Shermin Sherkat, Lior Skoury, Andreas Wortmann,
Thomas Wortmann

Artificial Intelligence Automated Task Planning for Fabrication

Abstract: Sequencing tasks for robotic fabrication processes of buildings is not automated, contributing to its inefficiency and negative environmental impacts. Automated Task Planning (ATP) is an Artificial Intelligence (AI) method that sequences discrete tasks automatically to achieve a particular goal using a planner (AI planner). However, Construction and Fabrication (CF) has yet to exploit ATP techniques fully, partly because representing 3D geometry in ATP remains challenging. To address this problem, this paper (1) investigates the challenges of using an ATP technique (classical ATP) related to integrating 3D geometry in a fabrication process task planning, (2) models a fabrication domain using the Planning Domain Definition Language (PDDL) – a standard planning language, and (3) generates a plan for a limited version of a fabrication process with a generic AI planner. Thus, fabrication practitioners can get familiar with the advantages and limits of classical ATP technique in fabrication domains.

1 Introduction

Automated Task Planning (ATP) is an AI method that utilizes reasoning to organize sequences of tasks based on their anticipated outcome to achieve a particular goal [1]. This method involves (1) a model of the planning domain and problem created with a planning language, e. g., PDDL, and (2) a planner solver that takes the domain and problem model as input and searches for a task plan.

CF fields require task planning for processes involving labor allocation, equipment and material assignment, and robotic assembly [5]. However, the current methods for task planning in CF primarily rely on manual procedures or process-specific algorithms, resulting in limited efficiency and reusability [7].

Despite the need for automation, current literature on the application of ATP techniques in CF is limited. To our knowledge, only a few papers [8-13] have used ATP techniques in CF, of which only papers [9, 11-12] have leveraged planning languages and AI planners. Thus, ATP benefits in CF are yet to be explored.

ATP has different techniques, namely classical, numerical, and probabilistic planning [5], each defined with a suitable planning language and corresponding AI planners. However, planning languages primarily model abstract tasks and domains, which poses challenges when it comes to representing 3D geometry and its associated details.

https://doi.org/10.1515/9783111162683-019

Hence, the contribution of this paper is threefold:
- Highlighting challenges and potential use-cases of classical ATP techniques for defining a fabrication domain.
- Defining a fabrication planning domain and problem model using classical PDDL,
- Discussing how to model 3D geometry using classical ATP.

We conclude that classical planning techniques can generate task plans for a pick-and-place fabrication problem, enabling the stacking and assembly of individual objects while employing various end-effectors. Nevertheless, we cannot model stacking several objects on top of one because we need to include the number of objects. Additionally, we cannot stack one big object on top of several objects at the same level, as we must consider both the number and height of the objects or the ability to group objects based on their geometric characteristics.

2 Background

ATP techniques create task (action) sequences using a predefined domain and problem model. AI planner is the solver that inputs the domain and problem model, conducts reasoning, and generates a sequence of actions (plan) [2]. The domain model should be generic and reusable for similar problems. Conversely, a problem model varies for each problem and can be solved by various planners that support the modeling language [4].

PDDL is a formal knowledge representation language utilized to express planning models [3]. It is widely recognized as the standard planning language for specific planning problems, including classical planning because it enables the exploration of multiple generic AI planners, algorithms, and optimization techniques. Classical planning, an ATP technique used in discrete and deterministic problem environments such as fabrication [4, 6], will be briefly explained in the following, where we provide a concise overview of the domain and problem models using classical PDDL.

Domain

Main parts of a classical PDDL domain model are *predicates* and *actions* [4].

(1) *Predicates* describe the space and the relation between the objects in the space. For instance, the predicate "At(robot, A)" can indicate the robot's location at position A. Predicates can be true or false, and the ones that are not true are considered false [6].

(2) *Actions* change facts about the world and consist of parameters (variables), preconditions, and effects. For example, action Travel (robot, A, B) in Tab. 3 requires a robot and two locations as parameters to travel from one place to the other. Addi-

tionally, certain preconditions must be true for this action to occur, e. g., the robot being at the first location, denoted by the predicate "At(robot, A)". Then, if all the preconditions are met, the effects will be applied. In this case, the effects would be twofold: the robot is at the second position (B), so the predicate At (robot, B) will be added, and the predicate At (robot, A) will be negated.

