Reinforcement Learning for Sequential Assembly of SL-Blocks

Self-interlocking combinatorial design based on Machine Learning

Bastian Wibranek[1], Yuxi Liu[2], Niklas Funk[3], Boris Belousov[4], Jan Peters[5], Oliver Tessmann[6]
[1]The University of Texas at San Antonio [2,6]Digital Design Unit, TU Darmstadt
[3,4,5]Intelligent Autonomous Systems, TU Darmstadt
[1]bastian.wibranek@utsa.edu [2]yuxi.liu@stud.tu-darmstadt.de
[3,4,5]{niklas|boris|jan}@robot-learning.de [6]tessmann@dg.tu-darmstadt.de

Adaptive reconfigurable structures are seen as the next big step in the evolution of architecture. However, to achieve this vision, new tools are required that enable autonomous configuration of given elements based on a specified design objective. Various approaches have been considered in the past, ranging from rule-based methods to evolutionary optimization. Although successful in applications where search heuristics or informative objective functions can be provided, these methods struggle with long-term planning problems. In this paper, we tackle the problem of sequential assembly of SL-blocks which has the character of a combinatorial optimization problem. We explore the applicability of deep reinforcement learning algorithms that recently showed great success on combinatorial problems in other domains, such as board games and molecular design. We highlight the unique challenges presented by the architectural design setting and compare the performance to evolutionary computation and heuristic search baselines.

Keywords: *Reinforcement Learning, Architectural Assembly, Discrete Design, SL-blocks, Dry Joined*

INTRODUCTION

Reversible building elements offer unique opportunities for the construction industry to reduce its carbon footprint, minimize waste and decrease the use of non-renewable resources. Among these advantages, the use of dry-fitted connections between elements allows for fast assembly and disassembly of structures out of prefabricated modules, thereby unlocking the path towards a circular economy in the construction industry (Durmisevic, 2019).

Today, most reversible connections rely on fixation via screws or other adhesives. On the other hand, topological interlocking (Tessmann and Rossi, 2019) provides an alternative approach to construction, in which stability and mechanical fixation are achieved purely based on the arrangement and connectivity of

the building elements.

Construction elements are often fit for a limited set of arrangements and cannot be easily reused. To enable greater flexibility and component reuse for future constructions, concepts such as Combinatorial Design (Sanchez, 2016), Digital Materials (Gershenfeld, et al. 2015), and Programmable Matter (Tibbits and Cheung, 2012) are challenging the current use of preassigned building components, instead putting the emphasis on reusability and recombination of large amounts of repetitive elements. In this paper, we focus on the SL-block proposed by Shih (2016) as the basis for exploring combinatorial design possibilities. SL-blocks enable flexible design while at the same time providing the benefits of self-interlocking assembly and straightforward reassembly (Figure 1).

Although repetitive components that can be connected in many different ways are advantageous for reversible assembly, it introduces new challenges to the design process. The problem of finding an arrangement of interlocking elements in which they assemble into a stable structure is complicated for a human designer. Its complexity grows exponentially with the length of the assembly sequence. Therefore, advanced optimization approaches are required to tackle this sequential combinatorial problem.

Several automated approaches to optimization of interlocking assemblies have been considered in the past. The use of graph representations and heuristic-based search methods were investigated by Wang et al., (2018). However, classical graph search methods are restricted by the branching factor of the problem and quickly reach their limits when faced with large-scale designs as required by architectural applications. Therefore, we are interested in considering machine learning methods that have recently shown success in other challenging combinatorial optimization domains, such as games of Chess and Go (Silver et al., 2017). In general, there is a significant amount of work on applying machine learning to combinatorial problems, as demonstrated in the recent survey by Bengio et al., (2020). In the context of construction, the first steps have been made by Bapst et al., (2019), who applied reinforcement learning to sequential assembly problems, albeit in a 2D setting.

In this paper, reinforcement learning is utilized for the sequential interlocking of dry joined blocks. Different algorithms were tested for rebuilding various curves that serve as a design input. The previous tedious combinatorial task is algorithmically optimized to design complex interlocking structures with modular building blocks.

