Learn2Assemble with Structured Representations and Search for Robotic Architectural Construction

Niklas Funk, Georgia Chalvatzaki, Boris Belousov, and Jan Peters
Department of Computer Science, Technical University of Darmstadt, Germany
{niklas,georgia,boris}@robot-learning.de, mail@jan-peters.net

Abstract: Autonomous robotic assembly requires a well-orchestrated sequence of high-level actions and smooth manipulation executions. Learning to assemble complex 3D structures remains a challenging problem that requires drawing connections between target designs and building blocks, and creating valid assembly sequences considering structural stability and feasibility. To address the combinatorial complexity of the assembly tasks, we propose a multi-head attention graph representation that can be trained with reinforcement learning (RL) to encode the spatial relations and provide meaningful assembly actions. Combining structured representations with model-free RL and Monte-Carlo planning allows agents to operate with various target shapes and building block types. We design a hierarchical control framework that learns to sequence the building blocks to construct arbitrary 3D designs and ensures their feasibility, as we plan the geometric execution with the robot-in-the-loop. We demonstrate the flexibility of the proposed structured representation and our algorithmic solution in a series of simulated 3D assembly tasks with robotic evaluation, which showcases our method's ability to learn to construct stable structures with a large number of building blocks. Code and videos are available at: https://sites.google.com/view/learn2assemble

Keywords: Structured representations, Autonomous assembly, Manipulation

1 Introduction

Construction and manufacturing are becoming increasingly automated in the last decades. However, there is an essential need for sustainable autonomous architectural assembly [1], where a game-changer would come with intelligent robot assembly abilities that optimally decide over plans, actions, execution, and efficiency [2]. In this work, our main focus is the combinatorial optimization problem of autonomously assembling complex structures with robotic manipulators *without an a priori* defined task plan and goal poses for the sequential picking-placing actions. For constructing abstract target designs, we must consider the combinatorics of the growing action space w.r.t. the number of available modules and the size of the structure. Therefore, an effective representation of the assembly problem is essential. Moreover, the geometric execution of the picking and placing actions by the robot imposes constraints to the assembly sequence, as those actions are subject to the kinematic feasibility in the robot's workspace. Eventually, the problem of assembling structures lies in the area of long-horizon manipulation tasks, where most methods in the literature consider a known task plan, and focus on fine manipulability and structural stability, or learn action sequences from demonstrations [3, 4, 5, 6, 7, 8].

In this paper, we propose a novel algorithmic solution for robotic assembly that combines high-level decision-making on the construction sequence with the geometric execution by the robot that should ensure feasibility and stability. We propose a graph-based representation that captures the relations between target shapes and available building blocks. Notably, we design a multi-head attention-based graph neural network (GNN) architecture with a purposefully induced inductive bias for encoding the structural representation of the assembly task. The GNN is trained through reinforcement learning (RL) to explore feasible actions, resulting in an expressive and flexible representation. When combined as prior to Monte Carlo Tree Search (MCTS), it extrapolates to out-of-distribution (OOD) assembly tasks, i.e., tasks with a higher number and different types of blocks and vari-

5th Conference on Robot Learning (CoRL 2021), London, UK.

able target shapes. Ultimately, we provide a solution for the motion generation of the sequential picking-placing actions when having the robot-in-the-loop to ensure reachable and feasible actions, which do not disrupt the assembly process. The proposed *learn2assemble* algorithm provides a flexible, autonomous robotic assembly agent for constructing 3D shapes using more than ten blocks.

Our main contributions are threefold. (1) We propose a **multi-head attention-based graph representation** for the 3D assembly problem that is flexible enough for representing arbitrary, stable structures and their relations to different building blocks. (2) We design an **integrated long-horizon manipulation algorithm** that learns through **exploration**, which combined with **model-based search** leads to generalizable skills. Our method considers the **robot-in-the-loop**, integrating high-level action planning with low-level motion generation for learning policies that ensure the kinematic feasibility and stability of the constructed structure. Finally, (3) we developed a novel **benchmarking environment for 3D robot assembly**, which is modular for testing with and without the robot, for arbitrary target designs, and with adjustable types and number of objects. Our empirical results on a series of representative experiments showcase the generalization power of the proposed algorithmic solution, drawing interesting insights on the combination of integrated learning and planning for long-horizon manipulation that can apply to a range of robotic applications.

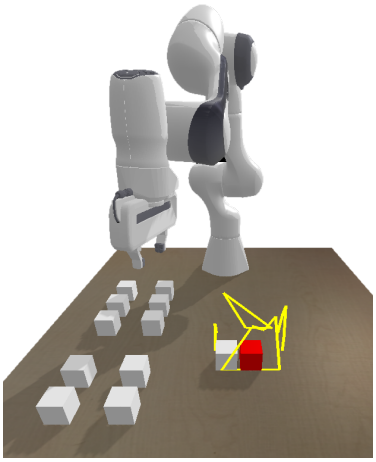


Figure 1: Simulated assembly environment with a 7-DoF manipulator and two sets of blocks: the unplaced white ones on the left and the "base" red block with some already placed blocks by the robot in the centre. The yellow silhouette denotes the target shape. The goal is to place the available blocks to fill the overall 3-dimensional target shape.

Related works. Autonomous robotic assembly is essential for manufacturing and construction, and therefore, many works tried to tackle the problem of automatizing task execution and fine manipulation. Autonomous assembly is a long-horizon manipulation task with multiple stages of decisions and subtask executions to be made alongside controlling the dynamic execution of the assembly process. Thus, researchers proposed methods for Task and Motion Planning (TAMP) [9], and RL [10] to address the challenges of combinatorial optimization over high-level action sequencing and low-level motion generation.

In [11, 12] a TAMP method is proposed for robotic architectural construction and extrusion, requiring though exact domain specifications to find an assembly sequence; TAMP with logic-geometric programming is proposed by [13] where the focus lies in optimizing the structure's stability. In [14], the authors propose a hierarchical planner using hybrid dynamics models for the "toy-airplane" assembly task. The authors of [15] optimize the assembly sequence of complex interlocking blocks through RL assuming a single fixed design. A method based on neuro-symbolic planning is proposed by [16] for learning to predict sequences of actions for stacking. The authors of [17] propose an end-to-end approach for sequential pick-and-place tasks using shape correspondences and learning to assemble by collecting demonstrations from a human operator disassembling. Conversely, we *learn2assemble* arbitrary designs from scratch, learning both the sequence of actions and discovering goal positions per building block.

Autonomous assembly appeals to the machine learning community due to its combinatorial complexity, which, depending on the structure's size and availability of building blocks and their possible combinations [18], can by far surpass the state-action combinatorics of problems like chess and Go [19, 20]. The relational representation [21, 22] and generalization power of GNNs was thoroughly explored in solving combinatorial optimization tasks [23], with successful applications in 2D assembly [24] when combined with RL. In robot learning, earlier works use deep RL for short-horizon challenges like peg-in-a-hole [10, 25, 26]. In [27] the use of structured representations in model-free RL is proposed to induce inductive biases in different stages of a curriculum for executing assembly tasks of building towers of variable heights. While the general motivation of this work is close to ours, [27] does not learn the sequence of the building blocks for constructing arbitrary structures, but it is essentially a goal-based method for learning pick-and-placing manipulation with known goal object positions. In a similar direction, the authors of [28] use learning from demonstrations to train two GNNs, one that selects objects in the scene and another one that selects a suitable goal state from a set of possible goal positions.

2 Proposed Method

The main research question we pose in this work is *"how can a robot perform combinatorial assembly tasks of abstract architectural designs?"*. Fig. 1 depicts a typical setup containing a 7DoF manipulator robot, a varying number of building blocks, and one possible architectural target design. We formalize the task of autonomous construction of abstract target shapes given some available building blocks as a Markov decision process (MDP) [29]. As illustrated in Figs. 1 & 4 we can build one- up to four-sided structures. The overall desired structure is given by each side's target topology, which is in turn defined by the coordinates of the target points spanning the area to be filled (see illustrative example in Fig. 2). The desired shape is thus parameterized by the set of target points $\mathcal{S}_T = \{\mathbf{x}_{T_i} | i \in N_i\}$ (yellow). The state also contains a set of blocks that are already placed in the scene $\mathcal{S}_P = \{\mathbf{x}_{P_j} | j \in N_j\}$ (red) and a set of unplaced blocks $\mathcal{S}_U = \{\mathbf{x}_{U_k} | k \in N_k\}$ (gray) that need to be added. The vector $\mathbf{x} \in \mathbb{R}^5$ includes the 3D position and two booleans indicating the element's properties (placed/unplaced, target point/building block). The state s_t at any time step t is thus given by the three sets $s = (\mathcal{S}_T, \mathcal{S}_P, \mathcal{S}_U)$ and has $N = N_i + N_j + N_k$ elements. The objective is to use the available blocks to construct a stable structure that fills the desired target design. We use a discrete action space and define five relative placement actions, i.e., any unplaced block can be put on top, on the left, on the right, behind, in front w.r.t. any already placed block. In the robot experiments, however, the action space is augmented to also allow the specification of the manipulator's orientation during grasping and placing. For most experiments, we choose over two possible grasping and placing orientations, resulting in four grasp-place combinations. Together with the placement actions, this yields $N_a = 4 \times 5 = 20$ actions when placing one block w.r.t. another. Since the possible actions a_t depend on the number of placed and unplaced blocks, we end up with a time-varying action space of size $N_j \times N_k \times N_a$. We assign a positive reward $r(s_t, a_t)$ on all actions that increase the filling of the target structure while preserving its stability. Given the problem's combinatorial complexity of discovering the optimal high-level action plan that will allow the assembly of complex 3D structures which are stable and kinematically feasible, we decompose it into three main problems: (i) finding an expressive representation for our state space, (ii) decoding states into meaningful actions, and (iii) ensuring stability and kinematic feasibility. Our research provides a thorough study of how we can tackle these problems, and we provide a novel 3D assembly algorithm with the robot-in-the-loop through combining a learned multi-head attention (MHA) representation with MCTS.

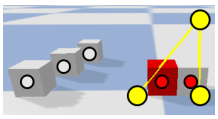


Figure 2: Setup with a one-sided triangular target shape defined by three points (yellow), three unplaced blocks (gray) and two placed blocks (red).

2.1 Multi-head attention graph representation

Graph-based representations [30] are an effective tool when dealing with combinatorial problems [31, 32, 33]. Compared to classical satisfiability solvers, their main advantage lies in their real-time capabilities, while their architectural properties allow generalization to problems of different sizes in contrast to most standard neural network architectures, which operate on fixed-size inputs and outputs. As those are essential properties for learning to assemble structures of combinatorial complexity, we will introduce below our proposed GNN model that is inspired by the combination of graphs and attention [34, 35]. GNNs receive as input a graph $G = (\mathcal{N}, \mathcal{E})$, and return a high-level encoding over nodes and edges to be further exploited for deciding which action to take. In our case, the set of nodes $\mathcal{N} = \{\mathcal{S}_T, \mathcal{S}_P, \mathcal{S}_U\} = \{n_i\}_{i=1..N}$ is given by the current state, and the connectivity information $\mathcal{E}$ is defined as a matrix of size $N \times N$. If there is an edge connection between nodes n_i and n_j entry $\mathcal{E}(i, j)$ equates to 1, otherwise it is 0.

