y}') \rangle} = 0.$$

ϕ as defined in Section 9.1.2 is actually overcomplete. This can now be used to bind the training error of the shifted integer parameters $\tilde{\boldsymbol{\theta}}^+$.

Theorem 2. *Let $-s$ be the smallest value in the vector $\mathbb{E}_{\mathcal{D}} \left[\phi(\boldsymbol{x})\right]$. Furthermore, let $\boldsymbol{\theta}^\star_i := s + \log \mathbb{E}_{\mathcal{D}} \left[\phi_i(\boldsymbol{x})\right]$ and $\tilde{\boldsymbol{\theta}}^+_i := s + \mathrm{bl}\, \mathbb{E}_{\mathcal{D}} \left[\phi_i(\boldsymbol{x})\right]$ then*

$$\ell(\boldsymbol{\theta}^\star) - \ell(\tilde{\boldsymbol{\theta}}^+) \leq \|\nabla f(\tilde{\boldsymbol{\theta}}^+)\|_1.$$

The result follows from the previous lemma, convexity, and the Cauchy-Schwarz inequality. Since each component of the gradient is a difference of two probabilities, its magnitude cannot be greater than 1. Hence, the gradient norm can be at most d. In the following section, the magnitude of the gradient relative to d is evaluated numerically.

Either due to restrictions in wordsize ω or for enlarging the number of representable marginal probabilities, a final scaling of the parameters might be desired. To allow an appropriate integer scaling, the parameter K is introduced. Let $s := \max_{1 \leq i \leq d} -\tilde{\boldsymbol{\theta}}_i$ be the negative component of $\tilde{\boldsymbol{\theta}}$ with the largest magnitude and $m := \max_{1 \leq i \leq d} \tilde{\boldsymbol{\theta}}_i$ be the positive component of $\tilde{\boldsymbol{\theta}}$ with the largest magnitude. The final integer parameters are computed by

$$\bar{\boldsymbol{\theta}}_{v=x} := \left\lfloor \frac{K}{s+m} \tilde{\boldsymbol{\theta}}^{+}_{v=x} \right\rfloor , \qquad \bar{\boldsymbol{\theta}}_{vu=xy} := \left\lfloor \frac{K}{s+m} \tilde{\boldsymbol{\theta}}^{+}_{vu=xy} \right\rfloor . \tag{9.20}$$

Thus, $\tilde{\boldsymbol{\theta}}^{+}$ is rescaled such that $\bar{\boldsymbol{\theta}} \in \{0, 1, \dots, K\}^{d}$, which may also be interpreted as implicit base change. Note, that unless $K = (s+m)$, the parameter vector is scaled and an additional error is added to the gradient. Hence, the impact of K is empirically evaluated in Section 9.1.4. The method of choosing parameters according to Equation 9.20 is called *direct integer estimation.*

Gradient-Based Estimation As already mentioned in Section 9.1.2, in certain situations, it might be desired to estimate the parameters with gradient-based methods. Unfortunately, the partial derivatives from Equation 9.7 are not integers. Hence, the expression must be rearranged to obtain an integer form. Let $\boldsymbol{f} := \sum_{\boldsymbol{x}\in\mathcal{D}} \phi(\boldsymbol{x})$, so that

$$\begin{aligned} &\left\lceil \sum_{x\in\mathcal{X}_v} \hat{M}^{\star}_{v}(x) \right\rceil |\mathcal{D}| \frac{\partial \ell(\boldsymbol{\theta} \mid \mathcal{D})}{\partial \boldsymbol{\theta}_{X_v = x_v}} \\ &= \left\lceil \sum_{x\in\mathcal{X}_v} \hat{M}^{\star}_{v}(x) \right\rceil \boldsymbol{f}_{v=x} - |\mathcal{D}|\, \hat{M}^{\star}_{v}(x_v). \end{aligned} \tag{9.21}$$

This scaled version of the partial derivative is an integer expression that can be computed by using only integer addition, multiplication, and binary bit shift. The common gradient descent update makes use of a step size η to determine how far the current weight vector should move in the direction of the gradient. The smallest possible step size in integer space is 1. This means that any parameter can either be increased or decreased by 1. Therefore, in the beginning of an integer gradient-based optimization, all the model parameters are 0 and the gradient will tend to increase a large number of parameters. This results in a rather slow convergence, since due to the fixed step size of 1, most of the parameters are worse than before the update. To compensate, we suggest updating, for each clique only the parameter for which the corresponding partial derivative has the largest magnitude. This method is used when estimating the CRF parameters in the following section.

9.1.4 Numerical Results

The previous sections pointed to various factors that may have an influence on the training error, test performance, or runtime of the integer approximation. In order to show that integer undirected models are a general approach for approximate learning in discrete state spaces, generative and discriminative variants of undirected models are evaluated on synthetic data and real-world data. We consider in particular the following methods:

RealMRF The classic generative undirected model as described in Section 9.1.2.

RealCRF The discriminative classifier as it is defined in [407, 655].

IntMRF The integer approximation of generative undirected models as described in Section 9.1.3.

IntCRF The integer approximation of discriminative undirected models. Further details are explained in Section 9.1.4.5.

Both real variants are based on floating-point arithmetic. In the MRF experiments, the model parameters are estimated from the empirical expectations by Equations 9.5 and 9.20. Parameters of discriminative models are estimated by stochastic gradient methods [655]. Each MRF experiment was repeated 100 times on random input distributions and graphs. In most cases, only the average is reported, since the standard deviation is too small to be visualized in a plot. Whenever MAP accuracy is reported, it corresponds to the percentage of correctly labeled vertices, where the prediction is computed with Equation 9.3.

Of course, the implementations of the above-mentioned methods are equally efficient, e.g. the message computation (and therefore the probability computation) executes exactly the same code for all methods, except for the arithmetic instructions. A subsets of the results is presented below. Unless otherwise explicitly stated, the experiments are done on an Intel Core i7-2600K 3.4 GHz (Sandy Bridge architecture, Table 9.1) with 16 GB 1 333 MHz DDR3 main memory. An implementation of integer Markov random fieldsis available as part of the Python package pxpy.[3]

Synthetic Data In order to achieve robust results that capture the *average behavior* of the integer approximation, a synthetic data generator has been implemented that samples random empirical marginals with corresponding MAP states. Therefore, a sequential algorithm for random trees with given degrees [57] generates random tree structured graphs. For a random graph, the weights $\boldsymbol{\theta}_i^\star \sim \mathcal{N}(0, 1)$ are sampled from a Gaussian distribution. Additionally, for each vertex, a random state is selected that gets a constant extra amount of weight, thus enforcing low entropy. The weights are then used to generate marginals and MAP states with the double precision floating-point variant of belief propagation. The so generated marginals provide empirical input distribution. The MAP state is then compared with the MAP state that is estimated by IntMRF and RealMRF for the given empirical marginals.

CoNLL-2000 Data This dataset was proposed for the shared task at the Conference on Computational Natural Language Learning in 2000. The train and test data consist

[3] https://pypi.org/project/pxpy.

of three columns separated by spaces. Each word has been put on a separate line and there is an empty line after each sentence. The first column contains the current word, the second contains its part-of-speech tag as derived by the Brill tagger, and the third contains its chunk tag as derived from the Wall Street Journal corpus. The chunk tags contain the name of the chunk type—I-NP for noun phrase words and I-VP for verb phrase words, say. Most chunk types have two types of chunk tags, B-CHUNK for the first word of the chunk and I-CHUNK for each other word in the chunk. In total, there are 22 chunk tags that correspond to the vertex states, i.e., $|\mathcal{X}| = 22$. For the computation of per chunk F_1-score, a chunk is treated as correct if and only if all consecutive tags that belong to the same chunk are correct. The dataset contains $8,936$ training instances and $2,012$ test instances. For each word, the surrounding words and part-of-speech tags are used as features. Because of the inherent dependency between neighboring vertex states, this dataset is especially well suited for evaluation if the dependency structure between vertices is preserved by the integer approximation.

9.1.4.1 The Impact of $|\mathcal{X}|$ and $|\mathcal{N}_v|$ on Quality and Runtime

In Section 9.1.3 an estimate of the error in marginal probabilities that are computed with bit length BP indicates that the size of a vertex state space $|\mathcal{X}_v|$ and the degree $|\mathcal{N}_v|$ have an impact on the training error. Figure 9.2 shows the training error in terms of normalized negative log-likelihood, the testing error in terms of MAP accuracy, and the runtime in seconds for various values of $|\mathcal{X}_v|$ and $|\mathcal{N}_v|$ for an increasing number of vertices on the synthetic data. Each curve is the average over 100 random trees with random parameters and $K = 8$. The results with varying $|\mathcal{X}_v|$ are generated with a maximum degree of 8 and the ones for varying $|\mathcal{N}_v|$ are generated with $|\mathcal{X}_v| = 4$.

In terms of training error, the top-left plot shows a clear offset between integer and floating-point estimates for the same number of states. In terms of varying degrees (center-left), the training error of the integer model shows a response to different neighborhood sizes, whereas the likelihood of the floating-point model is invariant against the degrees. A similar picture is drawn for the dependence of the test accuracy with $|\mathcal{X}|$ and $|\mathcal{N}_v|$ (top-right, top-center). The floating-point MAP estimate is not changed by an increasing number of states and neighbors, whereas the integer MRF shows a clear response. The accuracy of the integer MRF actually increases with increasing degrees. In general, the quality of the models seems to be independent of the number of vertices in the graph.

The floating-point model outperforms the approximate integer model in terms of the MAP accuracy and negative log-likelihood. However, the two plots at the bottom of Figure 9.2 show that the resource consumption in terms of clock cycles is largely reduced by the integer model. Time is measured for estimating parameters, computing the likelihood, and performing a MAP prediction. Since both algorithms (RealMRF and IntMRF) share exactly the same asymptotic complexity for these procedures, the substantial reduction in runtime that is shown by the results must be due to the reduction in clock cycles.

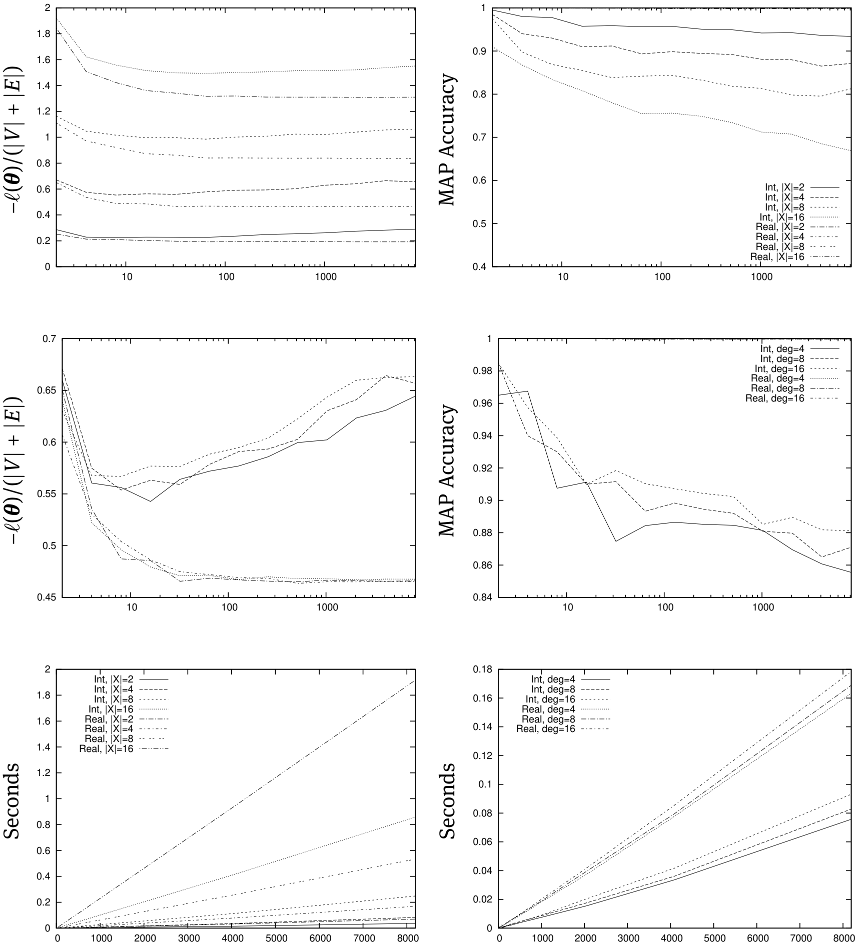


Fig. 9.2: MRF training error, test accuracy, and runtime in seconds for different choices of the state space size ($\mathcal{X}$) and maximum degree as a function of the number of vertices. Each of the top two rows shares legends and has an x-axis in logarithmic scale.

9.1.4.2 The Contribution of *K* to Quality

It might be convenient to scale the integer parameters such that the resulting parameter vector $\bar{\boldsymbol{\theta}}$ is in the set $\{0, 1, \ldots, K\}$. We illustrate the effect of such scaling by the response of the integer model in terms of training quality and test error, as shown in the two plots on the top of Figure 9.3. The training error seems to be a smooth function of K whereas the MAP accuracy is sensitive to the choices of K. This is what we expected, since a large K basically means that a larger number of marginal probabilities can be represented. One can also see that, as soon as K is large enough (i.e., $K = 8$ in the right plot at the top of Figure 9.3), a further increase does not show any significant impact on either training error and test accuracy. Both results where generated on graphs with a maximum degree of 8, but as already known from the previous experiment, the effect of different degrees on the model's quality is negligible. The bottom-left plot in Figure 9.3 shows the width of the intrinsic parameter space, i.e., the sum of the smallest and the largest integer parameter before rescaling is performed. It turns out, that the width of the parameter space is naturally bounded, since $s + m$ seems to converge on the same value for various configurations of n and $\mathcal{X}$. Plotting s and m separately shows that the dynamics in $s + m$ are mainly influenced by the smallest parameter s, i.e., the width of the parameter space must increase in order to represent smaller probabilities.

9.1.4.3 The Impact of *K* on the Gradient Norm

As indicated by the analysis of the training error in Section 9.1.3, the distance between the maximum likelihood estimate and the result of the direct integer parameter estimation is basically bounded by the gradient norm of the integer parameters $\bar{\boldsymbol{\theta}}$. Since the components of the gradient cannot exceed 1, a trivial upper bound for the gradient norm is d, the dimension of the parameter vector. A very strong observation can be made in the rightmost picture which shows the relative gradient norm for an increasing number of vertices and various values K. This result suggests that there exists a bound on the relative gradient norm that is independent of the number of vertices and that this bound decreases with increasing K.

9.1.4.4 Integer Models on Resource-Constrained Devices

The motivation for the integer model was to save resources in terms of clock cycles. We can now demonstrate that the impact of this reduction is larger, if the underlying architecture is weaker, i.e., has slower floating-point arithmetic. The two bar charts in Figure 9.4 show a runtime comparison of the integer MRF on two different CPU architectures. One is Sandy Bridge, which has also been the platform for all the other experiments; the other is a Raspberry Pi device with ARM11 architecture. As expected, the integer model actually speeds up the execution on the Pi device more than on the other architecture, i.e., the Raspberry Pi gains a speedup of 2.56× and Sandy Bridge a speedup of 2.34×. In terms of standard deviation, the ARM11 architecture is more stable than the Sandy Bridge, which might be a result of a more sophisticated out-of-order instruction execution in the latter architecture.

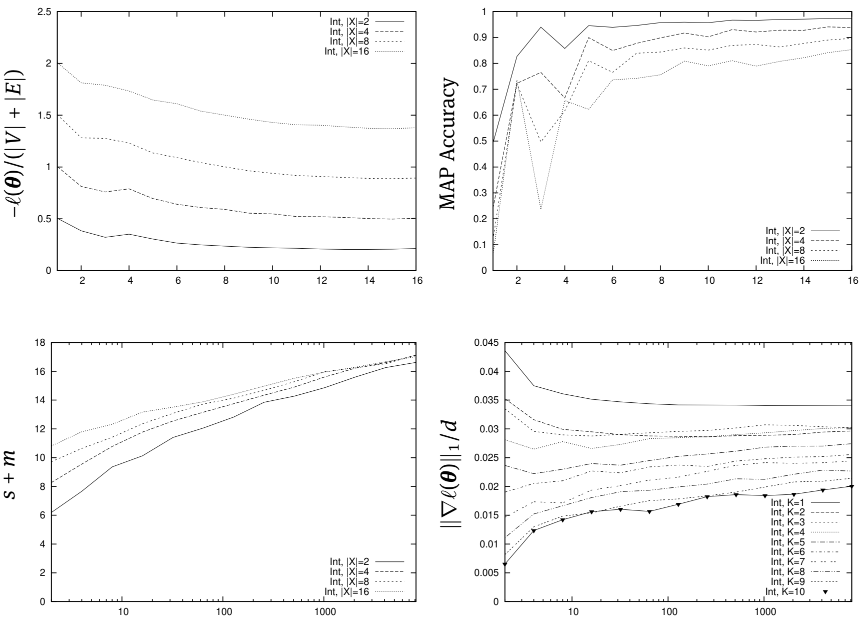


Fig. 9.3: Top: negative log-likelihood and MAP accuracy of MRF as a function of K. Bottom: the left plot shows how the width of the parameter space behaves as a function of the number of vertices (in log-scale) for different state space sizes, whereas $-s$ is the smallest and m the largest element of the corresponding estimated parameter vector. The relative norm of the gradient for various values of K is shown on the right.

9.1.4.5 Training Integer CRF with Stochastic Integer Gradient Descent

In the last evaluation, the randomized stochastic gradient training of discriminative models is investigated. An integer linear-chain CRF is constructed and trained by a stochastic gradient-descent algorithm. In case of the integer CRF, the parameter updates are computed by means of the scaled integer gradient (cf. the end of Section 9.1.3). Both algorithms perform 20 passes over the training data, each pass looping through the training instances in random order. This was repeated 50 times in order to compute an estimate of the expected quality of the randomized training procedure. The parameter update for the floating-point CRF is computed with the step size $\eta = 10^{-1}$. The ratio of quality per runtime is presented in Figure 9.4, where the negative log-likelihood is averaged over all training instances and the accuracy is computed with regard to the chunk tags. Chunk type precision, recall, and F_1-score are shown in Table 9.2, whereby the overall F_1-score for a model with $\boldsymbol{\theta} = \mathbf{0}$ is ≈ 26 %. As desired, the performance of the integer approximation is reasonable. Except for one chunk type (INTJ), precision, recall, and F_1-score have a relatively small standard deviation of about 2 %. The precision is three times better using integer CRF; the recall and F_1-score are one time better using integer CRF than real CRF. IntCRF is substantially worse than RealCRF only for the

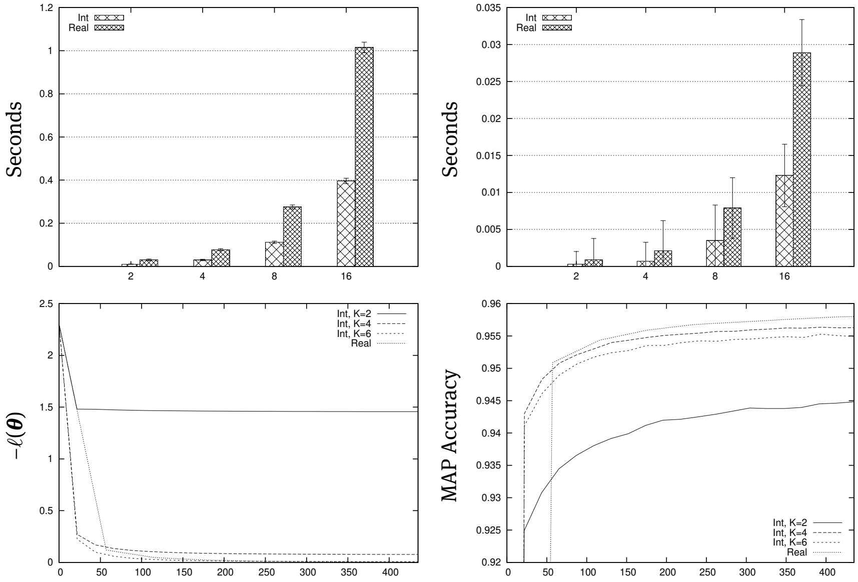


Fig. 9.4: Top: Runtime comparison of integer and floating-point MRF on two architectures for a varying number of states. Left: Raspberry PI @ 700 MHz (ARM11). Right: Intel Core i7-2600K @ 3.4 GHz (Sandy Bridge). Bottom: progress of stochastic gradient training in terms of training error and test accuracy of the CRFs over running seconds.

verb phrase (VP). For many real-world applications, this is a price that can be paid for IntCRF being about twice as fast as RealCRF.

9.1.5 Conclusion

In this contribution, integer undirected graphical models have been introduced, together with algorithms for probabilistic inference and parameter estimation that rely only on integer arithmetic. Generative and discriminative models have been evaluated in terms of prediction quality and runtime. We learned that optimal integer model parameters typically take values with small magnitude, reducing the storage requirement when compared with 64-bit double precision numbers. This allows us to sample from high-dimensional generative models and to use structured discriminative classifiers, even on computational devices without or with a slow floating-point unit, or in situations where energy has to be saved.

Tab. 9.2: The difference of RealCRF and IntCRF precision (Prec), recall (Rec), and F_1-score (F_1) averaged over 50 repetitions on the CoNLL-2000 dataset. Values are in percentage, i.e., a value of −0.29 means that the integer CRF was on average 0.29 % better at the corresponding measure than its floating-point counterpart.

	ADJP	**ADVP**	**CONJP**	**INTJ**	**LST**	**NP**	**PP**	**PRT**	**SBAR**	**VP**	**Overall**
Prec	−0.29	−0.46	4.81	−15.56	0	1.11	0.33	3.78	1.15	6.09	0.73
Rec	1.61	2.69	−3.55	0	0	0.34	0.13	1.82	1.78	6.31	0.48
F_1	0.74	1.17	1.60	−4.03	0	0.73	0.23	2.85	1.50	6.20	0.61

9.2 Power Consumption Analysis and Uplink Transmission Power

Robert Falkenberg

Abstract: The penetration of wireless communication systems in industrial and private environments is constantly increasing due to their flexible and mobile application possibilities. Wearables, smartphones, or industrial systems for tagging, tracking, and sensing are only a few examples from the tremendous variety of such systems. However, unleashing these systems from the power grid also means that the available energy is a limited resource that must be conserved and managed prudently.

The estimation of energy consumption by the communication system differs significantly from the other components, as it is strongly dependent on external influences. These include the quality of the radio channel, the channel access scheme, and the utilization of the shared transmission medium by other participants, who are often not part of the actual system. Data transmissions can last longer, require a higher transmission power, or fail due to collisions, so that they have to be repeated. The consequence is a longer activity time of the transceiver and a shorter dwell time in the efficient power saving mode. Therefore, realistic simulation models are required at design time, which take into account the properties of the communication interface as well as the external environment.

In the following, methods for modeling power consumption for different communication technologies are discussed. This includes decentralized narrow-band communication in the Short Range Devices (SRD) band and the comprehensive modeling of cellular technologies such as Long Term Evolution (LTE), LTE-Advanced (LTE-A) and Narrow Band Internet of Things (NB-IoT) by a Context-Aware Power Consumption Model (CoPoMo).

It is shown that a decentralized channel access with brisk activity on the radio channel leads to an increased power consumption of all waiting subscribers, if the channel occupancy is to be tracked continuously to keep the transmission latency as low as possible. Conversely, in centrally organized cellular radio networks, the energy consumption of the User Equipment (UE) is dominated by uplink transmissions, especially when high transmission power is required. The proportion for reception, however, depends mainly on the duration of the transmission. In fact, adding an additional reception path via Carrier Aggregation (CA) not only increases the data rate, but also reduces the energy consumption of the UE .

Since the knowledge of the transmit power is essential for the estimation of the power consumption, but most UEs do not provide this information to the application layer, a Machine Learning (ML)-based method for estimating the transmit power from

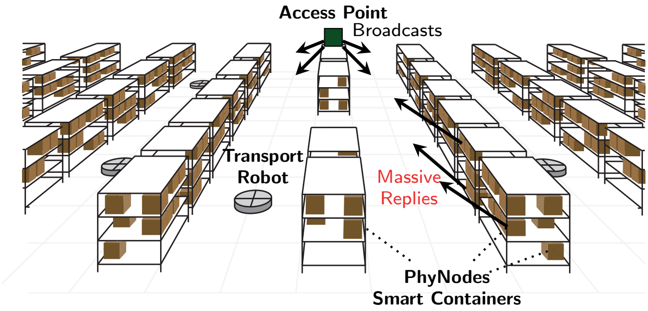


Fig. 9.5: Warehouse scenario. ©[2018] IEEE. Reprinted, with permission, from [206].

the available parameters such as strength and quality of the received signal, is also presented.

9.2.1 Introduction

Instead of treating inventory items as static resources, future intelligent warehouses will turn containers to become Cyber Physical Systems (CPSs) that actively and autonomously participate in the optimization of the logistical processes. Consequently, new challenges that are system-immanent for the massive Internet of Things (IoT), such as channel access in a shared communication medium, have to be addressed.

An example of such a warehouse scenario is shown schematically in Figure 9.5. A wide variety of autonomous transport systems are used to transport goods into or out of the warehouse. The individual goods are stored in smart containers that can provide information about their current contents and the goods they contain at any time by radio. Energy supply is a particular challenge for the embedded systems used for this purpose. Mains and battery operation are ruled out due to the size of the location, so that the platforms must obtain their energy for operation and communication through *energy harvesting*, using photovoltaic, say, and must manage it extremely efficiently.

To fetch a specific inventory, distributed Access Points (APs) transmit inventory queries to the warehouse. These are answered by containers with matching contents, specifying the quantity contained. Subsequently, the transport systems bring the requested quantity to a picking point for further use.

Since such requests can lead to massive replies depending on the distribution of goods and inventory, channel access must be coordinated to avoid collisions during transmission. Distributed channel access methods quickly reach their capacity limits and increase the energy consumption of network subscribers due to collisions, multiple transmissions, and prolonged waiting for a free channel. In this area, CRC 876 has developed and brought together innovative methods for recording, analyzing, and optimizing energy consumption [206, 209], which are discussed in the following subsections.

This contrasts with mobile communications networks that have centrally organized channel access, which are discussed later in this section. If we accept the restriction of operating only one specific technology on a frequency band, higher spectral efficiency can be achieved in return. Techniques such as central power control or inter-cell interference coordination enable resource-efficient transmission even at high subscriber density. Numerous studies of the CRC 876 have shown that the energy consumption of current mobile radio terminals (UE) is dominated by the transmission of data, especially at high transmission power. The specially developed CoPoMo enables a wide variety of trade-offs, e.g., between transmission time, energy consumption, and spectral resource requirements, for different frequency ranges, building densities, and mobility profiles. Section 9.2.4 introduces the basic concepts of CoPoMo and presents two studies, one dealing with a trade-off between transmission bandwidth and energy consumption, and the other presenting an ML-based method for the UE-based estimation of transmission power using available quality indicators.

9.2.2 Power Consumption with Distributed Channel Access

In unlicensed bands, a distributed channel access method is often used to enable fair coexistence of different technologies. These bands include the Industrial Scientific Medical (ISM) band at 2.4 GHz and the SRD band at 868 MHz. The latter is used for the communication of the PhyNode. Distributed channel access is based on the Listen Before Talk (LBT) principle, which is known in a similar form as Carrier Sense Multiple Access Collision Avoidance (CSMA/CA) for WLAN and ZigBee networks. For the SRD band, channel access is specified by European Telecommunications Standards Institute (ETSI), which is shown schematically in Figure 9.6. Stations with a transmission intent hold back their transmission until the transmission channel is free. When the channel becomes free, the system waits an additional backoff time $t_L = t_F + t_{PS}$ with $t_F = 5$ ms. Thereby t_{PS} is randomly selected for each startup in the interval 0 ms to 5 ms. If the channel is still free after t_L has elapsed, the transmission is carried out. Otherwise, the system waits again for a free channel including a newly selected backoff time t_L. For acceleration in case of low channel utilization, a station can set t_{PS} to 0 ms for the first transmission attempt if the channel is continuously free between the initial transmission request and the expiration of t_F.

Figure 9.7 shows the channel occupation by three stations in the radio spectrum. At the beginning of the recording, the channel is continuously occupied by a jammer. The three stations already have a transmission intent and hold back their transmission for the time being. After switching off the jammer, the three stations transmit one after the other according to the access scheme and the random backoff intervals.

However, short s result in low channel utilization and thus in reduced spectral efficiency due to the relatively long waiting times. In addition, the channel access method requires continuous monitoring of the radio channel between the arrival of the

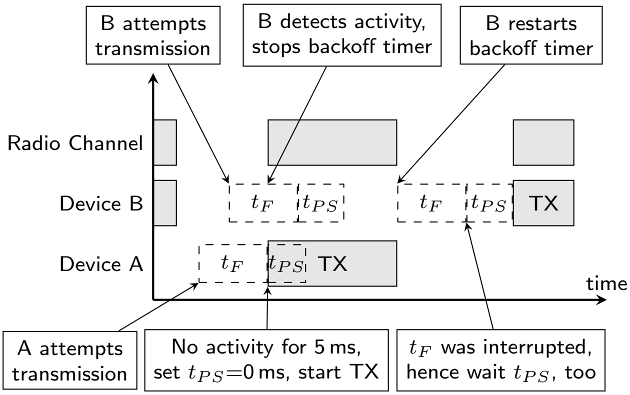

B attempts transmission
B detects activity, stops backoff timer
B restarts backoff timer
Radio Channel
Device B
Device A
t_F
t_{PS}
TX
time
A attempts transmission
No activity for 5 ms, set t_{PS}=0 ms, start TX
t_F was interrupted, hence wait t_{PS}, too

Fig. 9.6: Timeline of the LBT access scheme. Dashed blocks represent back-off intervals and solid blocks indicate an occupied channel by a transmission over the air. ©[2017] IEEE. Reprinted, with permission, from [209].

transmission intent and the actual execution of the transmission. The duration of this monitoring increases with the utilization of the channel. Since the receive circuits must be active during this time, the power consumption of all competing stations increases significantly.

Figure 9.8 shows the distribution of the transceiver's energy consumption as a function of the number of simultaneously active devices responding to 10 product requests in the warehouse scenario (cf. Figure 9.5). The energy accounting is obtained from an energy-aware driver model. The measurement includes the constant part for the reception of the 10 requests and an additional message to terminate the measurement after 11.75 s, as well as the variable energy consumption for sending the replies. Compared with the empty channel, the energy consumption increases by up to a factor of 10 in case of more than 30 stations.

To enable an optimization of energy consumption given the scarce resource, a simulative hardware-in-the-loop design space exploration framework was developed, which is discussed in more detail in the following section.

9.2.3 Simulative Access-Scheme Optimization

In this section, we present a multi-methodological system model that brings together testbed experiments for measuring real hardware properties and simulative evaluations for large-scale considerations [206]. As a case study, we focus on parametrization of the 802.15.4-based radio communication system, which has to be energy-efficient due to the scarce amount of harvested energy, but avoid latencies for the maintenance of scalability of the overlaying warehouse system. The results show that a modification of

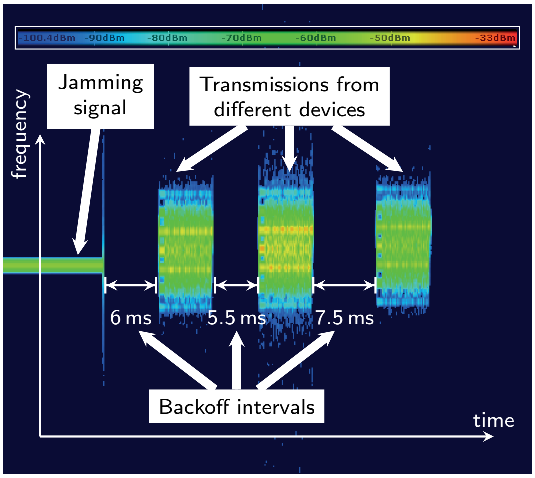


Fig. 9.7: Spectral view of LBT scheme in action. Three stations organize their pending transmissions when the channel becomes free (simulated by disabling a jamming signal).

the initial backoff time can lead to both energy and time savings in the order of 50 % compared with the standard.

Figure 9.9 shows the underlying modeling principle. On the right side is the physical system in the form of the PhyNetLab, in which a field evaluation can be performed [206]. The left side comprises the OMNeT++ simulation system, which models the communication system including the application layer in the form of a simulation. For a given system scenario with information on energy consumption and dwell time of individual operating states, data volume, available energy, and the number of network subscribers, a simulative optimization of the communication system is carried out that enables a trade-off between conflicting target variables, e.g. latency and energy consumption. This configuration is transferred to the physical testbed and evaluated in field experiments.