MODULAR DESIGN AND ASSEMBLY WITH SL-BLOCKS

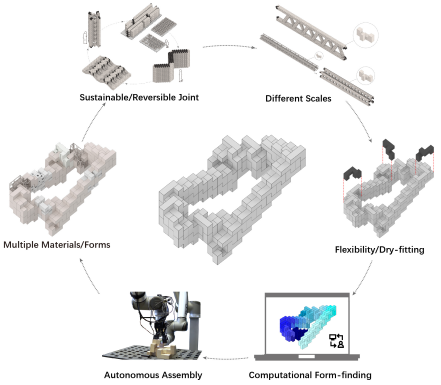


Figure 1
Advantages of SL-block in modular design.

The SL-block is an octocube consisting of one S-shaped and one L-shaped tetracube attaching along their sides (Shih 2016). The specific shape of the SL-block provides the most extensive space of possible combinations compared to other octa cubes (Chou, 2020). The basic relationship of two attaching SL-blocks is called engagement. There are six types of engagements defined as geometric transformations between the host pair and the new pair that is to be added. An SL-Strand is represented as a sequential concatenation of engagements, which specify the assembly process starting with an initial SL-block and adding blocks one by one. (Shih et al. 2019). For ex-

ample, the string aaaa specifies a string of four engagements lining up to form the configuration presented in Figure 2.

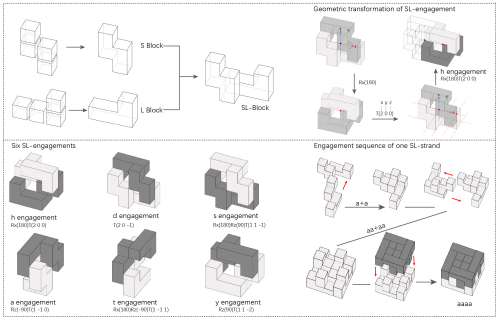


Figure 2
The arrangement of the S and L blocks and their various engagements.

With multiple SL-strands, various interlocking building elements (e.g., column, wall, and slab) can be configured, where only blocks highlighted in black can be removed without breaking the whole structure (Figure 3). Currently, these elements are created by manually defining the engagements, which is a very tedious task.

When designing and constructing with SL-blocks, one significant difficulty is to draw a connection between the discrete engagements and their effect on the overall structure. Since the geometric transformations are defined relative to the previously placed block, from a global point of view, selecting the same engagement may produce a different effect depending on the previous actions.

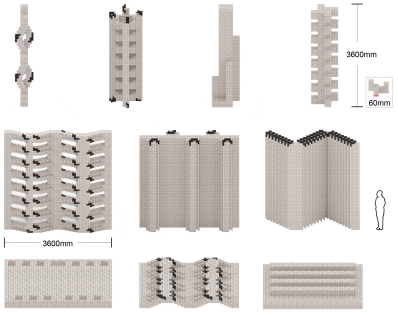


Figure 3
Different building elements from SL-blocks.

To circumvent this complexity, we propose to use curves as an interface to indicate the desired shape of the structure that needs to be assembled with SL-blocks. Parametrized curves take a leading role in design and architecture, being one of the main and delimiting elements of the form. Most iconically, Greg Lynn proposed to use curves for computer-aided design and has recently been transferred into the Digital and Discrete domain (Retsin, 2016).

In order to create an internal connection between the curve and the SL-blocks, our approach needs to transform the given curve into a sequential assembly sequence which is represented by a string of letters where each letter denotes the respective engagement. However, without the help of algorithms, this string generation process must be explored manually. Thus, in this paper, we focus on using different algorithms that can simplify the design process. In particular, we propose reinforcement learning as a promising approach for solving sequential assembly problems that provide a natural interface between the designer and modular building structures.

OPTIMIZATION ALGORITHMS FOR SEQUENTIAL ASSEMBLY

The problem of sequential assembly with SL-blocks falls into the scope of performance-driven architectural design (Shi & Yang, 2013). This approach aims to use numerical optimization to generate a structure based on provided design criteria. There are already several methods available to the community through Grasshopper plugins and external programs, such as evolutionary algorithms, direct search, and model-based methods, as surveyed by Wortmann & Nannicini (2017). However, these approaches do not take into account the sequential nature of the assembly problem. They treat the whole assembly sequence as one evaluation point and compute the fitness of the resulting structure only at the end of the assembly process.

