Graph-based Reinforcement Learning meets Mixed Integer Programs: An application to 3D robot assembly discovery

Niklas Funk, Svenja Menzenbach, Georgia Chalvatzaki, and Jan Peters

***Abstract*— *Robot assembly discovery (RAD)* is a challenging problem that lives at the intersection of resource allocation and motion planning. The goal is to combine a predefined set of objects to form something new while considering task execution with the robot-in-the-loop. In this work, we tackle the problem of building arbitrary, predefined target structures entirely from scratch using a set of Tetris-like building blocks and a robotic manipulator. Our novel hierarchical approach aims at efficiently decomposing the overall task into three feasible levels that benefit mutually from each other. On the high level, we run a classical mixed-integer program for global optimization of block-type selection and the blocks' final poses to recreate the desired shape. Its output is then exploited to efficiently guide the exploration of an underlying reinforcement learning (RL) policy. This RL policy draws its generalization properties from a flexible graph-based representation that is learned through Q-learning and can be refined with search. Moreover, it accounts for the necessary conditions of structural stability and robotic feasibility that cannot be effectively reflected in the previous layer. Lastly, a grasp and motion planner transforms the desired assembly commands into robot joint movements. We demonstrate our proposed method's performance on a set of competitive simulated RAD environments, showcase real-world transfer, and report performance and robustness gains compared to an unstructured end-to-end approach.**

I. INTRODUCTION

A common desire amongst many industry sectors is to increase resource efficiency. The construction industry is a key sector that could significantly reduce its environmental impact by re-using existing material more efficiently, moving towards the ideas of circular economy [1]. There is a fundamental need for combining intelligent algorithms for reasoning on how existing material can be recombined to form something new, with autonomous execution [2].

Herein, we are concerned with the problem of autonomous *robotic assembly discovery* (RAD), where a robotic agent should reason about abstract 3D target shapes that need to be fulfilled given a set of available building blocks (cf. Fig. 1). Unlike other assembly problems with known instructions, in RAD, the agent does neither have any prior information about which blocks to use and their final poses, nor about the execution sequence. Contrarily, the RAD agent should *discover* the possible ways of combining the building blocks, find appropriate action sequences, and put them into practice. RAD can thus be structured into two difficulty levels. On the high level, a goal-defined resource allocation problem has to be solved, which is typically NP-complete for discrete resources, and can be viewed as a real-world version of the Knapsack Problem [3]. The low level requires solving a constrained motion planning problem, considering kinematic feasibility and structural stability.

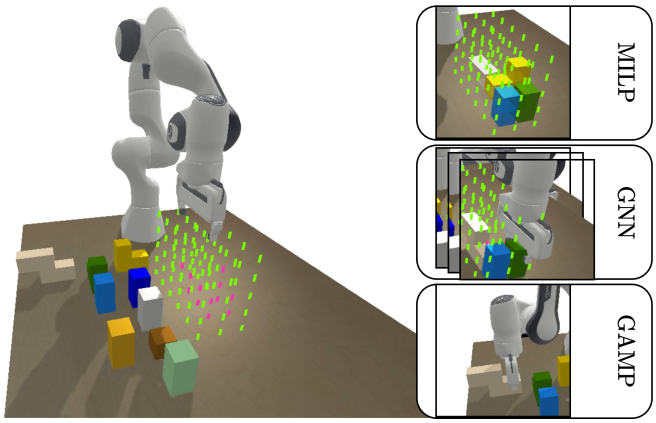

Fig. 1. Simulated RAD environment (left) and all three components of our proposed hierarchical approach (right). On the highest level, we solve an MILP to determine the building blocks' poses to optimally fill the voxelized, desired target shape (visualized in pink/green). Next follows a learned GNN policy determining which block to move next based on the scene information and MILP solution. Lastly, we run a GAMP to determine how to grasp the chosen block and realize the robot movement on the joint level.

One way of approaching this problem are end-to-end approaches that directly map from problem definition to low level actions [4], [5]. Such approaches are typically straightforward to design, and draw their generalization properties from learned graph-based representations. Yet, they often require extensive training due to the huge combinatorial action space, and are typically hard to debug and interpret. On the other end of the spectrum are Task and Motion Planning (TAMP) approaches [6], [7], which naturally represent the problem's hierarchical nature and necessitate full prior knowledge of geometry and kinematics. Yet, they are usually unsuitable for real-time reactive control, as the full joint optimization suffers from combinatorics and non-convex constraints.

We propose a novel hierarchical and hybrid method for 3D RAD that addresses both, resource allocation and motion planning. Namely, on the high level, a model-based mixed-integer linear program (MILP), handling the process of block-type selection and optimizing the blocks' final poses

*This work is supported by the AICO grant by the Nexplore/Hochtief Collaboration with TU Darmstadt, and the Emmy Noether DFG Programme (No. 448644653). Calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt.