Problem

In the problem file, we define an instance of problem using the predefined reusable domain. The main parts of a problem model are the initial and goal states. The initial state includes a conjunction of true predicates before the planning. In the previous example, the initial state describes the robot at position A: "At (robot, A)". The goal state is a conjunction of predicates describing what we want as our goal: "At (robot, B)". Hence, the planner's job is to search for a sequence of actions that would result in all the predicates in the goal state being true. The plan consists of one action in the example provided: Travel (robot, A, B) [6].

Related Works

To our knowledge, only papers [8-13] have utilized AI planning in construction or fabrication. Among these papers, [9, 12] have employed a planning modeling language similar to PDDL that works with a planner they are developing to address long-horizon planning. In the study conducted by Cheng and Hammad [11], a planning language called KQML (Knowledge Query and Manipulation Language) is used to plan the tasks of two cranes working together to lift a heavy object. Huang et al. [8] employed a flowchart for task planning that incorporates structural and geometrical data of a fabrication domain. Therefore, the potential of ATP techniques in CF remains largely unexplored.

Case Study

This paper utilizes a part of the BUGA pavilion fabrication process to test classical ATP. The BUGA Wood Pavilion, constructed on the summer island at the National Horticultural Show 2019 in Heilbronn, is a research demonstrator building. It features a digitally designed structure with a segmented hollow cassette form [17] (Fig. 1). The BUGA pavilion consists of modular parts called cassettes. The fabrication process for building cassettes involves picking objects from their initial position, placing them on a work table, and manipulating them using different end-effectors.

Fig. 1: The BUGA pavilion.

3 Methodology

This section provides an overview of various tasks and setups involved in the fabrication process. Then, it presents a model for the planning domain and formulates a corresponding problem, specifically focusing on describing geometry using classical PDDL.

The Fabrication Setup

The fabrication environment in this example is non-observable, deterministic, sequential, static, discrete, and single-agent [5, 6]. We create a task plan for one cassette, including a lower plate, a beam, and a top plate (Fig. 2).

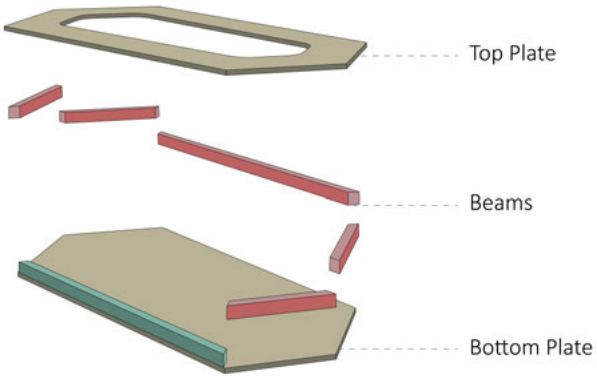

Fig. 2: The components of a cassette, including the lower plate, top plate, and beams.

This example needs three different end-effectors (tools):
- A vacuum gripper that picks the objects
- A glue gun that pours glue
- A nail gripper that puts nails into the pieces

The robot should pick up individual pieces, accurately stack them on a designated worktable, and join them using nails and glue. Once a complete cassette is assembled, the robot picks up the cassette and stores it on a separate table (Fig. 3 and 4).

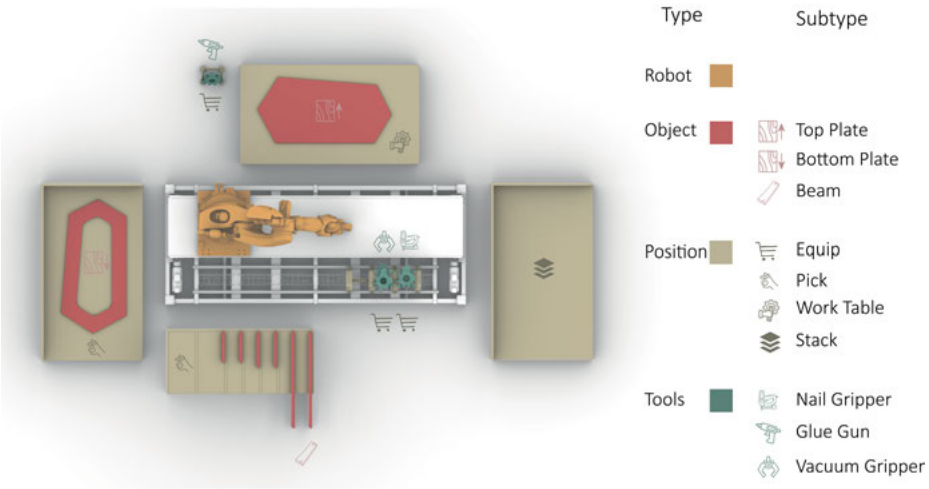

Fig. 3: The various types of objects and the general setup in the pick and place problem.