In contrast, reinforcement learning can be viewed as an alternative to black-box optimization,

where the agent makes decisions at every step of the assembly and thereby uses the fitness signal (reward function) more efficiently. The advantages of reinforcement learning have been recently demonstrated on extremely challenging combinatorial optimization problems, such as games of Chess, Shogi, and Go (Silver et al., 2017; Schrittwieser et al., 2020). Therefore, we aim to transfer the insights from the domain of reinforcement learning to architectural design since our problem has a similar combinatorial structure.

SEQUENTIAL ASSEMBLY AS A REINFORCEMENT LEARNING PROBLEM

We want to solve the task of realizing a user-defined curve. That is, given a curve, we want to find a sequence of engagements resulting in a chain of SL-blocks that follows the curve. The variations of this task differ in the complexity of the target curve. To formalize this problem in the language of reinforcement learning, we need to define action, observation, and reward.

In a reinforcement learning loop (Figure 4), the agent (Python script) takes an action (put the next block) and the environment (Grasshopper) returns an observation (e.g., positions of parts relative to the curve) and a reward (numerical value reflecting how well the curve is covered by SL-blocks). The agent keeps improving its actions based on observations and rewards with the objective to maximize the total sum of rewards in expectation.

Action in our case is an integer between zero and five, indicating which of the six possible engagements of SL-blocks should be executed next. Note that the action is relative to the previously placed block. Therefore, we need to keep track of the current state of the environment. Conceptually, the state is represented by the poses of all already placed parts and the target curve.

Our observation consists of three parts: i) list of normalized values that encode the distance from each of the 16 voxels of the current SL-Engagement to the curve, ii) flattened transformation matrix describing the pose of the current SL-engagement, and iii) coordinates of three points on the curve in front of the agent. Figure 5 provides a visual illustration of how the observation is constructed. It is essential to provide some local information about the future direction of the curve; therefore, we add three leading points to the observation. Adding more future points to the observation makes the problem of value estimation easier. However, at the same time, it increases the dimensionality of the observation space and causes the agent to overfit to the current curve. On the other hand, providing only one point makes the agent short-sighted, precluding it from taking into account the direction of the curve or obstacles a few steps ahead. In our experiments, we found that providing three points is sufficient to solve the considered tasks. In other scenarios, this parameter may be adjusted for improved performance.

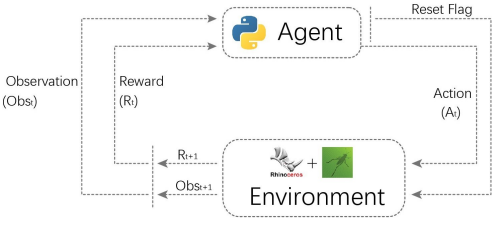


Figure 4
Reinforcement Learning Loop with Rhino/Grasshopper as Environment.

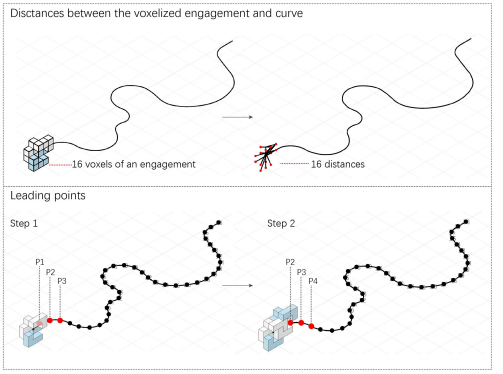


Figure 5
Two major parts of the observation: distances from each of the 16 voxels of the current engagement to the curve (top) and coordinates of 3 leading points (bottom). This additional information allows the agent to anticipate how the curve will evolve in the future.

The reward function and the reset flag are used to encourage the agent to follow the desired curve. In particular, the reward (defined as one minus the smallest

distance of the current block to the curve) is maximized when the structure perfectly follows the curve. If the agent deviates too much from the curve, the environment is reset to the initial state, which stops the accumulation of rewards and forces the agent to start anew.