The energy consumption of the communication system of the PhyNode is modeled in the simulation as a state machine with four states (cf. Figure 9.10). In the LISTEN state, the device periodically listens on the channel for preambles that indicate the beginning of a new packet for reception. When this occurs, the transceiver enters receive mode (RX) to receive the packet and then returns to LISTEN. If a packet is to be transmitted, it enters the BACKOFF state with repeated short dwelling times in the RX mode to wait for the free channel. After the backoff timer expires, it finally sends the packet in TX mode. The consumption values of the individual energy states can be automatically captured and fed into the simulation using the hardware-in-the-loop approach and the energy-aware driver models.

Based on the presented framework, the channel access procedure can be optimized for a given warehouse scenario. One of the most common processes in a self-inventory

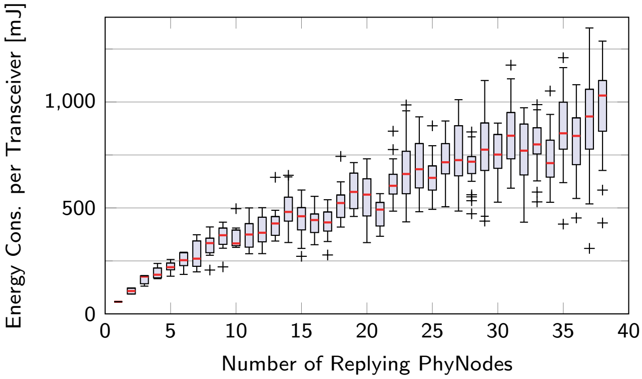


Fig. 9.8: Energy consumption of the radio transceiver for receiving 11 packets (queries) and transmitting 10 replies in a constant interval of 11.75s. ©[2017] IEEE. Reprinted, with permission, from [209].

warehouse is the request for specific products in a required quantity. For this purpose, a request is sent out as a broadcast and answered by matching containers, i.e. the communication systems located on them. Depending on the equipment of the warehouse and the usually requested quantity, only a fraction of the responses is sufficient to fulfill the requested product quantity. This value is called the Minimum Query Response Ratio ($\mathrm{QRR}_{\mathrm{min}} \in [0, 1]$).

The issued requests cause a large number of participants to attempt to send their responses at the same time. They select a random backoff and then send their packets. However, if the backoff can take on only a few discrete values compared with the number of subscribers, collisions inevitably occur, so that transmissions have to be repeated and the response time until $\mathrm{QRR}_{\mathrm{min}}$ is reached is increased as a result. To resolve such collisions, the 802.15.4 standard defines an exponential increase of the backoff window in the form of a backoff exponent BE, which is successively increased in case of collisions and then reset to the initial value BE_0. The minimum backoff exponent BE_0 is an optimization parameter that has to be chosen depending on the expected number of participants and the permitted delay in case of a smaller number of participants.

Figure 9.11 shows the exemplary application of the framework to optimize the initial backoff exponent BE_0 for different $\mathrm{QRR}_{\mathrm{min}}$.

The results show that for $\mathrm{QRR}_{\mathrm{min}} = 0.8$, an initial $\mathrm{BE}_0 = 8$ compared with $\mathrm{BE}_0 = 3$ for 420 responding nodes reduces the energy consumption by 49 % while reducing the time to fulfill the request by 56 %. However, even with smaller numbers of nodes, choosing a larger BE_0 has a positive effect on both objectives. However, at lower $\mathrm{QRR}_{\mathrm{min}} = 0.2$, a larger BE_0 leads to higher energy consumption in favor of reduced response time.

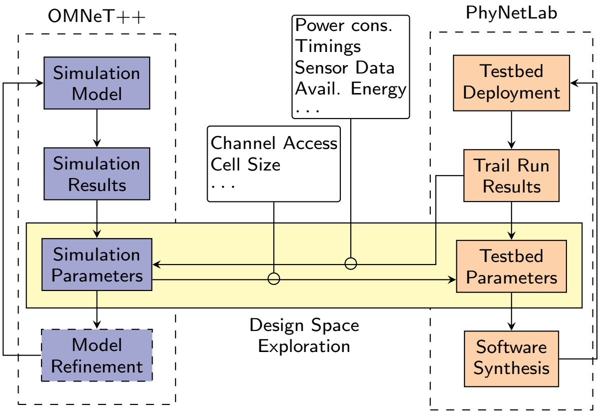
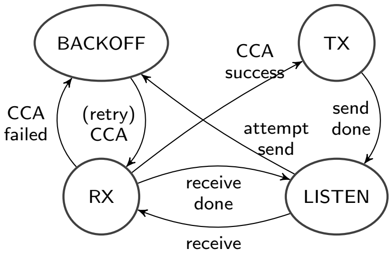


Fig. 9.9: System model for design-space exploration. ©[2018] IEEE. Reprinted, with permission, from [206].



State	Avg. Power
BACKOFF	1.5 μW
LISTEN	4.5 mW
RX	70 mW
TX	138 mW

Fig. 9.10: State machine model of the transceiver. ©[2018] IEEE. Reprinted, with permission, from [206].

9.2.4 Power Consumption in Cellular Networks

Due to the increasing spread and popularity of cellular radio networks for networking the smallest mobile devices, the analysis and optimization of energy efficiency is also gaining importance in this domain. Centralized control by static infrastructure, i.e. by the base station and backhaul, enables resource optimization without the intervention of the end devices. For example, it can be considered whether distant stations with poor channel conditions receive a greater short-term pensum of spectral resources to perform their transmission than stations with good channel conditions that can still achieve high data rates using lower transmit powers.

Optimizations of power consumption, however, require precise power consumption models that on the one hand provide accurate estimates and yet can be calculated efficiently. For this purpose, CRC 876 has contributed the CoPoMo [192], a Markovian power consumption model for calculating the power consumption of current LTE and LTE-A terminals. The calculation takes into account device-specific consumption char-

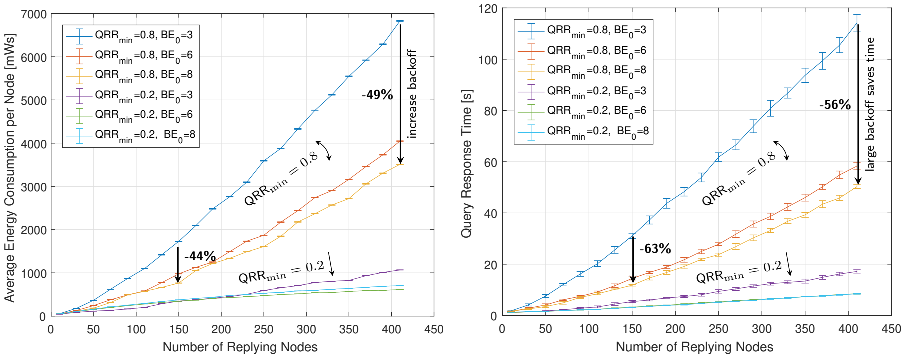


Fig. 9.11: Simulation results of the energy consumption (left) and query response time (right) as a function of the number of concurrently replying nodes for different backoff configurations and minimum query response ratio. ©[2018] IEEE. Reprinted, with permission, from [206].

acteristics as well as spectral resource utilization, the frequency range used, mobility, built environment, and the type of data traffic.

The following sections introduce the basic concepts of CoPoMo, and then present extensions and case studies for resource optimization.

9.2.4.1 Context-Aware Power Consumption Modeling

This section introduces the basic concepts of CoPoMo [192]. As in all communication systems, the power consumption of a UE depends on the current operating state, which in turn is influenced by numerous context and system parameters. The power consumption is caused by the digital signal processing and the operation of the High Frequency (HF) components for receiving and transmitting the radio signals. Due to the use of Application-Specific Integrated Circuits (ASICs), the signal processing is characterized by a relatively low power consumption, which scales only insignificantly with the effective data throughput, and can thus be assumed to be constant in many cases during an ongoing transmission. The consumption by the HF receiver is also usually not influenced by the received field strength and the effective data throughput, and thus also assumes a constant value for the respective frequency band during reception [191].

The power consumption of the UE is dominated by the consumption of the power amplifier for the transmission of messages in the uplink, especially at high transmission powers for overcoming a high path loss [191]. Figure 9.12 shows the average power consumption of a smartphone as a function of the transmit power of the power amplifier. The measurement covers the entire system, i.e. including the main processor, display, signal processing and all active HF components. Background activities and display brightness were reduced to a constant minimum.

A small increase in power at low transmitting powers and a steep increase in power consumption at high transmitting powers can be observed. The reason for this is the

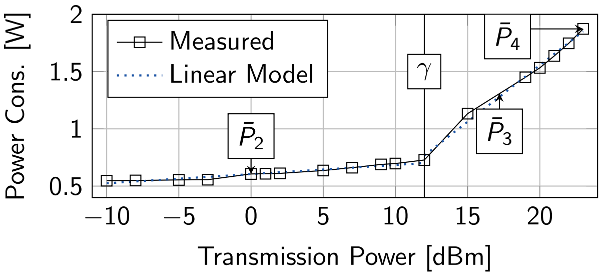


Fig. 9.12: Power consumption of a Samsung Galaxy S5 smartphone at 800 MHz in relation to transmission power. ©[2017] IEEE. Reprinted, with permission, from [208].

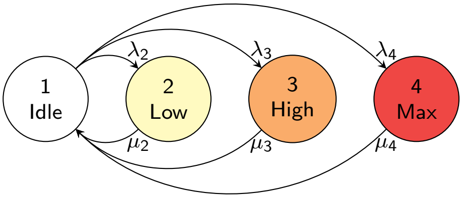


Fig. 9.13: Markovian power state model of LTE User Equipment (UE). ©[2017] IEEE. Reprinted, with permission, from [208].

typical use of two different power amplifiers with different operating ranges, which are switched to increase efficiency depending on a threshold value γ. Since the power consumption within the respective operating ranges is approximately linear, CoPoMo uses two linear models for power estimation, consisting of the respective slopes α, the y-intercept β, and the switching point γ, which are determined by empirical measurement series for each system and frequency band.

Further investigation has shown that the power consumption can be accurately estimated using four reference points of the linear model $\bar{P}_1$, $\bar{P}_2$, $\bar{P}_3$ and $\bar{P}_4$, so that the power consumption of an LTE UE can be described by a state model consisting of four corresponding states [190]. The state model is shown in Figure 9.13. State 1 represents the power consumption in idle state without outgoing data transmission. State 2 represents transmission with low transmit power (the point at 0 dB), state 3 represents transmission with high transmit power (midpoint between γ and maximum transmit power), and state 4 represents transmission with maximum transmit power of 23 dBm.

Transitions between states are given in terms of transition probabilities λ_i and μ_i and always lead over state 1, since state 2, 3, and 4 present states with outgoing transmission, during which the UE remains in this current state and is not able to switch into a different transmission-related power state. The state transitions are obtained

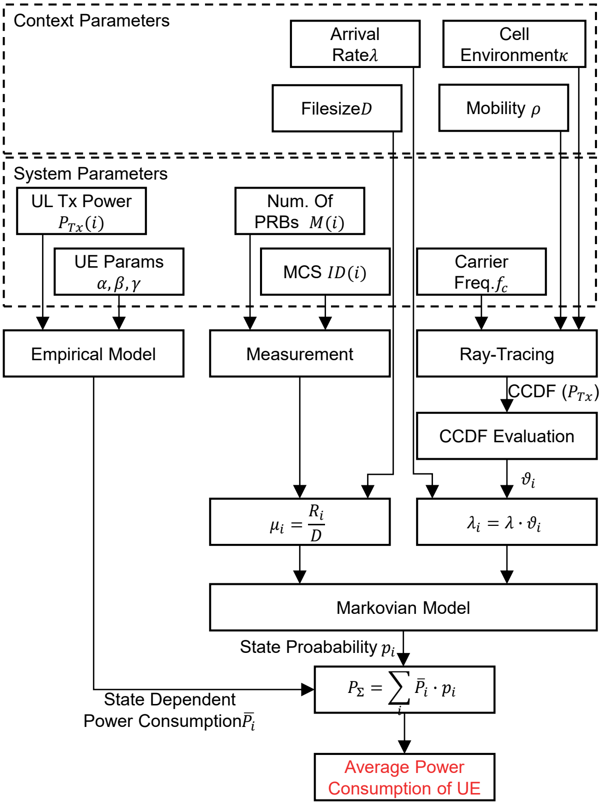


Fig. 9.14: Overview of CoPoMo. ©[2013] IEEE. Reprinted, with permission, from [192].

from the augmented overall model, which is shown in Figure 9.14. λ_i is calculated as $\lambda_i = \lambda \cdot \vartheta_i$, where $\frac{1}{\lambda}$ corresponds to the arrival rate of outgoing data transmissions and ϑ_i with $\sum_{i=2}^{4} \vartheta_i = 1$ indicates the distribution of the residence time in states 2, 3, and 4. The latter depends on the cell environment κ, mobility ρ, and carrier frequency f_c, and can be determined, say, by ray-tracing analysis and the statistical evaluation of path loss. μ_i is the inverse of the service rate and is calculated as $\mu_i = \frac{R_i}{D}$ with average file size D and the average uplink data rate R_i achieved in state i. The data rate in turn depends on the number of allocated RBs $M(i)$ and the Modulation and Coding Scheme (MCS) $ID(i)$, which are dynamically allocated according to the base station's scheduling strategy.

Finally, state probabilities can be determined from the transition probabilities, which then describe the average residence time in each state. The average power consumption of the UE can be determined together with the state-specific power consumption.

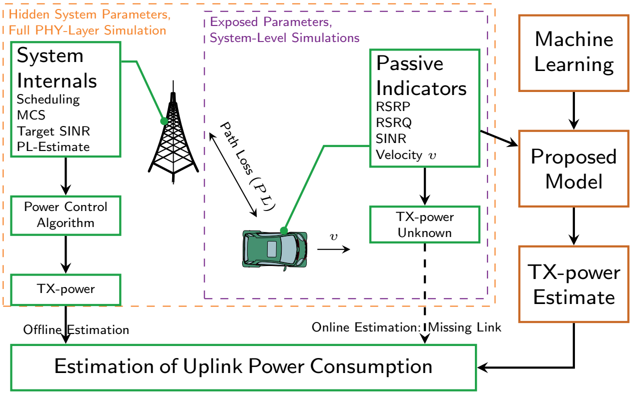


Fig. 9.15: Uplink power control of UE and its underlying system parameters are typically hidden to the application layers. However, this knowledge is crucial for predictions and estimations of the involved power consumption in energy-aware applications and system-level simulations. The proposed model derives this information from passive connectivity indicators. ©[2018] IEEE. Reprinted, with permission, from [207].

9.2.5 Uplink Power Prediction with Machine Learning

This section summarizes the work on ML-based uplink power prediction according to [207]. Energy-aware system design is an important optimization task for static and mobile IoT-based sensor nodes, especially for highly resource-constrained vehicles such as mobile robotic systems. For 4G/5G-based cellular communication systems, the effective transmission power of uplink data transmissions is of crucial importance for the overall system power consumption. Unfortunately, this information is usually hidden within off-the-shelf modems and mobile handsets and can therefore not be exploited for green communications. Moreover, the dynamic transmission power control behavior of the mobile device is not explicitly modeled in most of the established simulation frameworks.

In order to close this gap, we present a novel machine learning-based approach for forecasting the uplink transmission power used for data transmissions based on available passive network quality indicators and application-level information. A schematic illustration of the proposed solution approach is shown in Figure 9.15. The key idea is to leverage an SDR (Software-Defined Radio)-based measurement setup—capable of simultaneously determining the uplink transmission power P_{TX} and different network context indicators—in order to derive a machine learning-based prediction model that infers P_{TX} from the context measurements. This model can then be deployed to other platforms that are not capable of determining P_{TX} on their own.

The required machine learning model is derived from comprehensive field measurements of drive tests performed in a public cellular network and can be parameterized for integrating all measurements that a given target platform is able to provide for the

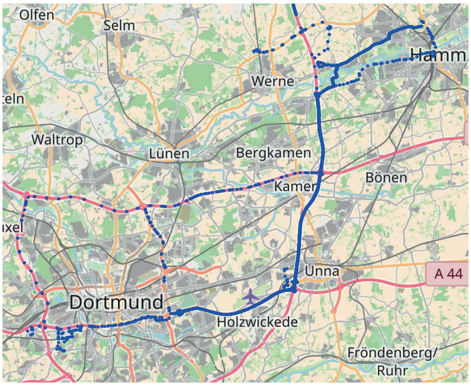


Fig. 9.16: Road map with locations of all data samples of the measurement campaign between two larger cities in Germany. Each blue point represents an intermediate status logging of all measured variables (cf. Table 9.3) during ongoing uplink transmissions. (Map: ©OpenStreetMap contributors, CC BY-SA). ©[2018] IEEE. Reprinted, with permission, from [207].

prediction process. Figure 9.16 shows a road map of the measurement points along a vehicular trajectory that covers urban, suburban, and rural environments. In total, 6172 have been acquired during the real world measurements. In focusing on the platform's sensing capabilities, we considered four different variants of feature sets. A summary of the feature sets and the implied impact factors of the contained features is given in Table 9.3.

For performing the actual prediction task, different regression models are considered:

Random Forest with 64 trees and a maximum depth of 32.

Deep Learning with three fully connected hidden layers, 64 neurons per layer, and Rectified Linear Unit (ReLU) activation function.

Ridge Regression with 12 model parameters (one per feature plus one bias term).

The results of the 10-fold cross validation are summarized in Figure 9.17. While the differences between the feature set variants are comparably small, larger differences between the machine learning models can be observed. The Random-Forest models thoroughly performed best with a mean average error of 3.166 dB. It can also be seen that the standard deviation between the different cross validation runs is small, which indicates a good model fit to unknown and independent data.

Tab. 9.3: Captured features and association with application-specific prediction models based on full-feature set 𝔽, practical sets ℙ1/ℙ2, and simulation set 𝕊. ©[2018] IEEE. Reprinted, with permission, from [207].

Parameter	Model	Indicated Influences(s)
Velocity	𝔽,ℙ1,ℙ2,𝕊	Distortions by fast fading
Upload size	𝔽,ℙ1,ℙ2,𝕊	Influence of TCP slow start
RSRP	𝔽,ℙ1,ℙ2,𝕊	Signal strength, distance
RSRQ, SINR	𝔽,ℙ1,ℙ2	Signal clarity, interference
Datarate	𝔽,ℙ1	Signal strength, allocated RBs *M*
RSSI	𝔽	Signal strength, distance
Frequency band	𝔽	Environment [349]
Number of neighbor cells (intra/inter freq.)	𝔽	Environment, cell density, interference
Cell bandwidth	𝔽	Exhaustion of TX-power headroom

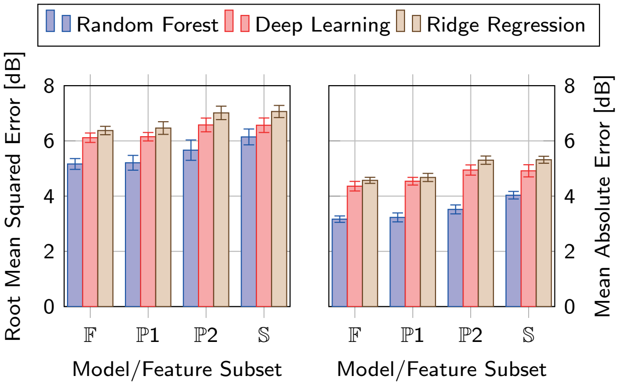


Fig. 9.17: Cross-validated error of trained prediction models for each feature subset (𝔽, ℙ1, ℙ2, 𝕊) and each machine learning method (Random Forest, Deep Learning, Ridge Regression) in terms of Root Mean Squared Error (RMSE) (left) and Mean Absolute Error (MAE) (right). Lower is better. ©[2018] IEEE. Reprinted, with permission, from [207].

Bibliography

[1] D. J. Abadi et al. “Aurora: A New Model and Architecture for Data Stream Management”. In: *The VLDB Journal – The Int. Journal on Very Large Data Bases* 12.2 (Aug. 2003), pp. 120–139 (cit. on p. 21).

[2] M. R. Ackermann, J. Blömer, and C. Sohler. “Clustering for Metric and Nonmetric Distance Measures”. In: *ACM Transactions on Algorithms* 6.4 (2010). Previously appeared in the Procs. of the Symposium on Discrete Algorithms 2008, 59:1–59:26 (cit. on pp. 201, 203, 204).

[3] C. Adam-Bourdarios, G. Cowan, C. Germain, I. Guyon, B. Kégl, and D. Rousseau. “The Higgs boson machine learning challenge”. In: *Advances in Neural Information Processing Systems: Procs. of the Workshop on High-energy Physics and Machine Learning 2015*. 2015, pp. 19–55 (cit. on p. 250).

[4] A. Adiththan, S. Ramesh, and S. Samii. “Cloud-assisted control of ground vehicles using adaptive computation offloading techniques”. In: *Procs. of the Design, Automation and Test in Europe Conference 2018*. Mar. 2018, pp. 589–592 (cit. on p. 363).

[5] P. K. Agarwal, R. Ben Avraham, H. Kaplan, and M. Sharir. “Computing the Discrete Fréchet Distance in Subquadratic Time”. In: *SIAM Journal on Computing* 43.2 (2014). Previously appeared in the Procs. of the Symposium on Discrete Algorithms 2013, pp. 429–449 (cit. on p. 199).

[6] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. “Approximating extent measures of points”. In: *Journal of the Association for Computing Machinery* 51.4 (2004), pp. 606–635 (cit. on pp. 87, 88).

[7] P. K. Agarwal and R. Sharathkumar. “Streaming Algorithms for Extent Problems in High Dimensions”. In: *Algorithmica* 72.1 (2015), pp. 83–98 (cit. on p. 87).

[8] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy. “X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.12 (2018) (cit. on p. 4).

[9] R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma. “Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages”. In: *Procs. of the Int. Conference on World Wide Web 2013*. 2013, pp. 13–24 (cit. on p. 272).

[10] B. Ahmadi, K. Kersting, and S. Natarajan. “Lifted Online Training of Relational Models with Stochastic Gradient Methods”. In: *Procs. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2012*. Ed. by P. Flach, T. D. Bie, and N. Cristianini. Vol. 7523. Bristol, UK: Springer, Sept. 2012, pp. 585–600. URL: http://link.springer.com/chapter/10.1007/978-3-642-33460-3_43 (cit. on p. 407).

[11] R. Ahmed, B. Buchli, S. Draskovic, L. Sigrist, P. Kumar, and L. Thiele. “Optimal Power Management with Guaranteed Minimum Energy Utilization for Solar Energy Harvesting Systems”. In: *ACM Transactions on Embedded Computing Systems* 18.4 (2019), pp. 1–26 (cit. on p. 47).

[12] H.-K. Ahn, H. Alt, M. Buchin, E. Oh, L. Scharf, and C. Wenk. “Middle curves based on discrete Fréchet distance”. In: *Computational Geometry: Theory and Applications* 89 (2020). Previously appeared in LATIN 2016, p. 101621 (cit. on pp. 197, 201, 202, 204, 205).

[13] Y. Akhremtsev, P. Sanders, and C. Schulz. “High-Quality Shared-Memory Graph Partitioning”. In: *IEEE Transactions on Parallel and Distributed Systems* 31.11 (2020), pp. 2710–2722 (cit. on p. 152).

[14] H. Alt and M. Godau. “Computing the Fréchet distance between two polygonal curves”. In: *Int. Journal of Computational Geometry and Applications* 5 (1995), pp. 75–91 (cit. on pp. 199, 211).

https://doi.org/10.1515/9783110785944-010

[15] F. Álvaro Muñoz, J. Sánchez Peiró, and J. Benedí Ruiz. “Recognition of On-line Handwritten Mathematical Expressions Using 2D Stochastic Context-Free Grammars and Hidden Markov Models”. In: *Pattern Recognition Letters* (2014), pp. 58–67 (cit. on p. 162).

[16] M. R. Anderberg. *Cluster analysis for applications*. Academic Press, 1973 (cit. on pp. 218, 222).

[17] H. Anderhub et al. “Design and operation of FACT - the first G-APD Cherenkov telescope”. In: *Journal of Instrumentation* 8.06 (June 2013), P06008. URL: http://iopscience.iop.org/1748-0221/8/06/P06008/ (cit. on pp. 351, 352).

[18] A. Andrae and T. Edler. “On Global Electricity Usage of Communication Technology: Trends to 2030”. In: *Challenges* 6.1 (2015), pp. 117–157 (cit. on p. 2).

[19] S. I. Ao et al. “CLUSTAG: hierarchical clustering and graph methods for selecting tag SNPs”. In: *Bioinformatics* 21.8 (2005), pp. 1735–1736 (cit. on p. 216).

[20] A. Arasu, S. Babu, and J. Widom. “The CQL Continuous Query Language: Semantic Foundations and Query Execution”. In: *The VLDB Journal – The Int. Journal on Very Large Data Bases* 15.2 (June 2006), pp. 121–142 (cit. on p. 21).

[21] A. Arasu et al. *STREAM: The Stanford Data Stream Management System*. Tech. rep. 2004-20. Stanford InfoLab, 2004. URL: http://ilpubs.stanford.edu:8090/641/ (cit. on p. 24).

[22] A. Arcangeli, I. Eidus, and C. Wright. “Increasing memory density by using KSM”. In: *Procs. of the Ottawa Linux Symposium 2009*. 2009, pp. 19–28 (cit. on p. 307).

[23] M. Arjovsky, S. Chintala, and L. Bottou. “Wasserstein Generative Adversarial Networks”. In: *Procs. of the Int. Conference on Machine Learning 2017*. 2017, pp. 214–223. URL: http://proceedings.mlr.press/v70/arjovsky17a.html (cit. on p. 260).

[24] ARM Limited. *Introducing NEON - Development Article*. URL: https://developer.arm.com/documentation/dht0002/a/ (cit. on p. 408).

[25] D. Arthur and S. Vassilvitskii. “*k*-means++: The advantages of careful seeding”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2007*. Association for Computing Machinery, 2007, pp. 1027–1035 (cit. on p. 188).

[26] N. Asadi, J. Lin, and A. P. De Vries. “Runtime optimizations for tree-based machine learning models”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.9 (2014), pp. 2281–2292 (cit. on p. 340).

[27] P. Axer and R. Ernst. “Stochastic response-time guarantee for non-preemptive, fixed-priority scheduling under errors”. In: *Procs. of the Design Automation Conference 2013*. 2013, 172:1–172:7 (cit. on p. 363).

[28] B. Weisfeiler and A. Leman. “The reduction of a graph to canonical form and the algebra which appears therein”. In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968). English translation by G. Ryabov is available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf, pp. 12–16 (cit. on pp. 117, 118).

[29] R. Babbar and B. Schölkopf. “Data scarcity, robustness and extreme multi-label classification”. In: *Machine Learning* 108.8 (2019), pp. 1329–1351 (cit. on pp. 275, 281, 284).

[30] R. Babbar and B. Schölkopf. “Dismec: Distributed sparse machines for extreme multi-label classification”. In: *Procs. of the ACM Int. Conference on Web Search and Data Mining 2017*. 2017, pp. 721–729 (cit. on p. 275).

[31] K. Bache and M. Lichman. *UCI Machine Learning Repository*. 2013. URL: http://archive.ics.uci.edu/ml (cit. on pp. 351, 352).

[32] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. “Streaming submodular maximization: Massive data summarization on the fly”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2014*. 2014 (cit. on pp. 76, 77).

[33] M. Badoiu and K. L. Clarkson. “Optimal core-sets for balls”. In: *Computational Geometry: Theory and Applications* 40.1 (May 2008), pp. 14–22 (cit. on p. 87).

[34] M. Badoiu and K. L. Clarkson. “Smaller core-sets for balls”. In: *Procs. of the Symposium on Discrete Algorithms 2003*. 2003, pp. 801–802 (cit. on p. 87).

[35] M. Badoiu, S. Har-Peled, and P. Indyk. “Approximate clustering via core-sets”. In: *Procs. of the ACM Symposium on Theory of Computing 2002*. 2002, pp. 250–257 (cit. on pp. 86, 87).

[36] K. R. Baker, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. “Preemptive Scheduling of a Single Machine to Minimize Maximum Cost Subject to Release Dates and Precedence Constraints”. In: *Operations Research* 31.2 (1983), pp. 381–386. DOI: https://doi.org/10.1287/opre.31.2.381 (cit. on p. 366).

[37] T. P. Baker. “Stack-based Scheduling of Realtime Processes”. In: *Real-Time Systems* 1 (1991), pp. 67–99 (cit. on p. 361).

[38] V. Balntas, E. Riba, D. Ponsa, and K. Mikolajczyk. “Learning local feature descriptors with triplets and shallow convolutional neural networks”. In: *Procs. of the British Machine Vision Conference 2016*. Ed. by R. C. Wilson, E. R. Hancock, and W. A. P. Smith. BMVA Press, 2016, pp. 119.1–119.11 (cit. on p. 168).

[39] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-hashimi, D. Brunelli, and L. Benini. “Hibernus : Sustaining Computation during Intermittent Supply for Energy-Harvesting Systems”. In: *IEEE Embedded Systems Letters* 7.1 (2015) (cit. on p. 53).

[40] A.-L. Barabasi and Z. N. Oltvai. “Network biology: Understanding the cell’s functional organization”. In: *Nature Reviews Genetics* 5.2 (2004), pp. 101–113 (cit. on pp. 116, 144).

[41] A. Barbalace, A. Iliopoulos, H. Rauchfuss, and G. Brasche. “It’s Time to Think About an Operating System for Near Data Processing Architectures”. In: *Procs. of the Workshop on Hot Topics in Operating Systems 2017*. HotOS ’17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 56–61. DOI: https://doi.org/10.1145/3102980.3102990 (cit. on p. 17).

[42] A. Barbalace, J. Picorel, and P. Bhatotia. “ExtOS: Data-Centric Extensible OS”. In: *Procs. of the ACM SIGOPS Asia-Pacific Workshop on Systems 2019*. APSys ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 31–39. DOI: https://doi.org/10.1145/3343737.3343742 (cit. on p. 17).

[43] R. C. Barros, A. C. P. L. F. de Carvalho, and A. A. Freitas. “Decision-Tree Induction”. In: *Automatic Design of Decision-Tree Induction Algorithms*. Springer International Publishing, 2015, pp. 7–45 (cit. on p. 340).

[44] P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe. “Convexity, classification, and risk bounds”. In: *Journal of the American Statistical Association* 101.473 (2006), pp. 138–156 (cit. on p. 274).

[45] L. Becchetti, M. Bury, V. Cohen-Addad, F. Grandoni, and C. Schwiegelshohn. “Oblivious dimension reduction for *k*-means: beyond subspaces and the Johnson-Lindenstrauss lemma”. In: *Procs. of the ACM SIGACT Symposium on Theory of Computing 2019*. Ed. by M. Charikar and E. Cohen. ACM, 2019, pp. 1039–1050. DOI: 10.1145/3313276.3316318. URL: https://doi.org/10.1145/3313276.3316318 (cit. on p. 213).

[46] S. Ben-Amor, D. Maxim, and L. Cucu-Grosjean. “Schedulability analysis of dependent probabilistic real-time tasks”. In: *Procs. of the Int. Conference on Real-Time Networks and Systems 2016*. 2016, pp. 99–107 (cit. on p. 363).

[47] J. L. Bentley and J. B. Saxe. “Decomposable Searching Problems I: Static-to-Dynamic Transformation”. In: *Journal of Algorithms* 1.4 (1980), pp. 301–358 (cit. on p. 87).

[48] E. Bernhardsson. *Annoy: Approximate Nearest Neighbors in C++/Python*. Python package version 1.13.0. 2018. URL: https://pypi.org/project/annoy/ (cit. on pp. 174, 175).

[49] N. Bertram, J. Ellert, and J. Fischer. “A Parallel Framework for Approximate Max-Dicut in Partitionable Graphs”. In: *Procs. of the Int. Symposium on Experimental Algorithms 2022*. Ed. by C. Schulz and B. Uçar. Vol. 233. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 10:1–10:15. DOI: 10.4230/LIPIcs.SEA.2022.10. URL: https://doi.org/10.4230/LIPIcs.SEA.2022.10 (cit. on p. 145).

[50] K. Bhatia, K. Dahiya, H. Jain, Y. Prabhu, and M. Varma. *The Extreme Classification Repository: Multi-label Datasets and Code*. 2016. URL: http://manikvarma.org/downloads/XC/XMLRepository.html (cit. on pp. 274, 282, 283).

[51] K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain. "Sparse Local Embeddings for Extreme Multi-Label Classification". In: *Procs. of the Int. Conference on Neural Information Processing Systems 2015*. NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 730–738 (cit. on p. 283).

[52] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi. "Graph Neural Networks with Convolutional ARMA Filters". In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1901.01343 (cit. on p. 136).

[53] B. Bischl, J. Bossek, D. Horn, and M. Lang. *Model-Based Optimization for mlr*. 2014. URL: https://github.com/berndbischl/mlrMBO (cit. on pp. 289, 294).

[54] B. Bischl, S. Wessing, N. Bauer, K. Friedrichs, and C. Weihs. "MOI-MBO: Multiobjective Infill for Parallel Model-Based Optimization". In: *Procs. of the Learning and Intelligent Optimization Conference 2014*. 2014 (cit. on p. 287).