Attention mechanisms [34] were introduced in GNNs [35] to enable nodes to attend over their neighbours' features and learn different weights for different nodes without requiring costly matrix operations. We want to exploit this flexibility as solving the assembly task necessitates drawing connections on multiple levels, i.e., between nodes of the same type to encode the already existing structure or the target shape, as well as between all nodes to come up with a meaningful representation for action-decision. In the following, we will introduce the proposed MHA architecture, that naturally reflects the necessary multi-level decision process of the assembly task, and has proven to be effective when combined with policy search for solving combinatorial problems, like the travelling salesman [32]. In the first step of MHA, the initial node embeddings $n_i^{(0)} = \mathbf{x}_i$ are projected into a higher dimensional space by

$$n_i^{(1)} = g(n_i^{(0)}) = \mathrm{ReLU}(\mathrm{FC}(n_i^{(0)})), \tag{1}$$

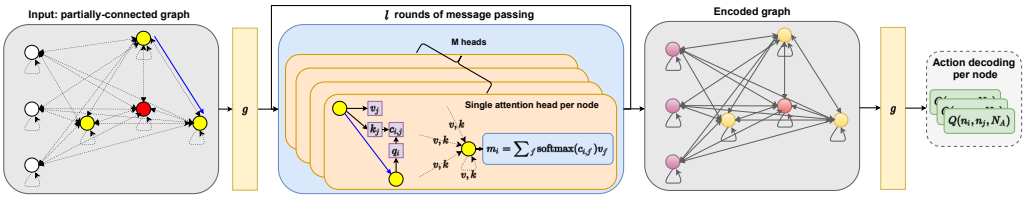


Figure 3: GNN architecture illustration, mapping from the input graph to Q-values. The coloring follows Fig. 2. After an initial projection into a higher dimensional space follow l rounds of message passing using MHA. This results in an encoded version of the graph, which is then exploited for action selection.

using a fully-connected (FC) layer followed by a rectified linear unit (ReLU) activation function. Note that the function g will be used repeatedly as we progress with the graph update, yet, assuming different weights on each further appearance. Next follow L rounds of message passing, i.e.,applying (2) L times to obtain the final embedding for each node i according to

$$n_i^{(l)} = h(g(h(\text{MHA}(\mathcal{N}^{(l-1)}, i)))), \tag{2}$$

with a skip connection layer $h(f(x)) = x + f(x)$, the current round of message passing l, all node embeddings from the previous round $\mathcal{N}^{(l-1)}$, and a MHA mechanism introduced below. As illustrated in Fig. 3, for a single out of the M attention heads with index o, we first compute three values – the key k, query q, and value v – using three different weight matrices $W_{k,o}$, $W_{q,o}$, $W_{v,o}$, respectively ($k_{i,o} = W_{k,o} {n_i}^{(l-1)}$, $q_{i,o} = W_{q,o} {n_i}^{(l-1)}$, $v_{i,o} = W_{v,o} {n_i}^{(l-1)}$). Multiplying the key and query of all nodes results in a compatibility score $c_{i,j}$ for the i^{th}-to-j^{th} node connection

$$c_{i,j,o} = \begin{cases} \frac{1}{d} q_{i,o}^T k_{j,o}, & \text{if } \mathcal{E}(i,j) = 1, \\ -\infty, & \text{otherwise,} \end{cases} \tag{3}$$

with d a normalizing constant. From this score, we can then compute the attention weights using a softmax $a_{i,j,o} = \frac{e^{c_{i,j,o}}}{\sum_{j'} e^{c_{i,j',o}}}$ to aggregate the values for each node, resulting in the output message $m_{i,o} = \sum_{j'} a_{i,j,o} v_{j,o}$. A weighted sum of all messages yields the final result of the MHA module

$$\text{MHA}(\mathcal{N}^{(l-1)}, i) = \sum_{o=1}^{M} W_{m,o} m_{i,o}, \tag{4}$$

with weights $W_{m,o}$ controlling the influence of each one of the single attention heads. Fig. 3 depicts a simplified example of the encoding performed by the MHA architecture.

2.2 Learning assembly policies

MHA-GNNs can only unfold their potential when combined with algorithms that refine their weights to form expressive representations exploited herein for action selection (Fig. 3). The GNN should thus be shaped based on the reward signal, resulting in a RL setup with the goal of obtaining performant policies. This powerful combination results in agents that (i) can be applied to different problem instances due to the representation's flexibility, (ii) are reactive, despite the problem's combinatorial complexity, and (iii) can be trained directly in simulation environments which include the nonlinearities of the robot and the contacts. Due to the problem's combinatorial complexity and its discrete, time-varying action space, we use model-free Q-learning, [36] which has been successfully applied to complex tasks such as playing Atari games from images. Moreover, we investigate its integration with model-based planning, as the addition of search can counteract the overoptimism of the Q-function approximation and result in more robust behaviour [19, 37].

Action Decoding. The encoded graph representation from Sec. 2.1 needs further processing to decide on the next action to take. As actions are defined relatively between unplaced and placed blocks (i.e., nodes) we can directly assign a value to all available actions

$$Q(n_i, n_j, N_a) = g\left({n_i}^{(l)}, {n_j}^{(l)}, \text{FC}\left(\frac{1}{N}\sum\nolimits_{j'} {n_{j'}}^{(l)}\right)\right) \qquad \forall n_i \in \mathcal{S}_\mathcal{U}, n_j \in \mathcal{S}_\mathcal{P} \tag{5}$$

with the total number of N nodes in the graph. Note that the Q-value does not only depend on the two nodes' embedding, but also on a global feature based on averaging over all embeddings. As all the operations are defined over the set of nodes, this encoding-decoding architecture can seamlessly generalize to different problem sizes, i.e.,different number of blocks or target shapes.

DQN: For our setting, we define the loss function as the smooth L1 loss between the current action-value estimate of the GNN, noted as $Q(n_i, n_j, a_s)$ and the value obtained from the rollouts using a

target network, noted as Q_T, i.e., $\hat{Q}(n_i, n_j, a_s) = r(s_t, a_t) + \gamma \max_{n_i \in \mathcal{S}_U, n_j \in \mathcal{S}_P} Q_T(n_i, n_j, a_s)$ with discount factor γ and the selected action $a_s \in N_a$.

MCTS: MCTS [38] has proven effective for solving tasks with discrete action spaces [39], hence, suitable for our problem. Contrary to model-free RL, MCTS relies on a model of the environment to perform tree search for action selection and does not require any form of function approximation. However, in scenarios with significant branching factors or expensive model evaluations, the time complexity quickly increases, and the use of MCTS becomes intractable. Consequently, [19, 20] exploit Q-learning for estimating the value of the nodes in MCTS and show impressive results in solving tasks of great complexity, like the game of Go. Also, in the context of accelerating and generalizing skill-learning, the combination of learning and planning has received increasing attention [20, 37, 40, 41]. In this work, inspired by these advantages, we explore different algorithmic variations of the interplay between Q-learning and planning, as described below. Pseudocode for all variants is given in Appendix B.

1) DQN+MCTS: Inspired by [19], we propose a variant of AlphaGo, that uses a pretrained Q-network as a prior for evaluating the leaf nodes of MCTS, leading to increased efficiency. As we seek to effectively combine learning and planning for autonomous assembly with the robot-in-the-loop, we use small search budgets for which the prominent solution for MCTS expansion - the exploration strategy of UCT [19, 42] - is unsuitable because it becomes over-optimistic [43]. Therefore, we use an ϵ-greedy expansion strategy during search, that allows better exploration than UCT.

2) Q-MCTS: We follow a generalization of Q-learning using samples based on planning. As in [20, 37], we combine DQN with MCTS during training to perform an informed exploration and collect good samples through search. However, as motivated previously, due to small search budgets, our approach uses an ϵ-greedy expansion strategy over unexplored states, instead of UCT. Let $Q_S(s_t, a_t) = Q_S(s_t, a_t) + r(s_t, a_t) + \gamma \max_{a'_t} Q(s'_t, a'_t)$ be the updated value of an expanded node, where s'_t is the new state arriving during search. The resulting Q-MCTS methods augments the DQN learning objective with a cross-entropy loss defined on the values explored during search, i.e.,$\mathbf{loss} = -\mathbf{softmax}(Q_S(s_t, a_t))^T \log(\mathbf{softmax}(Q(s_t, a_t)))$ which is intended to regularize and improve the Q-function estimate based on the experience collected while searching.

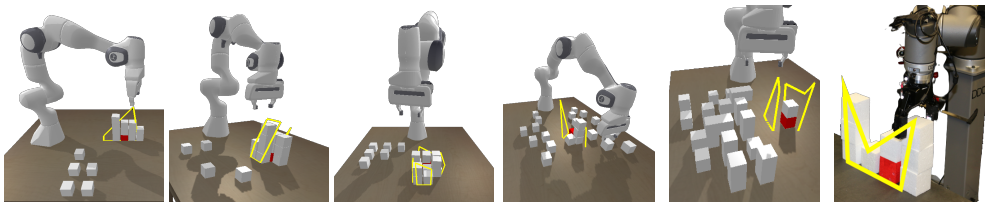
3) ϵ-MCTS: This implementation follows [37] more closely. The main differences w.r.t. the Q-MCTS policy are that first, there is an ϵ-greedy decision on whether to do search or uniformly sample a random action, whereas Q-MCTS always conducts search. Secondly, during search, ϵ-MCTS follows the UCT expansion strategy. The Q-learning objective remains the same as with Q-MCTS, however, the cross-entropy loss is computed only on the samples where search has been conducted. In essence, the method's difference is in the way of collecting the model-based samples.

2.3 Integrated learning and planning for robotic assembly

All previous components can now be combined to obtain an algorithm capable of training agents to build desired target shapes, i.e., a robot capable of abstracting the sequence of actions for building arbitrary stable target structures from individual elements while executing feasible actions in its workspace. Note that the only control component we assume given is a point-to-point control strategy based on the robot's inverse kinematics. To enable this combined decision-making strategy that touches on the ground of TAMP literature, we propose the combination of the previously described graph representations and learning algorithms to provide a novel *learn2assemble* method. Briefly, a single step of a learning episode starts with selecting actions (with or without tree-search depending on the learning algorithm), i.e., the next object to be placed, the grasping pose, the goal position, and orientation. A path from the picking to the placing position is computed, and the robot executes the placement, for which it receives a reward. Consequently, we store the current graph's state, action, reward, and new state in the replay memory to be later used for learning.

3 Experimental Results

For evaluating the different components of the proposed method and their respective contribution, we designed specific experimental scenarios. We start with an investigation over graph architectures, continuing with the different learning methods in environments for 3D assembly without including a robot yet. Selecting the best settings from the previous tests, we experiment with the robot-in-the-loop for our final empirical evaluations of the proposed algorithm.



(a) (b) (c) (d) (e) (f)

Figure 4: **(a)** Illustration of the single-sided 3D assembly environment, **(b)** the two-sided 3D assembly environment, **(c)** the four-sided 3D assembly environment with the robot-in-the-loop, **(d)** the two-sided 3D environment with the unplaced blocks placed at random, **(e)** the two-sided environment with different building blocks, and **(f)** the result of transferring a policy trained in simulation (a) to the real world and even a different manipulator.