INTERFACE BETWEEN GRASSHOPPER AND REINFORCEMENT LEARNING ALGORITHMS

To solve the above-defined reinforcement learning problem, we employ the stable-baselines3 library (Raffin et al. 2019), which is a de-facto standard for benchmarks in the RL community. We implemented a socket-based interface to use Grasshopper as an RL environment in combination with stable-baselines3, and we made it available for other researchers at https:github.com/b4be1/gh_gym. Among the available algorithms, we chose the two most suitable for our setting and that are known to work well for a wide range of hyperparameters: proximal policy optimization (PPO) and deep Q-learning (DQN). These algorithms represent two major classes of methods: actor-critic and value-based methods. Actor-critic methods learn a policy (actor) and a value function (critic), whereas value-based methods only learn the critic. The Python scripts and the Grasshopper files for running reinforcement learning on sequential assembly of SL-blocks are available at https:github.com/b4be1/rl-for-sequential-assembly. A video presenting our complete approach and describing how to use the interface can be found at https:youtu.be/owHATVWgNk4.

BASELINE: GREEDY SEARCH WITH A HEURISTIC

An alternative approach to optimizing a sequence of SL-Engagements is the heuristic-based search. Here the user provides a search heuristic that guides the agent at every step. The heuristic can be seen as the value function of the reinforcement learning agent, but instead of being learned through interaction with the environment, it is provided by the user.

In our curve following task, it is possible to provide a sufficiently good heuristic. Namely, the distance from the current engagement to a point slightly down the curve provides a strong signal, pulling the agent as a magnet towards placing the next block along the curve. We report the results of running this method as another baseline for evaluating the performance of reinforcement learning. We refer to this method as the greedy algorithm, as it greedily chooses the best action based on the heuristic's value. In this way, the greedy algorithm builds a single solution consisting of a sequence of SL-Engagements that follow the provided curve.

Related Work on Reinforcement Learning for Combinatorial Optimization in Construction

Performance-driven architectural design is a broad field (Shi & Yang, 2013). However, so far, mainly evolutionary algorithms have been explored (Wortmann & Nannicini, 2017; Fang, 2017; Costa et al., 2015). Only recently reinforcement learning started gaining interest in architecture for design (Mandow et al., 2020) and physical construction (Apolinarska et al., 2021). Our approach is conceptually related to that of Mandow et al. (2020) in that we use RL for design optimization; however, they use RL to evaluate designs generated by learned shape grammar, whereas we use RL to generate designs themselves.

Since the problem of sequential assembly belongs to the class of combinatorial optimization problems, research on methods for solving such problems, in general, is highly relevant. In recent years, there has been an increasing interest in applying deep learning techniques to combinatorial optimization problems (Bello et al. 2016, Khalil et al. 2017) as well as in using intelligent algorithms for construction and assembly tasks (Bapst et al. 2019, Thompson et al. 2020, Cho et al. 2020). However, most of the previous works on intelligent construction focused on automating the entire process end to end, i.e., without any means for the user to influence the design process (Bapst et al. 2019, Thompson et al. 2020). While this might result in an over-

all optimal design, it also clearly limits the interactivity of the approaches. Moreover, in those setups, the design objectives have to be defined rather passively through the reward function. On the contrary, the approach presented here allows the user to directly inform the algorithm by specifying the curve which needs to be built. This approach is more closely related to the setting presented by Janner et al. (2019), in which sequential assembly sequences were generated from image inputs.

RESULTS

This section presents evaluations and comparisons of PPO and DQN to evolutionary optimization and a greedy heuristic search algorithm on the curve following tasks with curves of varying degrees of complexity. Furthermore, evaluation on a task with obstacles is presented to demonstrate the adaptability of the reinforcement learning approach to constraints and additional objectives. Finally, the generated designs are validated in terms of stability and constructability by building a physical model using dry-fitted self-interlocking SL-blocks. The experiments were run on a laptop with Intel-Core-i7-4710HQ-Processor and 16GB-Memory.

