Research Article Open Access

Eduard Vatutin*, Oleg Zaikin, Stepan Kochemazov, and Sergey Valyaev

Using Volunteer Computing to Study Some Features of Diagonal Latin Squares

https://doi.org/10.1515/eng-2017-0052
Received October 14, 2017; accepted November 19, 2017

Abstract: In this research, the study concerns around several features of diagonal Latin squares (DLSs) of small order. Authors of the study suggest an algorithm for computing minimal and maximal numbers of transversals of DLSs. According to this algorithm, all DLSs of a particular order are generated, and for each square all its transversals and diagonal transversals are constructed. The algorithm was implemented and applied to DLSs of order at most 7 on a personal computer. The experiment for order 8 was performed in the volunteer computing project Gerasim@home. In addition, the problem of finding pairs of orthogonal DLSs of order 10 was considered and reduced to Boolean satisfiability problem. The obtained problem turned out to be very hard, therefore it was decomposed into a family of subproblems. In order to solve the problem, the volunteer computing project SAT@home was used. As a result, several dozen pairs of described kind were found.

Keywords: Latin square, diagonal Latin square, transversal, enumeration, orthogonality, SAT, volunteer computing, BOINC

1 Introduction

A Latin square of order N is a square table $N \times N$ filled with elements from some finite set of size N in such a way, that all elements within a single row or single column are distinct [7]. A Latin square is called diagonal if all elements in both its main diagonal and main antidiagonal are distinct.

Latin squares represent one of the most well studied combinatorial designs. However, even for relatively small orders there remain extremely hard open problems. That is why the researchers in the area of combinatorial designs often use computational experiments in their work. In the present study, two problems were considered. In the first one it is required to enumerate transversals of diagonal Latin squares of order up to 8. The second one consists in finding new pairs of orthogonal diagonal Latin squares of order 10. To solve the former a combinatorial algorithm was developed. The latter problem was reduced to Boolean satisfiability problem.

Both mentioned problems turned out to be quite hard, so high-performance computing was employed to solve them. In particular, they both were decomposed into a families of independent subproblems in accordance with the concept of embarassing parallelism [9]. In order to solve embarassingly parallel problems, desktop grids are usually employed [6]. Alternatively, enterprise desktop grids [12, 24] (as well as other types of desktop grids) could be employed to solve them. However, in the present study another type of desktop grid computing – volunteer computing [1] – was used. The defining characteristic of this type of computing is that it uses resources of volunteer's computers. Volunteer computing projects have been successfully used over the past two decades to solve problems from various areas (e.g., [2, 13, 35]). In the present study, two volunteer computing projects, Gerasim@home and SAT@home, were used to solve two mentioned combinatorial problems.

Let us give a brief outline of the paper. Section 2 describes some preliminaries on diagonal Latin squares and volunteer computing. In Section 3 the results on computing the minimal/maximal number of transversals of diagonal Latin squares of small order are presented. Section 4 describes how new pairs of orthogonal diagonal Latin squares of order 10 were found.

***Corresponding Author: Eduard Vatutin:** Southwest State University, Kursk, Russia, E-mail: evatutin@rambler.ru
Oleg Zaikin: Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences, Irkutsk, Russia, E-mail: zaikin.icc@gmail.com
Stepan Kochemazov: Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences, Irkutsk, Russia, E-mail: veinamond@gmail.com
Sergey Valyaev: Internet-portal BOINC.ru, Moscow, Russia, E-mail: serval@boinc.ru

2 Preliminaries

This section provides general information regarding diagonal Latin squares and volunteer computing, that is required to understand the following sections. In particular, it is described, which problems are considered and how exactly they are solved.

2.1 Diagonal Latin Squares

Without the loss of generality, a Latin square (LS) of order N is a square table $A = ||a_{ij}||$, $i, j = 1, \dots, N$, filled with elements a_{ij} from the set $\{0, \dots, N-1\}$, so that in each row and column each element appears exactly once.

Diagonal Latin squares (DLSs) form a special case of LS. In DLS both main diagonal and main antidiagonal contain every possible element from 0 to $N-1$. A DLS is called *normalized* if the elements in its first row are in an ascending order. It is easy to show that any DLS can be normalized by means of a bijective mapping (transposition) of elements from $\{0, \dots, N-1\}$. It follows from this fact that the corresponding set of DLSs forms an equivalence class containing $N!$ members.

A set of N entries, one selected from each row and each column of a LS of order N such that no two entries contain the same symbol, is called a *transversal* [32]. An example of a LS and the set of its transversals is shown in Figure 1. It is easy to see that both diagonals are transversals ($T_1 = \{a_{11}, a_{22}, a_{33}, a_{44}, a_{55}\}$ and $T_2 = \{a_{15}, a_{24}, a_{33}, a_{42}, a_{51}\}$).