[55] B. Bischl et al. "Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges". In: *arXiv: Computing Research Repository* (2021). DOI: arXiv:2107.05847v2 (cit. on p. 286). **SFB876-A3**

[56] G. E. Blelloch. *Prefix Sums and Their Applications*. Tech. rep. Carnegie Mellon University, 1990 (cit. on p. 394).

[57] J. Blitzstein and P. Diaconis. "A sequential importance sampling algorithm for generating random graphs with prescribed degrees." In: *Internet Mathematics* 6.4 (2011), pp. 489–522 (cit. on p. 416).

[58] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson. "A Flexible Real-Time Locking Protocol for Multiprocessors". In: *Procs. of the IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications 2007*. 2007 (cit. on pp. 361, 362, 368).

[59] S. Boettcher and A. G. Percus. "Extremal Optimization: Methods Derived from Co-Evolution". In: *Procs. of the Conference on Genetic and Evolutionary Computation 1999*. GECCO'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 825–832 (cit. on p. 145).

[60] J. Bolte, S. Sabach, and M. Teboulle. "Proximal alternating linearized minimization for nonconvex and nonsmooth problems". In: *Mathematical Programming* 146.1-2 (2014), pp. 459–494 (cit. on pp. 233, 234).

[61] P. Boncz, T. Neumann, and O. Erling. "TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark". In: *Procs. of the Technology Conference on Performance Evaluation and Benchmarking 2018*. Springer, 2018, pp. 61–76 (cit. on p. 400).

[62] H. Borchers. *adagio: Discrete and Global Optimization Routines*. R package version 0.6.5. 2016. URL: https://CRAN.R-project.org/package=adagio (cit. on p. 291).

[63] C. Borchert, D. Lohmann, and O. Spinczyk. "CiAO/IP: A Highly Configurable Aspect-Oriented IP Stack". In: *Procs. of the Int. Conference on Mobile Systems, Applications, and Services 2012*. New York, NY, USA: ACM, June 2012, pp. 435–448. DOI: http://doi.acm.org/10.1145/2307636.2307676 (cit. on p. 40). **SFB876-A1, SFB876-A4**

[64] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel. "Protein function prediction via graph kernels." In: *Bioinformatics* 21 Suppl 1 (2005), pp. i47–i56 (cit. on p. 116).

[65] K. Borgwardt and H.-P. Kriegel. "Shortest-Path Kernels on Graphs". In: *Procs. of the IEEE Int. Conference on Data Mining 2005*. 2005, pp. 74–81 (cit. on p. 124).

[66] J. Bossek. *smoof: Single and Multi-Objective Optimization Test Functions*. R package version 1.4. 2016. URL: https://CRAN.R-project.org/package=smoof (cit. on p. 293).

[67] G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein. “Improving Graph Neural Network Expressivity via Subgraph Isomorphism Counting”. In: *Procs. of the Int. Conference on Machine Learning 2020*. 2020 (cit. on p. 132).

[68] B. B. Brandenburg. “Multiprocessor Real-Time Locking Protocols: A Systematic Review”. In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1909.09600 (cit. on p. 362).

[69] B. B. Brandenburg and J. H. Anderson. “Optimality Results for Multiprocessor Real-Time Locking”. In: *Procs. of the IEEE Real-Time Systems Symposium 2010*. 2010, pp. 49–60. DOI: http://dx.doi.org/10.1109/RTSS.2010.17 (cit. on pp. 361, 362).

[70] V. Braverman, G. Frahling, H. Lang, C. Sohler, and L. F. Yang. “Clustering High Dimensional Dynamic Data Streams”. In: *Procs. of the Int. Conference on Machine Learning 2017*. Ed. by D. Precup and Y. W. Teh. PMLR, 2017, pp. 576–585. URL: http://proceedings.mlr.press/v70/braverman17a.html (cit. on pp. 10, 213). **SFB876-A2**

[71] L. Breiman. “Bagging Predictors”. In: *Machine Learning* 24.2 (Aug. 1996), pp. 123–140. DOI: https://doi.org/10.1023/A:1018054314350 (cit. on p. 341).

[72] L. Breiman. “Random Forests”. In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32 (cit. on pp. 341, 351).

[73] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Belmont, CA: Wadsworth, 1984 (cit. on p. 351).

[74] S. Breß, H. Funke, and J. Teubner. “Robust Query Processing in Co-Processor-Accelerated Databases”. In: *Procs. of the ACM SIGMOD Conference on Management of Data 2016*. San Francisco, CA, USA: ACM, June 2016 (cit. on pp. 381, 382, 386). **SFB876-C5**

[75] S. Breß, B. Köcher, H. Funke, S. Zeuch, T. Rabl, and V. Markl. “Generating Custom Code for Efficient Query Execution on Heterogeneous Processors”. In: *The VLDB Journal – The Int. Journal on Very Large Data Bases* 27.6 (Dec. 2018), pp. 797–822 (cit. on p. 394). **SFB876-A2**

[76] M. Brill, T. Fluschnik, V. Froese, B. J. Jain, R. Niedermeier, and D. Schultz. “Exact mean computation in dynamic time warping spaces”. In: *Data Mining and Knowledge Discovery* 33.1 (2019), pp. 252–291. DOI: https://doi.org/10.1007/s10618-018-0604-8 (cit. on p. 204).

[77] K. Bringmann. “Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails”. In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2014*. 2014, pp. 661–670 (cit. on p. 199).

[78] K. Bringmann, A. Driemel, A. Nusser, and I. Psarros. “Tight Bounds for Approximate Near Neighbor Searching for Time Series under the Fréchet Distance”. In: *Procs. of the Symposium on Discrete Algorithms 2022*. SIAM, 2022, pp. 517–550. DOI: 10.1137/1.9781611977073.25 (cit. on p. 203).

[79] K. Bringmann and W. Mulzer. “Approximability of the discrete Fréchet distance”. In: *Journal of Computational Geometry* 7.2 (2016). Previously appeared in Procs. of the Int. Symposium on Computational Geometry 2015, pp. 46–76. URL: http://jocg.org/index.php/jocg/article/view/261 (cit. on p. 200).

[80] E. Brochu, V. M. Cora, and N. De Freitas. “A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning”. In: *arXiv: Computing Research Repository* (2010). DOI: arXiv:1012.2599 (cit. on p. 95).

[81] L. D. Brown, T. T. Cai, and A. DasGupta. “Interval estimation for a binomial proportion”. In: *Statistical Science* (2001), pp. 101–117 (cit. on p. 79).

[82] G. von der Brüggen, J.-J. Chen, W.-H. Huang, and M. Yang. “Release Enforcement in Resource-Oriented Partitioned Scheduling for Multiprocessor Systems”. In: *Procs. of the Int. Conference on Real-Time Networks and Systems 2017*. 2017 (cit. on p. 368). **SFB876-B2**

[83] G. von der Brüggen, K.-H. Chen, W.-H. Huang, and J.-J. Chen. “Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments”. In: *Procs. of the IEEE Real-Time Systems Symposium 2016*. IEEE, 2016, pp. 303–314 (cit. on p. 372). **SFB876-A1**

[84] G. von der Brüggen, W.-H. Huang, and J.-J. Chen. “Hybrid self-suspension models in real-time embedded systems”. In: *Procs. of the IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications 2017*. IEEE, 2017, pp. 1–9. DOI: http://dx.doi.org/10.1109/RTCSA.2017.8046328 (cit. on p. 363). **SFB876-B2**

[85] G. von der Brüggen, N. Piatkowski, K.-H. Chen, J.-J. Chen, and K. Morik. “Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems”. In: *Procs. of the Euromicro Conference on Real-Time Systems 2018*. LIPIcs, 2018 (cit. on pp. 377, 378). **SFB876-A1, SFB876-B2**

[86] G. von der Brüggen, N. Ueter, J. Chen, and M. Freier. “Parametric utilization bounds for implicit-deadline periodic tasks in automotive systems”. In: *Procs. of the Int. Conference on Real-Time Networks and Systems 2017*. 2017, pp. 108–117 (cit. on p. 368).

[87] G. v. d. Brüggen, L. Schönberger, and J.-J. Chen. “Do Nothing, but Carefully: Fault Tolerance with Timing Guarantees for Multiprocessor Systems devoid of Online Adaptation”. In: *Procs. of the IEEE Pacific Rim Int. Symposium on Dependable Computing 2018*. Taipei, Taiwan, Dec. 2018. URL: https://ieeexplore.ieee.org/document/8639554 (cit. on p. 372).

[88] N. Bruno and S. Chaudhuri. “Physical design refinement: The ’merge-reduce’ approach”. In: *ACM Transactions on Database Systems* 32.4 (2007), p. 28 (cit. on p. 88).

[89] M. Bruynooghe. “Méthodes nouvelles en classification automatique de données taxinomiques nombreuses”. In: *Statistique et analyse des données* 2.3 (1977), pp. 24–42. URL: https://www.numdam.org/item/SAD_1977__2_3_24_0/ (cit. on pp. 218, 222).

[90] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. “A Tight Linear Time (1/2)-Approximation for Unconstrained Submodular Maximization”. In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2012*. 2012, pp. 649–658 (cit. on pp. 145, 153).

[91] N. Buchbinder, M. Feldman, and R. Schwartz. “Online submodular maximization with preemption”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2015*. Society for Industrial and Applied Mathematics. 2015, pp. 1202–1216 (cit. on p. 76).

[92] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. “Four Soviets Walk the Dog: Improved Bounds for Computing the Fréchet Distance”. In: *Discrete & Computational Geometry* 58.1 (2017). Previously appeared in Symposium on Discrete Algorithms 2014, pp. 180–216. DOI: https://doi.org/10.1007/s00454-017-9878-7 (cit. on p. 199).

[93] K. Buchin, A. Driemel, and M. Struijs. “On the Hardness of Computing an Average Curve”. In: *Procs. of the Scandinavian Symposium and Workshops on Algorithm Theory 2020*. Ed. by S. Albers. Vol. 162. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 19:1–19:19 (cit. on pp. 197, 202–205).

[94] K. Buchin, T. Ophelders, and B. Speckmann. “SETH Says: Weak Fréchet Distance is Faster, but only if it is Continuous and in One Dimension”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2019*. Ed. by T. M. Chan. SIAM, 2019, pp. 2887–2901. DOI: https://doi.org/10.1137/1.9781611975482.179 (cit. on p. 200).

[95] K. Buchin et al. “Approximating (k, ℓ)-center clustering for curves”. In: *Procs. of the Symposium on Discrete Algorithms 2019*. Ed. by T. M. Chan. ACM-SIAM, 2019, pp. 2922–2938. DOI: https://doi.org/10.1137/1.9781611975482.181 (cit. on pp. 197, 202–205, 210).

[96] K. Buchin et al. “Median Trajectories”. In: *Algorithmica* 66.3 (2013). Previously appeared in ESA 2010, pp. 595–614 (cit. on p. 202).

[97] M. Buchin, A. Driemel, and D. Rohde. “Approximating (k, ℓ)-Median Clustering for Polygonal Curves”. In: *Procs. of the Symposium on Discrete Algorithms 2021*. Ed. by D. Marx. ACM-

SIAM, 2021, pp. 2697–2717. DOI: https://doi.org/10.1137/1.9781611976465.160 (cit. on pp. 199, 203, 204).

[98] M. Buchin, N. Funk, and A. Krivošija. "On the complexity of the middle curve problem". In: *arXiv: Computing Research Repository* (2020). Presented in EuroCG 2020. DOI: arXiv:2001.10298 (cit. on pp. 197, 205). **SFB876-A2**

[99] M. Buchin, A. Krivošija, and A. Neuhaus. "Computing the Fréchet distance of trees and graphs of bounded tree width". In: *arXiv: Computing Research Repository* (2020). Presented in EuroCG 2020. DOI: arXiv:2001.10502 (cit. on p. 212). **SFB876-A2**

[100] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. "Recent Advances in Graph Partitioning". In: *Algorithm Engineering: Selected Results and Surveys*. Ed. by P. Kliemann Lasseand Sanders. Cham: Springer Int. Publishing, 2016, pp. 117–158 (cit. on p. 145).

[101] A. Burns and A. J. Wellings. "A Schedulability Compatible Multiprocessor Resource Sharing Protocol - MrsP". In: *Procs. of the Euromicro Conference on Real-Time Systems 2013*. 2013, pp. 282–291. DOI: http://dx.doi.org/10.1109/ECRTS.2013.37 (cit. on pp. 361, 362).

[102] M. Bury and C. Schwiegelshohn. "On Finding the Jaccard Center". In: *Procs. of the Int. Colloquium on Automata, Languages and Programming 2017*. Ed. by P. Indyk, F. Kuhn, and A. Muscholl. 2017 (cit. on p. 212). **SFB876-A2**

[103] M. Bury, C. Schwiegelshohn, and M. Sorella. "Sketch 'Em All: Approximate Similarity Search for Dynamic Data Streams". In: *Procs. of the ACM Int. Conference on Web Search and Data Mining 2018*. Ed. by Y. Maarek and Y. Liu. ACM, 2018 (cit. on p. 10). **SFB876-A2**

[104] M. Buschhoff. "Energy-Aware Design of Hardware and Software for Ultra-Low-Power Systems". PhD thesis. Dortmund: TU Dortmund University, 2019. DOI: http://dx.doi.org/10.17877/DE290R-20241 (cit. on pp. 35, 42, 43).

[105] M. Buschhoff, R. Falkenberg, and O. Spinczyk. "Energy-Aware Device Drivers for Embedded Operating Systems". In: *SIGBED Review* 16.3 (Nov. 2019), pp. 8–13. DOI: https://doi.org/10.1145/3373400.3373401 (cit. on pp. 6, 41). **SFB876-A4**

[106] M. Buschhoff, D. Friesel, and O. Spinczyk. "Energy Models in the Loop". In: *Procedia Computer Science* 130 (2018), pp. 1063–1068 (cit. on pp. 6, 43, 44). **SFB876-A4**

[107] S. Buschjäger. "Ensemble Learning with Discrete Classifiers on Small Devices". PhD thesis. TU Dortmund University, 2022 (cit. on p. 339).

[108] S. Buschjäger, K.-h. Chen, J.-j. Chen, and K. Morik. "Realization of Random Forest for Real-Time Evaluation through Tree Framing". In: 2018. URL: https://ieeexplore.ieee.org/document/8594826 (cit. on pp. 339, 341). **SFB876-A1, SFB876-B2, SFB876-C3**

[109] S. Buschjäger, P.-J. Honysz, L. Pfahler, and K. Morik. "Very Fast Streaming Submodular Function Maximization". In: *Procs. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2021*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 151–166. DOI: https://doi.org/10.1007/978-3-030-86523-8_10. URL: https://2021.ecmlpkdd.org/wp-content/uploads/2021/07/sub_178.pdf (cit. on pp. 74–76, 80, 81). **SFB876-A1**

[110] S. Buschjäger and K. Morik. "Decision Tree and Random Forest Implementations for Fast Filtering of Sensor Data". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65-I.1 (Jan. 2018), pp. 209–222. DOI: https://doi.org/10.1109/TCSI.2017.2710627 (cit. on pp. 7, 341, 342).

[111] S. Buschjäger, K. Morik, and M. Schmidt. "Summary Extraction on Data Streams in Embedded Systems". In: *Procs. of the ECML Workshop on IoT Large Scale Learning From Data Streams 2017*. 2017. URL: http://ceur-ws.org/Vol-1958/IOTSTREAMING3.pdf (cit. on p. 81). **SFB876-A1**

[112] S. Buschjäger, L. Pfahler, J. Buss, K. Morik, and W. Rhode. "On-Site Gamma-Hadron Separation with Deep Learning on FPGAs". In: *Procs. of the Joint European Conference on Ma-*

chine Learning and Knowledge Discovery in Databases 2020. Springer, 2020. URL: https://link.springer.com/content/pdf/10.1007%5C%2F978-3-030-67667-4_29.pdf (cit. on p. 10). **SFB876-A1, SFB876-C3**

[113] S. Buschjäger et al. "Margin-Maximization in Binarized Neural Networks for Optimizing Bit Error Tolerance". In: *Procs. of the Design, Automation and Test in Europe Conference 2021*. 2021. URL: https://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2021dateyayla.pdf (cit. on pp. 10, 326, 331, 332). **SFB876-A1**

[114] S. Buschjäger et al. *Towards Explainable Bit Error Tolerance of Resistive RAM-Based Binarized Neural Networks*. 2020. URL: https://www-ai.cs.tu-dortmund.de/PublicPublicationFiles/buschjaeger_etal_2020b.pdf (cit. on p. 331). **SFB876-A1**

[115] J. Buss, C. Bockermann, K. Morik, W. Rhode, and T. Ruhe. "FACT-Tools – Processing High-Volume Telescope Data". In: *Procs. of the Astronomical Data Analysis Software and Systems Conference 2019*. Ed. by M. Molinaro, K. Shortridge, and F. Pasian. Vol. 521. Astronomical Society of the Pacific, 2019, p. 584. URL: http://www.aspbooks.org/a/volumes/article_details/?paper_id=39082 (cit. on p. 339). **SFB876-C3**

[116] J. Byrka, T. Pensyl, B. Rybicki, A. Srinivasan, and K. Trinh. "An Improved Approximation for *k*-Median and Positive Correlation in Budgeted Optimization". In: *ACM Transactions on Algorithms* 13.2 (2017), 23:1–23:31. DOI: https://doi.org/10.1145/2981561 (cit. on p. 201).

[117] J. Cai, M. Fürer, and N. Immerman. "An optimal lower bound on the number of variables for graph identifications". In: *Combinatorica* 12.4 (1992), pp. 389–410 (cit. on p. 120).

[118] C. Cangea, P. Veličković, N. Jovanović, T. N. Kipf, and P. Liò. "Towards Sparse Hierarchical Graph Classifiers". In: *Advances in Neural Information Processing Systems 31: Procs. of the 2018 Conference*. 2018 (cit. on p. 135).

[119] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal. "Dynamic instrumentation of production systems". In: *Procs. of the Conference on USENIX Annual Technical Conference 2004*. Berkeley, CA, USA: USENIX Association, 2004, pp. 2–2 (cit. on pp. 17, 18).

[120] R. Caruana, N. Karampatziakis, and A. Yessenalina. "An empirical evaluation of supervised learning in high dimensions". In: *Procs. of the Int. Conference on Machine Learning 2008*. ACM. 2008, pp. 96–103 (cit. on p. 339).

[121] A. Chakrabarti and S. Kale. "Submodular Maximization Meets Streaming: Matchings, Matroids, and More". In: *arXiv: Computing Research Repository* (2013). DOI: arXiv:1309.2038 (cit. on p. 76).

[122] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy. "Machine Learning on Graphs: A Model and Comprehensive Taxonomy". In: *arXiv: Computing Research Repository* (2020). DOI: arXiv:2005.03675 (cit. on p. 116).

[123] M. Charikar, K. Chen, and M. Farach-Colton. "Finding frequent items in data streams". In: *Theoretical Computer Science* 312.1 (2004), pp. 3–15 (cit. on p. 88).

[124] C. Chekuri, S. Gupta, and K. Quanrud. "Streaming algorithms for submodular function maximization". In: *arXiv: Computing Research Repository* (2015). DOI: arXiv:1504.08024 (cit. on p. 76).

[125] F. Chen, J. Deng, Z. Pang, M. Baghaei Nejad, H. Yang, and G. Yang. "Finger angle-based hand gesture recognition for smart infrastructure using wearable wrist-worn camera". In: *Applied Sciences* 8.3 (2018), p. 369 (cit. on p. 61).

[126] J. Chen, J. Zhu, and L. Song. "Stochastic Training of Graph Convolutional Networks with Variance Reduction". In: *Procs. of the Int. Conference on Machine Learning 2018*. 2018 (cit. on pp. 136, 137, 139).

[127] J.-J. Chen, G. von der Brüggen, W.-H. Huang, and C. Liu. "State of the art for scheduling and analyzing self-suspending sporadic real-time tasks". In: *Procs. of the IEEE Int. Conference on*

Embedded and Real-Time Computing Systems and Applications 2017. (invited paper). IEEE, 2017, pp. 1–10. URL: 10.1109/RTCSA.2017.8046321 (cit. on p. 363). **SFB876-B2**

[128] J.-J. Chen et al. “Many suspensions, many problems: a review of self-suspending tasks in real-time systems”. In: *Real-Time Systems* (2018). preprint. URL: https://link.springer.com/article/10.1007%5C%2Fs11241-018-9316-9 (cit. on p. 363). **SFB876-B2**

[129] K. Chen, N. Ueter, G. v. der Brüggen, and J. Chen. “Efficient Computation of Deadline-Miss Probability and Potential Pitfalls”. In: *Procs. of the Design, Automation and Test in Europe Conference 2019*. 2019, pp. 896–901 (cit. on p. 377).

[130] K. Chen. “On Coresets for *k*-Median and *k*-Means Clustering in Metric and Euclidean Spaces and Their Applications”. In: *SIAM Journal on Computing* 39.3 (Aug. 2009). Previously appeared in Symposium on Discrete Algorithms 2006, pp. 923–947. URL: http://link.aip.org/link/?SMJ/39/923/1 (cit. on p. 201).

[131] K.-H. Chen and J.-J. Chen. “Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors”. In: *Procs. of the IEEE Int. Symposium on Industrial Embedded Systems 2017*. 2017, pp. 1–8. DOI: https://doi.org/10.1109/SIES.2017.7993392 (cit. on p. 377). **SFB876-B2**

[132] K.-H. Chen et al. “Efficient Realization of Decision Trees for Real-Time Inference (to appear, accepted)”. In: *ACM Transactions on Embedded Computing Systems* (2022) (cit. on p. 349). **SFB876-A1**

[133] L. Chen, Z. Wei, Z. Cui, M. Chen, H. Pan, and Y. Bao. “CMD: Classification-based Memory Deduplication Through Page Access Characteristics”. In: *Procs. of the ACM SIGPLAN/SIGOPS Int. Conference on Virtual Execution Environments 2014*. VEE ’14. ACM, 2014, pp. 65–76 (cit. on p. 307).

[134] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. “Simple and Deep Graph Convolutional Networks”. In: *Procs. of the Int. Conference on Machine Learning 2020*. 2020 (cit. on pp. 130, 132, 175).

[135] Y. Cheng and G. M. Church. “Biclustering of expression data.” In: *Procs. of the Int. Conference on Intelligent Systems for Molecular Biology 2000*. Vol. 8. 2000, pp. 93–103 (cit. on p. 233).

[136] C. Chevalier and D. Ginsbourger. “Fast Computation of the Multi-Points Expected Improvement with Applications in Batch Selection”. In: *Learning and Intelligent Optimization*. Ed. by G. Nicosia and P. Pardalos. Springer, 2013, pp. 59–69 (cit. on p. 287).

[137] W. L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C. J. Hsieh. “Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2019*. SIGKDD. 2019 (cit. on pp. 136, 141).

[138] Y.-D. Chih et al. “An 89TOPS/W and 16.3 TOPS/mm 2 All-Digital SRAM-Based Full-Precision Compute-In Memory Macro in 22nm for Machine-Learning Edge Applications”. In: *Procs. of the IEEE Int. Solid-State Circuits Conference 2021*. Vol. 64. IEEE. 2021, pp. 252–254 (cit. on p. 4).

[139] H. Cho, I. S. Dhillon, Y. Guan, and S. Sra. “Minimum sum-squared residue co-clustering of gene expression data”. In: *Procs. of the SIAM Int. Conference on Data Mining 2004*. SIAM. 2004, pp. 114–125 (cit. on p. 232).

[140] C. Choirat and R. Seri. “Estimation in Discrete Parameter Models”. In: *Statistical Science* 27.2 (2012), pp. 278–293 (cit. on p. 407).

[141] K. L. Clarkson. “Subgradient and sampling algorithms for ℓ_1 regression”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2005*. 2005, pp. 257–266 (cit. on pp. 87, 90).

[142] K. L. Clarkson, P. Drineas, M. Magdon-Ismail, M. W. Mahoney, X. Meng, and D. P. Woodruff. “The Fast Cauchy Transform and Faster Robust Linear Regression”. In: *SIAM Journal on*

Computing 45.3 (2016), pp. 763–810. DOI: https://doi.org/10.1137/140963698 (cit. on pp. 87, 90).

[143] K. L. Clarkson, R. Wang, and D. P. Woodruff. "Dimensionality Reduction for Tukey Regression". In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019, pp. 1262–1271 (cit. on p. 90).

[144] K. L. Clarkson and D. P. Woodruff. "Input Sparsity and Hardness for Robust Subspace Approximation". In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2015*. 2015, pp. 310–329. DOI: https://doi.org/10.1109/FOCS.2015.27 (cit. on pp. 87, 90).

[145] K. L. Clarkson and D. P. Woodruff. "Numerical linear algebra in the streaming model". In: *Procs. of the ACM Symposium on Theory of Computing 2009*. 2009, pp. 205–214 (cit. on pp. 89, 93, 94).

[146] K. L. Clarkson and D. P. Woodruff. "Sketching for *M*-Estimators: A Unified Approach to Robust Regression". In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2015*. 2015, pp. 921–939. DOI: https://doi.org/10.1137/1.9781611973730.63 (cit. on pp. 87, 90).

[147] M. B. Cohen, Y. T. Lee, C. Musco, C. Musco, R. Peng, and A. Sidford. "Uniform Sampling for Matrix Approximation". In: *Procs. of the Conference on Innovations in Theoretical Computer Science 2015*. 2015, pp. 181–190 (cit. on p. 87).

[148] V. Cohen-Addad, K. G. Larsen, D. Saulpic, and C. Schwiegelshohn. "Towards optimal lower bounds for *k*-median and *k*-means coresets". In: *Procs. of the ACM SIGACT Symposium on Theory of Computing 2022*. Ed. by S. Leonardi and A. Gupta. ACM, 2022, pp. 1038–1051. DOI: 10.1145/3519935.3519946. URL: https://doi.org/10.1145/3519935.3519946 (cit. on p. 212).

[149] V. Cohen-Addad, D. Saulpic, and C. Schwiegelshohn. "A new coreset framework for clustering". In: *Procs. of the Symposium on Theory of Computing 2021*. Ed. by S. Khuller and V. Vassilevska Williams. ACM, 2021, pp. 169–182. DOI: https://doi.org/10.1145/3406325.3451022 (cit. on p. 212).

[150] V. Cohen-Addad and C. Schwiegelshohn. "On the Local Structure of Stable Clustering Instances". In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2017*. Ed. by C. Umans. 2017 (cit. on p. 213). **SFB876-A2**

[151] V. Cohen-Addad, C. Schwiegelshohn, and C. Sohler. "Diameter and k-Center in Sliding Windows". In: *Procs. of the Int. Colloquium on Automata, Languages, and Programming 2016*. Ed. by M. Mitzenmacher, Y. Rabani, and D. Sangiorgi. Vol. 55. 2016, 19:1–19:12. DOI: https://doi.org/10.4230/LIPIcs.ICALP.2016.19 (cit. on p. 213). **SFB876-A2**

[152] A. Colin, E. Ruppel, and B. Lucia. "A reconfigurable energy storage architecture for energy-harvesting devices". In: *Procs. of the Int. Conference on Architectural Support for Programming Languages and Operating Systems 2018*. ACM, 2018 (cit. on pp. 53, 57).

[153] W. Cong, R. Forsati, M. Kandemir, and M. Mahdavi. "Minimal Variance Sampling with Provable Guarantees for Fast Training of Graph Neural Networks". In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2020*. 2020 (cit. on p. 137).

[154] G. Cormode and S. Muthukrishnan. "An improved data stream summary: The Count-Min sketch and its applications". In: *Journal of Algorithms* 55 (2004), pp. 29–38 (cit. on p. 88).

[155] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković. "Principal Neighbourhood Aggregation for Graph Nets". In: *Advances in Neural Information Processing Systems 33: Procs. of the 2020 Conference*. 2020 (cit. on pp. 132, 133, 139).

[156] M. Cucuringu, J. Puente, and D. Shue. *Model Selection in Undirected Graphical Models with the Elastic Net*. 2011 (cit. on p. 105).

[157] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi. "Calibrating probability with undersampling for unbalanced classification". In: *Procs. of the IEEE Symposium Series on Computational Intelligence 2015*. IEEE. 2015, pp. 159–166. URL: https://www.kaggle.com/mlg-ulb/creditcardfraud (cit. on p. 81).

[158] E. Dallago, A. Barnabei, A. Liberale, P. Malcovati, and G. Venchi. “An Interface Circuit for Low-Voltage Low-Current Energy Harvesting Systems”. In: *IEEE Transactions on Power Electronics* 30.3 (2015) (cit. on p. 53).

[159] E. Dallago, A. Lazzarini Barnabei, A. Liberale, G. Torelli, and G. Venchi. “A 300 mV Low-Power Management System For Energy Harvesting Applications”. In: *IEEE Transactions on Power Electronics* PP.99 (2015) (cit. on p. 53).

[160] J. N. Darroch and D. Ratcliff. “Generalized iterative scaling for log-linear models”. In: *The Annals of Mathematical Statistics* 43.5 (1972), pp. 1470–1480 (cit. on p. 105).

[161] A. Dasgupta, P. Drineas, B. Harb, and R. K. and Michael W. Mahoney. “Sampling algorithms and coresets for ℓ_p-regression”. In: *SIAM Journal on Computing* 38.5 (2009), pp. 2060–2078 (cit. on pp. 87, 90).

[162] D. Dato et al. “Fast ranking with additive ensembles of oblivious and non-oblivious regression trees”. In: *ACM Transactions on Information Systems* (2016) (cit. on p. 340).

[163] D. Davis, B. Edmunds, and M. Udell. “The Sound of APALM Clapping: Faster Nonsmooth Nonconvex Optimization with Stochastic Asynchronous PALM”. In: *Advances in Neural Information Processing Systems 29: Procs. of the 2016 Conference*. Vol. 29. 2016 (cit. on p. 247).

[164] M. Defferrard, X. Bresson, and P. Vandergheynst. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering”. In: *Advances in Neural Information Processing Systems 29: Procs. of the 2016 Conference*. 2016, pp. 3844–3852 (cit. on p. 136).

[165] N. Del Buono and G. Pio. “Non-negative Matrix Tri-Factorization for co-clustering: An analysis of the block matrix”. In: *Information Sciences* 301 (2015), pp. 13–26 (cit. on pp. 232, 238, 239).

[166] C. Delimitrou and C. Kozyrakis. “Quasar: Resource-efficient and QoS-aware Cluster Management”. In: *Procs. of the Int. Conference on Architectural Support for Programming Languages and Operating Systems 2014*. ACM, 2014, pp. 127–144 (cit. on p. 285).

[167] H. Dell, M. Grohe, and G. Rattan. “Lovász Meets Weisfeiler and Leman”. In: *Procs. of the Int. Colloquium on Automata, Languages, and Programming 2018*. 2018, 40:1–40:14 (cit. on p. 125).

[168] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei. “What does classifying more than 10,000 image categories tell us?” In: *Procs. of the European Conference on Computer Vision 2010*. Springer, 2010, pp. 71–84 (cit. on p. 272).

[169] Y. Deng, L. Song, and X. Huang. “Evaluating Memory Compression and Deduplication”. In: *Procs. of the IEEE Int. Conference on Networking, Architecture and Storage 2013*. IEEE Computer Society, 2013, pp. 282–286 (cit. on p. 307).

[170] Y. Deng, A. Kanervisto, J. Ling, and A. M. Rush. “Image-to-Markup Generation with Coarse-to-Fine Attention”. In: *Procs. of the Int. Conference on Machine Learning 2017*. 2017, pp. 980–989 (cit. on p. 162).

[171] E. Denton, J. Weston, M. Paluri, L. Bourdev, and R. Fergus. “User Conditional Hashtag Prediction for Images”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2015*. 2015 (cit. on p. 272).

[172] M. Desnoyers and M. R. Dagenais. “The LTTng tracer: A low impact performance and behavior monitor for GNU/Linux”. In: *Procs. of the Ottawa Linux Symposium 2006*. 2006, pp. 209–224 (cit. on p. 17).

[173] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Procs. of Conference of the North American Chapter of the Association for Computational Linguistics 2019*. Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: arXiv:1810.04805 (cit. on pp. 163, 166, 167, 169).

[174] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv: Computing Research Repository* (2018). DOI: arXiv:1810.04805 (cit. on p. 283).

[175] I. S. Dhillon, Y. Guan, and B. Kulis. “Weighted Graph Cuts without Eigenvectors: A Multilevel Approach”. In: 2007, pp. 1944–1957 (cit. on pp. 135, 140).

[176] I. S. Dhillon. “Co-clustering documents and words using bipartite spectral graph partitioning”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2001*. ACM. 2001, pp. 269–274 (cit. on p. 232).

[177] J. L. Díaz et al. “Stochastic Analysis of Periodic Real-Time Systems”. In: *Procs. of the IEEE Real-Time Systems Symposium 2002*. 2002, pp. 289–300 (cit. on p. 363).

[178] C. Ding, T. Li, W. Peng, and H. Park. “Orthogonal nonnegative matrix t-factorizations for clustering”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2006*. ACM. 2006, pp. 126–135 (cit. on p. 232).