EVALUATION OF REINFORCEMENT LEARNING ON THE CURVE FOLLOWING TASK

We compare two reinforcement learning algorithms, PPO and DQN, on the task of sequential assembly along a given curve. Three types of curves are considered: open 2D curve, open 3D curve, and closed 3D curve. For each curve type, we evaluate on two curve lengths: short (requires about 20 SL-Engagements) and long (requires about 40 SL-Engagements). Evaluation takes between 10-30 minutes depending on the difficulty of the curve. Table 1 shows representative samples obtained with PPO and DQN, as well as by the baseline algorithms - the genetic algorithm and the greedy algorithm.

While the genetic algorithm converges faster than reinforcement learning, in 5-10 minutes, it finds a reasonably good solution on 2D curves but fails on open and closed 3D curves (Figure 6). The reason for difficulties with 3D curves must lie in the higher complexity of the problem in 3D space because many more possibilities for placing SL-blocks are available. The algorithm cannot handle such an exponential increase in combinations.

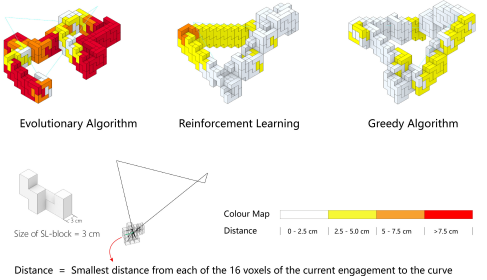
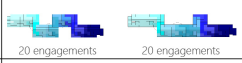
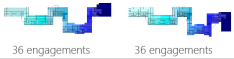
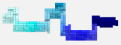
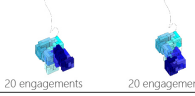
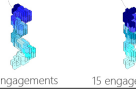
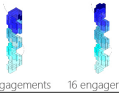
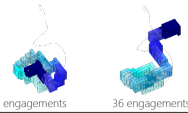
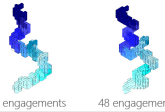
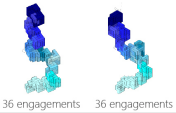
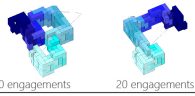
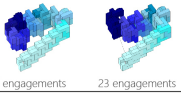
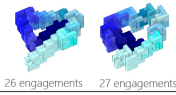
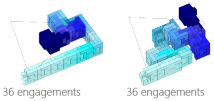
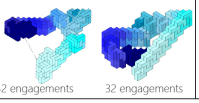
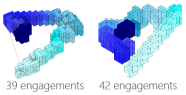


Figure 6
Colormap of the errors during the placement of SL-blocks occurring with the different algorithms on the 3D closed curve. The distances are based on a 3cm voxel-sized SL-block.

Compared to the evolutionary algorithm, the reinforcement learning algorithms succeed in finding sequential interlocking assemblies on curves of different lengths and dimensions, as shown in the middle columns of Table 1. Both presented algorithms, DQN and PPO, obtain similar solutions. The quality of the generated assembly sequences slightly degrades for the most complex problem instances, i.e., the closed 3D curves. This may be caused by insufficient training time since it was limited by 30 minutes to keep the interface interactive, as longer waiting times may be inconvenient to users. Increasing the training duration can improve the results on those complex problem instances. Currently, we only consider the standard RL setting where an agent is evaluated on the same task on which it is trained, i.e., we train an RL agent and evaluate it on the same curve. Ideally, we would like the agent to be able to generalize to novel curves without full retraining. The algorithms capable of such generalization are currently being researched in the fields of multitask and transfer learning, as well as meta-reinforcement-learning. We plan to investigate them in future work.

The rightmost column in Table 1 shows the results obtained by the greedy algorithm. As described earlier, this algorithm exploits an approximately opti-

Table 1
Evaluation results for two reinforcement learning algorithms, PPO and DQN, on the task of sequential assembly along curves. RL is compared to a genetic algorithm and a greedy optimization method with a manually provided heuristic.