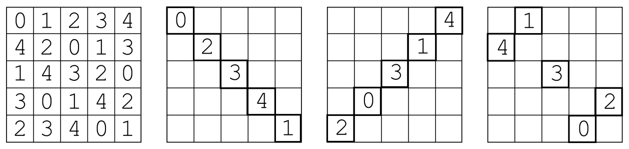
Figure 1: A Latin square of order 5 and its transversals.

A transversal $T^{(d)}$ is called `diagonal` if it contains exactly one element from the main diagonal of a LS and exactly one element from its main antidiagonal. A diagonal transversal is always a transveral of a general kind. An example of a LS and a set of its diagonal transversals is shown in Figure 2.

Two transversals T_1 and T_2 are called orthogonal if they do not contain the same elements of a LS. In other words:

$$T_1 \perp T_2 \leftrightarrow T_1 \cap T_2 = \emptyset.$$

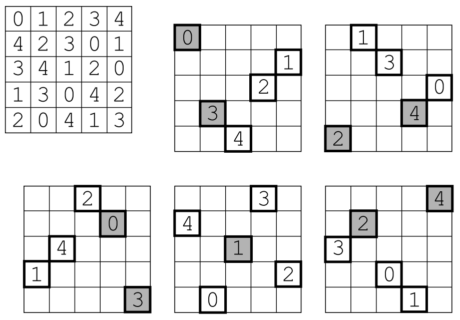
Figure 2: A Latin Square and its diagonal transversals. Transversals elements from main diagonal and main antidiagonal are marked with gray color.

In the example presented in Figure 1 no two transversals are orthogonal because they contain one common element a_{33}. At the same time all diagonal transversals in Figure 2 are pairwise orthogonal.

Two LSs $A = ||a_{ij}||$ and $B = ||b_{ij}||$ are called orthogonal if all ordered pairs $[a_{ij}, b_{ij}]$ are distinct. One of the most famous combinatorial problems is to prove the existence or non-existence of a triple of mutually orthogonal Latin squares (MOLS) of order 10. Note, that mutually orthogonal diagonal Latin squares (MODLS) are quite rare compared to MOLS.

It is known that a LS A of order N has an orthogonal mate B if and only if A has N mutually orthogonal transversals (for DLSs – N mutually orthogonal diagonal transversals). Having the corresponding set of N transversals it is easy to construct a square B: the elements corresponding to transversal T_i in square A are assigned with value $i-1$ in square B. Further, transversals are enumerated starting from 1 and the values of LS elements are enumerated starting from 0. Figure 3 shows the steps leading to construction of an orthogonal mate for the DLS from Figure 2.

2.2 Volunteer Computing

Volunteer computing [1] is one of the many types of distributed computing. It allows usual people – *volunteers* to participate in scientific studies that require large-scale computational experiments. Usually, within the context of volunteer computing volunteers donate the idle resources of their PCs to some project. A volunteer computing project is an arrangement which is organized specifically to solve one or several hard problems. Generally, volunteers are free to choose from several computing projects to give the available computing power to. The important detail is that volunteer computing projects are usually organized in such a way that the participation in a project does not in-

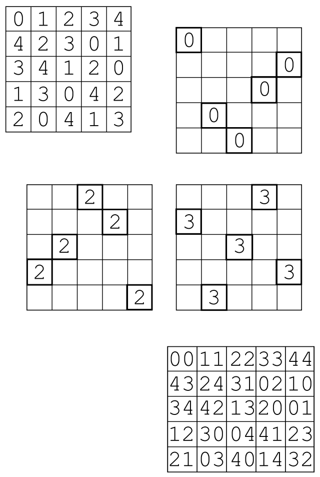
Figure 3: Constructing an orthogonal mate using the set of mutually orthogonal transversals

convenience a volunteer: the calculations are performed automatically, and the corresponding manager software allows fine tuning with regard to when it can employ the available resources, how many and for how long. Also, a feature of such projects is that by default only idle computational resources are used.

An evident limitation arising from the organization of volunteer computing projects consists in the fact that they can be used to solve only problems that allow embarrassing parallelism [9]. Since volunteers are usually not provided with any monetary rewards, the volunteer computing is quite cheap for scientists, compared to other distributing computing types. In fact, to maintain a volunteer computing project it is enough to have a special server coupled with a number of client applications, which function within the common infrastructure. Since volunteer computing has a strong social aspect, it means that providing feedback and answering the questions of volunteers is crucial for the success of a project. A distinctive advantage of volunteer computing consists in the fact that it allows to perform computational experiments for months or even years, unlike computing clusters or grids.