[179] S. Ding, L. Lin, G. Wang, and H. Chao. “Deep feature learning with relative distance comparison for person re-identification”. In: *Pattern Recognition* 48.10 (2015), pp. 2993–3003 (cit. on p. 166).

[180] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda. “Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed”. In: *Testbeds and Research Infrastructure. Development of Networks and Communities*. Ed. by T. Korakis, H. Li, P. Tran-Gia, and H.-S. Park. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, 2012, pp. 302–316 (cit. on p. 35).

[181] W. Dong, L. Yang, R. Gravina, and G. Fortino. “Soft wrist-worn multi-functional sensor array for real-time hand gesture recognition”. In: *IEEE Sensors Journal* (2021) (cit. on p. 61).

[182] B. Douillard, D. Fox, and F. T. Ramos. “A spatio-temporal probabilistic model for multi-sensor object recognition”. In: *Procs. of the IEEE/RSJ Int. Conference on Intelligent Robots and Systems 2007*. 2007, pp. 2402–2408 (cit. on p. 102).

[183] U. Drepper. *What Every Programmer Should Know About Memory*. 2007 (cit. on p. 343).

[184] A. Driemel and A. Krivošija. “Probabilistic embeddings of the Fréchet distance”. In: *Procs. of the Int. Workshop on Approximation and Online Algorithms 2018*. Ed. by L. Epstein and T. Erlebach. Vol. 11312. LNCS. Springer, 2018, pp. 218–237. DOI: https://doi.org/10.1007/978-3-030-04693-4_14 (cit. on p. 212). **SFB876-A2**

[185] A. Driemel, A. Krivošija, and C. Sohler. “Clustering time series under the Fréchet distance”. In: *Procs. of the Symposium on Discrete Algorithms 2016*. Ed. by R. Krauthgamer. SIAM, 2016, pp. 766–785. URL: http://epubs.siam.org/doi/10.1137/1.9781611974331.ch55 (cit. on pp. 197, 201–204). **SFB876-A2**

[186] A. Driemel and I. Psarros. “ANN for Time Series Under the Fréchet Distance”. In: *Procs. of the Int. Symposium on Algorithms and Data Structures 2021*. Ed. by A. Lubiw and M. R. Salavatipour. Vol. 12808. Lecture Notes in Computer Science. Springer, 2021, pp. 315–328. DOI: https://doi.org/10.1007/978-3-030-83508-8_23 (cit. on p. 203).

[187] D. Driggs, J. Tang, M. Davies, and C.-B. Schönlieb. “SPRING: A fast stochastic proximal alternating method for non-smooth non-convex optimization”. In: *arXiv: Computing Research Repository* (2020). DOI: arXiv:2002.12266 (cit. on pp. 238, 247).

[188] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. “Relative-Error CUR Matrix Decompositions”. In: *SIAM Journal on Matrix Analysis and Applications* 30.2 (2008), pp. 844–881. DOI: https://doi.org/10.1137/07070471X (cit. on pp. 87, 90, 92).

[189] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. “Sampling algorithms for ℓ_2 regression and applications”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2006*. 2006, pp. 1127–1136 (cit. on pp. 87, 90).

[190] B. Dusza. “Context-Aware Power Consumption Modeling for Energy Efficient Mobile Communication Services”. PhD thesis. TU Dortmund University, 2014. URL: http://www.shaker.de/de/content/catalogue/index.asp?lang=de%5C&ID=8%5C&ISBN=978-3-8440-2683-2 (cit. on p. 431). **SFB876-A4**

[191] B. Dusza, C. Ide, L. Cheng, and C. Wietfeld. “An Accurate Measurement-Based Power Consumption Model for LTE Uplink Transmissions”. In: *Procs. of IEEE INFOCOM 2013*. Turin, Italy: IEEE, Apr. 2013 (cit. on p. 430). **SFB876-A4, SFB876-B4**

[192] B. Dusza, C. Ide, L. Cheng, and C. Wietfeld. “CoPoMo: A Context-Aware Power Consumption Model for LTE User Equipment”. In: *Transactions on Emerging Telecommunications Technologies* 24.6 (Oct. 2013), pp. 615–632. URL: http://onlinelibrary.wiley.com/doi/10.1002/ett.2702/full (cit. on pp. 429, 430, 432). **SFB876-A4, SFB876-B4**

[193] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About A Highly Connected World*. Cambridge University Press, July 2010 (cit. on pp. 116, 144).

[194] A. Easwaran and B. Andersson. “Resource Sharing in Global Fixed-Priority Preemptive Multiprocessor Scheduling”. In: *Procs. of the IEEE Real-Time Systems Symposium 2009*. 2009, pp. 377–386. DOI: http://dx.doi.org/10.1109/RTSS.2009.37 (cit. on p. 368).

[195] I. J. Egielski, J. Huang, and E. Z. Zhang. “Massive Atomics for Massive Parallelism on GPUs”. In: *ACM SIGPLAN Notices* 49.11 (2015), pp. 93–103 (cit. on p. 391).

[196] F. C. Eigler et al. *Architecture of systemtap: a Linux trace/probe tool*. 2005. URL: http://www.cs.ucsb.edu/~grze/papers/profile/eigler05systemtap.bib (cit. on pp. 16, 18).

[197] T. Eiter and H. Mannila. *Computing discrete Fréchet distance*. Tech. rep. CD-TR 94/64. Christian Doppler Laboratory, 1994 (cit. on pp. 199, 211).

[198] J. G. Eldredge and B. L. Hutchings. “Density enhancement of a neural network using FPGAs and run-time reconfiguration”. In: *Procs. of the IEEE Workshop on FPGAs for Custom Computing Machines 1994*. IEEE. 1994, pp. 180–188 (cit. on pp. 253, 255).

[199] P. Emberson, R. Stafford, and R. I. Davis. “Techniques for the synthesis of multiprocessor tasksets”. In: *Procs. of the Int. Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems 2010*. 2010, pp. 6–11 (cit. on p. 368).

[200] T. A. Ensslin, M. Frommert, and F. S. Kitaura. “Information field theory for cosmological perturbation reconstruction and nonlinear signal analysis”. In: *Physical Review D* 80 (2009). DOI: http://dx.doi.org/10.1103/PhysRevD.80.105005 (cit. on p. 160).

[201] D. Eriksson, M. Pearce, R. Gardner Jacob R. and Turner, and M. Poloczek. “Scalable Global Optimization via Local Bayesian Optimization”. In: *Procs. of the Conference on Neural Information Processing Systems 2019*. 2019, pp. 5497–5508 (cit. on p. 96).

[202] Ú. Erlingsson, M. Peinado, S. Peter, M. Budiu, and G. Mainar-Ruiz. “Fay: Extensible Distributed Tracing from Kernels to Clusters”. In: *ACM Transactions on Computer Systems* 30.4 (Nov. 2012), 13:1–13:35 (cit. on p. 18).

[203] H. Esen, M. Adachi, D. Bernardini, A. Bemporad, D. Rost, and J. Knodel. “Control as a Service (CaaS): Cloud-Based Software Architecture for Automotive Control Applications”. In: *Procs. of the Int. Workshop on the Swarm at the Edge of the Cloud 2015*. Swarm at the Edge of the Cloud 2015. New York, NY, USA: Association for Computing Machinery, 2015, pp. 13–18. DOI: https://doi.org/10.1145/2756755.2756758 (cit. on p. 363).

[204] A. Esper, G. Nelissen, V. Nélis, and E. Tovar. “How Realistic is the Mixed-Criticality Real-Time System Model?” In: *Procs. of the Int. Conference on Real Time and Networks Systems 2015*. Real Time and Networks Systems 2015. New York, NY, USA: Association for Computing Machinery, 2015, pp. 139–148. DOI: https://doi.org/10.1145/2834848.2834869 (cit. on p. 372).

[205] B. O. Fagginger Auer and R. H. Bisseling. “A GPU Algorithm for Greedy Graph Matching”. In: *Facing the Multicore - Challenge II - Aspects of New Paradigms and Technologies in Parallel Computing*. 2011 (cit. on p. 135).

[206] R. Falkenberg, J. Drenhaus, B. Sliwa, and C. Wietfeld. “System-in-the-loop Design Space Exploration for Efficient Communication in Large-scale IoT-based Warehouse Systems”. In: *Procs. of the IEEE Int. Systems Conference 2018*. Vancouver, Canada: IEEE, Apr. 2018 (cit. on pp. 424, 426, 427, 429, 430). **SFB876-A4**

[207] R. Falkenberg, B. Sliwa, N. Piatkowski, and C. Wietfeld. “Machine Learning Based Uplink Transmission Power Prediction for LTE and Upcoming 5G Networks using Passive Downlink Indicators”. In: *Procs. of the IEEE Vehicular Technology Conference 2018*. Chicago, USA, Aug. 2018 (cit. on pp. 433–435). **SFB876-A4, SFB876-B4, SFB876-A1**

[208] R. Falkenberg, B. Sliwa, and C. Wietfeld. “Rushing Full Speed with LTE-Advanced is Economical - A Power Consumption Analysis”. In: *Procs. of the IEEE Vehicular Technology Conference 2017*. June 2017, pp. 1–7 (cit. on p. 431). **SFB876-A4**

[209] R. Falkenberg et al. “PhyNetLab: An IoT-based warehouse testbed”. In: *Procs. of the Federated Conference on Computer Science and Information Systems 2017*. Sept. 2017 (cit. on pp. 35, 424, 426, 428). **SFB876-A4**

[210] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. “LIBLINEAR: A library for large linear classification”. In: *Journal of Machine Learning Research* 9 (2008), pp. 1871–1874 (cit. on p. 275).

[211] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. J. V. Gool. “Random Forests for Real Time 3D Face Analysis”. In: *Int. Journal of Computer Vision* 101.3 (2013), pp. 437–458 (cit. on p. 339).

[212] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun. “CNP: An FPGA-based processor for convolutional networks”. In: *Procs. of the Int. Conference on Field Programmable Logic and Applications 2009*. IEEE. 2009, pp. 32–37 (cit. on p. 254).

[213] T. Feder and D. Greene. “Optimal algorithms for approximate clustering”. In: *Procs. of the Symposium on Theory of Computing 1988*. ACM, 1988, pp. 434–444 (cit. on p. 200).

[214] U. Feige. “A Threshold of Ln N for Approximating Set Cover”. In: *Journal of the Association for Computing Machinery* 45.4 (July 1998), pp. 634–652. DOI: http://doi.acm.org/10.1145/285055.285059 (cit. on p. 75).

[215] U. Feige, V. S. Mirrokni, and J. Vondrák. “Maximizing non-monotone submodular functions”. In: *SIAM Journal on Computing* 40.4 (2011), pp. 1133–1153 (cit. on p. 76).

[216] D. Feldman, M. Faulkner, and A. Krause. “Scalable Training of Mixture Models via Coresets”. In: *Advances in Neural Information Processing Systems 24: Procs. of the 2011 Conference*. 2011, pp. 2142–2150. URL: http://papers.nips.cc/paper/4363-scalable-training-of-mixture-models-via-coresets (cit. on p. 87).

[217] D. Feldman and M. Langberg. “A unified framework for approximating and clustering data”. In: *Procs. of the ACM Symposium on Theory of Computing 2011*. Ed. by L. Fortnow and S. P. Vadhan. ACM, 2011, pp. 569–578. DOI: http://doi.acm.org/10.1145/1993636.1993712 (cit. on p. 212).

[218] D. Feldman, A. Munteanu, and C. Sohler. “Smallest enclosing ball for probabilistic data”. In: *Procs. of the Symposium on Computational Geometry 2014*. 2014, p. 214 (cit. on pp. 87, 212).

[219] D. Feldman, M. Schmidt, and C. Sohler. “Turning Big Data Into Tiny Data: Constant-Size Coresets for k-Means, PCA, and Projective Clustering”. In: *SIAM Journal of Computing* 49.3 (2020), pp. 601–657 (cit. on pp. 87, 91, 212). **SFB876-A2**

[220] M. Feldman, A. Norouzi-Fard, O. Svensson, and R. Zenklusen. “The one-way communication complexity of submodular maximization with applications to streaming and robustness”. In: *Procs. of the ACM SIGACT Symposium on Theory of Computing 2020*. 2020, pp. 1363–1374 (cit. on pp. 74, 75, 77).

[221] X. Feng, Y. Zhang, and J. Glass. “Speech feature denoising and dereverberation via deep autoencoders for noisy reverberant speech recognition”. In: *Procs. of the IEEE Int. Conference on Acoustics, Speech and Signal Processing 2014*. IEEE. 2014, pp. 1759–1763 (cit. on p. 260).

[222] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt. “Scalable kernels for graphs with continuous attributes”. In: *Advances in Neural Information Processing Systems 26: Procs. of the 2013 Conference*. Erratum available at http://image.diku.dk/aasa/papers/graphkernels_nips_erratum.pdf. 2013, pp. 216–224 (cit. on p. 124).

[223] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. “Do we need hundreds of classifiers to solve real world classification problems?” In: *Journal of Machine Learning Research* 15.1 (2014), pp. 3133–3181 (cit. on p. 339).

[224] M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec. “GNNAutoScale: Scalable And Expressive Graph Neural Networks via Historical Embeddings”. In: *Procs. of the Int. Conference on Machine Learning 2021*. 2021 (cit. on pp. 130, 137, 139, 140, 142). **SFB876-A6**

[225] M. Fey and J. E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *Procs. of the ICLR Workshop on Representation Learning on Graphs and Manifolds 2019*. 2019. DOI: arXiv:1903.02428 (cit. on pp. 126, 130, 131, 134). **SFB876-A6,SFB876-B2**

[226] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. “SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels”. In: *Procs. of the IEEE Conference on Computer Vision and Pattern Recognition 2018*. 2018. DOI: arXiv:1711.08920 (cit. on pp. 130, 133). **SFB876-B2, SFB876-A6**

[227] M. Fey, J.-G. Yuen, and F. Weichert. “Hierarchical Inter-Message Passing for Learning on Molecular Graphs”. In: *Procs. of the ICML Graph Representation Learning and Beyond (GRL+) Workhop 2020*. 2020 (cit. on p. 132).

[228] H. Fichtenberger, M. Gillé, M. Schmidt, C. Schwiegelshohn, and C. Sohler. “BICO: Birch meets Coresets for k-means”. In: *Procs. of the European Symposium on Algorithms 2013*. Ed. by H. L. Bodlaender and G. F. Italiano. Springer, 2013. URL: http://link.springer.com/chapter/10.1007%5C%2F978-3-642-40450-4_41 (cit. on p. 213). **SFB876-A2**

[229] R. Fischer, N. Piatkowski, C. Pelletier, G. Webb, F. Petitjean, and K. Morik. “No Cloud on the Horizon: Probabilistic Gap Filling in Satellite Image Series”. In: *Procs. of the IEEE Int. Conference on Data Science and Advanced Analytics 2020*. Ed. by G. Webb, L. Cao, Z. Zhang, V. S. Tseng, G. Williams, and M. Vlachos. Environmental and Geo-spatial Data Analytics. The Institute of Electrical and Electronics Engineers, Inc. (IEEE), Oct. 2020, pp. 546–555. URL: https://ieeexplore.ieee.org/document/9260084 (cit. on p. 101).

[230] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. “An analysis of approximations for maximizing submodular set functions—II”. In: *Polyhedral combinatorics* (1978), pp. 73–87 (cit. on pp. 75, 76).

[231] FIT consortium. *FIT IoT-LAB*. FIT IoT-LAB, 2021. URL: https://iot-lab.info/ (cit. on p. 35).

[232] S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing*. Springer New York, 2013 (cit. on p. 378).

[233] M. Fragkoulis, D. Spinellis, P. Louridas, and A. Bilas. “Relational Access to Unix Kernel Data Structures”. In: *Procs. of the European Conference on Computer Systems 2014*. EuroSys ’14. New York, NY, USA: Association for Computing Machinery, 2014. DOI: https://doi.org/10.1145/2592798.2592802 (cit. on pp. 17, 19).

[234] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. M. Bronstein, and F. Monti. “SIGN: Scalable Inception Graph Neural Networks”. In: *Procs. of the Int. Conference on Machine Learning 2020*. 2020 (cit. on pp. 130, 137).

[235] P. I. Frazier. “A tutorial on Bayesian optimization”. In: *arXiv: Computing Research Repository* (2018). DOI: arXiv:1807.02811 (cit. on pp. 95, 96).

[236] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell. “Optimal Assignment Kernels for Attributed Molecular Graphs”. In: *Procs. of the Int. Conference on Machine Learning 2005*. 2005, pp. 225–232 (cit. on p. 121).

[237] H. Funke and J. Teubner. “Data-Parallel Query Processing on Non-Uniform Data”. In: *Procs. of the VLDB Endowment* (2020) (cit. on pp. 265, 268). **SFB876-A2**

[238] P. Gai, G. Lipari, and M. D. Natale. “Minimizing Memory Utilization of Real-Time Task Sets in Single and Multi-Processor Systems-on-a-Chip”. In: *Procs. of the IEEE Real-Time Systems Symposium 2001*. 2001, pp. 73–83. DOI: http://dx.doi.org/10.1109/REAL.2001.990598 (cit. on pp. 361, 362).

[239] L. Galli and C.-J. Lin. “A Study on Truncated Newton Methods for Linear Classification”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2021) (cit. on p. 278).

[240] P. R. Gankidi and J. Thangavelautham. “FPGA architecture for deep learning and its application to planetary robotics”. In: *Procs. of the IEEE Aerospace Conference 2017*. Mar. 2017, pp. 1–9 (cit. on p. 254).

[241] H. Gao and S. Ji. “Graph U-Net”. In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019 (cit. on p. 135).

[242] Y. Gao et al. “Estimating gpu memory consumption of deep learning models”. In: *Procs. ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering 2020*. 2020, pp. 1342–1352. URL: https://www.microsoft.com/en-us/research/uploads/prod/2020/09/dnnmem.pdf (cit. on p. 9).

[243] W. Gaul and M. Schader. “A new algorithm for two-mode clustering”. In: *Data Analysis and Information Systems*. Springer, 1996, pp. 15–23 (cit. on p. 231).

[244] F. Geerts, B. Goethals, and T. Mielikäinen. “Tiling databases”. In: *Procs. of the Int. Conference on Discovery Science 2004*. Springer. 2004, pp. 278–289 (cit. on p. 233).

[245] L. N. Geppert, K. Ickstadt, A. Munteanu, J. Quedenfeld, and C. Sohler. “Random projections for Bayesian regression”. In: *Statistics and Computing* 27.1 (2017), pp. 79–101. DOI: http://doi.org/10.1007/s11222-015-9608-z (cit. on pp. 86, 89, 93, 94). **SFB876-C4**

[246] L. N. Geppert, K. Ickstadt, A. Munteanu, and C. Sohler. “Streaming statistical models via Merge & Reduce”. In: *Int. Journal of Data Science and Analytics* 10.4 (2020), pp. 331–347. DOI: https://doi.org/10.1007/s41060-020-00226-0 (cit. on pp. 87, 88, 90). **SFB876-C4**

[247] L. N. Geppert. “Bayesian and Frequentist Regression Approaches for Very Large Data Sets”. PhD thesis. TU Dortmund University, 2018. DOI: http://dx.doi.org/10.17877/DE290R-19931 (cit. on pp. 89, 94, 95).

[248] L. Gerhorst, B. Herzog, S. Reif, W. Schröder-Preikschat, and T. Hönig. “AnyCall: Fast and Flexible System-Call Aggregation”. In: *Procs. of the Workshop on Programming Languages and Operating Systems 2021*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1–8. DOI: https://doi.org/10.1145/3477113.3487267 (cit. on p. 17).

[249] F. Gessert, W. Wingerath, S. Friedrich, and N. Ritter. “NoSQL database systems: a survey and decision guidance”. In: *Computer Science - Research and Development* 32.3-4 (2017), pp. 353–365 (cit. on p. 89).

[250] P. Geurts, D. Ernst, and L. Wehenkel. “Extremely randomized trees”. In: *Machine Learning* 63.1 (2006), pp. 3–42 (cit. on pp. 341, 351).

[251] S. Gidaris, P. Singh, and N. Komodakis. “Unsupervised Representation Learning by Predicting Image Rotations”. In: *Procs. of the Int. Conference on Learning Representations 2018*. 2018, pp. 1–16 (cit. on p. 163).

[252] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. “Neural Message Passing for Quantum Chemistry”. In: *Procs. of the Int. Conference on Machine Learning 2017*. 2017, pp. 1263–1272 (cit. on pp. 116, 125, 126, 130–132).

[253] D. Ginsbourger, J. Janusevskis, and R. Le Riche. *Dealing with asynchronicity in parallel Gaussian Process based global optimization*. Tech. rep. 2011. URL: https://hal.archives-ouvertes.fr/hal-00507632 (cit. on pp. 288, 292).

[254] D. Ginsbourger, R. Le Riche, and L. Carraro. "Kriging is Well-Suited to Parallelize Optimization". In: *Computational Intelligence in Expensive Optimization Problems*. Springer, 2010, pp. 131–162 (cit. on pp. 287, 288, 290, 292, 301).

[255] X. Glorot, A. Bordes, and Y. Bengio. "Deep sparse rectifier neural networks". In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2011*. 2011, pp. 315–323 (cit. on p. 261).

[256] M. X. Goemans and D. P. Williamson. "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming". In: *Journal of the Association for Computing Machinery* 42.6 (1995), pp. 1115–1145 (cit. on pp. 145, 147, 148, 153).

[257] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1983 (cit. on p. 150).

[258] R. Gomes and A. Krause. "Budgeted Nonparametric Learning from Data Streams". In: *Procs. of the Int. Conference on Machine Learning 2010*. Vol. 1. 2010, p. 3 (cit. on p. 76).

[259] A. Gomez, L. Sigrist, M. Magno, L. Benini, and L. Thiele. "Dynamic Energy Burst Scaling for Transiently Powered Systems". In: *Procs. of the Design, Automation and Test in Europe Conference 2016*. DATE '16. San Jose, CA, USA: EDA Consortium, 2016, pp. 349–354. URL: http://dl.acm.org/citation.cfm?id=2971808.2971888 (cit. on pp. 49, 57).

[260] A. Gomez. "On-demand communication with the batteryless MiroCard: demo abstract". In: *Procs. of the Conference on Embedded Networked Sensor Systems 2020*. 2020 (cit. on p. 57).

[261] A. Gomez, L. Sigrist, T. Schalch, L. Benini, and L. Thiele. "Efficient, Long-Term Logging of Rich Data Sensors Using Transient Sensor Nodes". In: *ACM Transactions on Embedded Computing Systems* 17.1 (2017), 4:1–4:23 (cit. on p. 47).

[262] A. Gomez, A. Tretter, P. A. Hager, P. Sanmugarajah, L. Benini, and L. Thiele. "Data-Flow Driven Partitioning of Machine Learning Applications for Optimal Energy Use in Batteryless Systems". In: *ACM Transactions on Embedded Computing Systems* (Feb. 2022). DOI: https://doi.org/10.1145/3520135 (cit. on p. 54).

[263] X. Gong and F. Wang. "Three Improvements on KNN-NPR for Traffic Flow Forecasting". In: *Procs. of the Int. Conference on Intelligent Transportation Systems 2002*. 2002, pp. 736–740 (cit. on p. 102).

[264] T. F. Gonzalez. "Clustering to Minimize the Maximum Intercluster Distance". In: *Theoretical Computer Science* 38 (1985), pp. 293–306 (cit. on pp. 200, 203).

[265] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (cit. on p. XI).

[266] I. Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27: Procs. of the 2014 Conference*. 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf (cit. on p. 250).

[267] M. Gorlatova, A. Wallwater, and G. Zussman. "Networking Low-Power Energy Harvesting Devices: Measurements and Algorithms". In: *IEEE Transactions on Mobile Computing* 12.9 (2013), pp. 1853–1865 (cit. on p. 47).

[268] A. B. Graf and S. Borer. "Normalization in support vector machines". In: *Pattern Recognition: Procs. of the German Association for Pattern Recognition Symposium 2001*. Springer. 2001, p. 277 (cit. on p. 81).

[269] R. L. Graham. "Bounds on Multiprocessing Timing Anomalies". In: *SIAM Journal of Applied Mathematics* 17.2 (1969), pp. 416–429 (cit. on p. 366).

[270] C. Gregg and K. Hazelwood. “Where Is the Data? Why You Cannot Debate CPU vs. GPU Performance Without the Answer”. In: *Procs. of the IEEE Int. Symposium on Performance Analysis of Systems and Software 2011*. IEEE Press, 2011, pp. 134–144 (cit. on p. 380).

[271] M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017 (cit. on p. 117).

[272] M. Grohe. “Word2vec, Node2vec, Graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data”. In: *arXiv: Computing Research Repository* (2020). DOI: arXiv:2003.12590 (cit. on p. 116).

[273] M. Grohe, K. Kersting, M. Mladenov, and A. E. Selman. “Dimension Reduction via Colour Refinement”. In: *Procs. of the European Symposium on Algorithms 2014*. 2014. URL: http://link.springer.com/chapter/10.1007/978-3-662-44777-2_42 (cit. on p. 118).

[274] P. Grosjean and S. Urbanek. *R Benchmark 2.5 Suite*. 2008. URL: http://r.research.att.com/benchmarks/R-benchmark-25.R (cit. on p. 315).

[275] S. Gu and Y. Yang. “A Deep Learning Algorithm for the Max-Cut Problem Based on Pointer Network Structure with Supervised Learning and Reinforcement Learning Strategies”. In: *Mathematics* 8.2 (2020) (cit. on p. 145).

[276] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. “Continuous deep Q-learning with model-based acceleration”. In: *Procs. of the Int. Conference on Machine Learning 2016*. 2016, pp. 2829–2838 (cit. on p. 259).

[277] F. Guidi and C. Sacerdoti Coen. “A survey on retrieval of mathematical knowledge”. In: *Procs. of the Int. Conference on Intelligent Computer Mathematics 2015*. Springer, 2015, pp. 296–315 (cit. on p. 161).

[278] C. Guo, W. Luk, and W. Xu. “Non-linear function evaluation reusing matrix-vector multipliers”. In: *Procs. of the IEEE Int. Conference on ASIC 2019*. IEEE, pp. 1–4 (cit. on pp. 251, 256).

[279] C. Guo et al. “Breaking the glass ceiling for embedding-based classifiers for large output spaces”. In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. Vol. 32. 2019 (cit. on p. 283).

[280] A. Gupta, K. Ni, O. Prakash, X. S. Hu, and H. Amrouch. “Temperature Dependence and Temperature-Aware Sensing in Ferroelectric FET”. In: *Procs. of the IEEE Int. Reliability Physics Symposium 2020*. 2020 (cit. on p. 328).

[281] A. Gupta, R. Krauthgamer, and J. R. Lee. “Bounded Geometries, Fractals, and Low-Distortion Embeddings”. In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2003*. IEEE. 2003, pp. 534–543. DOI: 10.1109/SFCS.2003.1238226 (cit. on p. 201).

[282] U. Gupta et al. “Chasing Carbon: The Elusive Environmental Footprint of Computing”. In: *Procs. of the IEEE Int. Symposium on High-Performance Computer Architecture 2021*. 2021, pp. 854–867. DOI: https://doi.org/10.1109/HPCA51647.2021.00076 (cit. on p. 3).

[283] B. Haasdonk and C. Bahlmann. “Learning with Distance Substitution Kernels”. In: *Pattern Recognition*. Vol. 3175. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 220–227 (cit. on p. 123).

[284] L. Habel, A. Molina, T. Zaksek, K. Kersting, and M. Schreckenberg. “Traffic simulations with empirical data – How to replace missing traffic flows?” In: *Traffic and Granular Flow ’15*. Ed. by V. L. Knoop and W. Daamen. Springer, May 2016, pp. 491–498. DOI: http://dx.doi.org/10.1007/978-3-319-33482-0_62 (cit. on p. 92). **SFB876-B4**

[285] Haci Bayhan, Dietmar Ebel, Timo Erler, Lorenz Kiebler, Kira Schmeltzpfenning, and Robert Schulze Forsthövel. *Logistik IT im Wandel: Einbindung dezentraler IT-Strukturen am Beispiel eines Cyberphysischen Produktionssystems (CPPS)*. 2021. URL: https://leistungszentrum-logistik-it.de/wp-content/uploads/2021/02/Whitepaper_Logistik-IT-im-Wandel.pdf (cit. on p. 45).

[286] F. Hadiji, A. Molina, S. Natarajan, and K. Kersting. “Poisson Dependency Networks: Gradient Boosted Models for Multivariate Count Data”. In: *Machine Learning Journal* 100.2 (2015), pp. 477–507. DOI: http://dx.doi.org/10.1007/s10994-015-5506-z (cit. on p. 92). **SFB876-B4**

[287] S. F. Hafstein, R. Chrobok, A. Pottmeier, and M. S. and F. Mazur. “A High-Resolution Cellular Automata Traffic Simulation Model with Application in a Freeway Traffic Information System”. In: *Computer-Aided Civil and Infrastructure Engineering* 19.5 (2004), pp. 338–350 (cit. on p. 101).

[288] R. T. Haftka, D. Villanueva, and A. Chaudhuri. “Parallel surrogate-assisted global optimization with expensive functions – a survey”. In: *Structural and Multidisciplinary Optimization* 54.1 (2016), pp. 3–13 (cit. on p. 287).

[289] L. A. Hall and D. B. Shmoys. “Jackson’s Rule for Single-Machine Scheduling: Making a Good Heuristic Better”. In: *Mathematics of Operations Research* 17.1 (1992), pp. 22–35. DOI: https://doi.org/10.1287/moor.17.1.22 (cit. on p. 365).

[290] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst. “Communication Centric Design in Complex Automotive Embedded Systems”. In: *Procs. of the Euromicro Conference on Real-Time Systems 2017*. 2017, 10:1–10:20 (cit. on p. 368).

[291] W. L. Hamilton. “Graph Representation Learning”. In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (2020), pp. 1–159 (cit. on pp. 130, 145).

[292] W. L. Hamilton, R. Ying, and J. Leskovec. “Inductive Representation Learning on Large Graphs”. In: *arXiv: Computing Research Repository* (2017). DOI: arXiv:1706.02216 (cit. on pp. 125, 126, 130, 132, 136, 141).

[293] J. Han, K. Song, F. Nie, and X. Li. “Bilateral k-Means Algorithm for Fast Co-Clustering.” In: *Procs. of the AAAI Conference on Artificial Intelligence 2017*. 2017, pp. 1969–1975 (cit. on p. 232).

[294] S. Har-Peled. “A Simple Algorithm for Maximum Margin Classification, Revisited”. In: *arXiv: Computing Research Repository* (2015). DOI: arXiv:1507.01563 (cit. on p. 87).

[295] S. Har-Peled and S. Mazumdar. “On coresets for k-means and k-median clustering”. In: *Procs. of the ACM Symposium on Theory of Computing 2004*. ACM, 2004, pp. 291–300 (cit. on pp. 86–88).

[296] S. Har-Peled and B. Raichel. “The Fréchet Distance Revisited and Extended”. In: *ACM Transactions on Algorithms* 10.1 (2014). Previously appeared in Procs. of the Int. Symposium on Computational Geometry 2011, 3:1–3:22. DOI: http://doi.acm.org/10.1145/2532646 (cit. on p. 202).

[297] S. Har-Peled, D. Roth, and D. Zimak. “Maximum Margin Coresets for Active and Noise Tolerant Learning”. In: *Procs. of the Int. Joint Conference on Artificial Intelligence 2007*. 2007, pp. 836–841 (cit. on p. 87).

[298] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. 2nd. Statistics. Springer, 2009 (cit. on p. XI).

[299] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. New York, USA: Springer, 2001 (cit. on p. 184).

[300] B. He et al. “Relational Joins on Graphics Processors”. In: *Procs. of the ACM SIGMOD Int. Conference on Management of Data 2008*. Vancouver, BC, Canada, June 2008, pp. 511–524 (cit. on p. 387).

[301] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. M. Kadie. “Dependency Networks for Collaborative Filtering and Data Visualization”. In: *Procs. of the Conference on Uncertainty in Artificial Intelligence 2000*. 2000, pp. 264–273. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1%5C&smnu=2%5C&article_id=31%5C&proceeding_id=16 (cit. on p. 92).

[302] M. Heimel, M. Saecker, H. Pirk, S. Manegold, and V. Markl. “Hardware-Oblivious Parallelism for In-Memory Column-Stores”. In: *Procs. of the VLDB Endowment* 6.9 (2013), pp. 709–720 (cit. on p. 382).

[303] M. Heinrich, A. Munteanu, and C. Sohler. “Asymptotically exact streaming algorithms”. In: *arXiv: Computing Research Repository* (2014). DOI: arXiv:1408.1847 (cit. on p. 90).

[304] H. Hellbrück, M. Pagel, A. Köller, D. Bimschas, D. Pfisterer, and S. Fischer. “Using and Operating Wireless Sensor Network Testbeds with WISEBED”. In: *Procs. of the IFIP Annual Mediterranean Ad Hoc Networking Workshop 2011*. 2011, pp. 171–178 (cit. on p. 35).