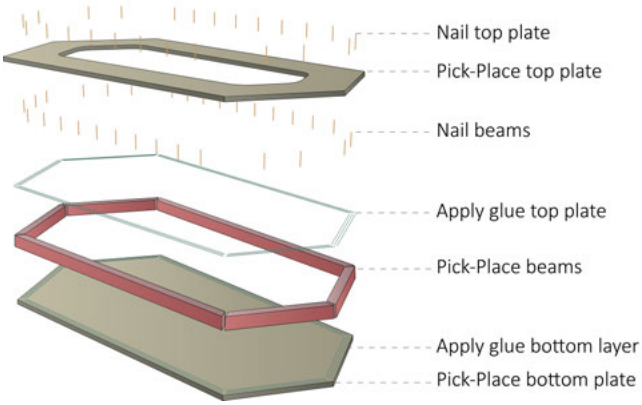

Fig. 4: A complete cassette.

Domain Model

The domain model includes types, predicates, and actions:

Types

Type feature lets us assign types and subtypes to the objects we define. Within the fabrication process domain, four main types are employed: *tools, positions, objects*, and *robots*. Additionally, subtypes are defined for these main types, as outlined below:

- Tools: Vacuum gripper, Nail Gripper, and glue gun
- Positions: EquipPosition, FirstPosition, WorkTablePosition, StackPosition
- Objects: module, LowPlate, TopPlate, Beams

Objects and *Positions* represent geometric information in this process. Notably, these positions and objects are discretized, indicating that they do not store numerical data such as dimensions. Instead, we define discrete positions with known coordinates, and the planner distinguishes between these positions based on their types and subtypes.

Predicates

Predicates are defined with a name, variables, and the type of each variable. For instance, in the predicate "At (?client -Robot ?p -positions)", "?client" is a variable of type Robot. As per the objects defined in the problem model, these variables are subsequently substituted with the corresponding objects. Table 1 presents the 15 predicates we defined.

Tab. 1: Description of predicates.

	Predicate	Description
1	(At ?Client -Robots ?pp -Positions)	a client (robot) is at a location
2	(AtPlace ?Object - Objects ?p -Positions)	an object is at a location
3	(AtTool ?tool -Tools ?ep -EquipPosition)	a tool is at a location
4	(OnTable ?Obj - Objects ?p -Positions)	an object is on a position
5	(TableFree ?t -Positions)	a location is not occupied
6	(Clear ?Obj - Objects)	no object is on top of this object
7	(OnTop ?Obj1 - Objects ?Obj2 - Objects)	an object is on top of another object
8	(Have ?Client -Robots ?Tool -Tools)	a robot is equipped with a tool
9	(Empty ?Client - Robots)	a robot does not have any tool mounted
10	(Active ?tool - Tools)	a tool is activated
11	(Holding ?Client - Robots ?Object - Object)	a robot is holding an object
12	(VG_Empty ?Client - Robots)	vacuum gripper is empty
13	(Finish ?module - Module)	a cassette is built
14	(Glued ?Obj - Objects)	an object is glued
15	(Nailed ?Obj - Objects)	an object is nailed

In Tab. 1, predicates one to four indicate the specific location of each object. These predicates tie a location to a single object. However, they do not give any information regarding other objects. For instance, if object A is not at location P, we do not know if any other object exists in that position. Hence, we need other predicates, such as the number 5, that give us more general information. The same logic goes for predicates six and seven. They are defined to check if one object is on top of another object. However, since classical task planning does not accept numbers as variables, we cannot know how many objects are stacked on top of an object and when the lower object can get the predicate "not (Clear ?obj - Objects)".

Actions are defined with parameters, preconditions, and effects. Table 2 displays the defined actions along with their corresponding parameters. The "Stack (obj1, obj2)" action necessitates the predicate "Clear (obj2)" as precondition and adds the effect "(not(clear obj2))". Hence, the planner cannot stack another object on top of obj2 since the precondition "Clear (obj2)" is negated. Additionally, Obj2 should have the predicate "(not(clear obj2))" only when all the required beams are in place. However, classical PDDL cannot model this information, and the same issue happens with the "unstack (obj1, obj2)" action.