One of the key milestones in the development and popularization of volunteer computing was the birth of the Berkeley Open Infrastructure for Network Computing (BOINC) [1], developed in Berkeley in 2002. The majority of modern projects are based on BOINC. Currently, there are about 70 active BOINC projects with total performance exceeding 11 PFLOPs. The structure of a BOINC project includes a server database, website, client applications and a number of server daemons. The latter include *work generator*, *validator* and *assimilator*. The work generator by definition generates tasks which are sent to volunteers. The validator is used to check if the results obtained from volunteers are correct or not. The correct results are transferred to the assimilator daemon to be processed. The client applications are the workhorses that perform actual computations. The more computing platforms are covered, the more performance a project can hope to achieve. The performance can be also boosted by several actions connected with volunteers (competitions, badges, etc., see [14]).

3 Enumerating Transversals for DLSs of small order in Gerasim@home

This section describes the algorithm for enumerating transversals for DLSs. It was implemented and applied to analyze DLSs of order up to 8. For order 8 the corresponding experiment was performed in a volunteer computing project.

3.1 Algorithm for Enumerating Transversals of Diagonal Latin Squares

From the constraint on the uniqueness of pairs of indices within a transversal (see Section 2) it is easy to construct the upper bound for their number. It is equal to $N!$, the number of transpositions of elements from $\{0, \dots, N-1\}$. The process of constructing transversals is quite similar to that of solving the widely known rooks puzzle – with each filled element the imposed constraints significantly reduce the size of the remaining search space. Thus, for a particular LS it is not hard to find the set of all its transversals.

The number of transversals significantly differs from square to square. There are no known analytical results regarding how the minimal and maximal number of transversals depend on the order of a LS. Nevertheless, due to computational experiments, sequences of minimal/maximal numbers of transverals for LSs are presented in Online Encyclopedia of Integer Sequences. In particular, entry A091323 shows the sequence of minimal number of transversals in a LS of order $2n + 1$, and entry A090741 shows the sequence of maximum number of transversals in a LS of order n [20]. The question, whether the maximal number of transversals for a LS of order 10 is 5504 or not, is still considered to be an open problem [5].

Similar estimations for the numbers of transverals and diagonal transversals for DLSs are unknown and can be determined in the course of a computational experiment. For this purpose, one needs to generate all possible DLSs of order N, and for each DLS to construct the sets of its transversals and diagonal transversals. In order to generate all possible DLSs, it is possible to employ the highly efficient algorithm, developed by authors of the present paper for this specific purpose. The algorithm and its implementation have several special optimizations that take into account algorithmic features of this problem:

– it employs a specific order of filling LS elements based on the number of available variants [11, 31];
– it uses static data structures that help to avoid placing critical data into dynamic memory;
– it tracks the sets of possible variants of cells values for unfilled cells and terminates the exploration of branches of the search tree if there are square cells with empty sets of variants;
– it employs auxiliary data structures (one-dimensional arrays) and bit arithmetic to determine the sets of possible variants fast [29].

The construction of a set of diagonal transversals for a specific DLS is done using the following recurrent algorithm:

1. **Initialization.** Specify initial values for a set of transversals $S := \emptyset$, current recursion depth $d := 0$, set of available columns $C := \{0, \dots, N-1\}$, set of available elements values $E := \{0, \dots, N-1\}$.
2. **Condition for recursion end.** If $d = N$ then add current transversal T to a set of found transversals $S := S \cap \{T\}$, decrease current recursion level $d := d - 1$, go to $3c$.
3. For all values $i = \overline{1, N}$ such that $(i \in C) \wedge (a_{di} \in E)$:
   (a) Add a_{di} to transversal T: $T[d] := i$; mark i-th column as used: $C := C \backslash \{i\}$; mark a_{di} as used: $E := E \backslash \{a_{di}\}$.
   (b) **Recurrent descent** Increase current recursion depth value: $d := d + 1$; go to 2.
   (c) Mark i-th column as available: $C := C \cup \{i\}$; mark a_{di} as available: $E := E \cup \{a_{di}\}$.
4. **End of algorithm.**

This algorithm finds all possible transversals of a given LS A. If it is necessary to find only diagonal transversals, then the corresponding checks are added to the 2-nd point of the algorithm. In accordance with the branch and bound strategy it is also possible (but not necessary) to add an additional condition which checks whether the current transversal contains LS elements from its main diagonal and main antidiagonal. The latter can increase the algorithm performance by early rejection of non-diagonal transversals.

3.2 Computational experiment

The algorithm proposed in Subsection 3.1 was used to organize a computational experiment aimed at determining minimal and maximal number of transversals and diagonal transversals of DLSs of small order.

Table 1: Minimal and maximal number of transversals for DLS.