[305] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau. “Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning”. In: *Journal of Machine Learning Research* 21.248 (2020), pp. 1–43. URL: http://jmlr.org/papers/v21/20-312.html (cit. on p. 6).

[306] S. Henwood, F. Leduc-Primeau, and Y. Savaria. “Layerwise Noise Maximisation to Train Low-Energy Deep Neural Networks”. In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1912.10764 (cit. on pp. 326, 327).

[307] D. Herr, Q. Han, S. Lohmann, and T. Ertl. “Visual Clutter Reduction through Hierarchy-based Projection of High-dimensional Labeled Data”. In: *Procs. of the Graphics Interface Conference 2016*. 2016, pp. 109–116 (cit. on p. 216).

[308] S. Hess, W. Duivesteijn, P.-J. Honysz, and K. Morik. “The SpectACl of Nonconvex Clustering: a Spectral Approach to Density-Based Clustering”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2019*. 2019 (cit. on p. 231). **SFB876-C1**

[309] S. Hess and K. Morik. “C-SALT: Mining Class-Specific ALTerations in Boolean Matrix Factorization”. In: *Procs. of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases 2017*. Springer, 2017. URL: https://link.springer.com/content/pdf/10.1007/978-3-319-71249-9_33.pdf (cit. on pp. 228, 238). **SFB876-C1**

[310] S. Hess, K. Morik, and N. Piatkowski. “The PRIMPING routine—Tiling through proximal alternating linearized minimization”. In: *Data Mining and Knowledge Discovery* 31.4 (July 2017), pp. 1090–1131. DOI: https://doi.org/10.1007/s10618-017-0508-z (cit. on pp. 228, 234, 237–239, 246). **SFB876-A1, SFB876-C1**

[311] S. Hess, N. Piatkowski, and K. Morik. “The Trustworthy Pal: Controlling the False Discovery Rate in Boolean Matrix Factorization”. In: *Procs. of the 2018 SIAM Int. Conference on Data Mining 2018*. SIAM. 2018, pp. 405–413. DOI: https://doi.org/10.1137/1.9781611975321.46 (cit. on pp. 228, 237, 238). **SFB876-A1, SFB876-C1**

[312] S. Hess, G. Pio, M. Hochstenbach, and M. Ceci. “BROCCOLI: overlapping and outlier-robust biclustering through proximal stochastic gradient descent”. In: *Data Mining and Knowledge Discovery* (2021), pp. 1–35 (cit. on pp. 228, 238, 244, 246).

[313] J. Hester and J. Sorber. “Flicker: Rapid prototyping for the batteryless internet-of-things”. In: *Procs. of the SenSys Conference 2017*. ACM, 2017 (cit. on p. 57).

[314] S. Hido and H. Kashima. “A Linear-Time Graph Kernel”. In: *Procs. of the IEEE Int. Conference on Data Mining 2009*. 2009, pp. 179–188 (cit. on p. 125).

[315] T. Hirtzlin et al. “Implementing Binarized Neural Networks with Magnetoresistive RAM without Error Correction”. In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1908.04085 (cit. on p. 327).

[316] T. Hirtzlin et al. “Outstanding Bit Error Tolerance of Resistive RAM-Based Binarized Neural Networks”. In: *Procs. of the Int. Conference on Artificial Intelligence Circuits and Systems 2019*. 2019 (cit. on pp. 327, 330, 331).

[317] D. S. Hochbaum and D. B. Shmoys. “A best possible heuristic for the *k*-center problem”. In: *Mathematics of Operations Research* 10.2 (1985), pp. 180–184 (cit. on p. 200).

[318] S. Hochreiter et al. “FABIA: Factor Analysis for Bicluster Acquisition”. In: *Bioinformatics (Oxford, England)* 26 (Apr. 2010), pp. 1520–7 (cit. on p. 239).

[319] W. Hoeffding. “Probability inequalities for sums of bounded random variables”. In: *Journal of the American Statistical Association* 58.301 (1963), pp. 13–30 (cit. on p. 377).

[320] W. Hu et al. “Open Graph Benchmark: Datasets for Machine Learning on Graphs”. In: *arXiv: Computing Research Repository* (2020). DOI: arXiv : 2005 . 00687 (cit. on pp. 129, 141). **SFB876-A6**

[321] Q. Huang, H. He, A. Singh, S. N. Lim, and A. R. Benson. “Combining Label Propagation and Simple Models Out-performs Graph Neural Networks”. In: *Procs. of the Int. Conference on Learning Representations 2021*. 2021 (cit. on pp. 130, 137).

[322] R. Huang, V. Pavlovic, and D. Metaxas. “A New Spatio-Temporal MRF Framework for Video-based Object Segmentation”. In: *Procs. of the Int. Workshop on Machine Learning for Vision-based Motion Analysis 2008*. Marseille, France, 2008 (cit. on p. 102).

[323] W. Huang, T. Zhang, Y. Rong, and J. Huang. “Adaptive Sampling Towards Fast Graph Representation Learning”. In: *Advances in Neural Information Processing Systems 31: Procs. of the 2018 Conference*. 2018 (cit. on p. 136).

[324] W.-H. Huang, J.-J. Chen, H. Zhou, and C. Liu. “PASS: Priority Assignment of Real-Time Tasks with Dynamic Suspending Behavior under Fixed-Priority Scheduling”. In: *Procs. of the Design Automation Conference 2015*. 2015 (cit. on p. 363). **SFB876-B2**

[325] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. “Binarized Neural Networks”. In: *Advances in Neural Information Processing Systems 29: Procs. of the 2016 Conference*. 2016, pp. 4107–4115 (cit. on pp. 10, 329).

[326] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. “Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations”. In: *Journal of Machine Learning Research* 18 (2018) (cit. on p. 253).

[327] J. H. Huggins, T. Campbell, and T. Broderick. “Coresets for Scalable Bayesian Logistic Regression”. In: *Advances in Neural Information Processing Systems 29: Procs. of the 2016 Conference*. 2016, pp. 4080–4088. URL: http://papers.nips.cc/paper/6486-coresets-for-scalable-bayesian-logistic-regression (cit. on pp. 87, 90, 91).

[328] F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Parallel Algorithm Configuration”. In: *Learning and Intelligent Optimization*. Ed. by Y. Hamadi and M. Schoenauer. Lecture Notes in Computer Science 7219. Springer Berlin Heidelberg, 2012, pp. 55–70. URL: http://link.springer.com/chapter/10.1007/978-3-642-34413-8_5 (cit. on p. 287).

[329] F. Hutter, H. H. Hoos, and K. Leyton-Brown. “Sequential Model-Based Optimization for General Algorithm Configuration”. In: *Learning and Intelligent Optimization*. Ed. by C. A. Coello. Lecture Notes in Computer Science 6683. Springer Berlin Heidelberg, 2011, pp. 507–523. URL: http://link.springer.com/chapter/10.1007/978-3-642-25566-3_40 (cit. on p. 292).

[330] T. I. “Longest Common Extensions with Recompression”. In: *Procs. of the Symposium on Combinatorial Pattern Matching 2017*. Ed. by J. Kärkkäinen, J. Radoszewski, and W. Rytter. Vol. 78. Leibniz Int. Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, 18:1–18:15 (cit. on p. 157).

[331] A. Ihler and D. McAllester. “Particle Belief Propagation”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2009*. Ed. by D. van Dyk and M. Welling. 2009, pp. 256–263 (cit. on p. 407).

[332] A. T. Ihler, J. W. Fischer III, and A. S. Willsky. “Loopy Belief Propagation: Convergence and Effects of Message Errors”. In: *Journal of Machine Learning Research* 6 (Dec. 2005), pp. 905–936 (cit. on p. 413).

[333] P. Indyk. “High-dimensional Computational Geometry”. PhD thesis. Stanford University, 2000 (cit. on p. 201).

[334] Intel Corporation. *Intel 64 and IA-32 Architectures Optimization Reference Manual*. 2016. URL: http://www.intel.de/content/www/de/de/architecture-and-technology/64-ia-32-architectures-optimization-manual.html (cit. on p. 408).

[335] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Procs. of the Int. Conference on Machine Learning 2015*. 2015, pp. 448–456. URL: http://jmlr.org/proceedings/papers/v37/ioffe15.html (cit. on p. 166).

[336] H. Jain, Y. Prabhu, and M. Varma. "Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications". In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2016*. 2016, pp. 935–944 (cit. on p. 284).

[337] K. Jain, M. Mahdian, and A. Saberi. "A New Greedy Approach for Facility Location Problems". In: *Procs. of the Symposium on Theory of Computing 2002*. ACM, 2002, pp. 731–740. DOI: http://doi.acm.org/10.1145/509907.510012 (cit. on p. 200).

[338] J. Janusevskis, R. Le Riche, and D. Ginsbourger. *Parallel Expected Improvements for Global Optimization: Summary, Bounds and Speed-Up*. Tech. rep. 2011, pp. 1–21. URL: https://hal.archives-ouvertes.fr/hal-00613971 (cit. on p. 292).

[339] J. Janusevskis, R. Le Riche, D. Ginsbourger, and R. Girdziusas. "Expected Improvements for the Asynchronous Parallel Global Optimization of Expensive Functions: Potentials and challenges". In: *Learning and Intelligent Optimization*. Springer, 2012, pp. 413–418 (cit. on pp. 288, 292).

[340] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. "What is the best multi-stage architecture for object recognition?" In: *Procs. of the IEEE Int. Conference on Computer Vision 2009*. IEEE Computer Society, 2009, pp. 2146–2153 (cit. on p. 242).

[341] H. Jayakumar, A. Raha, and V. Raghunathan. "QUICKRECALL: A Low Overhead HW/SW Approach for Enabling Computations across Power Cycles in Transiently Powered Computers". In: *Procs. of the Int. Conference on VLSI Design 2014* (2014) (cit. on p. 53).

[342] A. Jez. "Recompression: A Simple and Powerful Technique for Word Equations". In: *Journal of the Association for Computing Machinery* 63.1 (Feb. 2016) (cit. on pp. 154, 157).

[343] H. Jiang, T. Kathuria, Y. T. Lee, S. Padmanabhan, and Z. Song. "A Faster Interior Point Method for Semidefinite Programming". In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2020*. 2020, pp. 910–918 (cit. on p. 150).

[344] W. Jiang and S. G. Kong. "Block-based neural networks for personalized ECG signal classification". In: *IEEE Transactions on Neural Networks* 18.6 (2007), pp. 1750–1761 (cit. on p. 254).

[345] W. Jiang, S. G. Kong, and G. D. Peterson. "Continuous heartbeat monitoring using evolvable block-based neural networks". In: *Procs. of the IEEE Int. Joint Conference on Neural Networks 2006*. IEEE. 2006, pp. 1950–1957 (cit. on p. 254).

[346] W. Jiang, S. G. Kong, and G. D. Peterson. "ECG signal classification using block-based neural networks". In: *Procs. of the IEEE Int. Joint Conference on Neural Networks 2005*. Vol. 1. IEEE. 2005, pp. 326–331 (cit. on p. 254).

[347] Z. Jiang, H. Liu, B. Fu, and Z. Wu. "Generalized ambiguity decompositions for classification with applications in active learning and unsupervised ensemble pruning". In: *Procs. of the AAAI Conference on Artificial Intelligence 2017*. 2017, pp. 2073–2079 (cit. on p. 340).

[348] D. R. Jones, M. Schonlau, and W. J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13.4 (1998), 455–492. URL: http://link.springer.com/article/10.1023/A:1008306431147 (cit. on p. 286).

[349] P. Joshi, D. Colombi, B. Thors, L. E. Larsson, and C. Törnevik. "Output Power Levels of 4G User Equipment and Implications on Realistic RF EMF Exposure Assessments". In: *IEEE Access* 5 (2017), pp. 4545–4550 (cit. on p. 435).

[350] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. “Bag of Tricks for Efficient Text Classification”. In: *Procs. of the Conference of the European Chapter of the Association for Computational Linguistics 2017*. 2017, pp. 1–55. DOI: arXiv:1511.09249 (cit. on pp. 175, 176).

[351] B. D. Jovanovic and P. S. Levy. “A Look at the Rule of Three”. In: *The American Statistician* 51.2 (1997), pp. 137–139. URL: https://www.tandfonline.com/doi/abs/10.1080/00031305.1997.10473947 (cit. on p. 79).

[352] S. Jung et al. “A crossbar array of magnetoresistive memory devices for in-memory computing”. In: *Nature* 601.7892 (2022), pp. 211–216 (cit. on p. 4).

[353] T. Kalibera, P. Maj, F. Morandat, and J. Vitek. “A Fast Abstract Syntax Tree Interpreter for R”. In: *Procs. of the ACM SIGPLAN/SIGOPS Int. Conference on Virtual Execution Environments 2014*. VEE ’14. ACM, 2014, pp. 89–102 (cit. on p. 315).

[354] R. Kannan, S. Vempala, and D. P. Woodruff. “Principal Component Analysis and Higher Correlations for Distributed Data”. In: *Procs. of the Conference on Learning Theory 2014*. 2014, pp. 1040–1057 (cit. on p. 89).

[355] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. “A local search approximation algorithm for *k*-means clustering”. In: *Computational Geometry: Theory and Applications* 28.2-3 (2004). Previously appeared in Procs. of the Int. Symposium on Computational Geometry 2002, pp. 89–112 (cit. on p. 184).

[356] S. Karaev, P. Miettinen, and J. Vreeken. “Getting to know the unknown unknowns: Destructive-noise resistant Boolean matrix factorization”. In: *Procs. of the SIAM Int. Conference on Data Mining 2015*. SIAM. 2015, pp. 325–333 (cit. on p. 239).

[357] O. Kariv and S. Hakimi. “An Algorithmic Approach to Network Location Problems. II: The p-Medians”. In: *SIAM Journal on Applied Mathematics* 37.3 (1979), pp. 539–560 (cit. on p. 183).

[358] T. Karnagel, R. Mueller, and G. M. Lohman. “Optimizing GPU-Accelerated Group-By and Aggregation”. In: *Procs. of the Int. Workshop on Accelerating Data Management Systems at the Int. Conference on Very Large Data Bases 2015*. 2015, pp. 13–24 (cit. on pp. 384, 392).

[359] R. M. Karp. “Reducibility among Combinatorial Problems”. In: *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*. Springer US, 1972, pp. 85–103 (cit. on p. 144).

[360] V. Kartsch, S. Benatti, M. Mancini, M. Magno, and L. Benini. “Smart wearable wristband for EMG based gesture recognition powered by solar energy harvester”. In: *Procs. of the Int. Symposium on Computer Architecture 2018*. IEEE. 2018, pp. 1–5 (cit. on p. 61).

[361] G. Karypis and V. Kumar. “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”. In: *SIAM Journal on Scientific Computing* 20.1 (1998), pp. 359–392 (cit. on p. 140).

[362] S. Kato, R. Rajkumar, and Y. Ishikawa. “A loadable real-time scheduler suite for multicore platforms”. In: *Technical Report CMU-ECE-TR09-12* (2009) (cit. on p. 374).

[363] L. Kaufman and P. J. Rousseeuw. “Clustering by means of medoids”. In: *Statistical Data Analysis Based on the L_1 Norm and Related Methods*. Ed. by Y. Dodge. North-Holland, 1987, pp. 405–416 (cit. on pp. 182, 188).

[364] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 1990 (cit. on pp. 215, 218).

[365] L. Kaufman and P. J. Rousseeuw. “Partitioning Around Medoids (Program PAM)”. In: *Finding Groups in Data*. John Wiley&Sons, 1990. Chap. 2, pp. 68–125 (cit. on pp. 182, 188).

[366] J. Kays, A. Seack, T. Smirek, F. Westkamp, and C. Rehtanz. “The Generation of Distribution Grid Models on the Basis of Public Available Data”. In: 32.3 (2017), pp. 2346–2353 (cit. on p. 185).

[367] E. Kazemi, M. Mitrovic, M. Zadimoghaddam, S. Lattanzi, and A. Karbasi. “Submodular Streaming in All Its Glory: Tight Approximation, Minimum Memory and Low Adaptive Complexity”. In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019, pp. 3311–3320. URL: http://proceedings.mlr.press/v97/kazemi19a.html (cit. on pp. 76, 77).

[368] S. S. Keerthi, D. DeCoste, and T. Joachims. “A modified finite Newton method for fast solution of large scale linear SVMs.” In: *Journal of Machine Learning Research* 6.3 (2005) (cit. on p. 278).

[369] K. Kersting, B. Ahmadi, and S. Natarajan. “Counting belief propagation”. In: *Procs. of the Conference on Uncertainty in Artificial Intelligence 2009*. 2009, pp. 277–284 (cit. on p. 407).

[370] A. H. Khan. “Lightweight Neural Networks”. In: *arXiv: Computing Research Repository* (2017). DOI: arXiv:1712.05695 (cit. on p. 61).

[371] S. Khandagale, H. Xiao, and R. Babbar. “Bonsai: diverse and shallow trees for extreme multi-label classification”. In: *Machine Learning Journal* 109.11 (2020) (cit. on p. 283).

[372] S. Khot, G. Kindler, E. Mossel, and R. O’Donnell. “Optimal Inapproximability Results for MAX-CUT and Other 2-Variable CSPs?” In: *SIAM Journal on Computing (SICOMP)* 37.1 (2007), pp. 319–357 (cit. on p. 145).

[373] C. Kim et al. “FAST: Fast architecture sensitive tree search on modern CPUs and GPUs”. In: *Procs. of the ACM SIGMOD Int. Conference on Management of Data 2010*. ACM. 2010, pp. 339–350 (cit. on p. 341).

[374] D. Kim, T. Na, S. Yalamanchili, and S. Mukhopadhyay. “DeepTrain: A Programmable Embedded Platform for Training Deep Neural Networks”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2360–2370 (cit. on pp. 251, 253).

[375] T. N. Kipf and M. Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *Procs. of the Int. Conference on Learning Representations 2017*. 2017 (cit. on pp. 125, 130, 132, 136, 141).

[376] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. “Neural Relational Inference for Interacting Systems”. In: *Procs. of the Int. Conference on Machine Learning 2018*. Ed. by J. Dy and A. Krause. Vol. 80. Procs. of Machine Learning Research. PMLR, June 2018, pp. 2688–2697 (cit. on p. 99).

[377] H. Kise, T. Ibaraki, and H. Mine. “Performance Analysis of six Approximation Algorithms for the One-Machine Maximum Lateness Scheduling Problem with Ready Times”. In: *Journal of the Operations Research Society of Japan* 22.3 (1979), pp. 205–224 (cit. on p. 365).

[378] J. Klicpera, A. Bojchevski, and S. Günnemann. “Predict then Propagate: Graph Neural Networks meet Personalized PageRank”. In: *Procs. of the Int. Conference on Learning Representations 2019*. 2019 (cit. on pp. 130, 132, 136, 137).

[379] O. Koch, N. M. Kriege, and L. Humbeck. “Chemical Similarity and Substructure Searches”. In: *Encyclopedia of Bioinformatics and Computational Biology - Volume 2*. Ed. by S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach. Elsevier, 2019, pp. 640–649. DOI: https://doi.org/10.1016/b978-0-12-809633-8.20195-7 (cit. on p. 116).

[380] P. Koch, M. Dreier, M. Maass, M. Böhme, H. Phan, and A. Mertins. “A recurrent neural network for hand gesture recognition based on accelerometer data”. In: *Procs. of the Int. Conference of the IEEE Engineering in Medicine and Biology Society 2019*. IEEE. 2019, pp. 5088–5091 (cit. on p. 61).

[381] S. Koppula et al. “EDEN: Enabling Energy-Efficient, High-Performance Deep Neural Network Inference Using Approximate DRAM”. In: *Procs. of the Int. Symposium on Microarchitecture 2019*. 2019 (cit. on pp. 327, 329).

[382] A. Kopytov. *SysBench: a system performance benchmark*. 2004. URL: https://github.com/akopytov/sysbench (cit. on p. 28).

[383] I. Korb, H. Kotthaus, and P. Marwedel. “mmapcopy: Efficient Memory Footprint Reduction using Application-Knowledge”. In: *Procs. of the ACM Symposium on Applied Computing 2016*. 2016. DOI: https://dl.acm.org/doi/pdf/10.1145/2851613.2851736 (cit. on pp. 307, 310). **SFB876-A3**

[384] H. Kotthaus and M. Lang. *BenchR: Set of Benchmark of R*. TU Dortmund University. 2018. URL: https://github.com/allr/benchR (cit. on pp. 314, 315).

[385] H. Kotthaus. “Methods for Efficient Resource Utilization in Statistical Machine Learning Algorithms”. PhD thesis. Dortmund: TU Dortmund University, 2018. DOI: http://dx.doi.org/10.17877/DE290R-18928 (cit. on pp. 289, 293, 295–299, 301–303, 307, 309–312, 314, 317, 318, 320–323). **SFB876-A3**

[386] H. Kotthaus, I. Korb, M. Engel, and P. Marwedel. “Dynamic Page Sharing Optimization for the R Language”. In: *Procs. of the Symposium on Dynamic Languages 2014*. DLS ’14. Portland, Oregon, USA: ACM, 2014, pp. 79–90. DOI: https://dl.acm.org/doi/10.1145/2661088.2661094 (cit. on p. 307). **SFB876-A3**

[387] H. Kotthaus, I. Korb, M. Lang, B. Bischl, J. Rahnenführer, and P. Marwedel. “Runtime and Memory Consumption Analyses for Machine Learning R Programs”. In: *Journal of Statistical Computation and Simulation* 85.1 (2014), pp. 14–29. URL: http://www.tandfonline.com/eprint/T3mgYXAWdY4kWuDeSv2A/full (cit. on pp. 306, 323). **SFB876-A3**

[388] H. Kotthaus, J. Richter, A. Lang, M. Lang, and P. Marwedel. *Resource-Aware Scheduling Strategies for Parallel Machine Learning R Programs through RAMBO*. Stanford University, Palo Alto, California, July 2016, p. 195. URL: http://user2016.r-project.org/files/abs-book.pdf (cit. on p. 11).

[389] H. Kotthaus, L. Schönberger, A. Lang, J.-J. Chen, and P. Marwedel. “Can Flexible Multi-Core Scheduling Help to Execute Machine Learning Algorithms Resource-Efficiently?” In: *Procs. of the Int. Workshop on Software and Compilers for Embedded Systems 2019*. SCOPES ’19. ACM, 2019, pp. 59–62. DOI: https://dl.acm.org/doi/10.1145/3323439.3323986 (cit. on pp. 289, 292, 295). **SFB876-A3, SFB876-C5**

[390] H. Kotthaus et al. “RAMBO: Resource-Aware Model-Based Optimization with Scheduling for Heterogeneous Runtimes and a Comparison with Asynchronous Model-Based Optimization”. In: *Procs. of the Int. Conference on Learning and Intelligent Optimization 2017*. 2017, pp. 180–195. URL: https://www.springerprofessional.de/en/rambo-resource-aware-model-based-optimization-with-scheduling-fo/15164982 (cit. on pp. 289, 292). **SFB876-A3**

[391] M. Koyutürk and A. Grama. “PROXIMUS: a framework for analyzing very high dimensional discrete-attributed datasets”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2003*. ACM. 2003, pp. 147–156 (cit. on pp. 232, 233).

[392] S. Kramer, D. Ziegenbein, and A. Hamann. “Real world automotive benchmark for free”. In: *Procs. of the Int. Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems 2015*. 2015 (cit. on p. 368).

[393] A. Krause and D. Golovin. *Submodular function maximization*. 2014. URL: http://www.cs.cmu.edu/afs/.cs.cmu.edu/Web/People/dgolovin/papers/submodular_survey12.pdf (cit. on p. 76).

[394] N. Kriege and P. Mutzel. “Subgraph Matching Kernels for Attributed Graphs”. In: *Procs. of the Int. Conference on Machine Learning 2012*. Omnipress, 2012 (cit. on p. 124).

[395] N. Kriege, P. Giscard, and R. C. Wilson. “On Valid Optimal Assignment Kernels and Applications to Graph Classification”. In: *arXiv: Computing Research Repository* (2016). DOI: arXiv:1606.01141 (cit. on pp. 121, 122). **SFB876-A6**

[396] N. M. Kriege. “Deep Weisfeiler-Lehman Assignment Kernels via Multiple Kernel Learning”. In: *Procs. of the European Symposium on Artificial Neural Networks 2019*. 2019 (cit. on p. 122). **SFB876-A6**

[397] N. M. Kriege, P.-L. Giscard, F. Bause, and R. C. Wilson. "Computing Optimal Assignments in Linear Time for Approximate Graph Matching". In: *Procs. of the IEEE Int. Conference on Data Mining 2019*. 2019 (cit. on p. 117). **SFB876-A6**

[398] N. M. Kriege, F. D. Johansson, and C. Morris. "A Survey on Graph Kernels". In: *Applied Network Science* 5.1 (2020), p. 6. DOI: https://doi.org/10.1007/s41109-019-0195-3 (cit. on p. 116). **SFB876-A6**

[399] H. Kriegel, E. Schubert, and A. Zimek. "The (black) art of runtime evaluation: Are we comparing algorithms or implementations?" In: *Knowledge and Information Systems (KAIS)* 52.2 (2017). Online first 2016, paginated 2017, pp. 341–378. DOI: https://doi.org/10.1007/s10115-016-1004-2 (cit. on pp. 192, 223).

[400] R. Krishnakumar. "Kernel Korner: Kprobes – a Kernel Debugger". In: *Linux Journal* 2005.133 (May 2005), p. 11 (cit. on pp. 17, 18).

[401] A. Krivošija. "On clustering and related problems on curves under the Fréchet distance". PhD thesis. TU Dortmund University, 2021. DOI: http://dx.doi.org/10.17877/DE290R-22055 (cit. on p. 204).

[402] A. Krivošija and A. Munteanu. "Probabilistic smallest enclosing ball in high dimensions via subgradient sampling". In: *Procs. of the Int. Symposium on Computational Geometry 2019*. Ed. by G. Barequet and Y. Wang. Vol. 129. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, 47:1–47:14. DOI: https://doi.org/10.4230/LIPIcs.SoCG.2019.47 (cit. on pp. 87, 212). **SFB876-A2, SFB876-C4**

[403] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25: Procs. of the 2012 Conference*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105 (cit. on p. 254).

[404] F. Kschischang, S. Member, B. J. Frey, and H.-a. Loeliger. "Factor Graphs and the Sum-Product Algorithm". In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 498–519 (cit. on pp. 105, 407, 411, 413).

[405] M. Kumar, R. Ghani, and Z.-S. Mei. "Data mining to predict and prevent errors in health insurance claims processing". In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2010*. ACM. 2010, pp. 65–74 (cit. on pp. 201, 212).

[406] S. Kumar, S. Gollakota, and D. Katabi. "A Cloud-Assisted Design for Autonomous Driving". In: *Procs. of the MCC Workshop on Mobile Cloud Computing 2012*. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 41–46. DOI: https://doi.org/10.1145/2342509.2342519 (cit. on p. 363).

[407] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Procs. of the Int. Conference on Machine Learning 2001*. Ed. by C. E. Brodley and A. P. Danyluk. Morgan Kaufmann, 2001, pp. 282–289 (cit. on pp. 106, 416).

[408] K. P. Lakshmi and M. Subadra. "A survey on FPGA based MLP realization for on-chip learning". In: *International Journal of Scientific & Engineering Research* 4.1 (2013), pp. 1–9 (cit. on p. 253).

[409] W. H. K. Lam, Y. F. Tang, and M. Tam. "Comparison of two non-parametric models for daily traffic forecasting in Hong Kong". In: *Journal of Forecasting* 25.3 (2006), pp. 173–192 (cit. on p. 102).

[410] P. Lancaster and M. Tismenetsky. *The Theory of Matrices*. Academic Press, 1985 (cit. on p. 150).

[411] G. N. Lance and W. T. Williams. "A General Theory of Classificatory Sorting Strategies: 1. Hierarchical Systems". In: *The Computer Journal* 9.4 (Feb. 1967), pp. 373–380. DOI: https://doi.org/10.1093/comjnl/9.4.373 (cit. on pp. 215, 216).

[412] G. N. Lance and W. T. Williams. “A Generalized Sorting Strategy for Computer Classifications”. In: *Nature* 212.5058 (Oct. 1966), pp. 218–218. DOI: https://doi.org/10.1038/212218a0 (cit. on pp. 215, 216).

[413] A. Lang and E. Schubert. “BETULA: Fast Clustering of Large Data with Improved BIRCH CF-Trees”. In: *Information Systems* (2021). DOI: https://doi.org/10.1016/j.is.2021.101918 (cit. on pp. 216, 219–221). **SFB876-A2**

[414] A. Lang and E. Schubert. “BETULA: Numerically Stable CF-Trees for BIRCH Clustering”. In: *Procs. of the Int. Conference on Similarity Search and Applications 2020*. best paper candidate. 2020, pp. 281–296. DOI: https://doi.org/10.1007/978-3-030-60936-8_22 (cit. on pp. 216, 219, 220). **SFB876-A2**

[415] M. Lang, B. Bischl, and D. Surmann. “batchtools: Tools for R to Work on Batch Systems”. In: *The Journal of Open Source Software* 2.10 (2017) (cit. on p. 294).

[416] M. Langberg and L. J. Schulman. “Universal ϵ-approximators for Integrals”. In: *Procs. of the ACM-SIAM Symposium on Discrete Algorithms 2010*. 2010, pp. 598–607 (cit. on p. 91).

[417] M. Langhammer and B. Pasca. “Activation Function Architectures for FPGAs”. In: *Procs. of the Int. Conference on Field Programmable Logic and Applications 2017*. IEEE. 2017, pp. 1–6 (cit. on p. 253).

[418] J. Lässig, K. Kersting, and K. Morik. *Computational Sustainability*. Ed. by J. Lässig, K. Kersting, and K. Morik. Springer, 2016. URL: http://link.springer.com/book/10.1007/978-3-319-31858-5 (cit. on p. 6).

[419] L. Lazzeroni and A. Owen. “Plaid models for gene expression data”. In: *Statistica sinica* (2002), pp. 61–86 (cit. on p. 233).

[420] Y. LeCun. “The MNIST database of handwritten digits”. In: *http://yann. lecun. com/exdb/mnist/* (1998) (cit. on pp. 351, 352).

[421] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444 (cit. on pp. 253, 255).

[422] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. “Efficient backprop”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48 (cit. on p. 252).

[423] D. D. Lee and H. S. Seung. “Learning the parts of objects by non-negative matrix factorization”. In: *Nature* 401.6755 (1999), pp. 788–791 (cit. on p. 229).

[424] H. G. Lee and N. Chang. “Powering the IoT: Storage-less and converter-less energy harvesting”. In: *Procs. of the Asia and South Pacific Design Automation Conference 2015*. Jan. 2015, pp. 124–129 (cit. on p. 53).

[425] V. Leis, P. Boncz, A. Kemper, and T. Neumann. “Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age”. In: *Procs. of the SIGMOD Int. Conference on the Management of Data 2014*. ACM, 2014, pp. 743–754 (cit. on p. 386).

[426] M. Lewin, D. Livnat, and U. Zwick. “Improved Rounding Techniques for the MAX 2-SAT and MAX DI-CUT Problems”. In: *Procs. of the Int. Conference on Integer Programming and Combinatorial Optimization 2002*. Ed. by W. J. Cook and A. S. Schulz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 67–82 (cit. on pp. 145, 147).

[427] H. Li and Z. Lin. “Accelerated proximal gradient methods for nonconvex programming”. In: *Advances in Neural Information Processing Systems 28: Procs. of the 2015 Conference*. 2015, pp. 379–387 (cit. on p. 247).

[428] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang. “A High Performance FPGA-based Accelerator for Large-Scale Convolutional Neural Networks”. In: *Procs. of the Int. Conference on Field-Programmable Logic and Applications 2016*. 2016 (cit. on p. 254).

[429] H. L. Li Yi and Nguyen and D. P. Woodruff. “Turnstile streaming algorithms might as well be linear sketches”. In: *Procs. of the Symposium on Theory of Computing 2014*. 2014, pp. 174–183 (cit. on p. 89).

[430] J. Li, J.-J. Chen, K. Agrawal, C. Lu, C. D. Gill, and A. Saifullah. "Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks". In: *Procs. of the Euromicro Conference on Real-Time Systems 2014*. 2014, pp. 85–96 (cit. on p. 367).

[431] J. Li, T. Luong, and D. Jurafsky. "A Hierarchical Neural Autoencoder for Paragraphs and Documents". In: *Procs. of the Meeting of the Association for Computational Linguistics and the Int. Joint Conference on Natural Language Processing 2015*. Vol. 1. 2015, pp. 1106–1115 (cit. on p. 251).

[432] M. Li, G. L. Miller, and R. Peng. "Iterative Row Sampling". In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2013*. 2013, pp. 127–136 (cit. on p. 87).