Department of Computer Science, Technical University of Darmstadt, {niklas,georgia,jan}@robot-learning.de

for optimally resembling the desired target shape, is solved. The MILP's solution is then used as a *guiding exploration* signal in a graph-based Reinforcement Learning (RL) framework. We define a graph neural network (GNN) for capturing the geometric, structural, and physical relationships between building blocks, robot, and target shape, thereby incorporating all effects that have not been modelled on the higher level. The GNN is trained through model-free Q-learning and can, therefore, efficiently reason about the action sequencing, besides enabling the integration with tree search for improved long-term decisions [8]. To put the previous reasoning into practice, at the lowest level, we rely on a simple grasp and motion planning (GAMP) method [9] jointly optimizing grasp pose selection and end-pose motion generation.

To summarize, our proposed approach **(i)** benefits from the combination of global structural reasoning together with local, sequential decision-making, **(ii)** deals efficiently with the huge action space by skipping the complexity of determining the assembly sequencing on the high level, though inducing a strong *inductive bias* for the RL problem while exploiting GAMP at the lowest level, **(iii)** allows transfer and generalization to instances with different target shapes and types/numbers of blocks, as all levels are invariant to problem size, and **(iv)** provides the flexibility of adding search to further increase reliability and robustness. We present an empirical evaluation of our proposed approach in a set of competitive simulated RAD tasks and demonstrate real-world transfer. The results show superior performance against both empirical and learned baselines, thereby underlining its effectiveness.

II. RELATED WORKS

Due to their practical relevance, assembly and resource allocation tasks are studied amongst a wide area of communities. Researchers have proposed methods based on TAMP [6], [7] to solve the problem of motion generation for long-horizon tasks [10]. The tasks involve the combination of multiple manoeuvres, such as handovers, tool use, or even multi-robot coordination [11], [12]. Yet, TAMP suffers from a combinatorial barrier in the search space and high computational costs as TAMP attempts the full optimization of a hybrid problem, in which the high level variables influence the lower level through constraints. Thus, TAMP approaches rely on full prior knowledge about geometry, kinematics, and desired goal state. To reduce the combinatorial barrier, recently, methods combining learned heuristics with classical optimization were proposed [13], [14], [15], [16]. In contrast to TAMP problems, RAD comes at the additional complexity of not only having to decide on the sequencing of the placement actions and generating feasible motions, but also requires the optimization for the part placement poses.

Another line of research exclusively investigates the problem of optimal part placement [17], [18], [19] due to its practical relevance in logistics. A common theme along these works is the main focus on placing rectilinear objects into a convex domain for which they present MILP formulations. There are two types of formulations: position-based ones in which all objects can be placed at continuous positions, and grid-based ones based that voxelize the packing volume. The latter formulations are usually more practical as their relaxed solutions are tighter, simplifying the challenge of finding an integer feasible solution [17]. The authors of [17] introduce formulations for both cases and provide a discussion on how irregular items can be approximated with Tetris-like shapes, while [19] presents extensions for incorporating additional constraints. Recently, [20] investigated the problem of bin-packing irregular-shaped objects arriving in a nondeterministic order. While they present effective heuristic strategies, taking time and sequencing into account, they do not consider the physical action execution with a robot.

Reasoning about how to combine elements to resemble a given structure is also a crucial challenge amongst the machine learning community [21]. Variants of this problem have been explored in the context of RL, e.g. in [22], [23], [4]. In particular, GNNs in RL frameworks are very promising. Due to their ability to learn relational encodings [24], [25] and invariant representations, they can overcome combinatorial barriers [26], and be combined with search for improved generalization and robustness [5], [4], [8]. Indeed, in our prior work [5], we proposed a novel multi-head attention GNN, that when trained with Q-learning and combined with Monte Carlo Tree Search (MCTS), can effectively learn to solve RAD instances in an end-to-end fashion. Yet, the action-space combinatorics induce a very hard exploration problem and thus limit end-to-end RAD. Apart from our own work, the previous methods do not consider 3D scenarios and robotic part placement.

Contrarily, [27] proposes a method for local assembly from camera images through template prediction. Similarly, [28] focuses on building block towers of variable heights through structured representations and model-free RL. The authors of [29] also address block-stacking but use learning from demonstrations to train two GNNs to split the tasks of selecting the next object and the respective pre-defined goal location. [30] learn an end-to-end policy from human demonstrations for executing sequential picking and placing given shape correspondences. A method for next-best pose estimation for stone stacking is presented in [31]. While these works all consider robotic execution, they do not consider the additional challenge of placing versatile blocks to achieve a desired global goal configuration, which is only specified on an abstract level and does not reveal how the individual parts need to be arranged.

To the best of our knowledge, only our previous work [5] addresses RAD from a robot learning perspective. Yet, herein, we explore a different direction through proposing a novel structured, hierarchical approach by fusing global MILP optimization with learning local graph-based RL assembly policies, that combined with low level GAMP can reliably deal with complex RAD instances.