N	Min. number, corresponding DLS	Max. number, corresponding DLS
4	8 $\begin{pmatrix} 0\,1\,2\,3 \\ 3\,2\,1\,0 \\ 1\,0\,3\,2 \\ 2\,3\,0\,1 \end{pmatrix}$	8 $\begin{pmatrix} 0\,1\,2\,3 \\ 3\,2\,1\,0 \\ 1\,0\,3\,2 \\ 2\,3\,0\,1 \end{pmatrix}$
5	3 $\begin{pmatrix} 0\,1\,2\,3\,4 \\ 4\,2\,0\,1\,3 \\ 1\,4\,3\,2\,0 \\ 3\,0\,1\,4\,2 \\ 2\,3\,4\,0\,1 \end{pmatrix}$	15 $\begin{pmatrix} 0\,1\,2\,3\,4 \\ 4\,2\,3\,0\,1 \\ 3\,4\,1\,2\,0 \\ 1\,3\,0\,4\,2 \\ 2\,0\,4\,1\,3 \end{pmatrix}$
6	32 $\begin{pmatrix} 0\,1\,2\,3\,4\,5 \\ 4\,2\,5\,0\,3\,1 \\ 3\,5\,1\,2\,0\,4 \\ 5\,3\,0\,4\,1\,2 \\ 2\,4\,3\,1\,5\,0 \\ 1\,0\,4\,5\,2\,3 \end{pmatrix}$	32 $\begin{pmatrix} 0\,1\,2\,3\,4\,5 \\ 4\,2\,5\,0\,3\,1 \\ 3\,5\,1\,2\,0\,4 \\ 5\,3\,0\,4\,1\,2 \\ 2\,4\,3\,1\,5\,0 \\ 1\,0\,4\,5\,2\,3 \end{pmatrix}$
7	7 $\begin{pmatrix} 0\,1\,2\,3\,4\,5\,6 \\ 6\,2\,3\,5\,0\,1\,4 \\ 4\,5\,1\,6\,2\,3\,0 \\ 2\,3\,6\,4\,5\,0\,1 \\ 1\,6\,5\,0\,3\,4\,2 \\ 5\,0\,4\,2\,1\,6\,3 \\ 3\,4\,0\,1\,6\,2\,5 \end{pmatrix}$	133 $\begin{pmatrix} 0\,1\,2\,3\,4\,5\,6 \\ 4\,2\,6\,0\,5\,1\,3 \\ 3\,5\,1\,6\,0\,4\,2 \\ 5\,6\,3\,4\,1\,2\,0 \\ 6\,4\,5\,2\,3\,0\,1 \\ 1\,3\,0\,5\,2\,6\,4 \\ 2\,0\,4\,1\,6\,3\,5 \end{pmatrix}$
8	8 $\begin{pmatrix} 0\,1\,2\,3\,4\,5\,6\,7 \\ 1\,2\,0\,4\,3\,7\,5\,6 \\ 4\,7\,5\,1\,6\,0\,2\,3 \\ 3\,5\,7\,6\,1\,2\,0\,4 \\ 5\,4\,6\,2\,7\,1\,3\,0 \\ 7\,6\,4\,0\,5\,3\,1\,2 \\ 2\,3\,1\,7\,0\,6\,4\,5 \\ 6\,0\,3\,5\,2\,4\,7\,1 \end{pmatrix}$	384 $\begin{pmatrix} 0\,1\,2\,3\,4\,5\,6\,7 \\ 1\,2\,3\,0\,6\,7\,5\,4 \\ 6\,5\,7\,4\,1\,3\,2\,0 \\ 5\,3\,4\,1\,2\,0\,7\,6 \\ 2\,7\,0\,6\,5\,4\,3\,1 \\ 3\,4\,1\,5\,7\,6\,0\,2 \\ 7\,0\,6\,2\,3\,1\,4\,5 \\ 4\,6\,5\,7\,0\,2\,1\,3 \end{pmatrix}$

The experiment for $1 \le N \le 7$ was performed on one core of Intel Core i7-4770 (Haswell) CPU. The algorithm for estimating the number of transversals for DLS of order $N = 8$ has an average performance of about 350 DLS per second (for single-threaded implementation on Delphi language on one core of the mentioned CPU). Taking into account the fact, that the number of DLSs of order 8 is 7 447 587 840 [31], this experiment was estimated to take about 246 days on one core or about 1 month on 8 cores.

When searching for diagonal transversals for DLS of order $N = 8$ the performance is about 440 DLS per second. It is significantly higher than for nondiagonal transversals. Apparently the reason for this is that there are more constraints and, because of this, less branches in the corresponding search trees.

