Learn2Assemble with Structured Representations and Search for Robotic Architectural Construction

Anonymous Author(s)
Affiliation
Address
email

Abstract: Autonomous robotic assembly requires a well-orchestrated sequence of high-level actions and smooth manipulation executions. The problem of learning to assemble complex 3D structures remains challenging, as it requires drawing connections between target shapes and available building blocks, as well as creating valid assembly sequences with respect to stability and kinematic feasibility in the robot's workspace. We design a hierarchical control framework that learns to sequence the building blocks to construct arbitrary 3D designs and ensures that they are feasible, as we plan the geometric execution with the robot-in-the-loop. Our approach draws its generalization properties from combining graph-based representations with reinforcement learning (RL) and ultimately adding tree-search. Combining structured representations with model-free RL and Monte-Carlo planning allows agents to operate with various target shapes and building block types. 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.

Keywords: Structured representations, Autonomous assembly, Manipulation

1 Introduction

Sustainable autonomous architectural assembly is an essential challenge for industry 4.0 [1]. While construction and manufacturing have been significantly automated in the last decades, the game-changer in architectural construction will come when robots exhibit intelligent assembly abilities for deciding over plans, actions, execution, and efficiency [2]. Among the different aspects of autonomous construction, our main focus in this work is the problem of autonomously assembling complex structures with robot manipulators *without a priori* defined task plans, i.e., the picking-placing sequences. For constructing abstract designs of different volumes, we must consider the combinatorics of possible actions, since the action space grows together with the number of available modules. Also, we need to check the assembled structure for stability, which requires a physics simulator. Moreover, the geometric execution of the picking and placing actions by the robot imposes constraints in 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 structure 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 devise a graph-based representation that captures the relations between target shapes and available building blocks. Particularly, we design a graph neural network (GNN) architecture with a purposefully induced inductive bias. The GNN is trained through RL to explore feasible actions, resulting in an expressive and flexible representation. Notably, when combined as a prior with 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 variable target shapes. Ultimately, we provide a robot-in-the-loop solution for the geomet-

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

ric motion generation of the sequential picking-placing actions to ensure that the agent decides on actions that are reachable and feasible for the robotic manipulator and that 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 in the scene. Our main contributions are threefold. (1) We propose an **effective graph representation** for the 3D assembly problem that is flexible enough for representing arbitrary, stable structures. (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 assembly 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, providing interesting insights on the combination of integrated learning and planning for long-horizon manipulation.

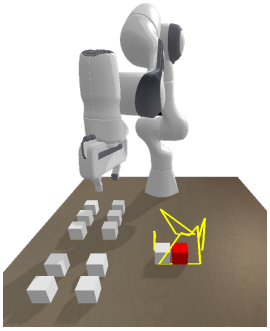


Figure 1: Simulation environment for robotic architectural assembly consisting of a 7-DoF manipulator and two sets of blocks: the unplaced white ones on the left and the "base" red block with some already placed blocks by the robot in the center. The yellow silhouette denotes the target shape that needs to be built, defined by a set of corner points. Here, the front silhouette is a pentagon, while on the back is a triangle (randomly generated). 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. Construction is a long-horizon manipulation task for which there are multiple stages of decisions and subtask executions to be made alongside the control of the dynamic execution of the assembly. 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 specific 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. A method based on neuro-symbolic planning is proposed by [15] for learning to predict sequences of actions for stacking. The authors of [16] propose an end-to-end approach for sequential pick-and-place tasks using shape correspondences and learning to assemble by collecting demonstrations from disassembling. While this work learns a sequence of high-level actions, the fact that it learns from disassembling implies that a human operator has already provided the goal of the task. Conversely, we *learn2assemble* arbitrary designs from scratch, devising high-level task sequences.