III. PROBLEM DEFINITION

We formulate the problem of having to combine a set of rectlinear, Tetris-like blocks into a desired target shape (cf.

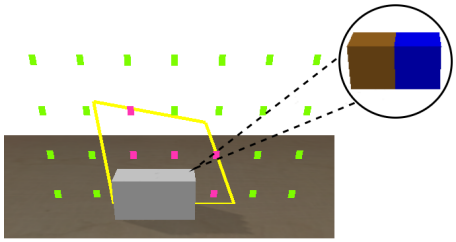

Fig. 2. Simplified 2D RAD scene with one placed block consisting of two primitive units (underlined by additionally visualizing them in brown and blue in the top right). Those two primitive units form the set of placed elements $\mathcal{S}_P$. In yellow, we showcase the area that is to be filled, which is discretized through a grid. The grid cells are visualized through their centre points. The pink points correspond to the set of target grid-cells ($\mathcal{T}_F$), i.e. the set of cells that should still be filled as they are part of the desired shape, while the green points represent the so-called non-target grid-cells ($\mathcal{T}_E$) that should ideally remain unoccupied as they are not part of the target shape and as another goal is to also avoid unnecessary placements.

Fig. 1) as a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$ [32] with state and action space, $\mathcal{S}, \mathcal{A}$, transition probabilities p, reward function r, and discount factor γ. To capture the assembly scene's current configuration, the state s is given by the combination of four sets, $s = (\mathcal{S}_U, \mathcal{S}_P, \mathcal{T}_F, \mathcal{T}_E)$, with $|\mathcal{S}_U|{=}N_U, |\mathcal{S}_P|{=}N_P, |\mathcal{T}_F|{=}N_F, |\mathcal{T}_E|{=}N_E$, where the set $\mathcal{S}_U$ encodes the unplaced primitive units that are still available for construction, $\mathcal{S}_P$ the primitive units that have already been used, $\mathcal{T}_F$ and $\mathcal{T}_E$ contain the so-called target grid-cells and non-target grid-cells, respectively. As shown in Fig. 2, we voxelize the building area around the desired target shape and, thus, end up with these grid-cells, which are parameterized through their respective 3D center coordinate $\boldsymbol{x} \in \mathbb{R}^3$, i.e. $\mathcal{T}_F{=}\{\boldsymbol{x}_i, i \in N_F\}$, $\mathcal{T}_E{=}\{\boldsymbol{x}_i, i \in N_E\}$ (visualized in pink and green). While the target grid-cells (pink) are part of the target shape and should ideally all be filled during RAD, the non-target grid-cells (green) should remain unoccupied. By projecting all grid-cells centre coordinate $\boldsymbol{x}_i$ into the yellow target shape, we decide whether it should be occupied or remain unoccupied during RAD.

In this work, we assume that all building blocks are a combination of primitive units. More specifically, we consider that there is only one type of primitive unit: a unit cube $u_c = 1^3$. Thus, all the blocks in the scene are a combination of primitive units (cf. Fig. 2), i.e., block i is defined by the union of N_{b_i} primitive units, $b_i = \bigcup_{j=1}^{N_{b_i}} u_c$. Representing blocks as concatenations of primitives allows for a universal interface with graph-based representations, as any Tetris-like block can be easily represented. Simply put, each primitive unit induces a node in the graph, and the connectivity information encodes whether or not multiple primitive units form a lager block (cf. two leftmost frames in Fig. 3). This choice also allows us to describe the placed and unplaced blocks through the primitive units' 3D positions $\boldsymbol{x}_k$, and connectivity information $y_k = [y_{k,1}, ..., y_{k,N_U}]$, i.e., $\mathcal{S}_U{=}\{(\boldsymbol{x}_k, y_k), k \in N_U\}$. If primitive unit k is connected with primitive unit 1 to form a larger block $y_{k,1}$ equals 1, otherwise $y_{k,1}{=}0$. We follow the same procedure for the set of already placed elements $\mathcal{S}_P$.

For placing blocks in the scene, we use a discrete, time-varying action space. In particular, every primitive unit which is at the moment unplaced, can be placed w.r.t. all available grid-cells. As more complicated blocks might also require rotations, we augment all placement actions with four rotational actions, i.e. rotating the block by $0, \pm 90$, or 180 degrees around the upward-pointing z-axis. Furthermore, we add one termination action that enables the agent to indicate that the current assembly is finished or not possible to continue, as there are no feasible actions left. Thus, the resulting action space contains $N_a = N_U \times (N_F + N_E) \times 4 + 1$ actions. Note that the MDP is focused on high level decision making. It does not account for the low level motion generation, namely grasp selection and robot motion planning, as this would further increase the already large action space. Nevertheless, given the action, the motion generation problem is well defined as it specifies the block that is to be moved, the required relative change in orientation, and its placement location. After every placement action, all primitive units belonging to the moved block are transferred from the set of unplaced elements to the set of placed ones. We also update the set of grid-cells by removing all cells that are now occupied.