The mentioned experiment was performed in the BOINC-based volunteer computing project Gerasim@home [10, 30]. In total, 3 003 workunits were generated for this purpose. In each workunit the first 5 elements of the second row of a DLS were fixed. Thus, an original problem was decomposed on the server by varying these 5 elements. Average runtime of each workunit was about 1 hour on 1 CPU core. The computing application had to generate all normalized DLSs with fixed 5 elements (filled in accordance with the given workunit), and determine for them the required characteristics. The results of experiments for orders $1 \le N \le 8$ are shown in Tables 1 and 2.

For DLS of order $N = 9$ the average performance of our algorithm is 370 DLS per second. It means that the current implementation will take about 900 years in Gerasim@home in its present form.

It is easy to see that for $1 \le N \le 8$ the maximal number of transversals for DLS coincides with that for LS (sequence A090741 in the on-line encyclopedia of integer sequences (OEIS [25])), excluding the orders 2 and 3 for which there are no DLS. It is likely that this feature will be repeated for DLS of higher orders. The minimal number of transversals is the new sequence (1, 0, 0, 8, 3, 32, 7, 8), which has not yet been represented in OEIS.

The obtained sequences of minimal (1, 0, 0, 4, 1, 2, 0, 0) and maximal number of diagonal transversals (1, 0, 0, 4, 5, 6, 27, 120) have not been represented in OEIS before.

Table 2: Minimal and maximal number of diagonal transversals for DLS.

N	Min. number, corresponding DLS	Max. number, corresponding DLS
4	4 $\begin{pmatrix}0&1&2&3\\3&2&1&0\\1&0&3&2\\2&3&0&1\end{pmatrix}$	4 $\begin{pmatrix}0&1&2&3\\3&2&1&0\\1&0&3&2\\2&3&0&1\end{pmatrix}$
5	1 $\begin{pmatrix}0&1&2&3&4\\4&2&0&1&3\\1&4&3&2&0\\3&0&1&4&2\\2&3&4&0&1\end{pmatrix}$	5 $\begin{pmatrix}0&1&2&3&4\\4&2&3&0&1\\3&4&1&2&0\\1&3&0&4&2\\2&0&4&1&3\end{pmatrix}$
6	2 $\begin{pmatrix}0&1&2&3&4&5\\4&2&5&0&3&1\\3&5&1&2&0&4\\5&3&0&4&1&2\\2&4&3&1&5&0\\1&0&4&5&2&3\end{pmatrix}$	6 $\begin{pmatrix}0&1&2&3&4&5\\4&2&5&1&3&0\\3&5&1&2&0&4\\5&3&0&4&1&2\\2&4&3&0&5&1\\1&0&4&5&2&3\end{pmatrix}$
7	0 $\begin{pmatrix}0&1&2&3&4&5&6\\6&2&4&5&1&0&3\\5&0&1&6&2&3&4\\3&5&6&4&0&1&2\\2&6&5&1&3&4&0\\4&3&0&2&5&6&1\\1&4&3&0&6&2&5\end{pmatrix}$	27 $\begin{pmatrix}0&1&2&3&4&5&6\\4&2&6&0&5&1&3\\3&5&1&6&0&4&2\\5&6&3&4&1&2&0\\6&4&5&2&3&0&1\\1&3&0&5&2&6&4\\2&0&4&1&6&3&5\end{pmatrix}$
8	0 $\begin{pmatrix}0&1&2&3&4&5&6&7\\1&2&0&4&3&7&5&6\\7&0&1&6&5&4&2&3\\4&6&7&5&1&0&3&2\\3&7&5&0&6&2&4&1\\5&4&6&7&2&3&1&0\\6&3&4&2&0&1&7&5\\2&5&3&1&7&6&0&4\end{pmatrix}$	120 $\begin{pmatrix}0&1&2&3&4&5&6&7\\2&3&0&1&6&7&4&5\\1&5&4&0&7&3&2&6\\5&4&7&6&1&0&3&2\\3&7&6&2&5&1&0&4\\7&6&5&4&3&2&1&0\\4&0&1&5&2&6&7&3\\6&2&3&7&0&4&5&1\end{pmatrix}$

4 Search for MODLS in SAT@home

Problems from various areas (hardware and software verification, cryptography, combinatorics) can be effectively reduced to Boolean satisfiability problem (SAT) [18]. In 2011 the authors of the present paper started the BOINC-based volunteer computing project SAT@home [26, 27]. It is aimed at solving hard SAT instances. In SAT@home each SAT instance is decomposed into a family of independent subproblems according to the concept of embarassing parallelism. The client application in SAT@home is based on Conflict-Driven Clause Learning SAT solvers [18].