[433] S. Li and O. Svensson. "Approximating k-Median via Pseudo-Approximation". In: *SIAM Journal on Computing* 45.2 (2016). Previously appeared in the Procs. of the Symposium on Theory of Computing 2013, pp. 530–547. DOI: https://doi.org/10.1137/130938645 (cit. on p. 201).

[434] T. Li. "A general model for clustering binary data". In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2005*. ACM. 2005, pp. 188–197 (cit. on p. 232).

[435] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. "Gated Graph Sequence Neural Networks". In: *Procs. of the Int. Conference on Learning Representation 2016*. 2016 (cit. on p. 135).

[436] Y. Li, T. Li, R. A. Patel, X.-D. Yang, and X. Zhou. "Self-powered gesture recognition with ambient light". In: *Procs. of the ACM Symposium on User Interface Software and Technology 2018*. 2018, pp. 595–608 (cit. on p. 61).

[437] Y. Li and A. Pedram. "CATERPILLAR: Coarse Grain Reconfigurable Architecture for Accelerating the Training of Deep Neural Networks". In: *Procs. of the Int. Conference on Application-specific Systems, Architectures and Processors 2017*. IEEE. 2017, pp. 1–10 (cit. on pp. 253, 255).

[438] F. Liang, C. Liu, and N. Wang. "A robust sequential Bayesian method for identification of differentially expressed genes". In: *Statistica Sinica* (2007), pp. 571–597 (cit. on p. 95).

[439] P. Libuschewski. "Exploration of Cyber-Physical Systems for GPGPU Computer Vision-Based Detection of Biological Viruses". PhD thesis. Dortmund, Germany: TU Dortmund University, 2017. DOI: http://dx.doi.org/10.17877/DE290R-17952 (cit. on p. 339). **SFB876-B2**

[440] T. Liebig, Z. Xu, M. May, and S. Wrobel. "Pedestrian Quantity Estimation with Trajectory Patterns". In: *Procs. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2012*. Springer, 2012, pp. 629–643. URL: http://link.springer.com/chapter/10.1007%5C%2F978-3-642-33486-3_40 (cit. on p. 102).

[441] M. Lippi, M. Bertini, and P. Frasconi. "Collective Traffic Forecasting". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by J. L. Balcázar, F. Bonchi, A. Gionis, and M. Sebag. Vol. 6322. LNCS. Springer, 2010, pp. 259–273 (cit. on p. 101).

[442] C. Liu and J.-J. Chen. "Bursty-interference analysis techniques for analyzing complex real-time task models". In: *Procs. of the IEEE Real-Time Systems Symposium 2014*. IEEE. 2014, pp. 173–183 (cit. on p. 363).

[443] F. T. Liu, K. M. Ting, and Z. Zhou. "Isolation Forest". In: *Procs. of the IEEE Int. Conference on Data Mining 2008*. Dec. 2008, pp. 413–422 (cit. on p. 81).

[444] S. P. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (Mar. 1982), pp. 129–137 (cit. on p. 231).

[445] A. Lochmann, F. Bruckner, and O. Spinczyk. "Reproducible Load Tests for Android Systems with Trace-based Benchmarks". In: *Procs. of the ACM/SPEC on Int. Conference on Performance Engineering Companion 2017*. ICPE '17 Companion. New York, NY, USA: ACM Press, 2017, pp. 73–76 (cit. on p. 32). **SFB876-A1**

[446] A. Lochmann, H. Schirmeier, H. Borghorst, and O. Spinczyk. “LockDoc: Trace-Based Analysis of Locking in the Linux Kernel”. In: *Procs. of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2019*. 2019 (cit. on p. 33). **SFB876-A1**

[447] B. Long, Z. (Zhang, and P. S. Yu. “Co-Clustering by Block Value Decomposition”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2005*. KDD ’05. New York, NY, USA: Association for Computing Machinery, 2005, pp. 635–640 (cit. on p. 238).

[448] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. “The irace package: Iterated racing for automatic algorithm configuration”. In: *Operations Research Perspectives* 3 (2016), pp. 43–58. DOI: https://doi.org/10.1016/j.orp.2016.09.002 (cit. on p. 286).

[449] Z. Lu, X. Wu, X. Zhu, and J. Bongard. “Ensemble pruning via individual contribution ordering”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2010* (2010), pp. 871–880 (cit. on p. 340).

[450] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. “Quickscorer: A fast algorithm to rank documents with additive ensembles of regression trees”. In: *Procs. of the Int. ACM SIGIR Conference on Research and Development in Information Retrieval 2015*. ACM. 2015, pp. 73–82 (cit. on pp. 340, 341).

[451] C. Lucchese, S. Orlando, and R. Perego. “A Unifying Framework for Mining Approximate Top-*k* Binary Patterns”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.12 (2014), pp. 2900–2913 (cit. on p. 239).

[452] C. Lucchese, R. Perego, F. M. Nardini, N. Tonellotto, S. Orlando, and R. Venturini. “Exploiting CPU SIMD extensions to speed-up document scoring with tree ensembles”. In: *Procs. of the Int. ACM SIGIR Conference on Research and Development in Information Retrieval 2016*. 2016 (cit. on p. 340).

[453] M. Lucic, O. Bachem, and A. Krause. “Strong Coresets for Hard and Soft Bregman Clustering with Applications to Exponential Family Mixtures”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2016*. 2016, pp. 1–9 (cit. on p. 87).

[454] Y. Lv, Y. Duan, W. Kang, Z. Li, F.-Y. Wang, et al. “Traffic flow prediction with big data: A deep learning approach.” In: *IEEE Trans. Intelligent Transportation Systems* 16.2 (2015), pp. 865–873 (cit. on p. 260).

[455] Y. Ma and J. Tang. *Deep Learning on Graphs*. Cambridge University Press, 2020 (cit. on pp. 9, 130, 136).

[456] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. “Learning Word Vectors for Sentiment Analysis”. In: *Procs. of the Meeting of the Association for Computational Linguistics: Human Language Technologies 2011*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. URL: http://www.aclweb.org/anthology/P11-1015 (cit. on pp. 351, 352).

[457] Y. Maeda, H. Hirano, and Y. Kanata. “A learning rule of neural networks via simultaneous perturbation and its hardware implementation”. In: *Neural Networks* 8.2 (1995), pp. 251–259 (cit. on p. 254).

[458] Y. Maeda and T. Tada. “FPGA implementation of a pulse density neural network with learning ability using simultaneous perturbation”. In: *IEEE Transactions on Neural Networks* 14.3 (2003), pp. 688–695 (cit. on p. 254).

[459] S. Mahajan and H. Ramesh. “Derandomizing Approximation Algorithms Based on Semidefinite Programming”. In: *SIAM Journal on Computing* 28.5 (1999), pp. 1641–1663 (cit. on p. 145).

[460] M. Mahdavi, R. Zanibbi, H. Mouch, and C. Viard-gaudin. “ICDAR 2019 CROHME + TFD : Competition on Recognition of Handwritten Mathematical Expressions and Typeset Formula

Detection". In: *Procs. of the IAPR Int. Conference on Document Analysis and Recognition 2019*. 2019 (cit. on p. 162).

[461] M. W. Mahoney. "Randomized Algorithms for Matrices and Data". In: *Foundations and Trends in Machine Learning* 3.2 (2011), pp. 123–224. DOI: https://doi.org/10.1561/2200000035 (cit. on p. 93).

[462] T. Mai, C. Musco, and A. Rao. "Coresets for Classification - Simplified and Strengthened". In: *Advances in Neural Information Processing Systems 34: Procs. of the 2021 Conference*. 2021, pp. 11643–11654 (cit. on p. 91).

[463] C. Manning, P. Raghavan, and H. Schütze. "Introduction to information retrieval". In: *Natural Language Engineering* 16.1 (2010), pp. 100–103 (cit. on p. 273).

[464] B. Mansouri, D. W. Oard, C. L. Giles, and R. Zanibbi. "Tangent-CFT : An Embedding Model for Mathematical Formulas". In: *Procs. of the ACM SIGIR Int. Conference on Theory of Information Retrieval 2019*. 2019 (cit. on pp. 162, 174, 175).

[465] F. E. Maranzana. "On the Location of Supply Points to Minimize Transportation Costs". In: *IBM Systems Journal* 2.2 (1963), pp. 129–135 (cit. on p. 184).

[466] J. Marín, D. Vázquez, A. M. López, J. Amores, and B. Leibe. "Random Forests of Local Experts for Pedestrian Detection". In: *Procs. of the IEEE Int. Conference on Computer Vision 2013*. 2013, pp. 2592–2599 (cit. on p. 339).

[467] S. F. Marinosson, R. Chrobok, A. Pottmeier, J. Wahle, and M. Schreckenberg. "Simulation Framework for the Autobahn Traffic in North Rhine-Westphalia". In: *Cellular Automata – Procs. of the Int. Conf. on Cellular Automata for Research and Industry 2002*. Springer, 2002, pp. 2977–2980 (cit. on p. 101).

[468] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. "Provably Powerful Graph Networks". In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. 2019 (cit. on pp. 127, 132).

[469] M. A. Maruf and A. Azim. "Extending resources for avoiding overloads of mixed-criticality tasks in cyber-physical systems". In: *IET Cyber-Physical Systems: Theory Applications* 5.1 (2020), pp. 60–70 (cit. on p. 363).

[470] P. Marwedel. *Embedded System Design - Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. 4th ed. Springer, 2021. URL: https://link.springer.com/book/10.1007/978-3-030-60910-8 (cit. on p. XI). **SFB876-A3**

[471] M. Masoudinejad. "Modeling Energy Supply Unit of Ultra-Low Power Devices with Indoor Photovoltaic Harvesting". PhD Thesis. TU Dortmund University, 2020. URL: https://eldorado.tu-dortmund.de/handle/2003/39765 (cit. on p. 35). **SFB876-A4**

[472] M. Masoudinejad, A. K. Ramachandran Venkatapathy, J. Emmerich, and A. Riesner. "Procs. of the Int. Conference on Sensor Systems and Software 2016". In: *Sensor Systems and Software*. Springer, 2016. Chap. 4, pp. 41–52 (cit. on p. 35). **SFB876-A4**

[473] M. Masoudinejad, K. A. Venkatapathy Ramachandran, D. Tondorf, D. Heinrich, R. Falkenberg, and M. Buschhoff. "Machine Learning Based Indoor Localisation using Environmental data in PhyNetLab Warehouse". In: *Procs. of the SysTech European Conference on Smart Objects, Systems and Technologies 2018*. June 2018 (cit. on p. 45). **SFB876-A4**

[474] B. Matérn. "Spatial Variation: Stochastic Models and their Application to some Problems in Forest Surveys and other Sampling Investigations". In: *Meddelanden fran Statens Skogsforskningsinstitut* 49.5 (1960), p. 144 (cit. on p. 294).

[475] V. Maurizio. "Double k-means clustering for simultaneous classification of objects and variables". In: *Advances in Classification and Data Analysis*. Springer, 2001, pp. 43–52 (cit. on p. 232).

[476] D. Maxim and L. Cucu-Grosjean. "Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters". In: *Procs. of the IEEE Real-Time Systems Symposium 2013*. 2013, pp. 224–235 (cit. on pp. 363, 376).

[477] M. May, D. Hecker, C. Körner, S. Scheider, and D. Schulz. "A Vector-Geometry Based Spatial kNN-Algorithm for Traffic Frequency Predictions". In: *Procs. of the IEEE Int. Conference on Data Mining 2008*. IEEE Computer Society, 2008, pp. 442–447 (cit. on p. 102).

[478] J. McAuley and J. Leskovec. "Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text". In: *Procs. of the ACM Conference on Recommender Systems 2013*. Vol. 7. RecSys '13. New York, NY, USA: ACM, 2013, pp. 165–172 (cit. on pp. 280, 282).

[479] P. McCullagh and J. Nelder. *Generalized linear models*. 2nd ed. Chapman and Hall/CRC, Boca Raton, 1989 (cit. on pp. 90, 92).

[480] B. D. McKay and A. Piperno. "Practical graph isomorphism, II". In: *Journal of Symbolic Computation* 60 (2014), pp. 94–112. DOI: https://doi.org/10.1016/j.jsc.2013.09.003 (cit. on p. 117).

[481] M. D. McKay, R. J. Beckman, and W. J. Conover. "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code". In: *Technometrics* 42.1 (2000), pp. 55–61 (cit. on pp. 294, 301).

[482] L. L. McQuitty. "Elementary Linkage Analysis for Isolating Orthogonal and Oblique Types and Typal Relevancies". In: *Educational and Psychological Measurement* 17.2 (1957), pp. 207–229. DOI: https://doi.org/10.1177/001316445701700204 (cit. on pp. 216, 217).

[483] N. Megiddo and K. J. Supowit. "On the Complexity of Some Common Geometric Location Problems". In: *SIAM Journal on Computing* 13.1 (1984), pp. 182–196 (cit. on p. 200).

[484] N. Meinshausen and P. Bühlmann. "High-dimensional graphs and variable selection with the Lasso". In: *Annals of Statistics* 34.3 (2006), pp. 1436–1462 (cit. on p. 105).

[485] S. Meintrup, A. Munteanu, and D. Rohde. "Random projections and sampling algorithms for clustering of high-dimensional polygonal curves". In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. 2019, pp. 12807–12817. URL: https://papers.nips.cc/paper/9443-random-projections-and-sampling-algorithms-for-clustering-of-high-dimensional-polygonal-curves (cit. on pp. 75, 87, 203, 204). **SFB876-A2, SFB876-C4**

[486] A. K. Menon, A. S. Rawat, S. Reddi, and S. Kumar. "Multilabel reductions: what is my loss optimising?" In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. Vol. 32. 2019 (cit. on p. 274).

[487] P. Menon et al. "Relaxed Operator Fusion for In-Memory Databases: Making Compilation, Vectorization, and Prefetching Work Together At Last". In: *Procs. of the VLDB Endowment* 11.1 (2017) (cit. on p. 386).

[488] S. G. Merchant and G. D. Peterson. "Evolvable block-based neural network design for applications in dynamic environments". In: *VLSI Design* 2010 (2010), p. 4 (cit. on pp. 253, 254).

[489] D. Merrill. *CUB v1.7.0: CUDA Unbound, a Library of Warp-Wide, Block-Wide, and Device-Wide GPU Parallel Primitives*. 2017 (cit. on p. 393).

[490] R. Michalski, J. Carbonell, and T. Mitchell. *Machine Learning: An artificial intelligence approach*. Kaufman Publishers Inc., 1983 (cit. on p. XI).

[491] P. Miettinen, T. Mielikainen, A. Gionis, G. Das, and H. Mannila. "The discrete basis problem". In: *Knowledge and Data Engineering, IEEE Transactions on* 20.10 (2008), pp. 1348–1362 (cit. on pp. 233, 235).

[492] P. Miettinen and J. Vreeken. "MDL4BMF: Minimum description length for Boolean matrix factorization". In: *ACM Transactions on Knowledge Discovery from Data 2014* 8.4 (2014), p. 18 (cit. on p. 239).

[493] T. Mikolov, K. Chen, G. Corrado, and J. Dean. “Efficient Estimation of Word Representations in Vector Space”. In: *arXiv: Computing Research Repository* (2013). DOI: arXiv:1301.3781 (cit. on p. 167).

[494] B. Mirkin, P. Arabie, and L. J. Hubert. “Additive two-mode clustering: the error-variance approach revisited”. In: *Journal of classification* 12.2 (1995), pp. 243–263 (cit. on p. 231).

[495] B. Mirzasoleiman, S. Jegelka, and A. Krause. “Streaming Non-Monotone Submodular Maximization: Personalized Video Summarization on the Fly”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2018*. 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17014/15832 (cit. on p. 10).

[496] I. Misra and L. v. d. Maaten. “Self-supervised learning of pretext-invariant representations”. In: *Procs. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2020*. 2020, pp. 6707–6717 (cit. on p. 175).

[497] J. Misra and I. Saha. “Artificial neural networks in hardware: A survey of two decades of progress”. In: *Neurocomputing* 74.1 (2010), pp. 239–255 (cit. on p. 253).

[498] S. Mitra and T. Acharya. “Gesture recognition: A survey”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.3 (2007), pp. 311–324 (cit. on p. 60).

[499] S. Miyamoto, Y. Kaizu, and Y. Endo. “Hierarchical and Non-Hierarchical Medoid Clustering Using Asymmetric Similarity Measures”. In: *Procs. of the Int. Symposium on Soft Computing and Intelligent Systems 2016*. 2016, pp. 400–403 (cit. on p. 216).

[500] P. Moerland and E. Fiesler. *Neural network adaptations to hardware implementations*. Tech. rep. The Idiap Research Institute, Switzerland, 1997 (cit. on p. 253).

[501] A. Molina, A. Munteanu, and K. Kersting. “Core Dependency Networks”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2018*. 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16847 (cit. on pp. 87, 89, 92, 93). **SFB876-B4, SFB876-C4**

[502] S.-W. Moon and S.-G. Kong. “Block-based neural networks”. In: *IEEE Transactions on Neural Networks* 12.2 (2001), pp. 307–317 (cit. on p. 254).

[503] F. Morandat, B. Hill, L. Osvald, and J. Vitek. “Evaluating the design of the R language: objects and functions for data analysis”. In: *Procs. of the European Conference on Object-Oriented Programming 2012*. Springer-Verlag, 2012, pp. 104–131 (cit. on p. 306).

[504] C. Morris, G. Rattan, and P. Mutzel. “Weisfeiler and Leman Go Sparse: Towards Scalable Higher-Order Graph Embeddings”. In: *Procs. of the Int. Conference on Machine Learning 2020*. 2020 (cit. on pp. 123, 127, 128, 132).

[505] C. Morris, M. Fey, and N. M. Kriege. “The Power of the Weisfeiler-Leman Algorithm for Machine Learning with Graphs”. In: *Procs. of the Int. Joint Conferences on Artifical Intelligence 2021*. 2021 (cit. on p. 128). **SFB876-A6**

[506] C. Morris, K. Kersting, and P. Mutzel. “Glocalized Weisfeiler-Lehman Graph Kernels: Global-Local Feature Maps of Graphs”. In: *Procs. of the IEEE Int. Conference on Data Mining 2017*. 2017, pp. 327–336 (cit. on p. 123). **SFB876-A6**

[507] C. Morris, N. Kriege, K. Kersting, and P. Mutzel. “Faster Kernels for Graphs with Continuous Attributes via Hashing”. In: *Procs. of the IEEE Int. Conference on Data Mining 2016*. 2016, pp. 1095–1100 (cit. on p. 124). **SFB876-A6**

[508] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. “TUDataset: A collection of benchmark datasets for learning with graphs”. In: *Procs. of the Int. Conference on Machine Learning Workshop on Graph Representation Learning and Beyond 2020*. 2020. URL: www.graphlearning.io (cit. on pp. 124, 128). **SFB876-A6**

[509] C. Morris et al. “Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2019*. 2019. DOI: arXiv:1810.02244 (cit. on pp. 117, 127, 130, 132, 137, 139). **SFB876-A6**

[510] S. Muecke, N. Piatkowski, and K. Morik. “Hardware Accelerated Learning at the Edge”. In: *Procs. of Decentralized Machine Learning at the Edge 2019*. Ed. by M. Kamp, D. Paurat, and Y. Krishnamurthy. Springer, 2019. URL: https://dmle.iais.fraunhofer.de/papers/muecke2019hardware.pdf (cit. on p. 7).

[511] A. Munteanu. “On large-scale probabilistic and statistical data analysis”. PhD thesis. TU Dortmund University, 2018. DOI: http://dx.doi.org/10.17877/DE290R-19112 (cit. on pp. 90, 95).

[512] A. Munteanu, A. Nayebi, and M. Poloczek. “A Framework for Bayesian Optimization in Embedded Subspaces”. In: *Procs. of the Int. Conference on Machine Learning 2019*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Procs. of Machine Learning Research. Long Beach, California, USA: PMLR, June 2019, pp. 4752–4761. URL: http://proceedings.mlr.press/v97/nayebi19a.html (cit. on pp. 90, 96). **SFB876-C4**

[513] A. Munteanu, S. Omlor, and C. Peters. “p-Generalized Probit Regression and Scalable Maximum Likelihood Estimation via Sketching and Coresets”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2022*. 2022 (cit. on pp. 89, 90, 95). **SFB876-C4**

[514] A. Munteanu, S. Omlor, Z. Song, and D. P. Woodruff. “Bounding the Width of Neural Networks via Coupled Initialization - A Worst Case Analysis”. In: *Procs. of the Int. Conference on Machine Learning 2022*. 2022 (cit. on p. 90). **SFB876-C4**

[515] A. Munteanu, S. Omlor, and D. P. Woodruff. “Oblivious Sketching for Logistic Regression”. In: *Procs. of the Int. Conference on Machine Learning 2021*. 2021. URL: https://proceedings.mlr.press/v139/munteanu21a.html (cit. on pp. 89–92). **SFB876-C4**

[516] A. Munteanu and C. Schwiegelshohn. “Coresets - Methods and History: A Theoreticians Design Pattern for Approximation and Streaming Algorithms”. In: *KI - Künstliche Intelligenz* 32.1 (2018). KI special issue on ’Algorithmic Challenges and Opportunities of Big Data’, pp. 37–53. DOI: https://doi.org/10.1007/s13218-017-0519-3 (cit. on pp. 10, 86, 87, 200). **SFB876-A2, SFB876-C4**

[517] A. Munteanu, C. Schwiegelshohn, C. Sohler, and D. P. Woodruff. “On Coresets for Logistic Regression”. In: *Advances in Neural Information Processing Systems 31: Procs. of the 2018 Conference*. 2018. URL: http://papers.nips.cc/paper/7891-on-coresets-for-logistic-regression (cit. on pp. 75, 87, 89–91). **SFB876-A2, SFB876-C4**

[518] A. Munteanu and M. Wornowizki. “Correcting statistical models via empirical distribution functions”. In: *Computational Statistics* 31.2 (June 2016), pp. 465–495. URL: http://doi.org/10.1007/s00180-015-0607-5 (cit. on p. 95). **SFB876-C3, SFB876-C4**

[519] R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro. “Relational Pooling for Graph Representations”. In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019 (cit. on p. 132).

[520] F. Murtagh. “A Survey of Recent Advances in Hierarchical Clustering Algorithms”. In: *The Computer Journal* 26.4 (Nov. 1983), pp. 354–359. DOI: https://doi.org/10.1093/comjnl/26.4.354 (cit. on p. 222).

[521] F. Murtagh and P. Legendre. “Ward’s Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward’s Criterion?” In: *Journal of Classification* 31.3 (2014), pp. 274–295. DOI: https://doi.org/10.1007/s00357-014-9161-z (cit. on p. 216).

[522] S. Murugan, K. P. Lakshmi, J. Sundar, and K. MathiVathani. “Design and Implementation of Multilayer Perceptron with On-chip Learning in Virtex-E”. In: *AASRI Procedia* 6 (2014), pp. 82–88 (cit. on pp. 253, 262).

[523] S. Muthukrishnan. “Data Streams: Algorithms and Applications”. In: *Foundations and Trends in Theoretical Computer Science* 1.2 (2005) (cit. on pp. 71, 89).

[524] S. Naderiparizi, A. N. Parks, Z. Kapetanovic, B. Ransford, and J. R. Smith. “WISPCam : A Battery-Free RFID Camera”. In: *Procs. of the IEEE Int. Conference on RFID 2015*. 2015 (cit. on p. 53).

[525] A. Nath and E. Taylor. “*k*-Median Clustering Under Discrete Fréchet and Hausdorff Distances”. In: *Procs. of the Int. Symposium on Computational Geometry 2020*. Ed. by S. Cabello and D. Z. Chen. Vol. 164. Leibniz Int. Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 58:1–58:15 (cit. on pp. 203, 204).

[526] O. Neugebauer. *Energy Measurement made Simple on Embedded Systems*. Technical Report No. 2 for Collaborative Research Center SFB 876 - Graduate School. 2017. URL: https://sfb876.tu-dortmund.de/auto?self=%5C%24egw1pio6bk (cit. on p. 300).

[527] J. von Neumann. “First Draft of a Report on the EDVAC”. In: 1945 (cit. on p. 2).

[528] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting. “Propagation Kernels: Efficient Graph Kernels from Propagated Information”. In: *Machine Learning* 102.2 (Feb. 2016), pp. 209–245 (cit. on p. 125). **SFB876-A6**

[529] T. Neumann. “Efficiently Compiling Efficient Query Plans for Modern Hardware”. In: *Procs. of the VLDB Endowment* 4.9 (2011), pp. 539–550. URL: http://www.vldb.org/pvldb/vol4/p539-neumann.pdf (cit. on pp. 263, 385, 386, 395).

[530] J. Newling and F. Fleuret. “A Sub-Quadratic Exact Medoid Algorithm”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2017*. 2017, pp. 185–193. URL: http://proceedings.mlr.press/v54/newling17a.html (cit. on p. 184).

[531] A. Y. Ng and M. I. Jordan. “On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes”. In: *Advances in Neural Information Processing Systems 14: Procs. of the 2001 Conference*. Ed. by S. B. T. G. Dietterich and Z. Ghahramani. Vol. 14. Cambridge, MA: MIT Press., 2002, pp. 841–848 (cit. on p. 102).

[532] D. Nguyen, N. Ho, and I. Chang. “St-DRC: Stretchable DRAM Refresh Controller with No Parity-overhead Error Correction Scheme for Energy-efficient DNNs”. In: *Procs. of the Design Automation Conference 2019*. 2019, pp. 1–6 (cit. on p. 327).

[533] H. Nguyen and T. Maehara. “Graph Homomorphism Convolution”. In: *Procs. of the Int. Conference on Machine Learning 2020*. 2020, pp. 7306–7316 (cit. on p. 125).

[534] K. Ni, A. Gupta, O. Prakash, S. Thomann, X. S. Hu, and H. Amrouch. “Impact of Extrinsic Variation Sources on the Device-to-Device Variation in Ferroelectric FET”. In: *Procs. of the IEEE Int. Reliability Physics Symposium 2020*. 2020 (cit. on p. 328).

[535] K. Ni et al. “Ferroelectric ternary content-addressable memory for one-shot learning”. In: *Nature Electronics* 2.11 (2019), pp. 521–529 (cit. on p. 4).

[536] F. Nie, X. Wang, C. Deng, and H. Huang. “Learning a structured optimal bipartite graph for co-clustering”. In: *Advances in Neural Information Processing Systems 30: Procs. of the 2017 Conference*. 2017, pp. 4129–4138 (cit. on p. 232).

[537] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis. “Matching Node Embeddings for Graph Similarity”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2017*. 2017, pp. 2429–2435 (cit. on p. 125).

[538] J. Nocedal and S. J. Wright. *Numerical Optimization*. Second. Springer Series in Operations Research and Financial Engineering. Springer, 2006. URL: https://books.google.de/books?id=epc5fX0lqRIC (cit. on p. 105).

[539] N. Noorshams and M. Wainwright. “Stochastic belief propagation: Low-complexity message-passing with guarantees”. In: *Procs. of the Allerton Conference on Communication, Control, and Computing 2011*. 2011, pp. 269–276 (cit. on p. 407).

[540] A. Norouzi-Fard, J. Tarnawski, S. Mitrovic, A. Zandieh, A. Mousavifar, and O. Svensson. “Beyond 1/2-Approximation for Submodular Maximization on Massive Data Streams”. In: *Procs. of the Int. Conference on Machine Learning 2018*. July 2018, pp. 3829–3838. URL: http://proceedings.mlr.press/v80/norouzi-fard18a.html (cit. on pp. 76, 77).

[541] E. Nurvitadhi et al. “Can FPGAs beat GPUs in accelerating next-generation deep neural networks?” In: *Procs. of the ACM/SIGDA Int. Symposium on Field-Programmable Gate Arrays 2017*. ACM. 2017, pp. 5–14 (cit. on p. 253).

[542] NVIDIA Corp. *CUDA Toolkit Documentation v8.0, CUDA C Programming Guide*. 2016 (cit. on p. 408).

[543] P. E. O’Neil, E. J. O’Neil, and X. Chen. “The star schema benchmark (SSB)”. In: *Polymers for Advanced Technologies* 200.0 (2007), p. 50 (cit. on p. 381).

[544] L. Oettershagen, N. M. Kriege, C. Morris, and P. Mutzel. “Temporal Graph Kernels for Classifying Dissemination Processes”. In: *Procs. of the SIAM Int. Conference on Data Mining 2020*. 2020 (cit. on p. 128). **SFB876-A6**

[545] OmniSci Incorporated. *OmniSciDB*. `https://www.omnisci.com/`. 2019. URL: https://www.omnisci.com/platform/omniscidb (cit. on p. 270).

[546] A. Omondi and J. Rajapakse. *FPGA implementations of neural networks*. Springer, 2006 (cit. on p. 253).

[547] A. V. D. Oord, Y. Li, and O. Vinyals. *Representation Learning with Contrastive Predictive Coding*. Tech. rep. 2018. DOI: arXiv:1807.03748 (cit. on p. 175).

[548] P. Paatero and U. Tapper. “Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values”. In: *Environmetrics* 5.2 (1994), pp. 111–126 (cit. on p. 229).

[549] Y. Pan et al. “A Multilevel Cell STT-MRAM-Based Computing In-Memory Accelerator for Binary Convolutional Neural Network”. In: *IEEE Transactions on Magnetics* 54 (2018), pp. 1–5 (cit. on p. 327).

[550] C. H. Papadimitriou. “Worst-case and probabilistic analysis of a geometric location problem”. In: *SIAM Journal on Computing* 10.3 (1981), pp. 542–557 (cit. on p. 200).

[551] R. Parada and J. Melia-Segui. “Gesture detection using passive RFID tags to enable people-centric IoT applications”. In: *IEEE Communications Magazine* 55.2 (2017), pp. 56–61 (cit. on p. 61).

[552] N. Parikh and S. Boyd. “Proximal Algorithms”. In: *Foundations and Trends in Optimization* 1.3 (Jan. 2014), pp. 127–239. DOI: http://dx.doi.org/10.1561/2400000003 (cit. on p. 234).

[553] H. Park and C. Jun. “A simple and fast algorithm for K-medoids clustering”. In: *Expert Systems With Applications* 36.2 (2009), pp. 3336–3341 (cit. on p. 184).

[554] T. Park and G. Casella. “The Bayesian Lasso”. In: *Journal of the American Statistical Association* 103 (2008), pp. 681–686 (cit. on p. 95).

[555] A. Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. 2019 (cit. on p. 134).

[556] D. Patterson et al. *Carbon Emissions and Large Neural Network Training*. arXiv. 2021. DOI: 10.48550/arXiv.2104.10350 (cit. on p. 6).

[557] J. Paul, J. He, and B. He. “GPL: A GPU-Based Pipelined Query Processing Engine”. In: *Procs. of the ACM SIGMOD Int. Conference on Management of Data 2016*. 2016, pp. 1935–1950 (cit. on p. 386).

[558] K. Paul and S. Rajopadhye. “Back-propagation algorithm achieving 5 GOPs on the Virtex-E”. In: *FPGA Implementations of Neural Networks*. Springer, 2006, pp. 137–165 (cit. on pp. 253, 262).

[559] N. D. Pearce and M. P. Wand. “Penalized Splines and Reproducing Kernel Methods”. In: *The American Statistician* 60.3 (2006), pp. 233–240 (cit. on p. 105).

[560] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988 (cit. on pp. 105, 407).

[561] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 350, 351).

[562] L. Pfahler. “Some Representation Learning Tasks and the Inspection of Their Models”. PhD thesis. TU Dortmund University, 2022 (cit. on p. 160). **SFB876-A1**

[563] L. Pfahler and K. Morik. “Semantic Search in Millions of Equations”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2020*. ACM, 2020. DOI: https://dl.acm.org/doi/pdf/10.1145/3394486.3403056 (cit. on pp. 160, 162, 174, 176). **SFB876-A1**

[564] L. Pfahler, J. Schill, and K. Morik. “The Search for Equations - Learning to Identify Similarities between Mathematical Expressions”. In: *Procs. of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases 2019*. Springer, 2019. URL: https://link.springer.com/chapter/10.1007/978-3-030-46133-1_42 (cit. on pp. 162, 167, 168, 170–172, 174). **SFB876-A1**

[565] J. M. Phillips. “Coresets and sketches”. In: *Handbook of Discrete and Computational Geometry*. Chapman and Hall/CRC, 2017 (cit. on pp. 85, 87).

[566] N. Piatkowski, S. Lee, and K. Morik. “Spatio-temporal random fields: compressible representation and distributed estimation”. In: *Machine Learning* 93.1 (2013), pp. 115–139. URL: http://link.springer.com/article/10.1007%5C%2Fs10994-013-5399-7 (cit. on pp. 101, 112). **SFB876-A1**

[567] N. Piatkowski. “Exponential Families on Resource-Constrained Systems”. PhD thesis. Dortmund: TU Dortmund University, 2018. URL: https://eldorado.tu-dortmund.de/handle/2003/36877 (cit. on pp. 9, 101, 111, 112, 408).

[568] N. Piatkowski. “Hyper-Parameter-Free Generative Modelling with Deep Boltzmann Trees”. In: *Procs. of the European Conference on Machine Learning 2019*. Springer, 2019 (cit. on p. 100).