On every successful placement action, we assign a reward of $r(s_t, a_t) = 0.2(N_{F_t} - N_{F_{t+1}} + N_{E_{t+1}} - N_{E_t})$, thereby giving a positive signal when the action reduced the number of target grid-cells, while also penalizing unnecessary filling of non-target grid-cells, therefore actively enforcing resource efficiency. The conditions for a successful placement action are that the block can be placed by the robot without moving or colliding with any other block, and that it is placed in a stable configuration (i.e. the resulting structure is not falling apart due to gravity). If the agent acting in the environment decides upon an invalid action, the episode is terminated and a reward of -1 is assigned. Otherwise, the episode is terminated upon the events of i) the agent choosing the termination action, ii) no more available building blocks, or iii) the completion of RAD, i.e., the filling of all target grid-cells. As the last case corresponds to the desired behaviour, we increase the final reward by $+1$ upon this event. Finally, to reflect the long-horizon of the considered task, we set the discount factor γ to 0.999.

IV. METHOD

To reliably solve RAD, we introduce our proposed tri-level hybrid approach that efficiently handles the huge action space and combines global decision-making, considering the overall goal, with local decision-making regarding the assembly process and sequence (cf. Fig. 1). Those requirements are also inspired by the findings of our previous work [5] in which we discovered increasing difficulties when attempting to solve more complex RAD instances with different block types and bigger target shapes in an end-to-end fashion. As we attribute the difficulties to the fact that the combined learning of the global and local policy renders a challenging exploration problem, in this paper, we propose to address these issues through our novel structured hierarchical approach. In the following, we will describe the method's

all three levels, starting with the MILP formulation for resolving the global resource assignment problem, followed by a flexible, learned GNN for task sequencing, and conclude with the low level GAMP module for handling the robotic execution on the joint level.

A. MILP for optimal geometric target filling (high level)

In the first step, we solve an MILP which is targeted at optimizing the blocks' placing poses to optimally fill the desired shape in light of the problem's combinatorial complexity. However, to render the problem tractable, we do not consider the sequencing and robotic constraints, therefore, only reasoning on the geometric level. MILP formulations have been successfully applied for solving related tasks such as the container loading problem [17], thus, in the following, we present a formulation suitable for RAD. Based on the grid-based parametrization of the target shape and the definition of the reward (Sec. III), we can define the objective function of the MILP that is subject to maximization as

$$\mathcal{O}_{\text{MILP}} = \max_{\boldsymbol{g}} \boldsymbol{c}^T \boldsymbol{g}, \tag{1}$$

where vector $\boldsymbol{g}$ represents the grid-state, and $\boldsymbol{c}$ contains weighting factors that indicate whether a grid-cell should be filled or not. This step necessitates flattening the three-dimensional grid into a vector. For the exemplary 2D problem displayed in Fig. 2 (grid dimensions $n_x \times n_y$), the indices of all points can be flattened to a single index j through $j = d_x + d_y n_x$ with the discrete coordinates of every grid-cell, d_x, d_y. Therefore, the first three entries of $\boldsymbol{c}$ are set according to $c[0]{=}c[1]{=}-1$ and $c[2]{=}1$, as the two leftmost lowest grid-cells should not be occupied, whereas the neighboring cell on the x-axis should be. Adapting this flattening process to the 3D case is straightforward, i.e. $j = d_x + d_y n_x + d_z n_x n_y$, with the additional grid index for the z coordinate d_z. As every grid-cell can only be occupied at maximum by one primitive unit, we add the constraint

$$g[i] \leq 1 \quad \forall g[i] \in \boldsymbol{g}. \tag{2}$$

Next, we need to determine how every potential positional and rotational placement action influences the grid-state. Therefore, for each type of building block (i.e., without disambiguating between the same blocks that are just placed in a different initial pose in the environment), we first attempt placing it with all available actions and determine how the placement affects the grid-state. For example, placing the horizontal block from Fig. 2 in the lowest left position without changing the rotation results in a grid state of $\boldsymbol{p}^T_{i=1,k=1} = [1, 1, 0, ..., 0]$, with block type index i and placement action k. By additionally assigning an integer decision variable $w_{i,k}$ and taking all object types into account, we can define the change in the grid-state according to

$$\boldsymbol{g} = \sum_{\hat{i}=1}^{P} \sum_{\hat{k}=1}^{K(\hat{i})} w_{i=\hat{i},k=\hat{k}} \boldsymbol{p}_{i=\hat{i},k=\hat{k}} \tag{3}$$

with a total of P different block types and $K(i)$ admissible placement actions. Note that the number of admissible placement actions is block-type-dependent, as we require that upon any placement action all primitive units stay within the grid boundaries. Considering Fig. 2, it is for instance possible to place a block consisting of a single primitive unit in the lower right-hand corner, whereas this action is not admissible for the horizontal block, as the block's right primitive unit would then end up outside the grid boundaries.