In the paper [5] the first three pairs of MODLS of order 10 were presented. In 2012 in SAT@home there was launched a computational experiment aimed at finding new pairs of MODLS of order 10. For this purpose a “naive” propositional encoding (e.g., see [17]) was constructed. It particular, the search for such pairs of DLSs where the first DLS was normalized (see Section 2) was encoded. This encoding contains 2000 Boolean variables and 434440 clauses.

Table 3 shows the results for 8 decomposition strategies, which were implemented in the project's work generator. In the first 7 decomposition strategies several cells in several rows of a DLS were filled from left to right. Since the first row is always fixed, it is reasonable to vary the values of row's cells starting from the second row. According to the 8th decomposition strategy, “2 diag., 9 cells each”, values of 18 cells from the main diagonal and the main antidiagonal of the first DLS were varied. For each partially filled first DLS the SAT solver was launched on a SAT instance encoding the problem of finding a pair of MODLS with specified constraints on the first square of the pair with the time limit of 5 minutes.

Table 3: The results of experiments (based on various decomposition strategies), aimed at finding new pairs of MODLS of order 10.

Decomposition	MODLS found	Time
1 row, 9 cells	1	1 month, 2015
2 rows, 2 cells each	-	1 day, 2015
2 rows, 3 cells each	-	3 days, 2015
2 rows, 4 cells each	-	2 weeks, 2015
2 rows, 5 cells each	26	4 months, 2015-2016
2 rows, 6 cells each	5	3 months, 2015-2016
2 rows, 8 cells each	17	9 months, 2012-2013
2 diag., 9 cells each	28	5 months, 2016

Let us comment the Table 3. The decomposition “2 rows, 8 cells each” corresponds to the decomposition that was used in the first stage of the experiment. The remaining 7 decompositions have been tested in 2015-2016. Using the decomposition strategies “2 rows, 2 cells each”, “2 rows, 3 cells each” and “2 rows, 4 cells each” it was not possible to find any pairs of MODLS at all. The “2 rows, 5 cells each” strategy turned out to be more effective than “2 rows, 8 cells each” used in 2012, because it allowed to find more pairs of MODLS per time unit (even taking into account the improved performance of the SAT@home project).

As a result of our experiments, in total 77 new pairs of MODLS of order 10 were found, which differ from that published in [5]. All found solutions are available online at the SAT@home web site [26]. It should be noted, that the presented results were obtained with the help of CluBORUn tool [34], which increased the project's performance by employing idle resources of several computing clusters.

Let us consider a triple of DLSs of the same order. Among all possible sets of ordered pairs of elements, that match orthogonality condition for all three pairs of squares simultaneously, the set with maximal power (called a *characteristics*) is chosen. In [33] the problem of constructing triples of MODLS of order 10 was considered as a SAT problem. For each known pair (at that moment there were only 20 of them) a separate CNF was constructed by assigning values to Boolean variables corresponding to the elements of the pair. Using parallel SAT solvers the triple with the characteristics of 73 was found. Using new pairs of MODLS of order 10, which were described in this section earlier, it was possible to find two more triples with the characteristics of 73.

5 Related Work

There are several examples of applications of high-performance computing in order to search for combinatorial designs based on Latin squares. For example, the fact, that there is no finite projective plane of order 10, was proven on a computing cluster [15]. The hypothesis about the minimal number of clues in Sudoku was first proven on a computing cluster too [19]. Later, the volunteer computing project Sudoku@vtaiwan was used to confirm it [16].

The papers [3, 22, 23] report on the numbers of Latin squares of order up to 11. Some of that results were obtained with the help of parallel computing. Diagonal Latin squares of order at most 8 were enumerated on a personal computer [31], while for order 9 it was done in the volunteer computing project Gerasim@home [29].

In [20] the transversals for Latin squares of order at most 9 were enumerated. Their algorithm takes into account the fact, that the space of Latin squares can be divided into isotopy classes (115 618 721 533 classes for order 9). Transversals were enumerated for each representative, this allowed to calculate the number of transversals for each isotopy class. The present study doesn't employ isotopy classes, because it deals with diagonal Latin squares. Therefore, it is required to generate all possible species of diagonal Latin squares of the considered orders.

In the paper [8] the enumeration of mutually orthogonal Latin squares of order 9 was performed. In [21] it was determined that Latin squares of order 10 from several particular families cannot participate in a triple of MOLS of order 10. In [36] a triple of diagonal Latin squares of order 10, that is the closest to being a triple of MODLS found so far, was provided. It was obtained by constructing all orthogonal mates for diagonal Latin squares generated according to a specific scheme.

In the survey [37] it is described how combinatorial designs can be found by SAT solvers. In particular, the author of that survey tried to find a triple of MOLS of order 10 via SAT solvers. In [28] a tool for SAT-based systems was proposed. This tool can be used to construct pandiagonal Latin squares.