Tab. 2: Description of actions.

	Action	Description
1	Travel (r?, p1?, p2?)	a robot travels from one location to another
2	Equip (r?, t?)	a robot takes a tool
3	DeEquip (r?, t?)	a robot unmounts a tool
4	InitializeTool (r?, t?)	a robot initializes a tool
5	CloseTool (r?, t?)	a tool is turned off
6	Pick (o?, p?, r?)	a robot picks an object (from a table)
7	Unstack (o?, o?, p?, r?)	a robot picks an object (unstack it from the top of another object)
8	Place (o?, p?, r?)	a robot puts an object on a table
9	Stack (o?, o?, p?, r?)	a robot puts an object on top of another object
10	Gluing (o?, p?, r?, gg?)	a robot glues an object with a gluing tool
11	Nailing (o?, p?, r?, ng?)	a robot nails an object with a nailing tool
12	Finish_Off_Cassette (o?, p?, r?, fm?)	Finishes off the assembly of a cassette

Problem model

A problem model consists of the objects, the initial and the goal state. We define the domain for creating one cassette. Thus, the following objects have been introduced (Tab. 4).

Tab. 3: Parameters, preconditions, and effects associated with the actions “Travel” and “Pick”.

Action	Parameters	Preconditions	Effects
Travel	?Client - Robots ?from -Positions ?to -Positions	(and (At ?Client ?from) (not (= ?from ?to))	(and (not (At ?Client ?from)) (At ?Client ?to))
Pick	?Obj - Objects ?P - Positions ?Client - Robots ?VG - VacuumGripper	(and (AtPlace ?obj ?P) (Clear ?Obj) (not(TableFree ?P)) (OnTable ?Obj ?p) ...)	(and (Holding ?Client ?Obj) (not (AtPlace ?obj ?P)) (not (Clear ?Obj)) (not(OnTable ?Obj ?p)) ...)

Tab. 4: Objects and their corresponding types in the problem model.

Main Type						
Objects		**Tools**		**Positions**		**Robot**
Name	Subtype	Name	Subtype	Name	Subtype	Name
LP1	LowPlate	VG	Vaccumegripper	FP1,FP2,FP3	FirstPosition	R1
B1	Beams	NG	NailGripper	P2	WorkTablePosition	
Tp1	TopPlate	GG	GlueGun	P3	StackPosition	
M1	module			E1,E2,E3	EquipPosition	

The initial state describes the scene before planning and includes predicates such as “OnTable (Tp1 FP3)” to indicate the object TP1 (top plate) is on FP3 (a first position). The goal state describes the desired outcome and is defined by four predicates (see Fig. 4):

(and
(AtPlace m1 P3)
(AtTool VG E1)
(AtTool NG E2)
(AtTool GG E3))

The predicates in the goal state guarantee that the robot unequips all end-effectors and completes the cassette assembly before relocating it to the final position.

Creating the plan

To create the plan, we utilize a generic planner called “FF-Planner”, which leverages the fast-forward algorithm to create the plan. The fast-forward algorithm is a heuristic

search algorithm that combines a relaxed planning graph and a goal regression technique to efficiently estimate the number of actions required to reach the goal state [2, 16]. Additionally, we use a PDDL VScode extension to define the domain and debug the plan [15]. The planner then successfully creates a plan for this problem, as described in the next section.

4 Results

This section discusses the results of applying classical ATP to a limited version of the BUGA pavilion fabrication process (Sec. 2). A plan is generated using the FF-Planner, a generic AI planner, based on the defined domain and problem models using classical PDDL. Next, we highlight the advantages and limitations of classical ATP for this fabrication process.

The domain model includes four main types (Tab. 4), 15 predicates (Tab. 1), and 12 actions (Tab. 2). The problem instance models one single cassette that uses one beam, as explained in the Methodology section (Sec. 3).