While the integer decision variables prohibit any partial block placement by definition, we still have to restrict that any block type can only be placed depending on its appearance in the current environment (N_i), i.e.

$$\sum_{\hat{k}=1}^{K(i)} w_{i,k=\hat{k}} \leq N_i, \quad \forall i \in P. \tag{4}$$

This constraint concludes the MILP formulation (optimize (1), with constraints (2-4)), which boils down to optimizing the integer decision variables $w_{i,k}$ through Gurobi [33]. Thus, the solution contains the quantity and final poses for every block type and is guaranteed to optimally fill the desired target shape. Yet, it neither resolves the challenge of determining the assembly order nor the combinatorial ambiguity regarding which block to use for each placement, as the scenes typically contain multiple blocks of same type.

B. GNN for task sequencing (medium level)

The high level MILP only partially resolves the combinatorial aspect of RAD. It lacks the placement actions' sequencing and the exact assignment of which block to use for each placement. Further, the MILP does not consider robotic feasibility, the blocks' initial positions, and neither structural stability during assembly. We thus require another level, capable of efficiently incorporating the MILP's prior knowledge, and deciding upon either executing one of the proposed actions or terminating the current assembly if none of the proposed actions is feasible. This can either be due to robotic constraints or due to stability considerations (e.g., robot colliding with the structure while placing a block / creating an unstable structure). Still, having the MILP's solution allows for efficiently shrinking the action space that has to be considered on this level and thus allows for using an approach based on *guided exploration* and model-free RL, capable of reflecting all real-world constraints without requiring any further simplifications.

Thus, we propose to employ an approach based on a combination of GNN and Q-learning [4], [5], for the following reasons. The graph-based representation is capable of providing the required flexibility on the representation level and invariance to the problem size, while performing the action selection based on Q-learning is desirable as i) the herein considered action space is discrete, but remains tractable for exploration due to the prior knowledge from the MILP solution, ii) the state-action-based formulation allows to efficiently incorporate the prior knowledge by masking out all actions that are not inside the MILP solution, iii) potential multimodalities in the MILP solution are not problematic and do not erroneously bias this Q-function estimator, and iv) it allows easy and time-effective combination with search-based methods, such as MCTS to improve robustness and performance despite the combinatorial action-space [8]. Moreover, since the overall method only requires

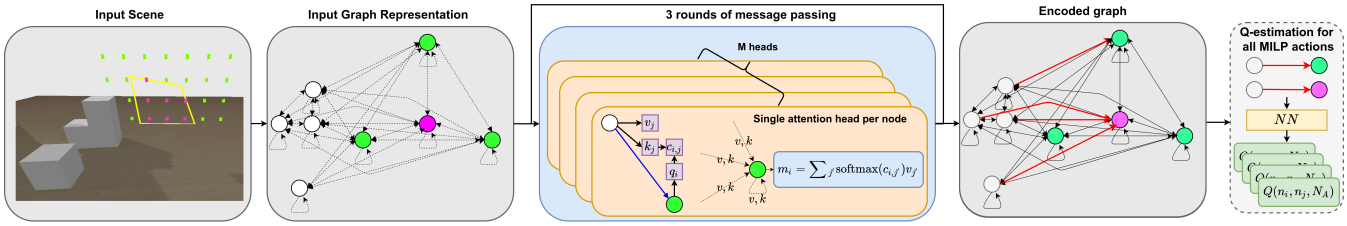

Fig. 3. Illustrating the process of action selection using the GNN. First, the current scene is transformed into a graph. Note that we only visualize a subset of the target (pink) and non-target (green) grid-cells. The white nodes depict the primitive units-to-be-placed. Next follow the 3 rounds of message passing in which all the nodes' features are updated using the attention mechanism (see [5] for details), which results in an encoded version of the graph. Finally, we perform action selection by predicting the Q-values for all the actions that are part of the MILP solution (visualized through the red arrows). The Q-values are predicted based on the nodes' features of the respective primitive units-to-be-placed and the grid-cells using a feedforward NN.

running the potentially time-consuming process (depending on problem size) of solving the MILP once, this level should be reactive w.r.t. changes in the blocks' positions.