Table 1: Comparison of different architectures on the single-sided environment without the robot and only one type of block. R is the cumulative discounted return, f the ratio of runs that ended with failure, i.e., the structure colliding, and b the ratio of runs that ended without success and no more blocks remaining. The star(*) marks the environment where the agents were trained in.

	3-by-3 grid, 20-24 blocks*			3-by-3 grid, 30-34 blocks			4-by-4 grid, 20-24 blocks		
Method	R	b	f	R	b	f	R	b	f
MHA (FC)	**3.22** (0.04)	0.01	**0.15**	**3.44** (0.04)	0.00	**0.21**	**3.66** (0.05)	0.18	**0.41**
S2V (FC)	2.36 (0.08)	0.49	0.47	2.15 (0.10)	0.08	0.87	2.75 (0.20)	0.45	0.55

Simulation environments. The assembly environments are depicted in Fig. 4. We investigate, one-, two- and four-sided setups with each side of the target shape defined by the position of 3 (see Fig. 2), 4, or 5 target points. The number of target points and their locations are sampled randomly from grids of different sizes, ranging from 3-by-3 (i.e., the sampled target points can at maximum span an area of height and width of 3 times the cube's edge length) to 6-by-6. Due to the relative action space, every scene is always initialized with one initially placed element, marked in red, serving as the building base. For evaluating the assembly progression, we use depth cameras, placed on each side of the specified target structure. By projecting the target points into the images after each action, we obtain the change in the target shape's filling. The reward functions endorse actions that lead to improvement in the filling (cf. Appx. C.2). The construction process is finished, once the total coverage exceeds a threshold, whenever there are no more unplaced blocks available, or upon executing an invalid action, i.e., an action resulting in an unstable configuration, in destructing the current structure, or if it is kinematically infeasible. In all environments without the robot, the placing is done by directly specifying one of the five placement actions, resulting in a reduced action space of $N_a = 5$. If not stated differently we use partially connected graphs (see Fig. 3), inducing a stronger inductive bias on the structured representation compared to fully connected (FC) graphs (cf. Appx. A.3 & D.3). In the following tables, for evaluating the agent's performance, we report the cumulative discounted return R, the ratio of runs that ended with failure, i.e., upon an invalid action f, the ratio of runs that ended without success and no more blocks remaining b, as well as the mean number of actions conducted per run $\bar{a}$. The star(*) marks the agents' evaluation in the same setting as in training, while the rest are OOD experiments, i.e., exclusively evaluating the agents in settings with previously unseen target shapes or number of blocks. For more details, see Appx. C.

Graph architectures. We evaluate the proposed MHA representation against the commonly used Structure2Vector (S2V) architecture [24, 31] (cf. Appx. A.1) in a simple environment (see Fig. 4a) only considering one type of object but omitting the robot. The learning is conducted with DQN.

Results. As shown in the first column of Table 1, already in the original training environment, the MHA approach outperforms S2V significantly. The high rates of failure and exceeding the number of available blocks indicate that S2V cannot draw the connection between the target shape and the current structure. This might be due to S2V's different message passing, which cannot weigh the importance of different nodes as with the attention mechanism. When increasing the number of available blocks and the size of the structure to be built (columns 3 & 4), we see an evident advantage of MHA in handling OOD tasks. In Appx. D.2.1, we provide additional results when using only a single attention head, which confirm that using attention is advantageous, and MHA yields the best performance. We, thus, continue our experimentation using the MHA architecture.

Learning algorithms. To investigate the performance of the learning algorithms (Sec. 2.2), we will use the two-sided environment shown in Fig. 4b without the robot, thus using the reduced action space.

Results. Table 2 summarizes the results, starting without any search budget (i.e., no tree search) to evaluate the learned Q-functions. The DQN and ϵ-MCTS agents perform similarly, with DQN slightly outperforming ϵ-MCTS through lower failure rates across tasks. Moreover, DQN can solve

Table 2: Combining Q-learning and MCTS in the two-sided environment without the robot.

Search Budget	Method	3-by-3, grid 20-24 blocks* R	$\bar{a}$	f	3-by-3 grid, 30-34 blocks R	$\bar{a}$	f	4-by-4 grid, 30-34 blocks R	$\bar{a}$	f	5-by-5 grid, 40-44 blocks R	$\bar{a}$	f
0	DQN	**3.21** (0.05)	7.93	0.16	**3.44** (0.08)	8.65	0.17	**3.61** (0.06)	12.02	0.51	**3.63** (0.08)	13.66	0.93
	ϵ-MCTS	3.18 (0.03)	8.57	0.22	3.37 (0.10)	9.30	0.30	3.54 (0.09)	12.53	0.62	3.50 (0.13)	12.75	0.95
	Q-MCTS	2.80 (0.15)	7.33	0.51	2.89 (0.09)	7.60	0.64	2.66 (0.08)	7.47	0.92	2.34 (0.17)	6.45	0.99
10	DQN+MCTS	**3.47** (0.02)	8.16	**0.05**	**3.66** (0.03)	8.96	**0.06**	**3.96** (0.04)	13.92	**0.31**	**4.08** (0.09)	17.33	**0.87**
	ϵ-MCTS	3.21 (0.03)	8.48	0.20	3.37 (0.11)	9.10	0.31	3.60 (0.09)	12.65	0.61	3.63 (0.16)	13.16	0.93
	Q-MCTS	3.24 (0.04)	8.56	0.27	3.39 (0.08)	9.24	0.41	3.42 (0.07)	10.99	0.76	3.41 (0.13)	11.27	0.96
1000	UCT	0.54	4.00	1.00	-	-	-	-	-	-	-	-	-

the task with fewer actions. Contrarily, the Q-MCTS agents perform significantly worse. We believe that this difference in performance is due to the constant cross-entropy regularization in Q-MCTS from the beginning of the training, especially when search samples might be bad, while for the ϵ-MCTS agent the regularization is only slowly added as training proceeds and less random actions are taken (search is better), which seems to be beneficial. Note that without the addition of search, increasing the number of blocks, as well as the target size, results in a quick increase of the failure rate for all methods. Adding a search budget of 10 already counteracts this trend, especially considering the DQN agent, where the rates of failure can be reduced for all the tasks by using DQN+MCTS at test time. For the Q-MCTS agent, the same trend is noticed, while the gains of the ϵ-MCTS agent are marginal. This suggests that the ϵ-MCTS agent is overoptimistic and seems to choose similar actions as to when not using search at all. Overall, combining MCTS with a pretrained DQN results in the best performance in our experiments. The last row of the table underlines the problem's combinatorial complexity, illustrating that performing pure UCT without any prior and a search budget of 1000 performs significantly worse for the simplest experimental setting (more details in Appx. D.4).

***Learn2assemble* with the robot-in-the-loop.** Next, we evaluate the DQN agent combined with tree-search in our target environments, including the robot manipulator. The task's difficulty increases, as also the grasping and placing poses have to be specified while ensuring action feasibility by the robot and the structure's stability. We start with constructing single-sided designs (Fig. 4a) to illustrate the necessity of training with the robot-in-the-loop before evaluating the proposed method with multi-sided designs (Fig. 4b & 4c).

Results. We first compare two policies, one trained with, and the other without the robot using plain DQN without any search in a single-sided environment (Fig. 4a). The agent trained with the robot-in-the-loop outperforms the other one resulting in a significantly reduced failure rate of $15\,\%$ and consistently higher rewards (cf. Appx. D.5). This shows the necessity of including geometric planning during training for obtaining high-level decisions that are compatible with the low-level execution. It is thus not sufficient to only figure out where to place the parts; the kinematic constraints and robot motion have to be considered. In the subsequent experiments, we build 3D structures with the robot as shown in Figs. 4b & 4c. Our results in Tables 3 and 4 show that MCTS (search budget of 10) consistently improves performance in terms of higher returns and lower failure rate. In particular, the experiments demonstrate that our proposed pipeline can execute multiple sequential pick-and-placing actions, with a maximum of 17 correctly placed blocks for the 5-by-5 grid in the two-sided environment and up to 22 correct placements in the four-sided environment, using DQN+MCTS with the robot-in-the-loop. Compared to the previous experiments, without the robot-in-the-loop, the failure rate is higher, indicating the task's increased difficulty, and the need of adding a soft-placing controller.

Table 3: Comparing policies with and without tree search on the four-sided robotic environment.

Method	3 by 3 grid 10-18 blocks* R	$\bar{a}$	f	3 by 3 grid 16-24 blocks R	$\bar{a}$	f
DQN	2.67 (0.06)	6.91	0.30	2.55 (0.06)	7.25	0.35
DQN+MCTS	**3.08** (0.06)	7.59	**0.16**	**2.90** (0.07)	8.00	**0.20**

Generalization w.r.t. randomized scenes. To evaluate our algorithm's robustness w.r.t. changes in the scene, we transfer the previously trained policies and evaluate them in scenarios where the unplaced blocks are placed randomly around the structure to be built, as shown in Fig. 4d.

Results. Rows 3 & 4 of Table 4 reveal that the policies indeed generalize to these novel scenarios, as the percentage of unsuccessful experiments only increases at maximum by $11\,\%$ for the most complex scenario, compared to their performance in the original environment (rows 1 & 2). This confirms that our proposed method does not overfit the exact layout or geometry, but rather builds meaningful features that allow to successfully transfer the behaviour to scenes that are substantially different from those encountered during training.

Table 4: Comparing policies on the two-sided robotic environment. Rows 1&2 correspond to evaluating in the original environments (Fig. 4b), while rows 3&4 are the evaluation in randomly initialized scenes (Fig. 4d).

Method	Environment initialization	3 by 3 grid 10-14 blocks* R	$\bar{a}$	f	3 by 3 grid 14-18 blocks R	$\bar{a}$	f	5 by 5 grid 14-18 blocks R	$\bar{a}$	f
DQN	fixed	2.16 (0.06)	4.89	0.20	2.10 (0.05)	5.27	0.24	2.54 (0.15)	7.53	0.56
DQN+MCTS	fixed	**2.41** (0.05)	5.32	**0.09**	**2.32** (0.03)	5.64	**0.15**	**3.19** (0.11)	9.38	**0.36**
DQN	random	2.06 (0.10)	4.93	0.24	1.88 (0.13)	5.02	0.32	2.28 (0.11)	7.52	0.64
DQN+MCTS	random	2.33 (0.06)	5.38	**0.10**	2.14 (0.12)	5.54	**0.20**	2.85 (0.14)	9.31	**0.47**

Table 5: Evaluating the trained policies in the environments with multiple different objects available (Fig. 4e). The results in the first row correspond to using a modified environment without including the robot.