The planner inputs the domain and the problem models, and successfully creates a plan with 72 steps in 8.313 seconds1. The first 22 steps of the output plan are illustrated in Fig. 5.

```
0.00100: (travel r1 p2 e1)          0.01200: (equipe r1 gg e3)
0.00200: (equipe r1 vg e1)          0.01300: (travel r1 e3 p2)
0.00300: (initialize r1 vg)         0.01400: (initialize r1 gg)
0.00400: (travel r1 e1 fp1)         0.01500: (gluing lp1 p2 r1 gg)
0.00500: (grab lp1 fp1 r1 vg)       0.01600: (closetool r1 gg lp1)
0.00600: (travel r1 fp1 p2)         0.01700: (travel r1 p2 e3)
0.00700: (place lp1 p2 r1 vg)       0.01800: (deequip r1 gg e3)
0.00800: (closetool r1 vg lp1)      0.01900: (travel r1 e3 e1)
0.00900: (travel r1 p2 e1)          0.02000: (equipe r1 vg e1)
0.01000: (deequip r1 vg e1)         0.02100: (initialize r1 vg)
0.01100: (travel r1 e1 e3)          0.02200: (travel r1 e1 fp2)
```

Fig. 5: The first 22 tasks for assembling a cassette with one beam.

In this example, as we are using one cassette with only one beam, there is only one valid sequence of tasks and no optimization is required.

One of the main advantages of ATP is its ability to generate plans for similar planning processes using a reusable domain model. While it takes some time to create the domain model, generating the problem and adjusting the subtypes for other similar problems can be automated because the problem file is the only part that varies.

The domain we created enables planning task sequences for picking and placing single objects on top of each other and utilizes different end-effectors to operate on

these objects. It is crucial to define the domain as generally as possible and utilize the main object types for action parameters. This approach eliminates the need to modify the actions for different problems, requiring only adjustments to the subtypes. For instance, we might have other types, such as slabs instead of lower plates.

To create problem instances for different processes, we can categorize the objects based on the main types (objects, tools, and positions) and translate them to PDDL using a Grasshopper component. This component takes the fabrication geometry, assigns them the appropriate type, and generates the initial state and goal state.

However, we only test the domain using one beam because classical planning reaches its limits when stacking multiple objects on top of a larger object. For example, when stacking a beam on a lower plate, the Stack action adds the effect "(not (clear LP1))" and requires the lower plate to have the "(clear LP1)" predicate as a precondition. Therefore, it is not possible to stack the second beam on top of LP1 as the precondition "(not (clear LP1))" already exists. Additionally, when placing the top plate on top of multiple beams, classical planning cannot assign the "(not (clear o?))" precondition as an effect to all the beams underneath.

5 Discussion and Conclusion

Automating task planning in CF is necessary because manual task planning is inefficient and environmentally unfriendly [5]. Therefore, it is important to explore ATP techniques in CF. However, the challenge lies in modeling the fabrication domain, which heavily relies on 3D geometry, using a planning language to leverage ATP potentials in CF. This challenge should be clarified, as few papers have investigated ATP techniques in CF.

This paper (1) highlights classical ATP features and how to integrate 3D geometry for fabrication task planning, along with its limitations, (2) creates a classical PDDL domain for a limited version of the fabrication process of the BUGA pavilion, and (3) tests the domain using an AI planner called FF-planner.

The results show a task plan created to fabricate a single cassette that uses one beam in a static, deterministic, and observable environment. We define the domain using 12 actions, 15 predicates, and four main types. The FF-Planner then creates a plan in 72 steps in 8.313 seconds.

To model geometry using classical PDDL, it is necessary to discretize the values, which means continuous variables such as dimensions or coordinates cannot be included. Additionally, the geometric relations between objects must be represented using predicates that can have a true or false value. We conclude that classical planning using PDDL for a fabrication domain is suitable for (1) stacking single objects on top of each other (e. g., placing one beam on top of one lower plate) and (2) using different tools to assemble and operate on the pieces.

However, classical planning is unsuitable for (1) stacking multiple objects (beams) on top of a single object, as it requires keeping track of the number of top objects, and (2) stacking one large object on top of multiple objects simultaneously as it requires keeping track of the number of lower objects and the ability to group them to assign a predicate to all of them using a single action.

Discretizing geometry is not necessarily a downside, as the environment in fabrication domains is usually static and deterministic, and replanning is unnecessary. Additionally, action models must be generic to ensure reusability across different processes. In this paper, the action "Finish_Off_Cassette" is not generic, as it creates a new object when all the cassette parts are assembled. However, a function can be developed to generate and replace this action for different modules other than cassettes, based on their assembly requirements. Similarly, it is possible to create a domain with as many actions (pick, place, stack, unstack) as the number of beams and add the "Clear(o?)" predicate for the lower plate when the last beam is placed. This method can solve the issue of stacking more than one object on top of one but only works if we generate all these actions for each unique problem.