[569] N. Piatkowski. “iST-MRF: Interactive Spatio-Temporal Probabilistic Models for Sensor Networks”. In: *Procs. of the Int. Workshop at ECML PKDD 2012 on Instant Interactive Data Mining 2012*. 2012 (cit. on p. 105). **SFB876-A1**

[570] N. Piatkowski, S. Lee, and K. Morik. “Integer undirected graphical models for resource-constrained systems”. In: *Neurocomputing* 173.1 (Jan. 2016), pp. 9–23. URL: http://www.sciencedirect.com/science/article/pii/S0925231215010449 (cit. on p. 9). **SFB876-A1, SFB876-C1**

[571] N. Piatkowski, S. Lee, and K. Morik. “Spatio-Temporal Models For Sustainability”. In: *Procs. of the SustKDD Workshop within ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2012*. Ed. by M. Marwah, N. Ramakrishnan, M. Berges, and Z. Kolter. ACM, 2012. URL: http://wan.poly.edu/KDD2012/forms/workshop/SustKDD12/doc/SustKDD12_1.pdf (cit. on p. 105). **SFB876-A1, SFB876-C1**

[572] N. Piatkowski and K. Morik. “Fast Stochastic Quadrature for Approximate Maximum-Likelihood Estimation”. In: *Procs. of the Conference on Uncertainty in Artificial Intelligence 2018*. 2018 (cit. on pp. 105, 407). **SFB876-A1**

[573] N. Piatkowski and K. Morik. “Stochastic Discrete Clenshaw-Curtis Quadrature”. In: *Procs. of the Int. Conference on Machine Learning 2016*. JMLR: W&CP. JMLR.org, June 2016. URL: http://jmlr.org/proceedings/papers/v48/piatkowski16.html (cit. on pp. 105, 407). **SFB876-A1**

[574] N. Piatkowski, L. Sangkyun, and K. Morik. “The Integer Approximation of Undirected Graphical Models”. In: *Procs. of the Int. Conference on Pattern Recognition Applications and Methods 2014*. Ed. by M. De Marsico, A. Tabbone, and A. Fred. SciTePress, 2014, pp. 296–304. URL: http://www-ai.cs.uni-dortmund.de/PublicPublicationFiles/piatkowski_etal_2014a.pdf (cit. on p. 408). **SFB876-A1, SFB876-C1**

[575] N. Piatkowski and F. Schnitzler. “Compressible Reparametrization of Time-Variant Linear Dynamical Systems”. In: *Solving Large Scale Learning Tasks. Challenges and Algorithms -*

Essays Dedicated to Katharina Morik on the Occasion of Her 60th Birthday. 2016, pp. 234–250. DOI: http://dx.doi.org/10.1007/978-3-319-41706-6_12 (cit. on p. 11). **SFB876-A1**

[576] N. Piatkowski, J. Streicher, S. Olaf, and K. Morik. “Open Smartphone Data for Structured Mobility and Utilization Analysis in Ubiquitous Systems”. In: *Mining, Modeling and Recommending Things in Social Media*. Ed. by M. Atzmueller, A. Chin, C. Scholz, and C. Trattner. Vol. 8940. Lecture Notes in Computer Science. Springer, 2014. Chap. Open Smartphone Data for Structured Mobility and Utilization Analysis in Ubiquitous Systems, pp. 116–130. URL: http://link.springer.com/chapter/10.1007%5C%2F978-3-319-14723-9_7 (cit. on p. 32). **SFB876-A1**

[577] N. Piatkowski et al. “Generative Machine Learning for Resource-Aware 5G and IoT Systems”. In: *Procs. of the IEEE Int. Conference on Communications 2021*. 2021, pp. 1–6. URL: https://doi.org/10.1109/ICCWorkshops50388.2021.9473625 (cit. on p. 101).

[578] K. Pietrzak. “On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems”. In: *Journal of Computer and System Sciences* 67.4 (2003), pp. 757–771. URL: http://www.sciencedirect.com/science/article/pii/S0022000003000783 (cit. on p. 205).

[579] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. 3rd. Springer Publishing Company, Incorporated, 2008 (cit. on p. 300).

[580] G. Pio, M. Ceci, D. Malerba, and D. D’Elia. “ComiRNet: a web-based system for the analysis of miRNA-gene regulatory networks”. In: *BMC Bioinformatics* 16.S-9 (2015), S7 (cit. on p. 228).

[581] T. Pock and S. Sabach. “Inertial proximal alternating linearized minimization (iPALM) for nonconvex and nonsmooth problems”. In: *SIAM journal on imaging sciences* 9.4 (2016), pp. 1756–1787 (cit. on p. 247).

[582] J. Podani. “New combinatorial clustering methods”. In: *Numerical syntaxonomy*. Ed. by M. B. Mucina L.and Dale. Dordrecht: Springer Netherlands, 1989, pp. 61–77 (cit. on p. 219).

[583] C. N. Potts. “Analysis of a Heuristic for One Machine Sequencing with Release Dates and Delivery Times”. In: *Operations Research* 28.6 (1980), pp. 1436–1441 (cit. on pp. 365, 368).

[584] Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma. “Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising”. In: *Procs. of the World Wide Web Conference 2018*. 2018, pp. 993–1002 (cit. on p. 283).

[585] Y. Prabhu and M. Varma. “Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2014*. ACM. 2014, pp. 263–272 (cit. on p. 272).

[586] R. Prenger, B. Chen, T. Marlatt, and D. Merl. *Fast map search for compact additive tree ensembles (cate)*. Tech. rep. Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2013 (cit. on p. 340).

[587] M. Qaraei, E. Schultheis, P. Gupta, and R. Babbar. “Convex Surrogates for Unbiased Loss Functions in Extreme Classification With Missing Labels”. In: *Procs. of the World Wide Web Conference 2021*. 2021, pp. 3711–3720 (cit. on p. 284).

[588] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space”. In: *Advances in Neural Information Processing Systems 30: Procs. of the 2017 Conference*. 2017 (cit. on pp. 130, 132, 135).

[589] C. Qian, Y. Yu, and Z. H. Zhou. “Pareto ensemble pruning”. In: *Procs. of the National Conference on Artificial Intelligence* 4 (2015), pp. 2935–2941. URL: https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/aaai15prun.pdf (cit. on p. 340).

[590] P. Raghavan. “Probabilistic construction of deterministic algorithms: Approximating packing integer programs”. In: *Journal of Computer and System Sciences* 37.2 (1988), pp. 130–143 (cit. on pp. 145, 147).

[591] R. Rajkumar. “Real-time synchronization protocols for shared memory multiprocessors”. In: *Procs. of the Int. Conference on Distributed Computing Systems 1990*. 1990, pp. 116–123. DOI: http://dx.doi.org/10.1109/icdcs.1990.89257 (cit. on pp. 361, 362).

[592] R. Rajkumar, L. Sha, and J. P. Lehoczky. “Real-Time Synchronization Protocols for Multiprocessors”. In: *Procs. of the IEEE Real-Time Systems Symposium 1988*. 1988, pp. 259–269 (cit. on pp. 361, 362).

[593] Y. K. Ramadass et al. “A batteryless thermoelectric energy-harvesting interface circuit with 35mV startup voltage”. In: *IEEE J Solid-State Circuits* (2010) (cit. on p. 57).

[594] S. J. Reddi, B. Póczos, and A. J. Smola. “Communication Efficient Coresets for Empirical Loss Minimization”. In: *Procs. of the Conference on Uncertainty in Artificial Intelligence 2015*. 2015, pp. 752–761. URL: http://auai.org/uai2015/proceedings/papers/141.pdf (cit. on p. 87).

[595] A. P. Reynolds, G. Richards, B. de la Iglesia, and V. J. Rayward-Smith. “Clustering Rules: A Comparison of Partitioning and Hierarchical Clustering Algorithms”. In: *Journal of Mathematical Modelling and Algorithms* 5.4 (2006), pp. 475–504 (cit. on p. 184).

[596] J. Richter, H. Kotthaus, B. Bischl, P. Marwedel, J. Rahnenführer, and M. Lang. “Faster Model-Based Optimization through Resource-Aware Scheduling Strategies”. In: *Procs. of the Int. Conference: Learning and Intelligent Optimization 2016*. Vol. 10079. Lecture Notes in Computer Science (LNCS). Springer Int. Publishing, 2016, pp. 267–273. URL: http://link.springer.com/chapter/10.1007/978-3-319-50349-3_22 (cit. on p. 11). **SFB876-A3**

[597] B. Rieck, C. Bock, and K. M. Borgwardt. “A Persistent Weisfeiler-Lehman Procedure for Graph Classification”. In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019, pp. 5448–5458 (cit. on p. 125).

[598] Robotnik. *Mobile Robot RB-1 Base*. 2021. URL: https://www.robotnik.eu/mobile-robots/rb-1-base-2/ (cit. on p. 374).

[599] M. Roidl et al. “Performance Availability Evaluation of Smart Devices in Materials Handling Systems”. In: *Procs. of the IEEE ICCC Workshops on Internet of Things 2014*. Shanghai, China, Oct. 2014 (cit. on p. 35).

[600] Y. Rong, W. Huang, T. Xu, and J. Huang. “DropEdge: Towards Deep Graph Convolutional Networks on Node Classification”. In: *Procs. of the Int. Conference on Learning Representation 2020*. 2020 (cit. on pp. 130, 136).

[601] ROS. *Robot Operating System (ROS)*. 2021. URL: https://www.ros.org/ (cit. on p. 374).

[602] L. Rosasco, E. De, V. A. Caponnetto, M. Piana, and A. Verri. “Are loss functions all the same”. In: *Neural Computation* 15 (2004), p. 2004 (cit. on p. 332).

[603] K. E. Rosing, E. L. Hillsman, and H. Rosing-Vogelaar. “A Note Comparing Optimal and Heuristic Solutions To the p-Median Problem”. In: *Geographical Analysis* 11.1 (1979), pp. 86–89 (cit. on p. 184).

[604] R. A. Rossi and N. K. Ahmed. “The Network Data Repository with Interactive Graph Analytics and Visualization”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2015*. 2015. URL: http://networkrepository.com (cit. on p. 154).

[605] O. Roustant, D. Ginsbourger, and Y. Deville. “DiceKriging, DiceOptim: Two R packages for the Analysis of Computer Experiments by Kriging-based Metamodeling and Optimization”. In: *Journal of Statistical Software* 51.1 (2012), pp. 1–55 (cit. on p. 294).

[606] A. Sailer, S. Schmidhuber, M. Deubzer, M. Alfranseder, M. Mucha, and J. Mottok. “Optimizing the task allocation step for multi-core processors within AUTOSAR”. In: *Procs. of the Int. Conference on Applied Electronics 2013*. Sept. 2013, pp. 1–6 (cit. on p. 368).

[607] Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio. “ROSCH:Real-Time Scheduling Framework for ROS”. In: *Procs. of the IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications 2018*. Aug. 2018, pp. 52–58 (cit. on p. 374).

[608] F. Saki, A. Sehgal, I. M. S. Panahi, and N. Kehtarnavaz. “Smartphone-based real-time classification of noise signals using subband features and random forest classifier”. In: *Procs. of the Int. Conference on Acoustics, Speech and Signal Processing 2016*. 2016, pp. 2204–2208 (cit. on p. 339).

[609] E. Sari, M. Belbahri, and V. P. Nia. “How Does Batch Normalization Help Binary Training?” In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1909.09139 (cit. on p. 329).

[610] R. E. Schapire and Y. Freund. “Boosting: Foundations and algorithms”. In: *Kybernetes* (2012) (cit. on p. 341).

[611] L. Schönberger, W.-H. Huang, G. v. d. Brüggen, K.-H. Chen, and J.-J. Chen. “Schedulability Analysis and Priority Assignment for Segmented Self-Suspending Tasks”. In: *Procs. of the IEEE Int. Conference on Embedded and Real-Time Computing Systems and Applications 2018*. Hakodate, Japan, Aug. 2018 (cit. on p. 363). **SFB876-B2**

[612] L. Schönberger et al. “Offloading Safety- and Mission-Critical Tasks via Unreliable Connections”. In: *Procs. of the Euromicro Conference on Real-Time Systems 2020*. 2020 (cit. on p. 373). **SFB876-A1, SFB876-A3, SFB876-A4, SFB876-B4**

[613] E. Schubert. “HACAM: Hierarchical Agglomerative Clustering Around Medoids – and its Limitations”. In: *Procs. of the Lernen.Wissen.Daten.Analysen Workshops: FGWM, KDML, FGWI-BIA, and FGIR 2021*. Ed. by T. Seidl, M. Fromm, and S. Obermeier. Vol. 2993. CEUR Workshop Proceedings. CEUR-WS.org, 2021, pp. 191–204. URL: http://ceur-ws.org/Vol-2993/paper-19.pdf (cit. on pp. 216, 219).

[614] E. Schubert and M. Gertz. “Numerically Stable Parallel Computation of (Co-)Variance”. In: *Procs. of the Int. Conference on Scientific and Statistical Database Management 2018*. SSDBM 2018 best paper award. 2018, 10:1–10:12. DOI: https://doi.org/10.1145/3221269.3223036 (cit. on p. 220).

[615] E. Schubert, H.-P. Kriegel, and A. Zimek. “The (black) art of runtime evaluation: Are we comparing algorithms or implementations?” In: *Knowledge and Information Systems* 52.2 (2017), pp. 341–3778 (cit. on p. 339).

[616] E. Schubert and P. J. Rousseeuw. “Fast and Eager k-Medoids Clustering: O(k) Runtime Improvement of the PAM, CLARA, and CLARANS Algorithms”. In: *Information Systems* 101 (2021), p. 101804. DOI: https://doi.org/10.1016/j.is.2021.101804 (cit. on pp. 184, 188, 189, 191). **SFB876-A2**

[617] E. Schubert and P. J. Rousseeuw. “Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms”. In: *Procs. of the Int. Conference on Similarity Search and Applications 2019*. 2019, pp. 171–187. DOI: https://doi.org/10.1007/978-3-030-32047-8_16 (cit. on pp. 184, 188, 189).

[618] E. Schubert and A. Zimek. “ELKI: A large open-source library for data analysis - ELKI Release 0.7.5 "Heidelberg"”. In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1902.03616 (cit. on pp. 192, 223).

[619] M. Seeger. *Greedy forward selection in the informative vector machine*. Tech. rep. Technical report, University of California at Berkeley, 2004 (cit. on p. 81).

[620] T. K. Sellis. “Multiple-Query Optimization”. In: *ACM Transactions on Database Systems* 13.1 (1988), pp. 23–52 (cit. on p. 31).

[621] G. Sen, G. Namata, M. Bilgic, and L. Getoor. “Collective Classification in Network Data”. In: *AI Magazine* 29 (2008) (cit. on p. 136).

[622] S. Sengupta, M. Harris, and M. Garland. *Efficient Parallel Scan Algorithms for GPUs*. Tech. Rep. NVR-2008-003. NVIDIA, Santa Clara, CA, 2008 (cit. on pp. 393, 394).

[623] L. Sha, R. Rajkumar, and J. P. Lehoczky. “Priority Inheritance Protocols: An Approach to Real-Time Synchronization”. In: *IEEE Transactions on Computers* 39.9 (1990), pp. 1175–1185. DOI: http://dx.doi.org/10.1109/12.57058 (cit. on p. 361).

[624] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Procs. of the IEEE* 104.1 (Jan. 2016), pp. 148–175 (cit. on p. 95).

[625] S. Shao and W. Luk. "Customised pearlmutter propagation: A hardware architecture for trust region policy optimisation". In: *Procs. of the Int. Conference on Field Programmable Logic and Applications 2017*. IEEE. 2017, pp. 1–6 (cit. on p. 254).

[626] P. Sharma and P. Kulkarni. "Singleton: System-wide Page Deduplication in Virtual Environments". In: *Procs. of the Int. Symposium on High-Performance Parallel and Distributed Computing 2012*. HPDC '12. ACM, 2012, pp. 15–26 (cit. on p. 307).

[627] N. Shervashidze, P. Schweitzer, E. van Leeuwen, K. Mehlhorn, and K. Borgwardt. "Weisfeiler–Lehman Graph Kernels". In: *Journal of Machine Learning Research* 12 (2011), pp. 2539–2561 (cit. on pp. 117, 120, 121, 124).

[628] A. Siddiqa, A. Karim, and A. Gani. "Big Data storage technologies: A survey". In: *Frontiers of Information Technology & Electronic Engineering* 18.8 (2017), pp. 1040–1070 (cit. on p. 89).

[629] L. Sigrist et al. "Environment and Application Testbed for Low-Power Energy Harvesting System Design". In: *IEEE Transactions on Industrial Electronics* (2020) (cit. on p. 59).

[630] L. Sigrist, R. Ahmed, A. Gomez, and L. Thiele. "Harvesting-aware optimal communication scheme for infrastructure-less sensing". In: *ACM Transactions on Internet of Things* (2020) (cit. on pp. 54, 57, 59).

[631] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele. "Measurement and Validation of Energy Harvesting IoT Devices". In: *Procs. of the Design, Automation and Test in Europe Conference 2017*. 2017, pp. 1159–1164 (cit. on pp. 48, 50, 58).

[632] L. Sigrist, A. Gomez, and L. Thiele. *Long-Term Tracing of Indoor Solar Harvesting*. Aug. 2019. DOI: https://doi.org/10.5281/zenodo.3363925 (cit. on pp. 47, 49, 51).

[633] M. Simonovsky and N. Komodakis. "Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs". In: *Procs. of the Conference on Computer Vision and Pattern Recognition 2017*. 2017 (cit. on pp. 116, 135).

[634] P. H. A. Sneath. "The Application of Computers to Taxonomy". In: *Journal of General Microbiology* 17.1 (Aug. 1957), pp. 201–226 (cit. on pp. 216, 217).

[635] J. Snoek, H. Larochelle, and R. P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems 25: Procs. of the 2012 Conference*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 2951–2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf (cit. on p. 288).

[636] C. Sohler and D. P. Woodruff. "Strong Coresets for *k*-Median and Subspace Approximation: Goodbye Dimension". In: *Procs. of the IEEE Symposium on Foundations of Computer Science 2018*. Ed. by M. Thorup. IEEE Computer Society, 2018, pp. 802–813. DOI: https://doi.org/10.1109/FOCS.2018.00081 (cit. on p. 212). **SFB876-A2**

[637] C. Sohler and D. P. Woodruff. "Subspace embeddings for the L_1-norm with applications". In: *Procs. of the ACM Symposium on Theory of Computing 2011*. Ed. by L. Fortnow and S. P. Vadhan. ACM, 2011, pp. 755–764 (cit. on pp. 87, 90, 95). **SFB876-C4**

[638] R. Sokal and P. Sneath. *Principles of Numerical Taxonomy*. Books in biology. W. H. Freeman, 1963 (cit. on pp. 216, 217).

[639] J. Sompolski, M. Zukowski, and P. A. Boncz. "Vectorization vs. Compilation in Query Execution". In: *Procs. of the Int. Workshop on Data Management on New Hardware 2011*. Athens, Greece, June 2011, pp. 33–40 (cit. on p. 386).

[640] K. Song, X. Yao, F. Nie, X. Li, and M. Xu. "Weighted Bilateral K-means Algorithm for Fast Co-clustering and Fast Spectral Clustering". In: *Pattern Recognition* (2020), p. 107560 (cit. on p. 232).

[641] S. Song, D. Yan, and Y. Xie. “Design of control system based on hand gesture recognition”. In: *Procs. of the IEEE Int. Conference on Networking, Sensing and Control 2018*. IEEE. 2018, pp. 1–4 (cit. on p. 61).

[642] D. Sontag and T. Jaakkola. “Tree Block Coordinate Descent for MAP in Graphical Models”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2009*. Vol. 8. 2009, pp. 544–551 (cit. on p. 407).

[643] J. Spencer. *Ten Lectures on the Probabilistic Method*. 2nd Edition. Society for Industrial and Applied Mathematics, 1994 (cit. on pp. 145, 147).

[644] O. Spinczyk, A. Gal, and W. Schröder-Preikschat. “AspectC++: An Aspect-Oriented Extension to C++”. In: *Procs. of the Int. Conference on Technology of Object-Oriented Languages and Systems 2002*. Sydney, Australia, Feb. 2002, pp. 53–60 (cit. on p. 40).

[645] O. Spinczyk and D. Lohmann. “The Design and Implementation of AspectC++”. In: *Knowledge-Based Systems, Special Issue on Techniques to Produce Intelligent Secure Software* 20.7 (2007), pp. 636–651 (cit. on p. 40).

[646] B. K. Stöcker, T. Schäfer, P. Mutzel, J. Köster, N. M. Kriege, and S. Rahmann. “Protein Complex Similarity Based on Weisfeiler-Lehman Labeling”. In: *Procs. of the Int. Conference on Similarity Search and Applications 2019*. Ed. by G. Amato, C. Gennaro, V. Oria, and M. Radovanovic. Cham: Springer Int. Publishing, 2019, pp. 308–322 (cit. on pp. 116, 117). **SFB876-A6, SFB876-C1**

[647] J. Stokes et al. “A Deep Learning Approach to Antibiotic Discovery”. In: *Cell* 180 (Feb. 2020), 688–702.e13 (cit. on p. 116).

[648] M. Stolpe, K. Bhaduri, K. Das, and K. Morik. “Anomaly Detection in Vertically Partitioned Data by Distributed Core Vector Machines”. In: *Procs. of the European Conference on Machine Learning and Knowledge Discovery in Databases 2013*. Ed. by H. Blockeel, K. Kersting, S. Nijssen, and F. Železný. Springer, 2013, pp. 321–336 (cit. on p. 89). **SFB876-B3**

[649] J. Streicher. *Data Modeling of Ubiquitous System Software*. Tech. rep. 7. TU Dortmund University, Aug. 2014 (cit. on p. 32). **SFB876-A1**

[650] E. Strubell, A. Ganesh, and A. McCallum. “Energy and Policy Considerations for Modern Deep Learning Research”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2020, The Innovative Applications of Artificial Intelligence Conference 2020, The AAAI Symposium on Educational Advances in Artificial Intelligence 2020*. AAAI Press, 2020, pp. 13693–13696. URL: https://aaai.org/ojs/index.php/AAAI/article/view/7123 (cit. on p. 6).

[651] X. Sun, X. Peng, P. Chen, R. Liu, J. Seo, and S. Yu. “Binary neural network with 16 Mb RRAM macro chip for classification and online training”. In: *Procs. of the Asia and South Pacific Design Automation Conference 2018*. 2018, pp. 574–579 (cit. on p. 327).

[652] X. Sun et al. “Low-VDD Operation of SRAM Synaptic Array for Implementing Ternary Neural Network”. In: *Procs. of IEEE Transactions on Very Large Scale Integration Systems 2017* 25.10 (2017), pp. 2962–2965 (cit. on p. 326).

[653] Y. Sun, Y. Zhang, and G. Xue. “SmartWheelTag: Flexible and Battery-less User Interface for Drivers”. In: *Procs. of the IEEE Vehicular Networking Conference 2018*. IEEE. 2018, pp. 1–2 (cit. on p. 61).

[654] C. Sutton and A. McCallum. “An Introduction to Conditional Random Fields for Relational Learning”. In: *Introduction to Statistical Relational Learning*. Ed. by L. Getoor and B. Taskar. MIT Press, 2007. URL: http://www.cs.umass.edu/~mccallum/papers/crf-tutorial.pdf (cit. on p. 106).

[655] C. Sutton and A. McCallum. “An Introduction to Conditional Random Fields”. In: *Foundations and Trends in Machine Learning* 4.4 (2012), pp. 267–373 (cit. on pp. 410, 416).

[656] Synopsys, Inc. *https://www.synopsys.com/silicon/tcad.html*. 2022 (cit. on p. 328).

[657] B. Tanasa, U. D. Bordoloi, P. Eles, and Z. Peng. “Probabilistic Response Time and Joint Analysis of Periodic Tasks”. In: *Procs. of the Euromicro Conference on Real-Time Systems 2015*. 2015, pp. 235–246 (cit. on p. 363).

[658] M. B. Teitz and P. Bart. “Heuristic Methods for Estimating the Generalized Vertex Median of a Weighted Graph”. In: *Operations Research* 16.5 (1968), pp. 955–961 (cit. on p. 184).

[659] M. Thorup. “Quick *k*-Median, *k*-Center, and Facility Location for Sparse Graphs”. In: *SIAM Journal on Computing* 34.2 (2005), pp. 405–432 (cit. on p. 201).

[660] R. Tibshirani. “Regression Shrinkage and Selection via the Lasso”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 58.1 (1996), pp. 267–288 (cit. on p. 105).

[661] R. M. Timoney. *OpenMath LaTeX to OpenMath Converter*. Tech. rep. 1999, pp. 1–9. URL: https://www.maths.tcd.ie/%7B~%7Drichardt/openmath/ml2om.pdf (cit. on p. 162).

[662] S. Tobuschat, R. Ernst, A. Hamann, and D. Ziegenbein. “System-level timing feasibility test for cyber-physical automotive systems”. In: *Procs. of the IEEE Symposium on Industrial Embedded Systems 2016*. May 2016, pp. 1–10 (cit. on p. 368).

[663] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. M. Borgwardt. “Wasserstein Weisfeiler-Lehman Graph Kernels”. In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. 2019, pp. 6436–6446 (cit. on p. 123).

[664] E. Tolochinsky and D. Feldman. “Coresets For Monotonic Functions with Applications to Deep Learning”. In: *arXiv: Computing Research Repository* (2018). DOI: arXiv:1802.07382 (cit. on p. 87).

[665] A. Tozawa, M. Tatsubori, T. Onodera, and Y. Minamide. “Copy-on-write in the PHP Language”. In: *Procs. of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages 2009*. POPL ’09. ACM, 2009, pp. 200–212 (cit. on p. 308).

[666] P. Tözün and H. Kotthaus. “Scheduling Data-Intensive Tasks on Heterogeneous Many Cores”. In: *IEEE Data Engineering Bulletin* 42.1 (2019), pp. 61–72. URL: http://sites.computer.org/debull/A19mar/p61.pdf (cit. on pp. 285, 287, 289). **SFB876-A3, SFB876-C5**

[667] D.-S. Tran, N.-H. Ho, H.-J. Yang, E.-T. Baek, S.-H. Kim, and G. Lee. “Real-time hand gesture spotting and recognition using RGB-D camera and 3D convolutional neural network”. In: *Applied Sciences* 10.2 (2020), p. 722 (cit. on p. 61).

[668] M. Trentzsch et al. “A 28nm HKMG super low power embedded NVM technology based on ferroelectric FETs”. In: *Procs. of the IEEE Int. Electron Devices Meeting 2016*. IEEE. 2016, pp. 11–5 (cit. on p. 328).

[669] T. H. Trinh, M.-t. Luong, Q. V. Le, and G. Brain. “Selfie : Self-supervised Pretraining for Image Embedding”. In: (2019) (cit. on p. 163).

[670] H. Truong et al. “Capband: Battery-free successive capacitance sensing wristband for hand gesture recognition”. In: *Procs. of the ACM Conference on Embedded Networked Sensor Systems 2018*. 2018, pp. 54–67 (cit. on p. 61).

[671] S. Tschiatschek, P. Reinprecht, M. Mücke, and F. Pernkopf. “Bayesian Network Classifiers with Reduced Precision Parameters”. In: *Procs. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2012*. 2012 (cit. on p. 407).

[672] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei. “RANA: Towards Efficient Neural Acceleration with Refresh-Optimized Embedded DRAM”. In: *Procs. of the Int. Symposium on Computer Architecture 2018*. 2018, pp. 340–352 (cit. on p. 327).

[673] A. Turing. “Computing Machinery and Intelligence”. In: *Mind*. New Series 59.236 (1950), pp. 433–460. URL: http://www.jstor.org/stable/2251299 (cit. on p. 5).

[674] H. Turner, T. Bailey, and W. Krzanowski. “Improved biclustering of microarray data demonstrated through systematic performance tests”. In: *Computational Statistics & Data Analysis* 48.2 (2005), pp. 235–254 (cit. on p. 233).

[675] M. Tzoufras, M. Gajek, and A. Walker. “Magnetoresistive RAM for error resilient XNOR-Nets”. In: *arXiv: Computing Research Repository* (2019). DOI: arXiv:1905.10927 (cit. on p. 327).

[676] E. Ustinova and V. Lempitsky. “Learning Deep Embeddings with Histogram Loss”. In: *Advances in Neural Information Processing Systems 29: Procs. of the 2016 Conference*. 2016 (cit. on p. 168).

[677] O. Vaidya, S. Gandhe, A. Sharma, A. Bhate, V. Bhosale, and R. Mahale. “Design and Development of Hand Gesture based Communication Device for Deaf and Mute People”. In: *Procs. of the IEEE Bombay Section Signature Conference 2020*. IEEE. 2020, pp. 102–106 (cit. on p. 61).

[678] S. Valat, M. Pérache, and W. Jalby. “Introducing Kernel-level Page Reuse for High Performance Computing”. In: *Procs. of the ACM SIGPLAN Workshop on Memory Systems Performance and Correctness 2013*. MSPC ’13. ACM, 2013, 3:1–3:9 (cit. on p. 307).

[679] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger. “Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA?” In: *Procs. of the IEEE Int. Symposium on Field-Programmable Custom Computing Machines 2012*. IEEE. 2012, pp. 232–239 (cit. on p. 340).

[680] V. N. Vapnik. *The Nature of Statistical Learning Theory*. New York: Springer, 1995 (cit. on p. 10).

[681] A. Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30: Procs. of the 2017 Conference*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (cit. on pp. 163, 165, 166).

[682] S. A. Vavasis. “On the complexity of nonnegative matrix factorization”. In: *SIAM journal on optimization* 20.3 (2009), pp. 1364–1377 (cit. on p. 229).

[683] P. Veličković, G. Cucurull, A. Casanova, A. Romero, and Y. B. Liò. “Graph Attention Networks”. In: *Procs. of the Int. Conference on Learning Representation 2018*. 2018 (cit. on pp. 130, 132, 136).

[684] S. I. Venieris and C.-S. Bouganis. “fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs”. In: *Procs. of the Int. Symposium on Field-Programmable Custom Computing Machines 2016*. 2016 (cit. on p. 254).

[685] J.-P. Vert. “The optimal assignment kernel is not positive definite”. In: *arXiv: Computing Research Repository* (2008). URL: arXiv:0801.4061 (cit. on p. 121).

[686] A. F. Vincent et al. “Spin-Transfer Torque Magnetic Memory as a Stochastic Memristive Synapse for Neuromorphic Systems”. In: *Transactions on Biomedical Circuits and Systems* 9 (2015), pp. 166–174 (cit. on p. 327).

[687] O. Vinyals, S. Bengio, and M. Kudlur. “Order Matters: Sequence to Sequence for Sets”. In: *Procs. of the Int. Conference on Learning Representation 2016*. 2016 (cit. on p. 135).

[688] J. S. Vitter. “Random sampling with a reservoir”. In: *ACM Transactions on Mathematical Software (TOMS)* 11.1 (1985), pp. 37–57 (cit. on p. 76).

[689] M. Wahib and N. Maruyama. “Scalable Kernel Fusion for Memory-Bound GPU Applications”. In: *Procs. of the Int. Conference for High Performance Computing, Networking, Storage and Analysis 2014*. IEEE Press, 2014, pp. 191–202 (cit. on p. 388).

[690] M. Wainwright, T. Jaakkola, and A. Willsky. “A new class of upper bounds on the log partition function”. In: *IEEE Transactions on Information Theory* 51.7 (2005), pp. 2313–2335 (cit. on p. 105).

[691] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. “Tree-reweighted belief propagation algorithms and approximate ML estimation by pseudo-moment matching”. In: *Procs. of the Int. Conference on Artificial Intelligence and Statistics 2003*. 2003 (cit. on p. 407).

[692] M. J. Wainwright and M. I. Jordan. “Graphical Models, Exponential Families, and Variational Inference”. In: *Foundations and Trends in Machine Learning* 1.1–2 (2008), pp. 1–305. DOI: http://dx.doi.org/10.1561/2200000001 (cit. on pp. 103, 105, 110, 408, 411).

[693] A. X. Wang, C. Tran, N. Desai, D. Lobell, and S. Ermon. “Deep Transfer Learning for Crop Yield Prediction with Remote Sensing Data”. In: *Procs. of the ACM SIGCAS Conference on Computing and Sustainable Societies 2018*. COMPASS ’18. New York, NY, USA: ACM, 2018, 50:1–50:5. DOI: http://doi.acm.org/10.1145/3209811.3212707 (cit. on p. 162).

[694] H. Wang, F. Nie, H. Huang, and F. Makedon. “Fast nonnegative matrix tri-factorization for large-scale data co-clustering”. In: *Procs. of the Int. Joint Conference on Artificial Intelligence 2011*. 2011, p. 1553 (cit. on p. 232).

[695] J. Wang, T. Jebara, and S.-F. Chang. “Semi-supervised learning using greedy max-cut”. In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 771–800 (cit. on p. 145).