Method	Environment w/wo robot	5 by 5 grid 20-24 blocks* R	$\bar{a}$	f	5 by 5 grid 30-34 blocks R	$\bar{a}$	f	6 by 6 grid 30-34 blocks R	$\bar{a}$	f
DQN+MCTS	wo robot	**2.20** (0.02)	5.05	**0.06**	1.89 (0.03)	5.96	**0.15**	1.48 (0.09)	7.27	**0.27**
DQN+MCTS	w robot	1.42 (0.05)	3.53	**0.21**	0.77 (0.09)	3.27	**0.52**	0.61 (0.08)	4.38	**0.61**

Generalization w.r.t. different building blocks. We finally investigate our method's performance when using more complex objects (Fig. 4e). This makes the task significantly more difficult, as the agent has to not only learn each part's admissible grasps but also differentiate the building blocks to select and place the correct object type. All novel objects are a combination of primitive boxes which allows keeping the relative action space, i.e., placing an unplaced primitive box belonging to a larger object w.r.t. a placed primitive box results in moving the entire object. For the experiments without the robot, we adjust the action space to enable changing the object's orientation (cf. Appx. D.9). *Results.* The results in Table 5 illustrate the representation's flexibility and ability to successfully deal with the different building blocks. Despite the increased complexity, we achieve similar performance in the scenes without the robot as in previous experiments (Table 2). When training with the robot, we can still handle the task's complexity with a $\sim 80\%$ success rate in the simplest setting, but observe a drop in performance with a larger number of objects and bigger target shapes. While one cause for the performance drop is the setting's increased complexity, there are also several placement actions that would require 3D gripper orientation control allowing for a smooth insertion of the complex blocks. As our current top-down placing controller only offers planar orientation control, some placement actions cannot be executed appropriately resulting in increased failures.

Remarks. We have conducted extensive experiments to show the superiority of our proposed MHA-GNN approach for solving combinatorial assembly tasks with the robot-in-the-loop. The results demonstrate how strong inductive biases combined with attention can shape meaningful relational representations. When combined with deep Q-learning, this representation allows us to take decisions over long horizons despite an increasing action space. Adding search at test time improves performance across all experiments, demonstrating that the proposed method can generalize w.r.t. different target shape sizes, number of building blocks, and different scenes. The policies can also be transferred to the real world and to different manipulators, by initializing the simulation to mirror the real scene and executing the obtained actions both in simulation and reality (see Fig. 4f). While we show very promising results in combining learning high-level action decisions with planning geometric executions, we can see a combinatorial barrier in the decision-making that might be tackled with a more informed search in the graph space. Our experiments on using different objects underline the representation's flexibility, but also reveal current limitations in the definition of the action space, especially considering the robotic execution, which may be mitigated by enriching the action space through 6D grasping and placing. Moreover, several assemblies failed due to "rough" robot actions, meaning that a more sophisticated motion generator might be needed for finer placement.

4 Conclusion

We presented a new *learn2assemble* algorithm for learning autonomous robotic 3D assembly from scratch without prior knowledge of any task plan. For addressing the problem's combinatorial complexity while maintaining adaptability to different scenarios, we propose a graph-based multi-head attention representation that captures the spatial relationships between target construction designs and unplaced blocks, and is trained through deep Q-learning. The powerful representation forms the basis for our hierarchical controller that jointly conducts high-level learning over action sequences and goal specifications together with low-level path planning, ensuring the execution of long-horizon tasks. Our extensive experiments confirm the representation's effectiveness and show extrapolation to environments with previously unseen target shapes, larger numbers of available elements, and different object types. When combining the learned Q-network with MCTS with computationally tractable small search budgets, we manage to improve performance and reliability across all tasks. Notably, we resolve the sequential long-horizon character of the assembly task by including the robot-in-the-loop to decide over feasible grasps and placing actions that ensure the stability of the construction. Our algorithm manages to correctly build structures using up to 22 building blocks with good success rates. In the future, we want to extend the algorithm to allow for a richer set of 6D grasping and placing poses, learn fine-placing or even in-hand manipulation controllers on the low level, and investigate the implementation of an assembly/disassembly strategy, so that the robot can potentially re-use wrongly placed blocks or reconfigure existing structures.

Appendix
Learn2Assemble with Structured Representations and Search for Robotic Architectural Construction

Niklas Funk, Georgia Chalvatzaki, Boris Belousov, and Jan Peters
Department of Computer Science, Technical University of Darmstadt, Germany
{niklas,georgia,boris}@robot-learning.de, mail@jan-peters.net

In the following sections, we provide additional background information and implementation details concerning the methods and results presented in the paper. We start by describing the graph representations, before presenting the pseudocode for all learning algorithms and outlining the experimental setup in greater detail. Lastly, we provide hyperparameters, learning curves and additional material for all the experiments.

A Graph representations

In this section, we introduce the Structure2Vec architecture, how to obtain every node's initial encoding and the two possibilities for defining the graph's connectivity. All these details are used in the experimental evaluation in Sec. 3.

A.1 Alternative encoding - Structure2Vec (S2V)

Structure2Vec Encoding (S2V). S2V is one popular GNN architecture applied to combinatorial optimization algorithms [31]. In the first step of this method, the initial node embeddings $n_i^{(0)} = \mathbf{x}_i$ are projected into a higher dimensional space via

$$n_i^{(1)} = g(n_i^{(0)}) = \text{ReLU}(\text{FC}(n_i^{(0)})), \tag{1}$$

using a fully-connected (FC) layer followed by a rectified linear unit (ReLU) activation function. Note that the function g will be used repeatedly as we progress with the graph update, however, on each further appearance we assume a different set of weights.

For discovering where a block needs to be placed next in our assembly problem, solely relying on the individual node encodings is not sufficient. We also need to take the topology of the graph into account. Thus, for each node, another feature e_i is computed which accumulates the features of all its neighbors as follows

$$e_i(n_i)^{(1)} = g\left(\frac{1}{c(n_i{}^{(0)})} \sum_{\mathcal{E}(i,j)=1, \forall j \in 1..N} g(n_j{}^{(0)})\right), \tag{2}$$

where $c(n_i)$ is the connectivity count of the node n_i, indicating how many edges are connected to this node. Based on these two initial embeddings, for each node, we obtain high level features by performing l rounds of message passing, i.e. updating each encoding l times, according to

$$n_i^{(l)} = g\left(n_i{}^{(l-1)}, g\left(e_i(n_i)^{(1)}, \frac{1}{c(n_i{}^{(0)})} \sum_{\mathcal{E}(i,j)=1, \forall j \in 1..N} n_j{}^{(l-1)}\right)\right). \tag{3}$$

This high-level representation $n_i{}^{(l)}$ will be used subsequently to define our agent's action.

A.2 Insights for the node encoding

To encode both the available and the already placed blocks, as well as the points representing the target structure to be built, we use the following initial representation

$$n_i^{(0)} = \begin{bmatrix} \mathbf{x}_{\text{pos}} \\ \text{placed} \\ \text{target} \end{bmatrix} \tag{4}$$

5th Conference on Robot Learning (CoRL 2021), London, UK.

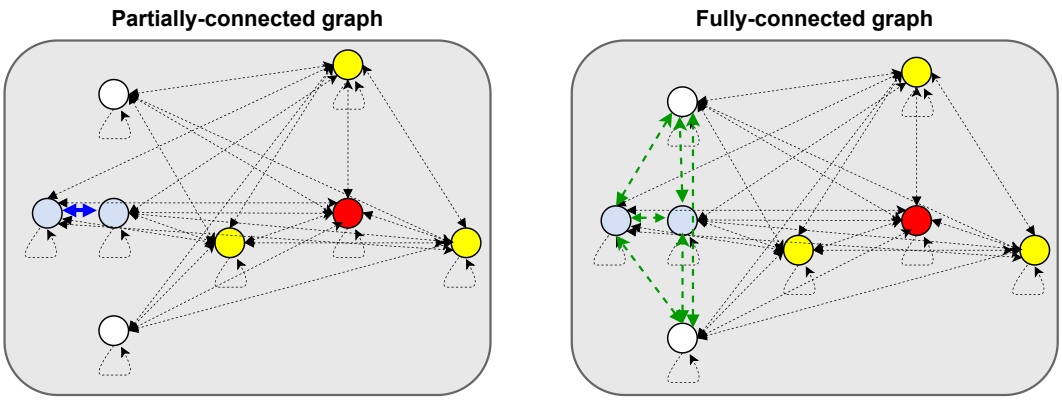


Figure 1: Illustrating the two different connectivity implementations. Note: The two nodes shaded in light blue illustrate one object made up of two primitive blocks. In the partial connectivity setup on the **left**, there are no connections in between the unplaced elements visualized on the left-hand side. Only the connection between the light blue nodes reveals them forming one object. On the contrary, in the fully-connected setup on the **right**, all nodes irrespective of their type are connected.

with the respective object's position ($\mathbf{x}_{\text{pos}} \in \mathbb{R}^3$), and the booleans placed and target. placed takes a value of 1 for target elements and already placed blocks, and -1 otherwise. target equates to 1 for target elements only, and to -1 otherwise. This initial representation forms the basis for obtaining the graph's higher-level encoding using either the MHA or the S2V architecture.

A.3 Graph connectivity

In Fig. 1, we illustrate the different connectivities investigated in this work. Namely, the partially-connected and the fully-connected setup. Whereas in the fully-connected setup all nodes are simply interconnected with each other, in the partially-connected setup, there are no connections in between the unplaced objects. We reason that message passing between the unplaced objects can be omitted, as it should be more crucial to make the connections between the placed elements and the target shape in order to figure out where to next place an unused block. Further, the partial connectivity also has the advantage to clearly mark objects formed from multiple boxes, by adding connections between the individual primitive elements without requiring any further one-hot encoding or similar representation.

B Learning algorithms - Pseudocode

This section provides the pseudocode of all the algorithms presented in Sec. 2.2. Alg. 1 represents the standard framework in which the learning is conducted, i.e. periodically collecting experience and updating the current Q-function which forms the basis for action selection. The difference between the algorithms lies in the action selection mechanisms and loss functions. Apart from that, all of them make use of a replay buffer during training and exploit the same simulation environments. For implementing the learning algorithms, we have used the mushroom reinforcement learning library [44].

The standard Q-learning algorithm, without conducting any model-based search, is provided in Alg. 2. Alg. 3 is an extension to Alg. 2 by adding tree search during test time. For expanding the initial node, it uses an epsilon-greedy expansion strategy. Algs. 5 and 4 differ from the others as also search is conducted while training the models. Whereas Q-MCTS conducts the tree search on every sample and uses an epsilon-greedy expansion, ϵ-MCTS first has the decision whether to do search or not and then eventually uses a UCT expansion strategy. Thus, the loss function of ϵ-MCTS is also more adaptive, since the cross-entropy loss term is only active when search has been conducted during training starting from the current state. As ϵ decreases over time, the influence of the cross-entropy term in ϵ-MCTS is increasing. Contrarily, Q-MCTS does not have this decision and the cross-entropy term is active throughout the whole training process, as for every sample search is conducted.