We now briefly describe the process of action selection, as also visualized in Fig. 3, but refer to our previous work [5] for the additional details. We first transform the environment's current state into a graph, by creating nodes for all primitive units and grid-cells (cf. Sec. III). The nodes' features contain the respective nodes' 3D position, as well as 2 indices that indicate the node type, i.e. placed/unplaced primitive unit, target/non-target grid-cell. Almost all nodes of the graph are fully-connected with each other – we only omit the connections in-between the unplaced primitive units if they do not belong to the same block for the purpose of explicitly encoding different blocks. Note that the edges solely define in between which nodes there is exchange of information. Upon graph creation follow three rounds of message passing using an attention mechanism [5], [26]. This process updates the nodes' values based on their neighbors' and can also be seen as building a meaningful graph encoding due to the flow of information. The obtained, updated node values are then used as the basis for computing Q-values for all available actions. Particularly, as we can place any unplaced primitive unit w.r.t. every grid-cell, for this final step, we use a standard feedforward NN that takes as input the encoded node values of i) the primitive unit-to-be-placed, and ii) the grid-cell, and outputs the Q-values for all the four rotational-placement actions in between these nodes. That way, we predict the quality of moving the primitive unit to the grid-cell, including the potential re-orientation. Note that this action moves the entire block that the primitive-unit-to-be-placed is part of. This process is repeated for all pairs of unplaced primitive units and grid-cells. To compute the Q-value for the termination action, we feed the average of all nodes' features through a different NN.

Action selection is done using an ϵ-greedy strategy, yet, only allowing to choose actions from the MILP solution and the termination action. Originally, the MILP's result only contains the final poses and quantities per block type (cf. Sec. IV-A). To make this information compatible with the current level, we infer all the translational and rotational actions w.r.t. the primitive units that will put the respective block into all the desired final poses. This ensures that all blocks of each type are considered for every placement, and it also directly yields the allowed actions between primitive units and grid-cells (cf. Fig. 3). After every placement, all actions that would put another block in the same pose are removed. Note that the MILP's prior knowledge is not considered at an earlier stage, as we view the message passing process as creating an holistic understanding of the RAD scene. Thus, we only exploit it on action selection.

The graph's weights are refined through temporal-difference learning. In particular, we minimize the smooth L1 loss between the current Q-value prediction of the GNN and the estimated value based on collected rollouts and a target network Q_T, i.e. $\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \max_{\tilde{a}} Q_T(s_{t+1}, \tilde{a})$, where the reward is defined according to Sec. III. While this Q-learning procedure by itself already results in good policies that can directly be used for action selection, at test time, we can additionally consider action selection based on the combination of Q-learning and MCTS (DQN+MCTS). This combination has proven very effective when dealing with combinatorial action spaces, and, in particular, improves policy performance, transferability and robustness [8], [4]. The combination is especially attractive, as the Q-function allows bootstrapping the depth of the Monte Carlo rollouts, which in turn allows exploring the effect of different actions without requiring costly simulated deep rollouts until episode termination. Again, following [5], we start the search from the current state by choosing an action according to an ϵ-greedy strategy and terminate every rollout directly after the first action, and estimate the next state's quality using the Q-function.

To conclude, in this intermediate level, we make use of a policy based on the combination of GNN, RL, and eventually add MCTS during test time. The policy benefits from the reduced combinatorial action space, determines the sequencing, and assigns the blocks to the placements, while considering all the environmental constraints. The last part that is missing, is a policy that turns these commands into joint-level signals to actually move the robot and the blocks.

C. Robot grasp and motion planning (low level)

The lowest level is tasked with converting the previous level's actions into robot joint commands, thereby handling the final robot execution of block grasping and placing. While it would be possible to add those decisions to the previous level, we consider motion generation separately, as it heavily depends on the actual robot manipulator, and we do not want to further increase the previous level's action space. Due to the scenes' layout (Fig. 1), i.e., the blocks

being close together initially and during the final placement, we first check the feasibility of a predefined set of top-down grasping poses and subsequently check if this grasp results in a feasible final placement pose. An action is feasible if the requested grasping/placement pose can be reached by the robot, i.e., considering the joint limits, and that there are no collisions with the other blocks in the scene in this configuration (which is computed using inverse kinematics (IK)). If there exists a pair of feasible grasping and placing poses, we move the robot by approaching the grasping pose from the top, then move to a position that is slightly above the placing location, and finally, approach the placement pose. Again, all intermediate waypoints are computed based on IK.

V. EXPERIMENTAL RESULTS

We now present the experimental evaluation of our proposed MILP-DQN method and potentially adding MCTS with a search budget of 5 (MILP-DQN-MCTS), as in [5]. We first evaluate in simulated RAD environments (using PyBullet [34], cf. Figs. 4 & 5) for answering two questions: 1) Does the MILP's guiding exploitation signal help to effectively boost the proposed method's performance compared to employing an end-to-end approach without any prior, i.e., is the high level MILP really needed? 2) How effective is the GNN policy of the medium level compared to using a heuristic approach for task sequencing, i.e., is the medium level GNN required? Lastly, we investigate whether our policies can be transferred to the real world.