6 Conclusions and Future Work

In the present study transversals for diagonal Latin squares of order at most 8 were enumerated. Also, several dozens pairs of orthogonal diagonal Latin squares of order 10 were found. These experiments were partially performed in two volunteer computing projects. The obtained results show, that volunteer computing suits well for solving such problems. In the future, it is planned to enumerate transversals for diagonal Latin squares of order 9. Currently, in Gerasim@home an experiment is being conducted that is aimed at finding orthogonal DLSs. It uses algorithms for solving the exact cover problem to find transversals and orthogonal pairs. After 4 months of the experiment more than 50 000 unique pairs of orthogonal DLSs were found.

Acknowledgement: Stepan Kochemazov and Oleg Zaikin are funded by Russian Science Foundation (project No. 16-11-10046). Eduard Vatutin is partially supported by Council for Grants of the President of the Russian Federation (grant MK-9445.2016.8) and by Russian Foundation for Basic Research (grant 17-07-00317-a). Authors thank all Gerasim@home and SAT@home volunteers, whose computers took part in the experiments. Authors also thank Ilya Kurochkin and Nickolay Khrapov from IITP RAS for their help in maintaining SAT@home.

Authors contribution: Eduard Vatutin and Sergey Valyaev – enumeration of transversals for DLS of order up to 8 in Gerasim@home (Section 3); Oleg Zaikin and Stepan Kochemazov – search for MODLS of order 10 in SAT@home (Section 4).

References

[1] David P. Anderson and Gilles Fedak. The computational and storage potential of volunteer computing. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006), 16-19 May 2006, Singapore*, pages 73–80. IEEE Computer Society, 2006.

[2] F. Asnicar, L. Erculiani, F. Galante, C. Gallo, L. Masera, P. Morettin, N. Sella, S. Semeniuta, T. Tolio, G. Malacarne, K. Engelen, A. Argentini, V. Cavecchia, C. Moser, and E. Blanzieri. Discovering candidates for gene network expansion by distributed volunteer computing. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 3, pages 248–253, 2015.

[3] Stanley E. Bammel and Jerome Rothstein. The number of 9×9 Latin squares. *Discrete Math.*, 11(1):93–95, January 1975.

[4] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[5] J.W. Brown, F. Cherry, L. Most, E.T. Parker, and W.D. Wallis. Completion of the spectrum of orthogonal diagonal Latin squares. *Lecture notes in pure and applied mathematics*, 139:43–49, 1992.

[6] Christophe Cerin and Gilles Fedak. *Desktop Grid Computing*. Chapman & Hall/CRC, 1st edition, 2012.

[7] Charles J. Colbourn and Jeffrey H. Dinitz. *Handbook of Combinatorial Designs, Second Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2006.

[8] Judith Egan and Ian M. Wanless. Enumeration of MOLS of small order. *Math. Comput.*, 85(298):799–824, 2016.

[9] Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[10] Gerasim@home: a volunteer computing project for research in discrete mathematics and logic control, http://gerasim.boinc.ru/.

[11] Solomon W. Golomb and Leonard D. Baumert. Backtrack programming. *J. ACM*, 12(4):516–524, October 1965.

[12] E. Ivashko and A. Golovin. Partition algorithm for association rules mining in boinc-based enterprise desktop grid. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9251:268–272, 2015.

[13] N. Khrapov, V. Roizen, M. Posypkin, A. Samtsevich, and A. R. Oganov. Volunteer computing for computational materials design. *Lobachevskii Journal of Mathematics*, 38(5):926–930, Sep 2017.

[14] Ilya Kurochkin and Anatoliy Saevskiy. BOINC forks, issues and directions of development1. *Procedia Computer Science*, 101(Supplement C):369 – 378, 2016. 5th International Young Scientist Conference on Computational Science, YSC 2016, 26-28 October 2016, Krakow, Poland.

[15] C.W.H. Lam, L. Thiel, and S. Swierz. The nonexistence of finite projective planes of order 10. *Canad. J. Math.*, 41:1117–1123, 1989.

[16] Hung-Hsuan Lin and I-Chen Wu. Solving the minimum Sudoku problem. In *The 2010 International Conference on Technologies and Applications of Artificial Intelligence*, TAAI '10, pages 456–461, Washington, DC, USA, 2010. IEEE Computer Society.

[17] Inês Lynce and Joël Ouaknine. Sudoku as a SAT problem. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2006, Fort Lauderdale, Florida, USA, January 4-6, 2006*, 2006.

[18] João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Biere et al. [4], pages 131–

153.