Algorithm 1 Learn2Assemble

for i in 1..NumberEpochs **do**
/* Collect experience
    $j = 0$, Buffer $\mathcal{B} = []$
/* Define number of samples to collect
    $\Gamma = 100$
    while $j < \Gamma$ **do**
        Sample unplaced elements $\mathcal{S}_U$, initially placed box $\mathcal{S}_P$, target shape $\mathcal{S}_T$
        finished=False
        while finished==False **do**
            Sample action $[a, Q_S, a_e] = act(Q, s)$ using Q-function approximator Q
            Move robot to pick and place the part - Update: $\mathcal{S}_U$, $\mathcal{S}_P$, obtain $r(s, a)$
            Receive next state s'
            $\mathcal{B}$.append($[s, a, Q_S, a_e, r(s, a), s']$)
            $j = j + 1$
            $s = s'$
            if $|\mathcal{S}_U| = 0$, or robot destroyed the structure, or $F(s_{t+1}) > \Delta$, or $j == \Gamma$ **then**
                finished=True
/* Update weights of Q-function
    $\pi = update(Q, \mathcal{B})$

Algorithm 2 DQN

Number of update steps χ
procedure $act(Q, s)$
    if $\mathrm{RandomVariable} < \epsilon$ **then**
        $a = \mathrm{RandomChoice(AllPossibleActions(s))}$
    else
        $a = \max_{a'} Q(s, a' | a' \in \mathrm{AllPossibleActions}(s))$
    return a
procedure $update(\pi, \mathcal{B})$
    Add $\mathcal{B}$ to Replay Memory
    for i in 1..χ **do**
        Sample random subset from Replay Memory
        $\mathrm{loss} = \mathrm{smoothL1}(Q(s, a) - (r(s, a) + \gamma \max_{a'} Q_T(s', a' | a' \in \mathrm{AllPossibleActions}(s))))$
        Update Q-function approximator Q with parameters θ
        $\theta = \theta - \alpha \frac{\partial \mathrm{loss}}{\partial \theta}$
    return Q

Algorithm 3 DQN + MCTS

/* Note, this is only during evaluation, for training see Alg. 2
Rollout Depth $\eta = 1$ if not stated otherwise
Search Budget τ
Number of update steps χ
procedure $act(Q, s)$
    Given: state s, set containing the explored actions $\mathcal{S}_A = \{\}$
    $\forall a$, Initialize $W(s, a) = 1$, $Q_S(s, a) = Q(s, a)$
    for i in 1..τ **do**
        if $\mathrm{RandomVariable} < \epsilon$ **then**
            $a = \mathrm{RandomChoice(AllPossibleActions(s)} | W(s, a) = 1)$
        else
            $a = \max_{a'} Q(s, a' | a' \in \mathrm{AllPossibleActions}(s), W(s, a') = 1)$
        Add a to $\mathcal{S}_A$, collect $r(s, a)$
        for j in 1..$\eta - 1$ **do**
            $a = \mathrm{DQN - act}(Q, s)$ (Alg. 2, Line 2), collect current single step reward $\tilde{r}$
            Update: $r(s, a) = r(s, a) + \gamma^j \tilde{r}$
        Update: $W(s, a) = W(s, a) + 1$, $Q_S(s, a) = \frac{1}{2}(Q_S(s, a) + r(s, a) + \gamma^\eta \max_{a'} Q(s', a'))$
    if $\mathrm{RandomVariable} < \epsilon$ **then**
        $a_r = \mathrm{RandomChoice}(\mathcal{S}_\mathrm{A})$
    else
        $a_r = \max_{a'} Q_S(s, a' | a' \in \mathcal{S}_A)$
    return $a_r, \{Q_S(s, a | a \in \mathcal{S}_A)\}, \{a | a \in \mathcal{S}_A\}$

Algorithm 4 Q-MCTS

Rollout Depth $\eta = 1$ if not stated otherwise
Search Budget τ
Number of update steps χ
procedure $act(Q, s)$
    Given: state s, set containing the explored actions $\mathcal{S}_A = \{\}$
    $\forall a$, Initialize $W(s,a) = 1$, $Q_S(s,a) = Q(s,a)$
    for i in $1..\tau$ **do**
        if $\mathrm{RandomVariable} < \epsilon$ **then**
            $a = \mathrm{RandomChoice}(\mathrm{AllPossibleActions(s)}|W(s,a) = 1)$
        else
            $a = \max_{a'} Q(s,a'|a' \in \mathrm{AllPossibleActions}(s), W(s,a') = 1)$
        Add a to $\mathcal{S}_A$, collect $r(s,a)$
        for j in $1..\eta - 1$ **do**
            $a = \mathrm{DQN} - \mathrm{act}(Q, s)$ (Alg. 2, Line 2), collect current single step reward $\tilde{r}$
            Update: $r(s,a) = r(s,a) + \gamma^j \tilde{r}$
        Update: $W(s,a) = W(s,a) + 1$, $Q_S(s,a) = \frac{1}{2}(Q_S(s,a) + r(s,a) + \gamma^\eta \max_{a'} Q(s',a'))$
    if $\mathrm{RandomVariable} < \epsilon$ **then**
        $a_r = \mathrm{RandomChoice}(\mathcal{S}_\mathrm{A})$
    else
        $a_r = \max_{a'} Q_S(s,a'|a' \in \mathcal{S}_A)$
    return $a_r, \{Q_S(s,a|a \in \mathcal{S}_A)\}, \{a|a \in \mathcal{S}_A\}$
procedure $update(\pi, \mathcal{B})$
    Add $\mathcal{B}$ to Replay Memory
    for i in $1..\chi$ **do**
        Sample random subset from Replay Memory, including the actions taken during search
        $\mathrm{loss} = \frac{1}{2}\mathrm{smoothL1}(Q(s,a) - (r(s,a) + \gamma \max_{a'} Q_T(s',a'|a' \in \mathrm{AllPossibleActions}(s))))$
            $-\frac{1}{2}\mathrm{softmax}(Q(s,a_e))^T \mathrm{softmax}(Q_S(s,a_e))$
        Update Q-function approximator Q with parameters θ
        $\theta = \theta - \alpha \frac{\partial \mathrm{loss}}{\partial \theta}$
    return Q

C Additional details on experimental setup

All the simulation environments have been implemented using PyBullet [45]. Every environment is characterized by the size and number of all the target shapes, as well as the number of blocks that are available. As most of the components used for the learning algorithms are implemented with fixed-size arrays, we decided that during training, every graph has a fixed size, i.e. a fixed number of nodes. However, as per side, the number of points describing the target shape is variable (3, 4 or 5), the number of available blocks is adapted accordingly. Thus, despite the choice of fixing the number of nodes during training, the algorithms still encounters a versatile set of graphs as the distribution over the node types changes depending on how many nodes are required for describing the desired shape.

C.1 Sampling target shape

For every side of the environment, we initially define its size. In this work, we experimented with grids ranging from size 3-by-3 up to sizes of 6-by-6. Exemplarily, a grid sized 3-by-3 has a maximum width of 3 blocks and a maximum height of 3 blocks. We assume to place the initial element (red block) always in the bottom center of this grid. If there are multiple sides to be built, the initial block is placed by randomly selecting one of them. From inside these grids, we sample 3, 4 or 5 points. Two out of these points are always sampled on the bottom, one to the left of the initial block and one to the right of it. For the upper points, we just sample points at a random height and width from inside the available grid.

Algorithm 5 ϵ-MCTS

Rollout Depth $\eta = 1$ if not stated otherwise
Search Budget τ
Number of update steps χ
procedure $act(Q, s)$
  if $\mathrm{RandomVariable} < \epsilon$ **then**
    $a = \mathrm{RandomChoice(AllPossibleActions(s))}$
    return a
  else
    Given: state s, set containing the explored actions $\mathcal{S}_A = \{\}$
    $\forall a$, Initialize $W(s,a) = 1$, $Q_S(s,a) = Q(s,a)$
    for i in $1..\tau$ **do**
      $a = \max_{a'} Q(s,a') + 2\sqrt{\frac{log(\sum_{a''} W(s,a''))}{W(s,a')}} | a' \in \mathrm{AllPossibleActions}(s), W(s,a') = 1$
      Add a to $\mathcal{S}_A$, collect $r(s,a)$
      for j in $1..\eta - 1$ **do**
        $a = \mathrm{DQN - act}(\pi, s)$ (Alg. 2, Line 2), collect current single step reward $\tilde{r}$
        Update: $r(s,a) = r(s,a) + \gamma^j \tilde{r}$
      Update: $W(s,a) = W(s,a) + 1$, $Q_S(s,a) = \frac{1}{2}(Q_S(s,a) + r(s,a) + \gamma^\eta \max_{a'} Q(s',a'))$
    $a_r = \max_{a'} Q_S(s, a' | a' \in \mathcal{S}_A)$
    return $a_r, \{Q_S(s,a|a \in \mathcal{S}_A)\}, \{a | a \in \mathcal{S}_A\}$
procedure $update(Q, \mathcal{B})$
  Add $\mathcal{B}$ to Replay Memory
  for i in $1..\chi$ **do**
    Sample random subset from Replay Memory, including the actions taken during search
/* Note: If the sample is collected without conducting search, then, for this sample, the cross entropy regularization term is omitted and instead equates to 0.
    $\mathrm{loss} = \frac{1}{2}\mathrm{smoothL1}(Q(s,a) - (r(s,a) + \gamma \max_{a'} Q_T(s', a' | a' \in \mathrm{AllPossibleActions}(s))))$
      $-\frac{1}{2}\mathrm{softmax}(Q(s,a_e))^T \mathrm{softmax}(Q_S(s,a_e))$
    Update Q-function approximator Q with parameters θ
    $\theta = \theta - \alpha \frac{\partial \mathrm{loss}}{\partial \theta}$
  return Q

C.2 Reward and reset function

Our method's objective is to use the available blocks to construct a stable structure that fills the target design up to a desired filling threshold Δ for which we view the experiment as successful. This results in a reset of the environment, i.e., sampling a new instance with new target points and blocks. The environment is also reset when there are no more unplaced blocks available, or if an invalid action has been taken. An action is invalid if it results in an unstable configuration, in destructing the previously built structure, or if it is not executable by the robot due to kinematic constraints. The outcome of every action is checked through the simulator as explained in Sec. C.5. To refine the stacking policies, we therefore have to provide a meaningful reward signal to incentivize the successful completion of the construction tasks. We have used two different reward definitions in this work, depending on the simulation environment. In the simple block stacking environments with only one type of block available, we made use of a discrete reward as introduced in Sec. C.2.1. However, in the more complicated environments with multiple different blocks available, we defined another reward function (see Sec. C.2.2) which not only returns a binary signal but actually encourages to fill the shape as efficiently as possible by providing a reward proportional to the increase in the filling.

C.2.1 Reward function in simple block stacking environments

In the simple environments with only one type of blocks available, we define the reward as

$$r(s_t, a_t) = \begin{cases} 1 & \text{if } F(s_{t+1}) - F(s_t) > 0, \\ -1 & \text{if an invalid action,} \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

where an invalid action is defined as described above in Sec. C.2. The coverage $F(s_t)$ is computed by first summing over the intersection areas for each side, i.e. the areas for which the target shape and current structure overlap, and divide it by the total area of the target shapes. The intuition behind this reward function is to provide an "intrinsic" motivation for actions that lead to improvement in the filling of the desired area, punish actions that disrupt the execution, and to not reward any actions that do not promote the assembly (e.g., placing blocks outside the desired area).

C.2.2 Reward function in environments with multiple different blocks

In environments with multiple different blocks available we want to incentivize filling the target shape using the different available modules as effectively as possible, and thus assign a slightly modified reward function according to

$$r(s_t, a_t) = \begin{cases} c_1(F(s_{t+1}) - F(s_t)) + 1 & \text{if } F(s_{t+1}) - F(s_t) > 0 \text{ and } F(s_{t+1}) > \Delta, \\ c_1(F(s_{t+1}) - F(s_t)) & \text{if } F(s_{t+1}) - F(s_t) > 0, \\ -1 & \text{if an invalid action destroying the structure,} \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

The coverage $F(s_t)$ is computed as introduced in Sec. C.2.1 and the hyperparameter (scaling constant) c_1 is set to 3. The constant has been determined empirically and is required to scale the reward signal to facilitate the learning of the action-value function. For example, setting this constant to 1 might make it difficult to differentiate an action that results in only marginally improving the filing from another one that does not improve it at all. Since exceeding the filling threshold Δ corresponds with successfully completing the experiment, we provide an additional bonus of +1 to the agent.