	Evolutionary Algorithm	Reinforcement Learning		Greedy Algorithm
Curves	Results_Genetic Algorithm	Results_PPO	Results_DQN	Results
2D short open curve	 20 engagements 20 engagements	 21 engagements 21 engagements	 24 engagements 25 engagements	 22 engagements
2D long open curve	 36 engagements 36 engagements	 35 engagements 30 engagements	 42 engagements 41 engagements	 38 engagements
3D short open curve	 20 engagements 20 engagements	 18 engagements 15 engagements	 18 engagements 16 engagements	 21 engagements
3D long open curve	 36 engagements 36 engagements	 33 engagements 48 engagements	 36 engagements 36 engagements	 37 engagements
3D short closed curve	 20 engagements 20 engagements	 25 engagements 23 engagements	 26 engagements 27 engagements	 22 engagements
3D long closed curve	 36 engagements 36 engagements	 32 engagements 32 engagements	 39 engagements 42 engagements	 38 engagements

mal heuristic for choosing the next action, and therefore its results may be considered as the target for evolutionary and reinforcement learning algorithms. Since the heuristic is provided, the algorithm runs very fast, finding a solution in under 2 minutes. In all cases, the algorithm finds assembly sequences that closely reproduce the specified curves.

ADAPTABILITY OF REINFORCEMENT LEARNING ON A TASK WITH OBSTACLES

An important advantage of reinforcement learning compared to the greedy optimization approach with heuristics is that RL algorithms are learning the value function autonomously. In contrast, for the greedy optimization a human designer needs to specify the heuristic. Although it is straightforward to provide a reasonable heuristic in the simple curve-following task, we are ultimately interested in more abstract tasks, such as optimizing a structure based on generic design criteria, and in that case, finding a robust heuristic may not be possible.

Figure 7 shows a straightforward modification of the curve following task by introducing obstacles. As illustrated, the greedy algorithm is not able to adapt to the task change, whereas RL finds a strategy that is able to solve the problem. In this scenario, following the rigid and predefined heuristic results in a suboptimal solution. To improve the result obtained by the greedy algorithm, the user would be required to manually adapt the heuristic to the novel scenario, which might be a cumbersome process. In contrast, reinforcement learning needs no further adaptations, and the exact same algorithm can be applied to the modified problem.

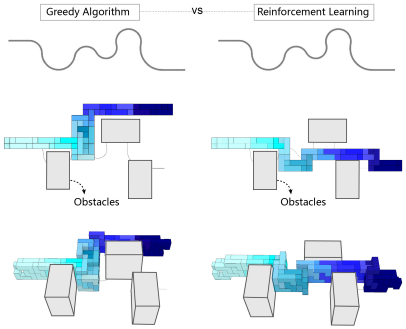


Figure 7
Comparing the assemblies generated by the greedy algorithm and DQN in the setting where three obstacles are added along the curve.

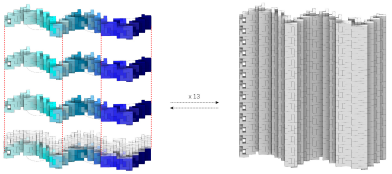


Figure 8
Exploiting the results from the RL algorithms on the 2D curve following the problem to obtain a wall made up of SL-blocks.

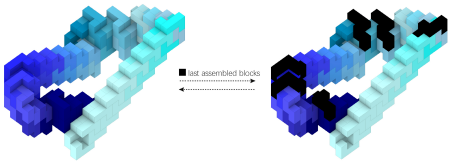


Figure 9
A color gradient shows the assembly sequence of a complex cantilevering structure from light to dark blue (left), and the key pieces for final fixation that are placed last indicated in black (right).

ARCHITECTURAL VERIFICATION OF THE GENERATED DESIGNS

The proposed interface where an architect specifies a curve and an algorithm finds an assembly sequence can be utilized in different ways as a creative tool in the design process. Here we provide an example of stacking the curves to generate a wall, as shown in Figure 8.

Furthermore, the designs generated by reinforcement learning for closed 3D curves can be used to fabricate and assemble complex cantilevering structures (Figures 9 and 10).

Thus, we confirm that the assembly sequence generated by the considered algorithm is feasible and leads to a physically constructible and stable structure. Therefore, reinforcement learning can support discrete form-finding and assembly based on the design preferences indicated by the architect.