Notably, 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, can by far surpass the combinatorics of problems like chess and Go [17, 18]. The generalization power of GNNs was thoroughly explored in solving combinatorial optimization tasks [19], with successful applications in 2D assembly [20] when combined with RL. In robot learning, earlier works use deep RL for short-horizon challenges like peg-in-a-hole [10, 21, 22]. In [23] 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, this work does not learn the sequence of the building blocks for constructing arbitrary structures, but it is essentially a goal-based method for sequential pick-and-placing with known goal conditions. In a similar direction, the authors of [24] 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?"*. A typical setup of the problem is illustrated in Fig. 1, which visualizes 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). In our MDP the desired shape that needs to be assembled consists of a set of points parameterizing this target shape $\mathcal{S}_T = \{\mathbf{x}_{T_i} | i \in N_i\}$, a set of blocks that are already placed in the scene $\mathcal{S}_P = \{\mathbf{x}_{P_j} | j \in N_j\}$, and a set of unplaced blocks $\mathcal{S}_U = \{\mathbf{x}_{U_k} | k \in N_k\}$ which are still to be added, where $\mathbf{x} \in \mathbb{R}^5$ includes the 3D position and two booleans indicating the type of block (placed/unplaced, target/non-target element). 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 actions per block contain five possible placements (on top, on the left, on the right, behind, in front) w.r.t. already placed blocks. Moreover, in our MDP, the execution of the block's placement is operated by the robot (Fig. 1). Hence, the action space is augmented with two possible grasping poses. As we need to accommodate both for the picking pose and the placing position and orientation, we have to decide over $N_a = 2 \times 2 \times 5 = 20$ actions per block. However, since the possible actions a_t depend on the currently placed blocks and the target shape we want to fulfill, we end up with a discrete, time-varying action space of size $N_j \times N_k \times N_a$. The objective is to use the available blocks to construct a stable structure that fills the desired target design in its entity, which can be defined by a reward function

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

with the coverage of the target shape $F(s_t)$. The coverage is computed based on depth images and is defined as the intersection of the target shape and current structure. The environment is reset if a threshold Δ is exceeded, or whenever there are no more unplaced blocks available, or after an invalid action. 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.

2.1 Graph representation

Graph-based representations [25] are an effective tool when dealing with combinatorial problems [26, 27, 28]. 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 [29, 30]. In general, a GNN receives as input a graph $G = (\mathcal{N}, \mathcal{E})$, and returns 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 of the environment, 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.

Multi-head Attention Encoding (MHA). Attention mechanisms [29] were introduced in GNNs [30] to enable nodes to attend over their neighbors' features and learn different weights to different nodes without requiring costly matrix operations. Such graph attention networks are effective for solving combinatorial problems, like traveling salesman, when combined with policy search [27]. This architecture provides more flexibility, especially in how the information is accumulated at each node, which can benefit our combinatorial assembly task. In the first step of this method, the initial node embeddings $n_i^{(0)} = \mathbf{x}_i$ are projected into a higher dimensional space by

$$n_i^{(1)} = g(n_i^{(0)}) = \mathrm{ReLU}(\mathrm{FC}(n_i^{(0)})), \tag{2}$$

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. Next follow l rounds of message passing to obtain the final node embedding according to the operation

$$n_i^{(l)} = h(g(h(\mathrm{MHA}({n_i}^{(l-1)})))), \tag{3}$$

with a skip connection layer $h(f(x)) = x + f(x)$ and a multi-head attention mechanism MHA introduced below. As also illustrated in Fig. 2, for a single attention head, from which there exist in total M, with input ${n_i}^{(l-1)}$ of the ith node, we first compute three values – the key k, query q, and

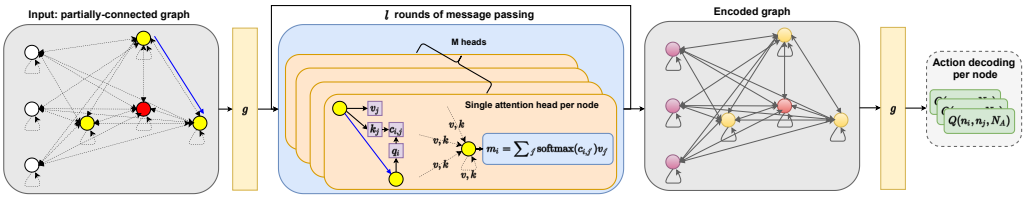


Figure 2: GNN architecture illustration, mapping from the input graph to Q-values used for action decision. The coloring follows Fig. 1. Yellow notes target elements, red the initially placed and white the unplaced blocks. After an initial projection into a higher dimensional space follow l rounds of message passing using MHA. This results in an encoded version of the graph, which is then exploited for action decoding and selection.