C.3 Populate environment without (wo) robot

Depending on the choice of target shape, as well as the number of available blocks, we have N_k blocks remaining that need to be placed. To ensure an equal spacing in between the blocks, we use Sobol sequences [46] to sample where the unplaced blocks are to be placed. Note that in the experiments without the robot, we did not enforce any measures to avoid collisions between the unplaced blocks, as they are not modelled by the physical simulator. They are only added to the simulated scene the moment they are placed.

C.4 Populate environment with (w) robot

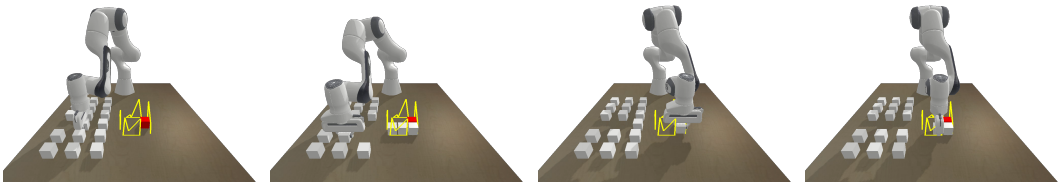
In the environment with the robot, the unplaced parts are added to the physics simulator from the very beginning, as the parts have to be grasped by the robot. To ensure graspability, we place them in rows with sufficient spacing.

C.5 Checking stability of overall structure

To check the stability of the overall structure, we track the velocities of all the parts in the scene. If blocks are colliding or falling down, the accumulated velocity will exceed a threshold, which will signal that the action has been invalid. This will reset the entire environment. In all non-robotic scenarios, this threshold on the velocity is also active for the block that is being placed, ensuring that the controller will not drop any block from above, as this will result in more inaccurate placements compared to just positioning it exactly at the right place. In the robot scenarios, and especially in the multidimensional ones, dropping a cube from a higher position is sometimes required as otherwise, the construction of enclosed shapes is impossible. Therefore, we only keep track of all the placed elements, ignoring the block that is currently being placed.

C.6 Robotic environment - Moving the robot

For moving the robot, we use trajectories that are defined by multiple waypoints. To map from the Cartesian space to joint coordinates, we use the Pinocchio library [47]. For grasping as well as placing the cubes, the respective goal positions are approached from the top, without running any additional obstacle avoidance module. As shown in Fig. 2, the objects are grasped from the top and there are two grasping poses, as well as two placing poses available. These four grasp-place



(a) (b) (c) (d)

Figure 2: Illustrating the different available grasping and placing poses in the four-sided robotic environment. **(a)&(b)** Two grasping poses. **(b)&(c)** Two placing poses.

Table 1: Hyperparameters used for training the policies

Parameter	Value
Number Training Epochs	5000
Discount Factor γ	0.8
Initial ϵ	1.0
Final & evaluation ϵ	0.05
Epoch final ϵ reached	1000
Size replay buffer	30000
Optimizer	adam
Learning rate	0.001
Batch size	32
Number of update steps χ	25
Update target (epochs)	50
Target filling threshold Δ	0.975

combinations allow to realize rotations by 0° (same orientation for grasping and placing), as well as $\pm 90^\circ$ around the z-axis.

D Additional experimental results

In this section, we will provide additional details and results underlining the main findings of our work. To better showcase the behavior of the individual agents, we also provide videos on our website `https://sites.google.com/view/learn2assemble`.

D.1 General information on training procedure

If not stated otherwise, all the results presented in the paper are based on evaluating 5 agents that have been trained with different random seeds using the parameters shown in Table 1.

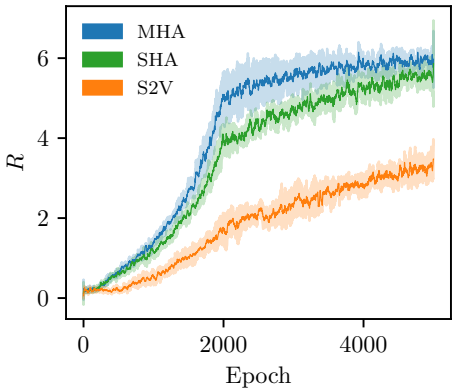
For all experimental results presented in this work, we report the mean values (and eventually the 95% confidence interval in brackets) from evaluating all the agents on building 100 randomly sampled target shapes. In the result tables in Sec.3, the star(*) indicates the environment in which the agents have been trained in, while the other experiments are OOD.

The value of ϵ is decreased linearly and the target Q-network Q_T is updated every 50 epochs, where each epoch consists of sampling 100 state-action transitions. To speed up the training process, which is especially crucial when search is added also during training time, we implemented a parallel sampling strategy by combining the code from [48] and [44]. In the following, we will also show the evolution of the discounted average return during the training process. Note that for obtaining those learning curves, we did not average over building 100 target shapes, but use the score collected from the 100 samples which are at the same time exploited to update the Q-function.

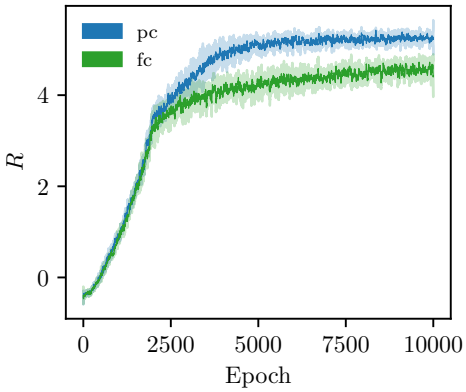
D.2 Evaluation of graph architectures

For evaluating the graph architectures, we used a slightly different set of parameters. Namely, the final exploration frame has been reached after episode 2000 and the replay buffer size has been set to 150000.

As illustrated in Fig. 3a, already throughout the training process, the discounted average return obtained by the MHA agents is significantly higher compared to using the S2V architecture. This hints that using the attention mechanism results in obtaining more powerful representations, which then result in obtaining higher rewards as the target shape can be filled more effectively. The videos



(a) Evolution of the discounted average return R when training the MHA, SHA and S2V models using the standard DQN algorithm (Alg. 2).



(b) Evolution of the discounted average return R when training the MHA in the fully-connected (fc) and partially-connected (pc) configuration using the standard DQN algorithm (Alg. 2).

Figure 3: Learning curves for different experiments.

Table 2: Comparison of different architectures on the single-sided environment without the robot and only one type of block, extending the results presented in Table 1 by also reporting scores for the single-head attention encoding (SHA). R is the cumulative discounted return, f the ratio of runs that ended with failure, i.e., the structure colliding and b, the ratio of runs that ended without success and no more blocks remaining.

Method	3-by-3 grid, 20-24 blocks*			3-by-3 grid, 30-34 blocks			4-by-4 grid, 20-24 blocks		
	R	b	f	R	b	f	R	b	f
MHA (FC)	**3.22** (0.04)	0.01	**0.15**	**3.44** (0.04)	0.00	**0.21**	**3.66** (0.05)	0.18	**0.41**
SHA (FC)	2.99 (0.09)	0.08	0.25	3.16 (0.10)	0.04	0.37	3.48 (0.08)	0.25	0.49
S2V (FC)	2.36 (0.08)	0.49	0.47	2.15 (0.10)	0.08	0.87	2.75 (0.20)	0.45	0.55

confirm these impressions and clearly show that the S2V architecture has difficulties drawing the connection between the target shape and the already placed blocks, resulting in placing many unnecessary blocks and failing to fill the target shape reliably. This underlines the findings presented in Sec. 3.

D.2.1 Additional experiment - Single-head attention (SHA)

To evaluate the effectiveness of using the multi-head attention approach, we ran one additional experiment using only a single attention head. The results are shown in Table 2. We reason that SHA performs better compared to S2V as it can explicitly compute the compatibility score between nodes which helps to disambiguate the different components encoded in the graph structure. Nevertheless, the superiority of MHA hints that having multiple attention heads with different weights allows to construct even more meaningful features which result in better performance.

D.3 Evaluation of graph connectivity

While it is quite straightforward to defined each node's features (position, type information, cf. Appx. A.2), there are different approaches for the graph connectivity. This choice is important as it influences the message passing between nodes. We investigated the effect of using a fully connected (FC) graph and compare it to using partial connections (see Fig. 1) omitting the connections between different unplaced blocks, inducing a stronger inductive bias on the structured representation.

To test the generalization abilities, we test the approaches in the setting of Fig. 4 with different types of blocks (i.e., rectangles of various lengths). Those novel objects are fixed concatenations of the already existing blocks, treating the individual boxes as primitive elements. This has the advantage of not needing any modifications to the action and observation space, while allowing us to create more complex and diverse objects.

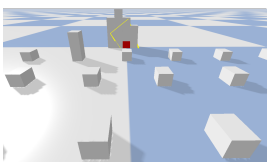


Figure 4

Table 3: Comparison of FC against partially connected MHA architecture on a task with variable sets of object-types available in the setting shown in Fig. 4.

Method	5-by-5 grid, 25-27 blocks *		5-by-5 grid, 35-37 blocks		6-by-6 grid, 35-37 blocks		5-by-5 grid, 25-27 blocks (different types)	
	R	f	R	f	R	f	R	f
MHA (FC)	1.94 (0.12)	0.39	1.69 (0.11)	0.51	1.32 (0.10)	0.61	1.88 (0.11)	0.42
MHA (partial)	**2.42** (0.03)	**0.23**	**2.22** (0.06)	**0.34**	**1.75** (0.08)	**0.51**	**2.40** (0.10)	**0.26**

We use the same set of parameters as in the previous subsection, but we trained the agents for 10000 epochs. Further, as this experiment considers also larger blocks, we use the adapted reward function introduced in Appx. C.2.2. Using the larger objects necessitates two new skills: learning to differentiate the different object types and handling them correctly.

Results. As shown in Table 3, partial connectivity outperforms the FC in all experiments. Interestingly, designing a priori meaningful edge connections favors the extrapolation of the method to unseen types of objects (last column), indicating that the chosen representation is modular and effective in solving more challenging problem settings. These results also indicate that reasoning about different block types in the FC setting is more difficult, as this has to happen based on the blocks' distances. Contrarily, in the partial setting, the connectivity information directly reveals the block type. As we want to keep the flexibility of handling more complex blocks, we will exclusively focus on the partially connected architecture.

Fig. 3b depicts the learning behaviors for the fully- and the partially-connected setup. While initially, there is hardly any difference between the two approaches, in the long run, the partially-connected setup clearly outperforms the fully-connected one. We reason that during the early stages of training, the connectivity only plays a minor role, as all the agents have to figure out which actions are admissible. However, as training proceeds, the partial connectivity gathers higher rewards, indicating that providing additional structure through the graph's connectivity facilitates learning. This trend is also visible in the accompanying videos.

D.4 Evaluation of the learning algorithms

When comparing the performance of the different learning algorithms, we use the exact same hyperparameters as introduced in Sec. D.1. Compared to the experiments presented in Secs. D.2 & D.3, we decided to decrease the size of the replay buffer and to increase the speed of decaying ϵ. Those measures were intended to increase memory efficiency as well as to improve the convergence speed. Further, for training the Q-MCTS and the ϵ-MCTS agents a search budget of 5 is being used.