CONCLUSION

Sequential assembly is a challenging combinatorial optimization problem. This paper has explored the applications of deep reinforcement learning to the automated design of open and closed curves based on interlocking dry-fitted SL-blocks. Two types of reinforcement learning methods were evaluated: actor-critic proximal policy optimization and value-based Q-learning. We described how an architectural problem can be translated into an RL problem and how the optimization can be performed using standard open-source Python packages of prominent RL algorithms.

Our results have demonstrated the potential of reinforcement learning, showing that it is able to generate significantly better designs on 3D curves compared to evolutionary algorithms, and we provided a real-world assembly based on the generated design to highlight its stability. For the curve following task considered in this paper, we designed an approximately optimal search heuristic that allowed us to compare the RL results against the greedy search approach. A notable observation here is that reinforcement learning achieves comparable results but

takes a significant amount of time to learn the value function on the order of tens of minutes. The key reason for this is the high number of trial-and-error attempts that RL algorithms require and the fact that Grasshopper is not optimized for such use and is therefore slowing down the pipeline.

We consider the results encouraging and in the future plan to explore the ways to speed up the simulation procedure in Grasshopper and consider more abstract optimization criteria, such as filling a 3D silhouette of a desired shape with SL-blocks, or optimizing based on architectural metrics such as the amount of lighting in an area or energy efficiency of construction.

Future research will have to shed light on the fabrication aspects for producing meaningful architectural structures (Figure 11). These investigations should include explorations of various materialities and fabrication techniques. The structural behavior and analysis of such assemblies should be examined for various load cases. The vast amount of assembly steps and the complex combinatorics are currently under investigation to be automated with robots.

RL integration for intricate design with SL-blocks makes it possible to design a dry-assembled structure that can be easily disassembled after usage. The adaptation of machine learning algorithms results in an informed design tool to obtain the assembly sequence of elements. The focus on reversible building elements that are self-interlocking requires architects to rethink certain design aspects without hindering their creative explorations. The presented approach is the first step in embedding the assembly procedure and charging it with various criteria to design with the help of machine intelligence.

ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 640554 and from the Forum for Interdisciplinary Research at TU Darmstadt for the project Multimaterial Modular Design Through Robot Learning and Tactile Sensing. The authors acknowledge the support from the Artificial Intelligence in Construction (AICO) grant by Nexplore/Hochtief Collaboration Lab at TU Darmstadt.

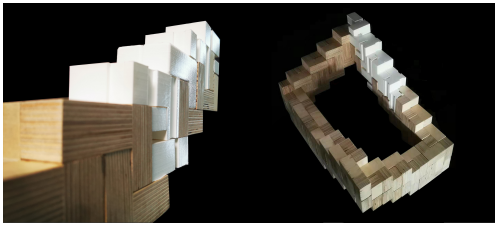


Figure 10
Cantilevering model made from veneer plywood and 3D-printed SL-blocks.

Figure 11
Proposal for a bridge design made from 8.000 SL-blocks. The blocks in contact with soil could be cast in concrete and the top part fabricated from plywood.

REFERENCES

Apolinarska, AA, Pacher, M, Li, H, Cote, N, Pastrana, R, Gramazio, F and Kohler, M 2021, 'Robotic assembly of timber joints using reinforcement learning', *Automation in Construction*, 125, p. 103569

Bapst, V, Sanchez-Gonzalez, A, Doersch, C, Stachenfeld, K, Kohli, P, Battaglia, P and Hamrick, J 2019 'Structured agents for physical construction', *International Conference on Machine Learning*, pp. 464-474

Battaglia, PW, et al. 2018, 'Relational inductive biases, deep learning, and graph networks', *arXiv preprint arXiv:1806.01261*, 1806, pp. 1-40

Bello, I, Pham, H, Le, QV, Norouzi, M and Bengio, S 2016, 'Neural combinatorial optimization with reinforcement learning', *arXiv preprint arXiv:1611.09940*, 1611, pp. 1-15

Bengio, Y, Lodi, A and Prouvost, A 2020, 'Machine learning for combinatorial optimization: a methodological tour d'horizon', *European Journal of Operational Research*, 290, pp. 405-421