value v – using three different weight matrices W_k, W_q, W_v, respectively ($k_i = W_k {n_i}^{(l-1)}$, $q_i = W_q {n_i}^{(l-1)}$, $v_i = W_v {n_i}^{(l-1)}$). Multiplying the key and query of all nodes results in a compatibility score $c_{i,j}$ for each node-to-node connection

$$c_{i,j} = \begin{cases} \frac{1}{d} q_i^T k_j, & \text{if } \mathcal{E}(i,j) = 1, \\ -\infty, & \text{otherwise,} \end{cases} \tag{4}$$

with d a normalizing constant. From this score we can then compute the attention weights using a softmax $a_{i,j} = \frac{e^{c_{i,j}}}{\sum_{j'} e^{c_{i,j'}}}$ to aggregate the values for each node resulting in the output $m_i = \sum_{j'} a_{i,j} v_j$. Aggregating and weighting the output messages from all M attention heads yields the final result of the MHA module

$$\text{MHA}({n_i}^{(l-1)}) = \sum_{o=1}^{M} W_{m,o} m_{i,o}, \tag{5}$$

with weights $W_{m,o}$ controlling the influence of each one of the M single-head attention heads. Fig. 2 depicts a simplified example of the encoding performed by the MHA architecture.

2.2 Learning algorithms

GNNs can only unfold their potential when combined with algorithms that refine their weights to form expressive representations, exploited herein for action selection (Fig. 2). As our goal is to obtain performant assembly policies, the representations should be shaped based on the reward signal, thus, resulting in a RL setup. This forms a powerful combination which results in agents that (i) can be applied to different problem instances due to the representations' 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 [31]. Moreover, we investigate its integration with model-based planning, as it has been shown to provide increasing advantages [17, 32].

Action Decoding. The encoded graph representation from Sec. 2.1 needs further processing to decide on the next action to take. As in this work, actions are defined relative between unplaced and placed blocks (i.e. nodes) we exploit their feature representations and assign a value to all the available actions

$$Q(n_i, n_j, N_a) = g\left({n_i}^{(l)}, {n_j}^{(l)}, \text{FC}\left(\frac{1}{N}\sum\nolimits_{j'} {n_{j'}}^{(l)}\right)\right) \qquad \forall n_i \in \mathcal{S}_\mathcal{U}, n_j \in \mathcal{S}_\mathcal{P} \tag{6}$$

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 problems with different sizes.

DQN: Model-free Q-learning has been successfully applied to complex tasks such as playing Atari games from images [31]. As we are dealing with a discrete set of actions, the use of this value-based algorithm is a natural choice. The algorithm is off-policy and comes with a replay memory that benefits learning, especially for our case where target shapes change every time the environment is reset. 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 $a_s \in N_a$ describing the relative placement.

MCTS: MCTS [33] has proven effective for solving tasks with discrete action spaces [34], 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. Thus, in its original setting, MCTS does not require any form of function approximation. However, in complex scenarios, where the branching factor is significant or the computational complexity of the model is high, the time complexity quickly increases, and the use of MCTS becomes intractable. Consequently, in [17, 18] the combination of Q-learning for computing the value of the nodes in MCTS showed 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 [18, 32, 35, 36]. In this work, inspired by these advantages, we explore different algorithmic variations of the interplay between Q-learning and planning, as described below.

1) DQN+MCTS: Inspired by [17], 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. The prominent solution for MCTS expansion is based on the exploration strategy of UCT [17, 37], which encourages an exploration-exploitation balance, providing optimal policies for exponentially increasing tree search. However, when UCT is restricted to smaller search budgets, it becomes over-optimistic [38]. As we seek to effectively combine learning and planning for autonomous assembly with the robot-in-the-loop, we use a small search budgets. 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. Following [18, 32], we combine DQN with MCTS during training to perform an informed exploration and collect good samples through tree search, that potentially allows us to explore unvisited states in the environment. However, as stated previously, small search budgets that keep MCTS computationally feasible, do not favor UCT implementation. Therefore, we have implemented a similar approach using an ϵ-greedy expansion strategy over unexplored states. 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. $\mathbf{loss} = -\mathbf{softmax}(Q_S(s_t, a_t))^T \log(\mathbf{softmax}(Q(s_t, a_t)))$. The intuition behind this objective is to not only shape the Q-function towards reasonable value estimates for the executed actions, but also for the actions solely explored during search. This should also keep the search itself efficient, as the expansion strategy heavily depends on the initial Q estimate for all the available actions.