Before diving into the results, we quickly explain the training procedure. As the training of our proposed approach requires knowledge of the MILP solution for every RAD scene, we decided to create a dataset prior to training. This dataset contains $50,000$ different scenes, describing the environment's initial state, upon which the agent is subsequently acting, modifying its state through the actions. For the experiments that do not consider the robot, we train all agents for $1,000,000$ steps, while we train for $1,500,000$ steps in the robot experiments. For all methods we train 5 agents using different seeds [35]. We describe the difficulty of our two-sided RAD environments (i.e., two target shapes must be filled) by specifying the maximum height and width of the admissible target shapes, e.g., Fig. 2 shows a potential one-sided target shape of height and width 3.

We will use a star(*) to denote the agents' evaluation in their training conditions, i.e., using target shapes of similar size, yet, using different initial environment states (i.e. blocks and poses) compared to training. The other experiments are even further out-of-distribution, as the target shapes are guaranteed to be bigger compared to the ones seen during training. As bigger shapes require more blocks, the number of initially placed blocks is also increasing. The results are obtained by averaging the agents' performance in 200 scenes. We report the discounted reward R, the fraction of runs that ended i) upon perfectly recreating the target shape d, ii) with selecting the termination action e, iii) upon failure f, i.e., trying to execute an action that is not feasible with the robot, or placing the block in an unstable configuration, or destroying the already existing structure, while differentiating between failing on grasp selection f_g, i.e., no feasible grasp exists, and on block placement f_p for the robot experiments. Finally, we report the desired target grid-cell coverage $\bar{a}$, i.e., the fraction of target grid-cells that were initially supposed to be filled and have actually been filled. We also provide videos of the experiments at `https://sites.google.com/view/rl-meets-milp`.

TABLE I

COMPARING OUR PROPOSED METHOD WITH TWO LEARNED BASELINES IN THE TWO-SIDED ENVIRONMENT WO ROBOT.

Grid Size	Method	R	e	d	$\bar{a}$
3*	DQN	0.63 (0.02)	0.59	0.27	0.71
	DQN-REL [5]	0.67 (0.01)	0.7	0.23	0.68
	MILP-DQN	**1.22** (0.01)	0.31	**0.53**	**0.87**
4	DQN	0.71 (0.08)	0.53	0.2	0.69
	DQN-REL [5]	0.75 (0.08)	0.65	0.14	0.66
	MILP-DQN	**1.56** (0.03)	0.25	**0.47**	**0.87**
5	**MILP-DQN**	**1.92** (0.05)	0.17	0.42	**0.85**

A) Is the high level MILP needed?

We consider a scenario without the robot-in-the-loop, which reduces the complexity as GAMP can be omitted. Thus, the task reduces to placing the blocks in a stable configuration while trying to optimally fill the desired shape. We compare our approach against two baselines that do not consider the MILP. The first one (DQN) can place any of the available blocks at all currently unoccupied grid-cells. The second one (DQN-REL) follows our previous work [5], in which the blocks can only be placed next to the already placed ones, thereby reducing the action space. In the first step only, we allow placing the blocks at any target grid-cell.

The results in Table I reveal that the MILP provides a strong inductive bias that is effective in guiding the exploration, as the agents trained using our proposed MILP-DQN approach outperform the two baselines. The baseline agents exhibit very similar performance, with DQN-REL yielding slightly higher rewards. Actually, the smaller action space of the DQN-REL agents results in better performance at the beginning of the training. Compared to the baseline agents, the agents trained using MILP-DQN achieve an increase in the success rate and discounted reward by a factor of 2 (grid size of 3, 4). These results confirm the task's combinatorial complexity. Performing an ϵ-greedy exploration without using an informed prior does not allow for discovering good action sequences. Therefore, the baseline agents learn to terminate more frequently and achieve significantly lower successes and rewards. The results also reveal that the MILP-DQN agents generalize well to the out-of-distribution environments as the desired target grid-cell coverage remains high at 0.87 and 0.85 (grid size of 4,5), despite the significant increase in task complexity, i.e., the average target grid-cells that should be filled increase from roughly 5 to 12 while increasing the grid size from 3 to 5. We also want to point out that some desired shapes contain configurations of target grid-cells and non-target grid-cells, such that the MILP's solution does not contain all the actions that would be needed to achieve $\bar{a}$=1.0 as the MILP optimizes a tradeoff between optimal coverage and resource efficiency. When correcting for this effect in the computation of $\bar{a}$, the values increase to 0.97, 0.91 & 0.87 for MILP-DQN (grid sizes of 3, 4 & 5).

TABLE II

COMPARING OUR PROPOSED METHOD WITH A HEURISTIC IN THE TWO-SIDED ENVIRONMENT WITH THE ROBOT-IN-THE-LOOP.