CHO, M, JUNGTAEK, K, HYUNSOO, C, LEE, JH and PARK, J 2020 'Combinatorial 3D Shape Generation via Sequential Assembly', *Advances in Neural Information Processing Systems*

Chou, LW 2019, *The Study on SL-Blocks*, National Chiayi University

Costa, A, Nannicini, G, Schroepfer, T and Wortmann, T 2015, 'Black-box optimization of lighting simulation in architectural design', in Cardin, MA, Krob, D, Lui, PC, Tan, YH and Wood, K (eds) 2015, *Complex systems design & management Asia*, Springer, pp. 27-39

Dai, H, Khalil, EB, Zhang, Y, Dilkina, B and Song, L 2017, 'Learning combinatorial optimization algorithms over graphs', *arXiv preprint arXiv:1704.01665*, 1704, pp. 1-24

Durmisevic, E 2019, *Circular economy in construction design strategies for reversible buildings*, Elma Durmisevic

Fang, Y 2017, *Optimization of Daylighting and Energy Performance Using Parametric Design, Simulation Modeling, and Genetic Algorithms*, Ph.D. Thesis, North Carolina State University

Gershenfeld, N, Carney, M, Jenett, B, Calisch, S and Wilson, S 2015, 'Macrofabrication with digital materials: Robotic assembly', *Architectural design*, 85(5), pp. 122-127

Janner, M, Levine, S, Freeman, WT, Tenenbaum, JB, Finn, C and Wu, J 2018, 'Reasoning about physical interactions with object-oriented prediction and planning', *arXiv preprint arXiv:1812.10972*, 1812, pp. 1-12

Sanchez, J 2016 'Combinatorial Design: Non-Parametric Computational Design Strategies', *ACADIA // 2016: POSTHUMAN FRONTIERS: Data, Designers, and Cognitive Machines [Proceedings of the 36th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*

Scarselli, F, Gori, M, Tsoi, AC, Hagenbuchner, M and Monfardini, G 2008, 'The graph neural network model', *IEEE transactions on neural networks*, 20(1), pp. 61-80

Schrittwieser, J, et al. 2020, 'Mastering atari, go, chess and shogi by planning with a learned model', *Nature*, 588(7839), pp. 604-609

Schulman, J, Wolski, F, Dhariwal, P, Radford, A and Klimov, O 2017, 'Proximal policy optimization algorithms', *arXiv preprint arXiv:1707.06347*, 1707, pp. 1-12

Shi, X and Yang, W 2013, 'Performance-driven architectural design and optimization technique from a perspective of architects', *Automation in Construction*, 32, pp. 125-135

Shih, S 2016 'On the Hierarchical Construction of SL Blocks A Generative System that Builds Self-Interlocking Structures', *Advances in Architectural Geometry 2016*

Shih, SG, Yen, CH and Chou, LW 2019 'Extensible Structures of Interlinking SL Strands', *Proceedings of Bridges 2019: Mathematics, Art, Music, Architecture, Education, Culture*, pp. 415-418

Silver, D, et al. 2017, 'Mastering the game of go without human knowledge', *nature*, 550(7676), pp. 354-359

Tessmann, O and Rossi, A 2019, 'Geometry as Interface: Parametric and Combinatorial Topological Interlocking Assemblies', *Journal of Applied Mechanics*, 86(11), pp. 1-12

Thompson, R, Ghalebi, E, DeVries, T and Taylor, GW 2020, 'Building LEGO Using Deep Generative Models of Graphs', *arXiv preprint arXiv:2012.11543*, 2012, pp. 1-12

Tibbits, S and Cheung, K 2012, 'Programmable materials for architectural assembly and automation', *Assembly Automation*, 32(3), pp. 216-225

Wang, Z, Song, P and Pauly, M 2018, 'DESIA: A general framework for designing interlocking assemblies', *ACM Transactions on Graphics (TOG)*, 37(6), pp. 1-14

Wortmann, T and Nannicini, G 2017, 'Introduction to architectural design optimization', in Karakitsiou, A, Migdalas2, A, Rassia, ST and Pardalos4, PM (eds) 2017, *City Networks*, Springer, pp. 259-278