PDDL 1.2 and ADL extension should be explored in future steps as the former can include numbers, and the latter allows adding conditionals which might solve the grouping issue. Additionally, while many actions can be the same for similar fabrication processes like "nailing" and "gluing", some actions could still be generalized more. For instance, actions pick and place only recognize one vacuum gripper tool. These actions must be adjusted if we require two different types of vacuum grippers for picking different objects.

Acknowledgments

This research was funded by the Ministry of Science, Research and the Arts Baden-Wuerttemberg in the Artificial Intelligence Software Academy (AISA). This research is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy -EXC 2120/1-390831618.

References

[1] Ghallab, M., D. S. Nau, and P. Traverso. *Automated Planning: Theory and Practice.* Amsterdam; Boston: Elsevier/Morgan Kaufmann, 2004.

[2] Geffner, H., and B. Bonet. A Concise Introduction to Models and Methods for Automated Planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 22. San Rafael: Morgan & Claypool Publishers, 2013.

[3] Garrett, C. R., R. Chitnis, R. Holladay, B. Kim, T. Silver, L. Pack Kaelbling, and T. Lozano-Pérez. Integrated Task and Motion Planning. *Annual Review of Control, Robotics, and Autonomous Systems,* 2021.

[4] Haslum, P., ed. An Introduction to the Planning Domain Definition Language. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 42. Morgan & Claypool Publishers, 2019.
[5] Sherkat, S., A. Wortmann, T. Wortmann, Potentials of Symbolic AI Planning for Construction.33rd Forum Building Informatics, Technische Universität München, 2022.
[6] Russell, S. J., P. Norvig, and E. Davis. *Artificial Intelligence: A Modern Approach.* 3rd ed. Prentice Hall Series in Artificial Intelligence. Upper Saddle River: Prentice Hall, 2010.
[7] Abioye, S. O., L. O. Oyedele, L. Akanbi, A. Ajayi, J. M. Davila Delgado, M. Bilal, O. O. Akinade, and A. Ahmed. Artificial Intelligence in the Construction Industry: A Review of Present Status, Opportunities and Future Challenges. *Journal of Building Engineering,* 2021
[8] Huang, Y., P. Y. V. Leung, C. Garrett, F. Gramazio, M. Kohler, and C. Mueller. The New Analog: A Protocol for Linking Design and Construction Intent with Algorithmic Planning for Robotic Assembly of Complex Structures. In *Symposium on Computational Fabrication,* 1–17. Virtual Event USA: ACM, 2021.
[9] Hartmann, V. N., O. S. Oguz, D. Driess, M. Toussaint, and A. Menges. Robust Task and Motion Planning for Long-Horizon Architectural Construction Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* Las Vegas, USA: IEEE, 2020.
[10] Pradhan, A., and B. Akinci. Planning-Based Approach for Fusing Data from Multiple Sources for Construction Productivity Monitoring. *Journal of Computing in Civil Engineering,* 2012.
[11] Cheng Z., and A. Hammad. Agent-Based Simulation for Collaborative Cranes. In *Winter Simulation Conference,* 2051–56. Washington, DC, USA: IEEE, 2007.
[12] Hartmann, V. N., A. Orthey, D. Driess, O. S. Oguz, and M. Toussaint. Long-Horizon Multi-Robot Rearrangement Planning for Construction Assembly. *IEEE Transactions on Robotics* 39, 2023
[13] Hu, W. Automatic Construction Process of Prefabricated Buildings on Geometric Reasoning. In *Construction Research Congress* 2005, 1–10. San Diego, California, United States: American Society of Civil Engineers, 2005.
[14] Skoury, L., F. Amtsberg, X. Yang, H. J. Wagner, A. Menges, and T. Wortmann. A Framework for Managing Data in Multi-Actor Fabrication Processes. In *Towards Radical Regeneration,* ed. C. Gengnagel, O. Baverel, G. Betti, M. Popescu, M. Ramsgaard Thomsen, and J. Wurm, 2023.
[15] Dolejsi, J., D. Long, M. Fox, G. Besançon, PDDL Authoring and Validation Environment for Building end-to-end Planning Solutions, ICAPS18, 2018.
[16] Hoffmann, J. FF: The Fast-Forward Planning System. *AI Magazine* 22, 2001.
[17] Wagner, H. J., M. Alvarez, A. Groenewolt, and A. Menges. Towards Digital Automation Flexibility in Large-Scale Timber Construction: Integrative Robotic Prefabrication and Co-Design of the BUGA Wood Pavilion. *Construction Robotics* 4, 2020.