[696] M. Wang et al. “Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs”. In: *Int. Conference on Learning Representations Workshop on Representation Learning on Graphs and Manifolds* (2019). URL: https://arxiv.org/abs/1909.01315 (cit. on p. 126).

[697] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. “Dynamic Graph CNN for Learning on Point Clouds”. In: *ACM Transactions on Graphics (TOG)* (2019) (cit. on pp. 130, 132, 135).

[698] Y. Wang et al. “Storage-less and Converter-less Photovoltaic Energy Harvesting with Maximum Power Point Tracking for Internet of Things”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* PP.99 (2015) (cit. on p. 53).

[699] S. Weber, A. Gelman, D. Lee, M. Betancourt, A. Vehtari, and A. Racine-Poon. “Bayesian aggregation of average data: An application in drug development”. In: *Annals of Applied Statistics* (2018) (cit. on p. 94).

[700] B. Weisfeiler. *On Construction and Identification of Graphs*. Lecture Notes in Mathematics, Vol. 558. Springer, 1976 (cit. on p. 118).

[701] B. Weisfeiler and A. A. Lehman. “A Reduction of a Graph to a Canonical Form and an Algebra Arising During this Reduction”. In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968) (cit. on pp. 132, 137).

[702] G. Werner-Allen, P. Swieskowski, and M. Welsh. “MoteLab: A Wireless Sensor Network Testbed”. In: *Procs. of the Int. Symposium on Information Processing in Sensor Networks 2005*. IPSN ’05. IEEE Press, 2005. URL: http://dl.acm.org/citation.cfm?id=1147685.1147769 (cit. on p. 35).

[703] J. Whittaker, S. Garside, and K. Lindveld. “Tracking and predicting a network traffic process”. In: *Int. Journal of Forecasting* 13.1 (Mar. 1997), pp. 51–61. URL: http://www.sciencedirect.com/science/article/B6V92-3SWXN18-6/2/97477ccea59869520e0f7a0fcd1c19bc (cit. on p. 101).

[704] A. Wieder and B. B. Brandenburg. “Efficient partitioning of sporadic real-time tasks with shared resources and spin locks”. In: *Procs. of the Int. Symposium on Industrial Embedded Systems 2013*. 2013, pp. 49–58. DOI: http://dx.doi.org/10.1109/SIES.2013.6601470 (cit. on p. 368).

[705] B. Williams and L. Hoel. “Modeling and Forecasting Vehicular Traffic Flow as a Seasonal ARIMA Process: Theoretical Basis and Empirical Results.” In: *Journal of Transportation Engineering* 129.6 (2003), pp. 664–672 (cit. on p. 101).

[706] R. Winkelmann. *Econometric Analysis of Count Data*. 5th ed. Springer, 2008 (cit. on pp. 92, 93).

[707] D. Wishart. “256. Note: An Algorithm for Hierarchical Classifications”. In: *Biometrics* 25.1 (1969), pp. 165–170. URL: http://www.jstor.org/stable/2528688 (cit. on pp. 216, 218).

[708] D. P. Woodruff. “Sketching as a Tool for Numerical Linear Algebra”. In: *Foundations and Trends in Theoretical Computer Science* 10.1-2 (2014), pp. 1–157 (cit. on p. 89).

[709] D. P. Woodruff and Q. Zhang. “Subspace Embeddings and ℓ_p-Regression Using Exponential Random Variables”. In: *Procs. of the Conference on Learning Theory 2013*. 2013, pp. 546–567 (cit. on pp. 87, 89, 90, 95).

[710] F. Wu, T. Zhang, A. H. de Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger. “Simplifying Graph Convolutional Networks”. In: *Procs. of the Int. Conference on Machine Learning 2019*. 2019 (cit. on pp. 130, 136, 137).

[711] H. Wu, G. Diamos, J. Wang, S. Cadambi, S. Yalamanchili, and S. Chakradhar. “Optimizing Data Warehousing Applications for GPUs Using Kernel Fusion/Fission”. In: *Procs. of the Parallel and Distributed Processing Symposium Workshops and PhD Forum 2012*. IEEE, 2012, pp. 2433–2442 (cit. on p. 384).

[712] J. Xie and J. Yang. “A survey of join processing in data streams”. In: *Data Streams: Models and Algorithms*. Ed. by C. C. Aggarwal. Springer, Jan. 2007, pp. 209–236 (cit. on p. 20).

[713] J. Xu et al. “Stacked sparse autoencoder (SSAE) for nuclei detection on breast cancer histopathology images”. In: *IEEE transactions on medical imaging* 35.1 (2016), pp. 119–130 (cit. on p. 260).

[714] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. “How Powerful are Graph Neural Networks?” In: *Procs. of the Int. Conference on Learning Representation 2019*. 2019 (cit. on pp. 117, 127, 130, 132, 133, 137, 139).

[715] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. “Representation Learning on Graphs with Jumping Knowledge Networks”. In: *Procs. of the Int. Conference on Machine Learning 2018*. 2018 (cit. on p. 130).

[716] J. Yang, H. Wang, W. Wang, and P. Yu. “An Improved Biclustering Method for Analyzing Gene Expression Profiles”. In: *Int. Journal on Artificial Intelligence Tools* 14 (Oct. 2005), pp. 771–790 (cit. on p. 239).

[717] L. Yang, D. Bankman, B. Moons, M. Verhelst, and B. Murmann. “Bit Error Tolerance of a CIFAR-10 Binarized Convolutional Neural Network Processor”. In: *Procs. of the Int. Symposium on Computer Architecture 2018*. 2018, pp. 1–5 (cit. on p. 326).

[718] Z. Yang, W. Cohen, and R. Salakhutdinov. “Revisiting Semi-supervised Learning with Graph Embeddings”. In: *Procs. of the Int. Conference on Machine Learning 2016*. 2016 (cit. on p. 136).

[719] W. Yao, A. S. Bandeira, and S. Villar. “Experimental performance of graph neural networks on random instances of max-cut”. In: *Procs. of the SPIE Optical Engineering + Application Conference, Wavelets and Sparsity XVIII, 2019*. Ed. by D. V. D. Ville, M. Papadakis, and Y. M. Lu. Vol. 11138. Int. Society for Optics and Photonics. SPIE, 2019, pp. 242–251 (cit. on p. 145).

[720] M. Yayla et al. “FeFET-based Binarized Neural Networks Under Temperature-dependent Bit Errors”. In: *IEEE Transactions on Computers* (2021). DOI: 10.1109/TC.2021.3104736. URL: https://ieeexplore.ieee.org/document/9513530 (cit. on p. 328). **SFB876-A1**

[721] T. Ye, H. Zhou, W. Y. Zou, B. Gao, and R. Zhang. “RapidScorer: Fast tree ensemble evaluation by maximizing compactness in data level parallelization”. In: *Procs. of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2018*. 2018 (cit. on p. 340).

[722] Y. Ye, K. A. Ross, and N. Vesdapunt. “Scalable Aggregation on Multicore Processors”. In: *Procs. of the Int. Workshop on Data Management on New Hardware 2011*. 2011, pp. 1–11 (cit. on p. 394).

[723] Z. Yin and R. Collins. “Belief Propagation in a 3D Spatio-temporal MRF for Moving Object Detection”. In: *IEEE Computer Vision and Pattern Recognition (CVPR)* (June 2007) (cit. on p. 102).

[724] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. “Hierarchical Graph Representation Learning with Differentiable Pooling”. In: *Advances in Neural Information Processing Systems 31: Procs. of the 2018 Conference*. 2018 (cit. on pp. 126, 135). **SFB876-A6**

[725] J. Yoo and S. Choi. “Orthogonal nonnegative matrix tri-factorization for co-clustering: multiplicative updates on Stiefel manifolds”. In: *Information processing & management* 46.5 (2010), pp. 559–570 (cit. on pp. 232, 239).

[726] L. Yu, J. Shen, J. Li, and A. Lerer. “Scalable Graph Neural Networks for Heterogeneous Graphs”. In: *arXiv: Computing Research Repository* (2020). DOI: arXiv : 2011 . 09679 (cit. on pp. 130, 137).

[727] S. Yu et al. “Binary neural network with 16 Mb RRAM macro chip for classification and online training”. In: *Procs. of the Int. Electron Devices Meeting 2016*. 2016, pp. 16.2.1–16.2.4 (cit. on p. 327).

[728] Y. Yuan, R. Lee, and X. Zhang. “The Yin and Yang of processing data warehousing queries on GPU devices”. In: *Procs. of the VLDB Endowment* 6.10 (2013), pp. 817–828 (cit. on pp. 384, 401).

[729] R. Zanibbi, G. Topi, M. Kohlhase, and K. Davila. “NTCIR-12 MathIR Task Overview”. In: *Procs. of the NTCIR Conference on Evaluation of Information Access Technologies 2016*. Tokyo, Japan, 2016 (cit. on pp. 162, 172).

[730] F. Zeidler. *Beitrag zur Selbststeuerung cyberphysischer Produktionssysteme in der auftragsbezogenen Fertigung*. Praxiswissen Service, 2019 (cit. on p. 45).

[731] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. “GraphSAINT: Graph Sampling Based Inductive Learning Method”. In: *Procs. of the Int. Conference on Learning Representation 2020*. 2020 (cit. on pp. 136, 141).

[732] H. Zeng et al. “Deep Graph Neural Networks with Shallow Subgraph Samplers”. In: *arXiv: Computing Research Repository* (2020). DOI: arXiv:2012.01.380 (cit. on p. 137).

[733] H. Zha, X. He, C. Ding, H. Simon, and M. Gu. “Bipartite graph partitioning and data clustering”. In: *Procs. of the Int. Conference on Information and Knowledge Management 2001*. ACM. 2001, pp. 25–32 (cit. on p. 232).

[734] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. “Optimizing FPGA-based accelerator design for deep convolutional neural networks”. In: *Procs. of the ACM/SIGDA Int. Symposium on Field-Programmable Gate Arrays 2015*. ACM. 2015, pp. 161–170 (cit. on p. 254).

[735] K. Zhang et al. “Hetero-DB: Next Generation High-Performance Database Systems by Best Utilizing Heterogeneous Computing and Storage Resources”. In: *Journal of Computer Science and Technology* 30.4 (2015), pp. 657–678 (cit. on p. 384).

[736] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. “An End-to-End Deep Learning Architecture for Graph Classification”. In: *Procs. of the AAAI Conference on Artificial Intelligence 2018*. 2018 (cit. on pp. 126, 135).

[737] T. Zhang, R. Ramakrishnan, and M. Livny. “Birch: A new data clustering algorithm and its applications”. In: *Data Mining and Knowledge Discovery* 1.2 (June 1997), pp. 141–182. URL: http://www.ece.nwu.edu/~harsha/Clustering/newkbspaper.ps (cit. on pp. 215, 219, 220).

[738] T. Zhang, R. Ramakrishnan, and M. Livny. “BIRCH: An Efficient Data Clustering Method for Very Large Databases”. In: *Procs. of the ACM SIGMOD Int. Conference on Management of Data 1996*. 1996, pp. 103–114. URL: http://www.ece.nwu.edu/~harsha/Clustering/sigmodpaper.ps (cit. on pp. 215, 219, 220).

[739] Y. Zhang, S. Burer, and W. N. Street. “Ensemble pruning via semi-definite programming”. In: *Journal of Machine Learning Research* 7 (2006), pp. 1315–1338. URL: http://www.jmlr.org/papers/volume7/zhang06a/zhang06a.pdf (cit. on p. 340).

[740] Z.-Y. Zhang, T. Li, C. Ding, X.-W. Ren, and X.-S. Zhang. “Binary matrix factorization for analyzing gene expression data”. In: *Data Mining and Knowledge Discovery* 20.1 (2010), pp. 28–52 (cit. on p. 232).

[741] Z.-Y. Zhang, Y. Wang, and Y.-Y. Ahn. “Overlapping community detection in complex networks using symmetric binary matrix factorization”. In: *Physical Review E* 87.6 (2013), p. 062803 (cit. on p. 232).

[742] Z. Zhang, C. Ding, T. Li, and X. Zhang. “Binary matrix factorization with applications”. In: *Procs. of the IEEE Int. Conference on Data Mining 2007*. IEEE. 2007, pp. 391–400 (cit. on p. 232).

[743] F. Zhao and N. Park. “Using Geographically Weighted Regression Models to Estimate Annual Average Daily Traffic”. In: *Journal of the Transportation Research Board* 1879.12 (2004), pp. 99–107 (cit. on p. 102).

[744] W. Zhao et al. “An FPGA-based Framework for Training Convolutional Neural Networks”. In: *Procs. of the Int. Conference on Application-specific Systems, Architectures and Processors 2016*. 2016 (cit. on pp. 254, 255).

[745] W. Zhong and R. Zanibbi. “Structural similarity search for formulas using leaf-root paths in operator subtrees”. In: *Procs. of the European Conference on Information Retrieval 2019*. Springer, 2019, pp. 116–129 (cit. on pp. 161, 162).

[746] J. Ziv and A. Lempel. “A universal algorithm for sequential data compression”. In: *IEEE Transactions on Information Theory* 23.3 (1977), pp. 337–343 (cit. on p. 157).

[747] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu. “Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks”. In: *Advances in Neural Information Processing Systems 32: Procs. of the 2019 Conference*. 2019 (cit. on p. 136).

[748] H. Zou and T. Hastie. “Regularization and variable selection via the Elastic Net”. In: *Journal of the Royal Statistical Society B* 67 (2005), pp. 301–320 (cit. on p. 105).

[749] M. Zukowski, P. A. Boncz, N. Nes, and S. Héman. “MonetDB/X100–A DBMS In The CPU Cache”. In: *IEEE Data Engineering Bulletin* 28.2 (2005), pp. 17–22 (cit. on p. 385).

[750] U. Zwick. “Analyzing the MAX 2-SAT and MAX DI-CUT approximation algorithms of Feige and Goemans”. In: (2000) (cit. on p. 145).

Index

5G, 433

Accelerated Processing Unit (APU), 380
Acceleration, 253
Accelerator, 2, 263, 360
Application-Specific Integrated Circuit (ASIC), 4

Backpropagation, 139, 250, 256, 329, 330
Bagging, 341
Battery-powered device, 37, 46, 48, 52
Bayes
– naive Bayes classifier, 45, 316
– optimization, 8, 95
Bayesian network, 407
Belief propagation, 407, 410, 411, 413
Bernoulli experiment, 342
Bidirectional Encoder Representations (BER), 163, 166, 167, 169, 174–176
Bit
– Error Rate (BER), 326, 330–334
– flip, 10, 329–332, 334
Boosting, 339, 340

Cellular network, 433
Central Processing Unit (CPU), 2, 26, 28, 31, 40, 41, 53, 259, 263, 264, 270, 271, 276, 277, 280, 281, 284, 287, 288, 291–304, 315, 340–343, 351, 352, 354–356, 406, 411, 419
– architecture, 351
Channel
– distributed channel access, 425
Classification, 5, 7, 81, 87, 174
– multi-label, 272
Classification And Regression Tree (CART), 351
Cluster Feature Tree (CF-Tree), 219, 220, 222
Clustering, 8, 10, 86, 87, 186, 187, 197, 228, 230, 290, 291
– (k,l), 202
– biclustering, 228
– graph, 140
– hierarchical, 215, 221
– k-means, 200, 204, 212
– k-medoids, 182, 185
Code generation, 263, 350
Communication
– awareness, 74
– network, 7, 425, 430
– technologies, 423
Compression, 11, 111, 112, 114, 145, 157–159
Confidence interval, 79
Coprocessor, 4, 271, 380
Coresets, 10, 86–92, 95, 200, 212, 213
Covariance, 94
Cyber-physical system, 3, 45, 89, 285, 369, 370

Data
– acquisition, 16–20, 28, 49, 62, 66
– packet, 52, 425
– parallelism, 257
– stream, 8, 10, 71, 74, 80, 85, 87, 88, 406
– stream algorithms, 5, 10, 71, 74, 88, 89, 213
– summary, 74, 86, 89
– volume, 272, 427
Deadlines, 291, 292, 367
Decision tree, 7, 11, 45, 341, 343, 344, 350, 355
Dependency Graph Approach (DGA), 360, 362
Design space exploration, 426
Directed-Acyclic Graph (DAG), 367
Dynamic Power Management (DPM), 52
Dynamic voltage scaling, 57

Embedded system, 2–8, 15, 38, 47, 52, 60, 74, 89, 101, 285, 286, 363, 424
Energy
– awareness, 34, 35, 39, 41, 47, 48, 52, 54, 55, 59, 65, 66
– consumption, 3–10, 32, 34, 37, 39–46, 48, 52–59, 61, 65, 66, 68, 74, 249, 281, 300, 302–304, 326, 327, 406, 407, 424–429
– efficiency, 2, 6, 34, 36–38, 41, 45, 46, 48, 49, 56, 62, 67, 68
– harvesting, 34, 37, 47–59, 61, 65–68, 424
– measurement, 6, 40, 43, 50, 52

https://doi.org/10.1515/9783110785944-011

Ensembles, 11, 339–342
– tree, 11
Error measure, 188
– accuracy, 294, 295
– mean average precision, 173
– precision, 173
Euclidean distance, 123, 216, 218, 220, 222, 290
Event streams, 15, 19, 20, 26
Evolutionary algorithms, 7, 286
Exponential families, 9, 11, 90, 104, 409

Ferroelectric Field-Effect Transistors (FeFET), 326–328
Field-Programmable Gate Array (FPGA), 7, 8, 10, 249, 250, 340
Finite State Machine (FSM), 41, 42, 65
Floating point arithmetic, 328

Gradient descent, 232, 415
– stochastic, 329, 420
Graph partitioning, 145, 152, 155
Graphical models, 89, 92, 103, 106, 408, 411, 416
– probabilistic, 408
Graphics Processing Unit (GPU), 2, 8, 9, 140, 249, 253, 263–265, 270, 271, 300, 340, 341, 360, 361, 379, 407

Heterogeneous processors, 300

Idle state, 431
Indicator function, 103
Inference, 10, 11, 108, 137, 139, 166, 340, 341, 350
– probabilistic, 105, 407, 410, 411, 413
– quadrature-based, 105
– variational, 105
Instrumentation, 16, 17, 19, 26, 42
Internet of Things (IoT), 3, 6, 38, 46–50, 74, 85, 423, 433

k-nearest Neighbor (kNN), 45, 102, 316
Kalman filter, 101
Kernel functions, 5, 81, 96, 102, 118, 121–123, 294, 347, 355
– graph, 116, 118, 120, 124, 125
– hash graph, 124
– RBF, 81
– triangular, 168
– Weisfeiler Leman graph, 120–125, 127
Knowledge, 304
– a priori, 93, 161
– background, 239

L1 norm, 95, 105, 111, 113
L2 norm, 94, 105, 111, 113
Latency, 30, 31, 277, 342, 343, 427
Learning
– contrastive, 174, 175
– embeddings, 161, 162, 167
– self-supervised, 161, 163, 167
– supervised, 42, 272, 341
– unsupervised, 10, 102
Learning tasks, 5, 9, 81, 96, 100, 163
– masking tasks, 163
Leave-one-out, 171
Light, 57, 59, 61, 68
Linkage
– centroid, 216–218
– complete, 216, 290
– median, 216–218
– single, 216
– Ward, 216
Locking protocol, 361
Log-likelihood, 105, 113
Long Term Evolution (LTE), 423

Markov Random Fields (MRFs), 9, 11, 102, 112–114, 408, 416, 420, 421
Matrix factorization, 11
– binary, 234
– Boolean, 235
– nonnegative, 229
– objectives, 230
Max-Cut, 144
Maximum a posteriori, 105, 408, 409, 417, 420
Maximum likelihood, 93, 105, 408, 409, 413, 419
McQuitty’s Weighted Pair-Group Method with Arithmetic mean (WPGMA), 216, 217, 222
Memory, 9, 10, 74–78, 82, 83, 136, 157, 184, 222, 307, 308, 313, 327, 344, 347
– allocation, 277, 284, 309–313
– architecture, 276, 277, 342
– bottleneck, 3, 277, 281, 284
– cache, 3, 281, 284, 340–345, 347
– capacity, 3
– footprint, 108, 110, 111, 137, 138, 140, 307, 309, 313, 314, 329

– hierarchy, 3, 276, 281, 284, 343
– layout, 27, 313, 341, 342, 345
– locality, 276, 277, 342
– Magnetoresistive Random Access Memory (MRAM), 4
– Non-Uniform Memory Access (NUMA), 276
– non-volatile, 10, 326
– physical, 308, 309
– scratchpad, 3, 381
– shared, 145, 152, 158, 310, 381
– Static Random-Access Memory (SRAM), 4, 67, 326, 328
– virtual, 308–311
Merge & Reduce, 87, 88
Message passing, 8, 10, 105, 125, 130–132, 143, 410, 412, 413
Model execution, 341
Monte Carlo, 287, 288, 294, 297
Multicore, 4, 8, 33, 249, 275, 285, 296, 304
Multilayer perceptrons, 8, 137, 250

Nearest-neighbor chain algorithm, 222
Nearest-neighbor search, 174
Neural Network
– Binarized Neural Networks (BNN), 10, 326, 328–333
– Convolutional Neural Networks (CNN), 62, 130, 162, 163, 333
– Deep Neural Networks (DNN), 15, 61, 328, 434
– Graph Neural Networks (GNN), 8–10, 116, 117, 125–128, 130–135, 137–141, 161–163, 165
Neural network
– Convolutional Neural Networks (CNN), 254

Offloading, 4, 369, 378
One-vs-Rest Classification, 274
Operating system kernel, 17–20, 22, 23, 27–30, 32, 33, 310
– data, 17, 22

Parallelism, 8, 249, 275, 287, 386
Partition function, 104
Peripheral Component Interconnect express (PCIe), 380
Permission, 32, 61, 310
Poisson dependency network, 92
Potential function, 105, 411
Power
– consumption, 2, 3, 6, 7, 48, 55, 58, 64, 65, 256, 300, 301, 426, 429–433
– measurement, 42, 45, 47, 49, 50, 58, 59, 300, 301
Probability density function (pdf), 95, 104
Processor utilization, 288
Proximal Optimization, 233
Pruning, 340

Quantization, 10
Query compilation, 386

Radio, 49, 425, 433
– mobile radio networks, 429
– Software-Defined Radio (SDR), 433
Random forest, 45, 341, 434
Real-time system, 34, 360, 361, 370, 378
Regression, 5, 6, 10, 43, 85–87, 90, 102, 285–292, 298, 316, 341, 434, 435
– Bayesian, 89, 93, 94
– generalized linear regression models, 89, 90, 92
– hierarchical, 94
– LASSO, 95
– linear, 6, 10, 86, 89, 90, 92, 93
– logistic, 90–92, 316
– ordinary least squares, 92
– Poisson, 92
– probit, 90, 95
Regularization, 5, 11, 93, 103, 105, 110–114, 234, 236–238, 275, 331, 333
Representation
– Term-frequency inverse-document-frequency (Tf-idf), 272
Representation learning, 129, 130, 167, 174, 176
Resource
– awareness, 288, 289, 292
– efficiency, 227, 286
– efficient transmission, 425
– optimization, 429, 430
– synchronization, 361
– utilization, 16, 263, 268, 285, 287, 297, 298, 304, 430
Resource Block (RB), 432
Resource-constrained, 47, 61
Resource-constrained learning, 101, 102,

108, 110, 111, 113, 136
Rule of Three, 79, 80
Runtime
– estimation, 289, 291

Sampling, 5, 6, 9, 10, 48, 64, 67, 85, 90–92, 130, 136, 139, 167, 168, 201, 203, 240, 287, 288, 294, 301
– importance, 86, 89, 91, 93, 136
– layer-wise, 136
– node-wise, 136
– reservoir, 76
– sub-sampling, 137, 139
– subgraph, 136
– uniform, 93
Scalability, 10, 87, 108, 130, 137, 140, 143, 223, 224, 293, 294, 304, 426
Scheduling, 8, 9, 26, 27, 39, 285, 286, 288–292, 294, 295, 297–300, 302, 304, 366, 411, 432
– Earliest Deadline First (EDF), 366
– federated, 367
– List-Earliest Deadline First (List-EDF), 366
– non-preemptive fixed-priority, 363
– Partitioned Earliest-Deadline-First (P-EDF), 367
– preemptive fixed-priority, 375
– Worst-Fit Partitioned Earliest-Deadline-First (WF-P-EDF), 368
Security, 31, 68
Sensor network, 6, 15, 34, 74, 101, 106
Shannon entropy, 104
Signal
– digital processing, 430
– Received Signal Strength Indicator (RSSI), 45
Similarity measure, 117, 121, 162, 197, 198, 200, 204
– cosine, 162, 174, 175
– dynamic time warping, 199
– Fréchet distance, 198, 199
Single Instruction, Multiple Data (SIMD), 386
Sketches, 10, 85, 86, 88–91, 93, 284
Spatio-temporal state prediction, 100, 102, 103, 106, 111, 113, 114
Spectral efficiency, 425
Speedup, 184, 223, 279, 281, 282, 322
Star Schema Benchmark (SSB), 381
Submodular functions, 10, 74–77, 80, 84
Sufficient statistics, 103, 104, 409, 410
Support vector data description, 212
Support Vector Machine (SVM), 10, 45, 316, 332
Synchronization, 4, 140, 284

Task period, 368
Tensor Processing Unit (TPU), 2
Test bed, 34–38, 40, 45
Thread parallelism, 275
Tiling, 214
Time series data, 62, 65, 101, 197, 198
Transformer, 163, 165, 166, 176, 283
Transmission power, 44, 425, 430, 433

Ultra-low power device, 9
Ultra-low power state, 44, 52, 59, 67
Uplink power, 433
User Equipment (UE), 423, 425, 430, 431

Variance, 96, 139, 291
von Neumann architecture, 2, 3

Wasserstein distance, 94, 122, 123
Wireless Sensor Network (WSN), 35, 36
Worst Case Execution Time (WCET), 363, 370
Worst-Fit Heuristic, 367

List of Contributors

Editors

Morik, Katharina, Prof. Dr. TU Dortmund University, Department of Computer Science, Head of the chair for Artificial Intelligence & Speaker of the Collaborative Research Center 876 on Providing Information by Resource-Constrained Data Analysis, katharina.morik@tu-dortmund.de

Marwedel, Peter, Prof. Dr. TU Dortmund University, Department of Computer Science, CRC 876, peter.marwedel@tu-dortmund.de

Contributors

Amrouch, Hussam, Jun.-Prof. Dr.-Ing. Universität Stuttgart, amrouch@iti.uni-stuttgart.de

Babbar, Rohit, Ph.D. Aalto University, Assistant Professor Department of Computer Science, rohit.babbar@aalto.fi

Bertram, Nico, M. Sc. TU Dortmund University, Department of Computer Science, CRC 876, nico.bertram@tu-dortmund.de

Borchert, Christoph, Dr.-Ing. Osnabrück University, Institute of Computer Science, CRC 876, christoph.borchert@uni-osnabrueck.de

Buschhoff, Markus, Dr.-Ing. TU Dortmund University, Department of Computer Science, Design Automation for Embedded Systems Group, CRC 876, markus.buschhoff@tu-dortmund.de

Buschjäger, Sebastian, M. Sc. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, sebastian.buschjaeger@tu-dortmund.de

Chen, Jian-Jia, Prof. Dr. TU Dortmund University, Department of Computer Sciences, Head of the chair for Design Automation for Embedded Systems, CRC 876, jian-jia.chen@cs.uni-dortmund.de

Chen, Kuan-Hsun, Dr.-Ing. University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science, CRC 876, k.h.chen@utwente.nl

Ellert, Jonas, M. Sc. TU Dortmund University, Department of Computer Science, Chair 11 Algorithm Engineering, CRC 876, jonas.ellert@tu-dortmund.de

Falkenberg, Robert, M. Sc. TU Dortmund University, Communications Networks Institute, CRC 876, robert.falkenberg@tu-dortmund.de

Fey, Matthias, M. Sc. TU Dortmund University, Department of Computer Science, Computer Graphics Lab, CRC 876, matthias.fey@tu-dortmund.de

Fischer, Johannes, Prof. Dr. TU Dortmund University, Department of Computer Science, Chair 11 Algorithm Engineering, CRC 876, johannes.fischer@cs.tu-dortmund.de

https://doi.org/10.1515/9783110785944-012

Funke, Henning, M. Sc. TU Dortmund University, Department of Computer Science, Databases and Information Systems Group, CRC 876, henning.funke@cs.tu-dortmund.de

Gómez, Andrés, Ph.D. University of St.Gallen, Interaction- and Communication-based Systems Group, andres.gomez@unisg.ch

Guo, Ce, Professor Imperial College London, Faculty of Engineering, Department of Computing, c.guo@imperial.ac.uk

Hess, Sibylle, Dr. Eindhoven University of Technology, Assistant Professor, Mathematics and Computer Science, Data Mining, CRC 876, s.c.hess@tue.nl

Kotthaus, Helena, Dr. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, helena.kotthaus@cs.tu-dortmund.de

Kriege, Nils M., Prof. Dr. University of Vienna, Faculty of Computer Science, Data Mining and Machine Learning (DM), CRC 876, nils.kriege@univie.ac.at

Krivošija, Amer, Dr. TU Dortmund University, Department of Statistics, Mathematical Statistics with Applications in Biometrics, Projekt FAIR, CRC 876, amer.krivosija@tu-dortmund.de

Lang, Andreas, M. Sc. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, andreas.lang@tu-dortmund.de

Lenssen, Lars, M. Sc. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, lars.lenssen@tu-dortmund.de

Lochmann, Alexander, M. Sc. TU Dortmund University, Department of Computer Science, Embedded System Software Group, CRC 876, alexander.lochmann@tu-dortmund.de

Luk, Wayne, Prof. Dr. Imperial College London, Faculty of Engineering, Department of Computing, w.luk@imperial.ac.uk

Masoudinejad, Mojtaba, Dr.-Ing. TU Dortmund University, Department of Computer Science, Design Automation for Embedded Systems Group, CRC 876, mojtaba.masoudinejad@tu-dortmund.de

Mayer, Simon, Prof. Dr. University of St. Gallen, Interaction- and Communication-based Systems Group, simon.mayer@unisg.ch

Morris, Christopher, Dr. RWTH Aachen, Faculty of Electrical Engineering and Information Technology, CRC 876, christopher.morris@tu-dortmund.de

Munteanu, Alexander, Dr. TU Dortmund University, Dortmund Data Science Center, Faculties of Statistics and Computer Science, CRC 876, alexander.munteanu@tu-dortmund.de

Pfahler, Lukas, M. Sc. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, lukas.pfahler@tu-dortmund.de

Piatkowski, Nico, Dr. Fraunhofer Institute for Intelligent Analysis and Information Systems, CRC 876, nico.piatkowski@iais.fraunhofer.de

Schubert, Erich, Prof. Dr. TU Dortmund University, Department of Computer Science, Artificial Intelligence Group, CRC 876, erich.schubert@cs.tu-dortmund.de

Schultheis, Erik, M. Sc. Aalto University, Department of Computer Science, erik.schultheis@aalto.fi

Shi, Junjie, M. Sc. TU Dortmund University, Department of Computer Science, Design Automation for Embedded Systems Group, CRC 876, junjie.shi@tu-dortmund.de

Sliwa, Benjamin, Dr.-Ing. TU Dortmund University, Senior researcher Communications Networks Institute, CRC 876, benjamin.sliwa@tu-dortmund.de

Spinczyk, Olaf, Prof. Dr.-Ing. Osnabrück University, Department for Mathematics and Computer Science, Leader of the Embedded Software Systems Group, CRC 876, olaf@uos.de

Streicher, Jochen, Dipl.-Inf. TU Dortmund University, Department of Computer Science, Embedded System Software Group, CRC 876, jochen.streicher@tu-dortmund.de

Suter, Lars, M.A. University of St.Gallen, Interaction- and Communication-based Systems Group, larskai.suter@student.unisg.ch

Teubner, Jens, Prof. Dr. TU Dortmund University, Department of Computer Science, Databases and Information Systems Group, CRC 876, jens.teubner@cs.tu-dortmund.de

Weichert, Frank, Priv.-Doz. Dr. TU Dortmund University, Department of Computer Science, Intelligent Sensing in Computer Graphics, CRC 876, frank.weichert@tu-dortmund.de

Yayla, Mikail, M. Sc. TU Dortmund University, Department of Computer Science, Design Automation for Embedded Systems Group, CRC 876, mikail.yayla@tu-dortmund.de

Technical Editors

Becker, Andreas, Dr. TU Dortmund University, Department of Computer Science, CRC 876, andreas3.becker@tu-dortmund.de

Buß, Jens, Dr. TU Dortmund University, Department of Computer Science, Managing Director of the Collaborative Research Center 876, jens.buss@tu-dortmund.de

Acknowledgment

Part of the work on this book is the result of research of the Collaborative Research Center 876 "Providing Information by Resource-Constrained Analysis", which was funded from 2011–2022 by the Deutsche Forschungsgemeinschaft (DFG) under DFG project number 124020371, see: https://gepris.dfg.de/gepris/projekt/124020371?language=en.