[19] Gary McGuire, Bastian Tugemann, and Gilles Civario. There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem via hitting set enumeration. *Experimental Mathematics*, 23(2):190–217, 2014.

[20] Brendan D. McKay, Jeanette C. McLeod, and Ian M. Wanless. The number of transversals in a Latin square. *Designs, Codes and Cryptography*, 40(3):269–284, 2006.

[21] Brendan D. McKay, Alison Meynert, and Wendy Myrvold. Small Latin squares, quasigroups, and loops. *Journal of Combinatorial Designs*, 15(2):98–119, 2007.

[22] Brendan D. McKay and Eric Rogoyski. Latin squares of order 10. *Electr. J. Comb.*, 2(1):1–4, 1995.

[23] Brendan D. McKay and Ian M. Wanless. On the number of Latin squares. *Annals of Combinatorics*, 9(3):335–344, oct 2005.

[24] Natalia Nikitina, Evgeny Ivashko, and Andrei Tchernykh. Congestion game scheduling implementation for high-throughput virtual drug screening using boinc-based desktop grid. In Victor Malyshkin, editor, *Parallel Computing Technologies*, pages 480–491, Cham, 2017. Springer International Publishing.

[25] The on-line encyclopedia of integer sequences (OEIS), https://oeis.org/.

[26] SAT@home: a volunteer computing project aimed at solving hard SAT instances, http://sat.isa.ru/pdsat/.

[27] Alexander Semenov and Oleg Zaikin. Algorithm for finding partitionings of hard variants of boolean satisfiability problem with application to inversion of some cryptographic functions. *SpringerPlus*, 5(1):1–16, 2016.

[28] Takehide Soh, Naoyuki Tamura, and Mutsunori Banbara. Scarab: A rapid prototyping tool for sat-based constraint programming systems. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 429–436. Springer, 2013.

[29] Eduard Vatutin, Stepan Kochemazov, and Oleg Zaikin. Applying volunteer and parallel computing for enumerating diagonal latin squares of order 9. In *Proc. of The Eleventh International Conference on Parallel Computational Technologies*, volume 753 of *Communications in Computer and Information Science*, pages 110–124. Springer, 2017.

[30] Eduard Vatutin, Sergey Valyaev, and Vitaly Titov. Comparison of sequential methods for getting separations of parallel logic control algorithms using volunteer computing. In *Second International Conference BOINC-based High Performance Computing: Fundamental Research and Development (BOINC:FAST 2015), Petrozavodsk, Russia, September 14-18, 2015*, volume 1502 of *CEUR-WS*, pages 37–51, 2015.

[31] Eduard Vatutin, Oleg Zaikin, Alexey Zhuravlev, Maxim Manzyuk, Stepan Kochemazov, and Vitaly Titov. Using grid systems for enumerating combinatorial objects on example of diagonal Latin squares. In *Selected Papers of the 7th International Conference Distributed Computing and Grid-technologies in Science and Education, Dubna, Russia, July 4-9, 2016*, volume 1787 of *CEUR-WS*, pages 486–490, 2016.

[32] Ian Murray Wanless. *Transversals in latin squares: a survey*, pages 403–437. Cambridge University Press, United Kingdom, 2011.

[33] Oleg Zaikin and Stepan Kochemazov. The search for systems of diagonal Latin squares using the SAT@home project. *International Journal of Open Information Technologies*, 3(11):4–9, 2015.

[34] Oleg Zaikin, Maxim Manzyuk, Stepan Kochemazov, Igor Bychkov, and Alexander Semenov. A volunteer-computing-based grid architecture incorporating idle resources of computational clusters. In Ivan Dimov, István Faragó, and Lubin G. Vulkov, editors, *Numerical Analysis and Its Applications - 6th International Conference, NAA 2016, Lozenetz, Bulgaria, June 15-22, 2016, Revised Selected Papers*, volume 10187 of *Lecture Notes in Computer Science*, pages 769–776, 2016.

[35] Oleg Zaikin, Pavel Petrov, Mikhail Posypkin, Vadim Bulavintsev, and Ilya Kurochkin. Using volunteer computing for sound speed profile estimation in underwater acoustics. In *Third International Conference BOINC-based High Performance Computing: Fundamental Research and Development (BOINC:FAST 2017). Petrozavodsk, Russia, August 28 - September 1, 2017*, volume 1973 of *CEUR-WS*, pages 43–48, 2017.

[36] Oleg Zaikin, Alexey Zhuravlev, Stepan Kochemazov, and Eduard Vatutin. On the construction of triples of diagonal Latin squares of order 10. *Electronic Notes in Discrete Mathematics*, 54:307–312, 2016.

[37] Hantao Zhang. Combinatorial designs by SAT solvers. In Biere et al. [4], pages 533–568.