The learning progress is very similar for all the introduced algorithms, as can be seen in Fig. 5. The learning curves for DQN as well as ϵ-MCTS are hardly distinguishable, which we think is due to the fact that both of them are using an ϵ-greedy decision whether to apply a random action or to either use the best action according to the Q-function (DQN) or according to the result of a quick searching procedure (ϵ-MCTS). Q-MCTS, in contrast, seems to learn slightly quicker during the early phase of the learning process due to more exploitation, as it always performs the tree search. However, this behavior might actually also result in less exploration, as always conducting tree search might result in overfitting to a suboptimal solution.

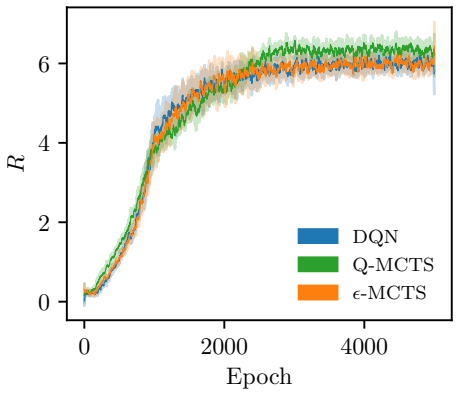


Figure 5: Evolution of the discounted average return R when training agents using the different learning algorithms.

Table 4 provides results from running additional experiments. The last two columns (most difficult tasks) reveal that using a search budget of 10, instead of 5, slightly improves performance. While using a search budget of 5 and playing the rollout until termination yields best performance across all tasks and methods, we still decided to use a search budget of 10 with only a single step of expansion for the subsequent experiments. The main reason for this choice being computational efficiency.

Table 4: Combining Q-learning and MCTS in the two-sided environment without the robot. R denotes the cumulative discounted return, $\bar{a}$, the mean number of actions conducted in this environment, f the ratio of runs that ended with failure, i.e. the structure colliding. The double star (**) indicates that the rollout has been played until termination.

Search Budget	Method	3-by-3, grid 20-24 blocks*			3-by-3 grid, 30-34 blocks			4-by-4 grid, 30-34 blocks			5-by-5 grid, 40-44 blocks		
		R	$\bar{a}$	f	R	$\bar{a}$	f	R	$\bar{a}$	f	R	$\bar{a}$	f
0	DQN	**3.21** (0.05)	7.93	0.16	**3.44** (0.08)	8.65	0.17	**3.61** (0.06)	12.02	0.51	**3.63** (0.08)	13.66	0.93
	ϵ-MCTS	3.18 (0.03)	8.57	0.22	3.37 (0.10)	9.30	0.30	3.54 (0.09)	12.53	0.62	3.50 (0.13)	12.75	0.95
	Q-MCTS	2.80 (0.15)	7.33	0.51	2.89 (0.09)	7.60	0.64	2.66 (0.08)	7.47	0.92	2.34 (0.17)	6.45	0.99
5	DQN+MCTS	**3.43** (0.04)	8.20	0.03	**3.67** (0.02)	8.99	**0.02**	**3.95** (0.04)	14.05	0.35	**3.96** (0.10)	16.05	0.91
	ϵ-MCTS	3.16 (0.06)	8.38	0.23	3.34 (0.08)	9.11	0.35	3.55 (0.09)	12.45	0.61	3.51 (0.18)	12.61	0.93
	Q-MCTS	3.07 (0.10)	8.03	0.40	3.20 (0.08)	8.35	0.50	3.10 (0.12)	8.97	0.85	2.85 (0.19)	8.27	0.98
5**	DQN+MCTS	**3.47** (0.02)	8.12	**0.02**	**3.71** (0.02)	8.83	**0.02**	**3.99** (0.04)	14.13	**0.28**	**4.04** (0.06)	17.08	**0.87**
	ϵ-MCTS	3.21 (0.04)	8.40	0.19	3.39 (0.06)	9.06	0.31	3.60 (0.06)	12.84	0.59	3.58 (0.12)	13.28	0.92
	Q-MCTS	3.20 (0.07)	8.00	0.27	3.28 (0.10)	8.48	0.43	3.28 (0.07)	9.60	0.77	3.08 (0.18)	9.23	0.98
10	DQN+MCTS	**3.47** (0.02)	8.16	0.05	**3.66** (0.03)	8.96	0.06	**3.96** (0.04)	13.92	0.31	**4.08** (0.09)	17.33	**0.87**
	ϵ-MCTS	3.21 (0.03)	8.48	0.20	3.37 (0.11)	9.10	0.31	3.60 (0.09)	12.65	0.61	3.63 (0.16)	13.16	0.93
	Q-MCTS	3.24 (0.04)	8.56	0.27	3.39 (0.08)	9.24	0.41	3.42 (0.07)	10.99	0.76	3.41 (0.13)	11.27	0.96
1000	UCT	0.54	4.00	1.00	-	-	-	-	-	-	-	-	-

Table 5: Comparing policies trained with (w) and without (wo) the robot-in-the-loop.

Method	5 by 5 grid 15-17 blocks*			6 by 6 grid 16-18 blocks		
	R	$\bar{a}$	f	R	$\bar{a}$	f
w robot	**2.71** (0.14)	5.40	**0.25**	**2.93** (0.08)	6.71	**0.42**
wo robot	2.14 (0.27)	3.92	0.40	2.40 (0.18)	4.91	0.58

The video material that we provide along this appendix illustrates the different findings, as well as the results from Sec. 3. Adding search to the DQN agent improves performance and results in slight gains compared to the ϵ-MCTS agents, whereas the Q-MCTS agents perform worse.

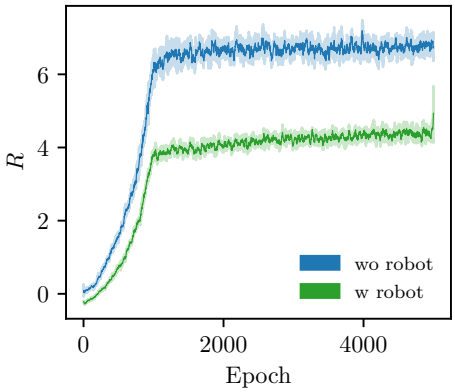
D.5 Evaluation of trainings with and without the robot-in-the-loop

In this section, we provide more details on the comparison between DQN agents trained with and without the robot-in-the-loop. Compared to the previous experiments, we decided to adapt the filling threshold indicating when to view a shape as being built successfully. For all experiments including the robot, we decided to lower this value to $\Delta = 0.75$. The reason lies in the fact that when using the robot to actually place the cubes, there will always remain some spacing in between the parts. Therefore, lowering the threshold is necessary.

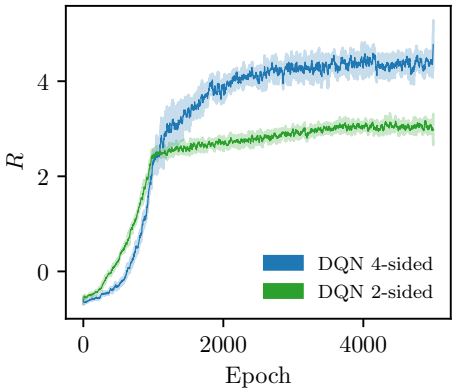
The difference in the magnitude of the obtained rewards in Fig. 6a can thus be explained by the two different threshold values used during training, as for the training without the robot, we still used the previous value of 0.975. Apart from this, there does not seem to be a distinction in the learning speed. However, when evaluating both of the policies in the environment with the robot-in-the-loop, agents trained without the robot fail significantly more often as they do not take the robot's kinematics into account (see Table 5). This results in the arm colliding with the structure as shown in the accompanying videos. Thus, as already pointed out in Sec. 3 of the main paper, for obtaining reliable policies for assembling architectural structures, it is essential to include the robot-in-the-loop already during training. The results also underline that the graph representations are flexible enough to take the robot's restrictions and nonlinearities into account while still successfully solving the tasks. Note that for this experiment we still considered the 5-dimensional action space (only consisting of the placement actions) as choosing a grasp perpendicular to the structure being built will ensure to not collide with it. This choice also ensures a fair comparison between the agents.

D.6 Evaluation of building complex shapes with the robot-in-the-loop

Lastly, we investigated building even more complex shapes with the robot-in-the-loop. Building those two- and four-sided shapes also requires grasp selection, and thus this setting is the most complex investigated in this work. Fig. 6b depicts the learning curves for the DQN agents trained on these difficult tasks. As can be seen, learning to build two-sided shapes seems to be more straightforward and results in faster training and a steeper learning curve. Nevertheless, towards the end of the training, the agents can achieve higher rewards in the four-sided environments as those contain more blocks to be placed. Since adding MCTS during test time improved performance throughout all of our experiments, we showcase the behavior of DQN+MCTS agents in the accompanying videos. Note that to better illustrate the performance of the agents, the threshold has been set to 0.9 for the



(a) Evolution of the discounted average return R when training the agents on the single-sided environment with and without the robot-in-the-loop using the standard DQN algorithm (Alg. 2).



(b) Evolution of the discounted average return R when training the agents in the two- and four-sided environment, including the robot using the standard DQN algorithm (Alg. 2).

Figure 6: Learning curves for different experiments with the robot-in-the-loop.

videos. As shown there, the agents are capable of building complex shapes and are successful at placing a large number of blocks.

D.7 Illustrating search

We present a series of pictures, which depict the searching process, i.e. which actions have been explored during search given the initial configuration shown in Fig. 7. As can be seen in Figs. 8 and 9 which show the explored grasping and placement actions respectively, the tree search explores a versatile set of actions and attempts to grasp different objects. Some of the grasps are invalid as the gripper fails to grasp the part due to collisions (see Fig. 8 a). The other grasps are valid but the associated placement actions illustrated in Fig. 9 b-d result in a different filling of the target structure. Finally, the highest reward action (grasp - Fig. 8 d, placement - Fig. 9 d)is selected and executed in the environment.

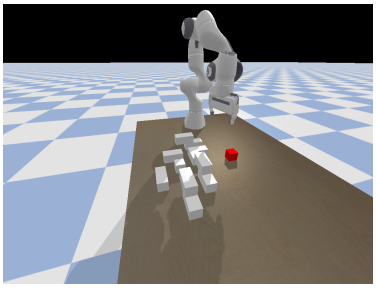
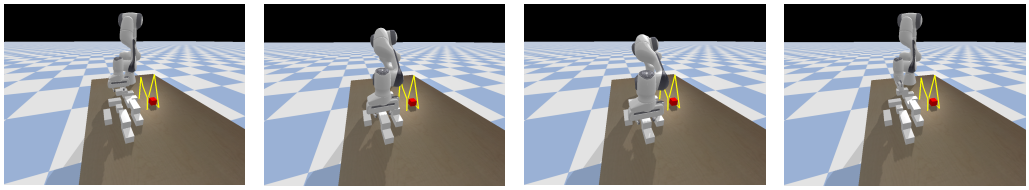
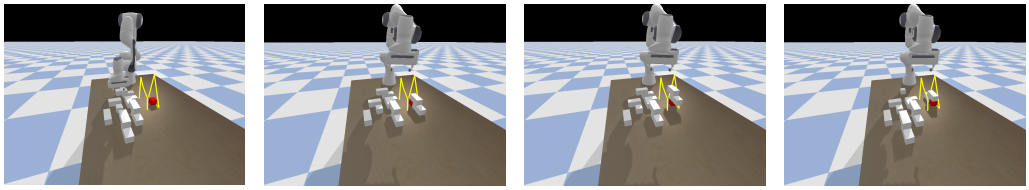


Figure 7: Initial configuration on which we start the search.