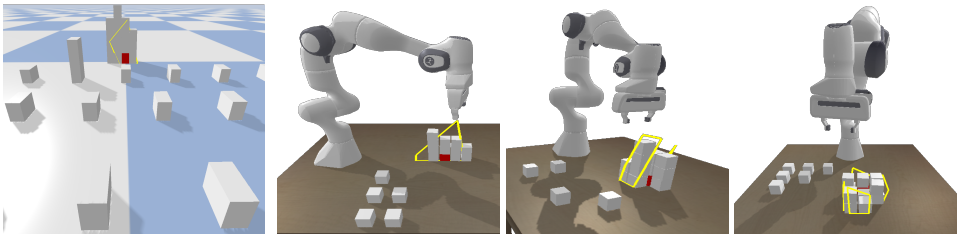
3) ϵ-MCTS: This implementation follows [32] more closely. The main differences w.r.t. the Q-MCTS policy are that first, there is an ϵ-greedy decision whether to do search or uniformly sample a random action, whereas Q-MCTS always conducts search. Secondly, when performing the search, ϵ-MCTS follows the UCT expansion strategy. The Q-learning objective remains the same as with Q-MCTS, however, only for samples where search has been conducted the cross entropy loss is added. 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 goal shapes, i.e., a robot able to abstract 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 according to (1). 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 learning algorithm, we designed specific experimental scenarios to progressively examine their contributions. We start with an investigation over graph architectures and their connectivity, continuing with the advantages of the different learning methods in environments for 3D assembly where a robot was not yet included. Selecting the best settings



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

Figure 3: **(a)** Example of the single-sided target-shape environment with different block types without robot. **(b)** Illustration of the single-sided 3D assembly environment, **(c)** the two-sided 3D assembly environment, and **(d)** the four-sided 3D assembly environment with the robot-in-the-loop.

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.

	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

Table 2: Comparison of FC against partially connected MHA architecture on a task with variable sets of object-types available in the setting shown in Fig. 3a.

	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)	
Method	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**

Star(*) marks the environment where the agents were trained.

from the previous tests, we experiment with the robot-in-the-loop for our final empirical evaluations of the proposed algorithm.

Simulation environments. The assembly environments are depicted in Fig. 3. They all consist of a set of already placed blocks, a set of unplaced blocks (placed away from the existing structure), as well as some points characterizing the target shape that needs to be built (yellow sketches). All objects in the scene, i.e. placed, unplaced blocks, and target points, are described by their 3D position and two booleans indicating the type. The scene is always initialized with one initially placed element, marked in red, serving as reference frame and building base. The target shape is represented by 3, 4, or 5 points on each side of the assembly scene, resulting in one-, two- and four-sided setups. The points representing the shape are sampled randomly from grids of different sizes, ranging from 3-by-3 (i.e. maximum height and width of the shape is 3 blocks) to 6-by-6. In all environments without the robot, the placing is done by directly setting the block's new position. Except for the two- and four-sided robotic scenario, the action space is reduced to $N_a = 5$ as the grasping and placing poses do not need to be specified. For evaluating the progression of the assembly, we use depth cameras, placed on each side of the specified target structure. By projecting the target shape points into the depth images after each action, we evaluate the change in the target shape's filling, assigning a reward according to (1).

Evaluation of Graph Architectures. We evaluate the proposed MHA representation against the commonly used Structure2Vector (S2V) architecture [20, 26]. To evaluate the different architectures, we use a simplified version of the environment shown in Fig. 3a, as we only consider one type of object and no robot. In this setting, the learning is conducted with DQN.

Results. Table 1 shows the obtained results. The first column, replicating the same setting as in training, shows that the MHA approach outperforms S2V significantly. The high rates of failure f and exceeding the number of available blocks indicate that the S2V architecture cannot represent the connection between the target shape and the current structure. This might be due to S2V's different message passing, as it 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. However, we must note that MHA is better in handling an increasing number of blocks compared to larger target structures. This might be due to the fact that MHA focuses more on the connection between the current structure and the target, rather than the available number of blocks. Due to our findings, we continue our experimentation using the MHA architecture.

Evaluation of Graph Connectivity. To successfully solve the downstream assembly task, the initial graph has to be transformed into an informative high-dimensional representation through the

Table 3: 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* 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
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	-	-	-	-	-	-	-	-	-

graph encoding-decoding architecture, described in Sec. 2. While it is straightforward to define the features of each graph node, there are different approaches for the nodes' connectivity. We investigated the effect of using two sets of connectivity information that are important as they influence the message passing between nodes. First, we devised a fully connected (FC) graph, including self-connections. Second, we designed a graph with partial connections (see Fig. 2), inducing a stronger inductive bias on the structured representation. To test the generalization abilities, we test the approaches in the setting of Fig. 3a with different types of blocks (i.e., rectangles of various lengths). We decided that 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, but it still allows us to create more complex and diverse objects.

Results. Table 2 shows the results of the connectivity experiments. 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 of the unplaced blocks also reveals their type. As we want to keep the flexibility of handling more complex blocks, we will exclusively focus on the partially connected architecture.