(a) (b) (c) (d)

Figure 8: **(a) - (d)** Illustrating four different grasping configurations explored during search. Only the grasp depicted in **(a)** is invalid as the gripper collides with the block.



(a) (b) (c) (d)

Figure 9: **(a) - (d)** Illustrating the result of executing the placement actions (including gripper orientation) starting from the grasps illustrated in Fig. 8. For **(a)** no placement is executed as the grasp is already invalid. Given the target shape, the placement action depicted in **(d)** results in the highest reward and is thus finally executed after the search.

D.8 Generalization with respect to randomized scenes

For the experiments on how well our learned representations scale to different scenes, we have used the exact same training configuration as for the results in Appx. D.6. In fact, we have simply evaluated the agents that have been trained in the original two-sided scenario and evaluate them in the randomized scenes. As described in the main paper, and as also shown in the accompanying videos (filling threshold set to 0.9), the agents transfer well to the novel settings.

D.9 Generalization with respect to different building blocks

Figure 10 illustrates the environment in which multiple blocks are available. As shown, there are in total 4 different building blocks available which are i) primitive box object (from previous experiments), ii) vertical block of length 2, consisting of 2 primitive blocks, iii) L-shaped object consisting of 3 primitive blocks, and iv) s-shaped object which is made up of 4 primitive blocks. In line with the previous experiments, the set of objects that is available is sampled at random from this set. This however implies that now there is no guarantee that the set of available blocks is actually sufficient to fill the shape up to the desired threshold. Also, for the objects that are not symmetric around the z-axis, i.e. L- and s-shaped block, we randomly choose an orientation from the set $\{0, \pm 90, 180\}$.

In the environments without the robot, we modify the action space, as it would otherwise be impossible for the agents to rotate the available blocks which might be crucial to successfully solve these more complicated tasks. We therefore also allow the agents to rotate the respective block around the z-axis with values of $\{0, \pm 90, 180\}$. The combination of those four rotational actions together with the five placement actions results in an action space of dimension $N_a = 4 \times 5 = 20$.

For training the agents in those complicated settings we use the reward function from Appx. C.2.2. We also found that applying the previous settings (cf. Table 1) results in converging to suboptimal

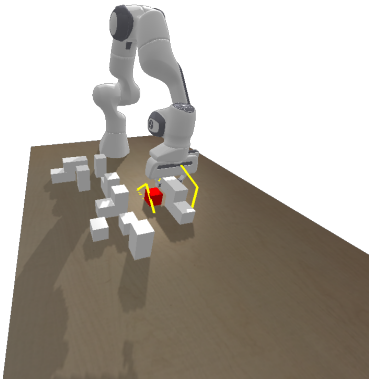


Figure 10: Initial configuration on which we start the search.

Table 6: Evaluating the trained policies in the environments with multiple different objects available. The results in row 3&4 correspond to using a modified environment without including the robot.

	5 by 5 grid 20-24 blocks*			5 by 5 grid 30-34 blocks			6 by 6 grid 30-34 blocks			
Method	R	$\bar{a}$	f	R	$\bar{a}$	f	R	$\bar{a}$	f	w/wo robot
DQN	1.25 (0.03)	3.18	0.25	0.28 (0.07)	2.95	0.67	0.11 (0.11)	3.12	0.75	w robot
DQN+MCTS	1.42 (0.05)	3.53	**0.21**	0.77 (0.09)	3.27	**0.52**	0.61 (0.08)	4.38	**0.61**	w robot
DQN	1.44 (0.21)	4.87	0.34	1.29 (0.13)	5.91	0.43	0.94 (0.15)	6.53	0.56	wo robot
DQN+MCTS	**2.20** (0.02)	5.05	**0.06**	1.89 (0.03)	5.96	**0.15**	1.48 (0.09)	7.27	**0.27**	wo robot

solutions. We thus increased the number of training epochs from 5000 to 10000, increased the epoch when the final value of ϵ is reached from 1000 to 2000, and also increased the discount factor to 0.95.

Table 6 complements the results in the paper by also reporting the performance of the DQN agents. As can be seen, also in those scenarios, the addition of search improves the agent's performance. However, in the experiments that include the robot, the addition of search is not as efficient as for the other experiments (especially considering the first column) which hints at the fact that there might be additional limiting factors. We reason that the action space might limit the agent's performance by not providing enough flexibility in terms of low-level placement actions, as well as grasp selection.

Acknowledgments

The authors acknowledge the support from the Artificial Intelligence in Construction (AICO) grant by the Nexplore/Hochtief Collaboration Lab at TU Darmstadt. This project has also received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 640554, and the Emmy Noether DFG Programme iROSA with Grant No. 448644653. The authors would like to thank the reviewers and metareviewer for their useful comments that helped to substantially improve the paper, and Prof. Dr. Oliver Tessmann from the Digital Design Unit (DDU) at the Department of Architecture at TU Darmstadt, for supporting us with the robotic experiments.

References

[1] B. G. de Soto and M. J. Skibniewski. Future of robotics and automation in construction. In *Construction 4.0*, pages 289–306. Routledge, 2020.

[2] S. Tibbits. *Autonomous assembly: designing for a new era of collective construction*. John Wiley & Sons, 2017.

[3] J. O. de Haro, V. N. Hartmann, O. S. Oguz, and M. Toussaint. Learning efficient constraint graph sampling for robotic sequential manipulation. *arXiv preprint arXiv:2011.04828*, 2020.

[4] S. Pirk, K. Hausman, A. Toshev, and M. Khansari. Modeling long-horizon tasks as sequential interaction landscapes. *arXiv preprint arXiv:2006.04843*, 2020.

[5] T. Ren, G. Chalvatzaki, and J. Peters. Extended task and motion planning of long-horizon robot manipulation. *arXiv preprint arXiv:2103.05456*, 2021.

[6] Y. Lee, E. S. Hu, Z. Yang, A. Yin, and J. J. Lim. Ikea furniture assembly environment for long-horizon complex manipulation tasks. *arXiv preprint arXiv:1911.07246*, 2019.

[7] A. Mandlekar, D. Xu, R. Martín-Martín, S. Savarese, and L. Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations. *arXiv preprint arXiv:2003.06085*, 2020.

[8] A. Simeonov, Y. Du, B. Kim, F. R. Hogan, J. Tenenbaum, P. Agrawal, and A. Rodriguez. A long horizon planning framework for manipulating rigid pointcloud objects. *arXiv preprint arXiv:2011.08177*, 2020.

[9] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[10] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana. Deep reinforcement learning for high precision assembly tasks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 819–825. IEEE, 2017.

[11] C. R. Garrett, Y. Huang, T. Lozano-Pérez, and C. T. Mueller. Scalable and probabilistically complete planning for robotic spatial extrusion. *arXiv preprint arXiv:2002.02360*, 2020.

[12] Y. Huang, C. Garrett, I. Ting, S. Parascho, and C. Mueller. Robotic additive construction of bar structures: Unified sequence and motion planning. *arXiv preprint arXiv:2105.11438*, 2021.

[13] V. N. Hartmann, O. S. Oguz, D. Driess, M. Toussaint, and A. Menges. Robust task and motion planning for long-horizon architectural construction planning. *arXiv preprint arXiv:2003.07754*, 2020.

[14] A. Jain and S. Niekum. Efficient hierarchical robot motion planning under uncertainty and hybrid dynamics. In *Conference on Robot Learning*, pages 757–766. PMLR, 2018.

[15] B. Wibranek, Y. Liu, N. Funk, B. Belousov, J. Peters, and O. Tessmann. Reinforcement learning for sequential assembly of sl-blocks - Self-interlocking combinatorial design based on machine learning. 2021.

[16] M. Burke, K. Subr, and S. Ramamoorthy. Action sequencing using visual permutations. *IEEE Robotics and Automation Letters*, 6(2):1745–1752, 2021.

[17] K. Zakka, A. Zeng, J. Lee, and S. Song. Form2fit: Learning shape priors for generalizable assembly from disassembly. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9404–9410. IEEE, 2020.

[18] J. Huang, G. Zhan, Q. Fan, K. Mo, L. Shao, B. Chen, L. Guibas, and H. Dong. Generative 3d part assembly via dynamic graph learning. *arXiv preprint arXiv:2006.07793*, 2020.

[19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[20] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[21] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

[22] T. Kipf, E. van der Pol, and M. Welling. Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*, 2019.

[23] Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*, 2021.

[24] V. Bapst, A. Sanchez-Gonzalez, C. Doersch, K. L. Stachenfeld, P. Kohli, P. W. Battaglia, and J. B. Hamrick. Structured agents for physical construction. *arXiv:1904.03177 [cs]*, May 2019. URL http://arxiv.org/abs/1904.03177. ZSCC: 0000017 arXiv: 1904.03177.

[25] G. Thomas, M. Chien, A. Tamar, J. A. Ojea, and P. Abbeel. Learning robotic assembly from cad. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3524–3531. IEEE, 2018.

[26] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, A. M. Agogino, A. Tamar, and P. Abbeel. Reinforcement learning on variable impedance controller for high-precision robotic assembly. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3080–3087. IEEE, 2019.

[27] R. Li, A. Jabri, T. Darrell, and P. Agrawal. Towards practical multi-object manipulation using relational reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4051–4058. IEEE, 2020.

[28] Y. Lin, A. S. Wang, and A. Rai. Efficient and interpretable robot manipulation with graph neural networks. *arXiv preprint arXiv:2102.13177*, 2021.

[29] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[30] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[31] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In *NIPS*, 2017.

[32] W. Kool, H. van Hoof, and M. Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[33] T. Barrett, W. Clements, J. Foerster, and A. Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3243–3250, 2020.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017.

[35] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[37] J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, T. Pfaff, T. Weber, L. Buesing, and P. W. Battaglia. Combining Q-Learning and Search with Amortized Value Estimates. *arXiv:1912.02807 [cs, stat]*, Jan. 2020. URL http://arxiv.org/abs/1912.02807. ZSCC: NoCitationData[s0] arXiv: 1912.02807.

[38] C. Browne, E. Powley, D. Whitehouse, S. Lucas, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES*, 4(1):50, 2012. ZSCC: 0002061.

[39] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[40] M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. *arXiv preprint arXiv:1906.08253*, 2019.

[41] A. S. Morgan, D. Nandha, G. Chalvatzaki, C. D'Eramo, A. M. Dollar, and J. Peters. Model predictive actor-critic: Accelerating robot skill acquisition with deep reinforcement learning. *arXiv preprint arXiv:2103.13842*, 2021.

[42] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[43] P.-A. Coquelin and R. Munos. Bandit algorithms for tree search. *arXiv preprint cs/0703062*, 2007.

[44] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, and J. Peters. Mushroomrl: Simplifying reinforcement learning research. https://github.com/MushroomRL/mushroom-rl, 2020.

[45] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[46] I. M. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 7(4):784–802, 1967.

[47] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard. The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *IEEE International Symposium on System Integrations (SII)*, 2019.

[48] F. Muratore. Simurlacra - a framework for reinforcement learning from randomized simulations. https://github.com/famura/SimuRLacra, 2020.