Evaluation of learning algorithms. To investigate the efficiency of the learning algorithms (Sec. 2.2), we will use the two-sided environment shown in Fig. 3c but without having the robot. However, we will now evaluate the agents in more challenging settings, i.e., exposing them to more difficult scenarios in terms of the size of the target shapes and the number of available blocks.

Results. Table 3 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 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 the methods. Adding a search budget of 5 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. Q-MCTS with its constant additional cross-entropy regularization throughout the whole training process results in worse performance, which we believe is due to overfitting to the respective task. The last row of the table confirms the combinatorial complexity of our problem, illustrating that performing pure UCT without any prior and a search budget of 1000 performs significantly worse, when tested to the simplest setting.

Evaluation of *learn2assemble* with robot-in-the-loop. Finally, we evaluate the DQN agent combined with tree-search in our original environment that contains the robot manipulator. Thus, the task difficulty increases as we now also have to specify the grasping and placing poses to ensure the feasibility and stability of the structure. We incrementally increase the experiment's difficulty. We start presenting results on constructing a single-sided design (Fig. 3b) to compare whether training with the robot-in-the-loop is necessary. Then, we evaluate the proposed method in OOD structures with multi-sided designs (Fig. 3c & 3d), aiming to showcase our pipeline's generalizability.

Results. In the first experiment, we compare two policies, one trained with and the other without the robot-in-the-loop, and in plain DQN (no addition of search). As shown in Table 4, the agent trained with the robot-in-the-loop outperforms the other agent. This result underlines the necessity of including geometric planning during training for high-level decisions to complete the construction task successfully. It is not sufficient to only figure out where to place the parts; the kinematic constraints and robot motion should be considered.

Table 4: Comparing policies trained with (w) and without (wo) the robot-in-the-loop.

	5 by 5 grid 15-17 blocks*			6 by 6 grid 16-18 blocks		
Method	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

Table 5: Comparing policies with and without tree search on the four-sided robotic environment.

	3 by 3 grid 10-18 blocks*			3 by 3 grid 16-24 blocks		
Method	R	$\bar{a}$	f	R	$\bar{a}$	f
DQN	2.57 (0.18)	7.25	0.45	2.44 (0.16)	7.15	0.52
DQN+MCTS	**2.91** (0.17)	8.09	**0.32**	**2.85** (0.09)	8.33	**0.38**

In the subsequent experiments, we build 3D structures with the robot using the entire action space as shown in Figs. 3c & 3d. We investigate again whether the combination with MCTS improves performance. Our results in Tables 5 and 6 show that MCTS (search budget of 10) consistently improves performance in terms of higher returns and lower failure rate. In particular, those 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 the DQN+MCTS approach 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 that adding a soft-placing controller might be beneficial. All algorithmic solutions are presented in detail in the appendix.

Table 6: Comparing policies with and without tree search on the two-sided robotic environment.

	3 by 3 grid 10-14 blocks*			3 by 3 grid 14-18 blocks			5 by 5 grid 14-18 blocks		
Method	R	$\bar{a}$	f	R	$\bar{a}$	f	R	$\bar{a}$	f
DQN	2.01 (0.23)	5.96	0.36	1.84 (0.23)	5.98	0.46	1.96 (0.31)	7.12	0.77
DQN+MCTS	**2.27** (0.24)	6.43	**0.27**	**2.17** (0.23)	6.82	**0.40**	**2.79** (0.27)	9.31	**0.55**

Remarks. We have investigated different graph representations and their combinations with learning methods for solving combinatorial assembly tasks with the robot-in-the-loop. Our results demonstrate how strong inductive biases combined with attention can shape a meaningful relational representation. When combined with deep Q-learning, this representation allows us to take decisions over long horizons despite an increasing action space. The addition of search with small budgets during training seems to hinder the expressivity of the learned Q-function, constraining it into learning locally poorer values. 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. 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. Our proposed algorithm benefits from an effective graph attention representation that captures the spatial relationships between target construction designs and unplaced blocks, which trained through deep Q-learning can extrapolate to previously unseen target shapes, deal with a larger number of available elements, and different object types. Moreover, when combining the learned Q-network with MCTS with small budgets, to keep the search computationally tractable, we manage to improve performance and reliability across all tasks. Our method is a hierarchical controller with high-level learning over action sequences and goal specifications combined with low-level path planning that ensures the execution of long-horizon 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 combine the high-level policy with a learned termination function, learn fine-placing 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

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

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

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

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

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