Grid Size	Method	R	e	f_g	f_p	$\bar{a}$
4*	HEUR (wo robo)	-	-	-	-	0.81
	HEUR	0.57 (0.04)	0.36	0.24	0.16	0.62
	MILP-DQN	1.03 (0.04)	0.49	0.08	0.08	0.7
	MILP-DQN-MCTS	**1.24** (0.03)	0.57	**0.02**	**0.03**	**0.75**
5	HEUR (wo robo)	-	-	-	-	0.78
	HEUR	0.34 (0.02)	0.29	0.34	0.24	0.47
	MILP-DQN	0.98 (0.06)	0.53	0.1	0.15	0.58
	MILP-DQN-MCTS	**1.38** (0.04)	0.65	**0.02**	**0.06**	**0.65**

B) How effective is the GNN policy for robotic execution?
We now consider scenarios with the robot-in-the-loop (Figs. 4, 5) and investigate the effectiveness of the medium level GNN policy. For this purpose, we compare the learned GNN with a heuristic (HEUR). The agents using HEUR perform action selection as follows: based on actions proposed by the MILP, the heuristic only considers those for which the block's placement will result in a stable configuration, i.e., all grid-cells below the block-to-be-placed are already filled. Subsequently, the HEUR selects one action from this subset at random. If there is no action available that satisfies these conditions, the termination action is selected. Additionally, we report the performance of the heuristic agent on the same problem instances, however, without using the robot for part placement (HEUR wo robot). We consider this agent as an oracle, as it is acting in a substantially simpler environment without having to consider any robotic constraints.

The results from these experiments are presented in Table II. In both versions of the environment, there is a significant difference between the oracle heuristic (HEUR wo robo) and the heuristic baseline (HEUR) that underlines the increased difficulty from having the robot-in-the-loop. Moreover, our proposed MILP-DQN & MILP-DQN-MCTS agents outperform the heuristic baseline (HEUR). Notably, already in the environment with less building blocks, i.e. with the grid size of 4, using the heuristic on the medium level results in 40% of all the rollouts terminating upon an invalid action. Slightly more failures, i.e., 24% can be attributed to grasp selection, i.e. the policy selecting a block for placement that cannot be grasped or placed without collisions, while 16% are due to collisions during placement. Such a failure is depicted in Fig. 5, where due to bad action sequencing by the HEUR agent, the two blocks collide. Those high rates of failure indicate that a more informed method for action sequencing is actually required. As can be seen in Table II, both versions of our proposed approach are capable of effectively reducing the percentage of failures, with MILP-DQN decreasing the rates roughly by a factors of 3 & 2 (for grasping and placing failures, respectively), while the addition of MCTS leads to an impressive decrease by factors of 12 & 5. Those results show that our learned graph-based representations are indeed capable of effectively capturing the state of the environment and make informed decisions regarding the action sequencing which is a crucial component of RAD. The clear advantages also prevail for the even more difficult scenarios, considering the grid sizes of 5, where again, both versions of our proposed algorithm also achieve significantly higher rewards and target grid-cell coverage compared to the HEUR baseline. While there is a slight drop in performance concerning the achieved coverage of the MILP-DQN-MCTS agent with the increase in environment difficulty, the increase in failures is marginal (only by 3% on placing). Moreover, when relating the target grid-cell coverage of MILP-DQN-MCTS with the performance of the oracle HEUR wo robot agent, our proposed agents achieve relative fillings of 0.93 and 0.83 respectively, therefore again underlining their effectiveness. For more complicated scenes with grid sizes of 6, the performance of MILP-DQN-MCTS slightly degenerates. Yet, we attribute this behavior to the combination of extremely cluttered scenes and the robot's limited workspace, and speculate that mobile manipulators could circumvent these issues given the strong generalization from the previous experiments. Overall, the experiments show that our proposed hierarchical approach is indeed capable of resolving the inherent difficulties of RAD, as also illustrated in Fig. 4 where we show the successful assembly of a desired target shape using 4 blocks of 3 different types.

C) Is real-world policy transfer possible?
Finally, we evaluate whether the obtained MILP-DQN-MCTS policies can be transferred to real-world RAD scenes (cf. Fig. 6). For the evaluation, we first register all of the building blocks' poses using OptiTrack and initialize a simulated RAD scene mirroring the real-world. The simulated twin environment is subsequently exploited for evaluating our policies and performing MCTS planning to decide upon the next action which is then executed in both, simulation and reality. As shown in Fig. 6 and in the supplementary videos, we find that our proposed MILP-DQN-MCTS agents can indeed be transferred to real-world RAD scenes. This once again underlines its robustness w.r.t. different scenes, in particular, w.r.t. scene initializations and part placements.

VI. CONCLUSION

We have presented a novel hierarchical approach for robot assembly discovery (RAD). Our proposed approach is based on the powerful combination of global reasoning through mixed-integer programming, with graph-based reinforcement learning and model-based search for local decision-making, together with grasp and motion planning for realizing the assembly actions on the manipulator's joint level. The hierarchy allows for the efficient decomposition of the original problem's huge combinatorial action space and thereby results in robust, reliable, and effective RAD policies. The proposed approach is validated in a set of simulated RAD experiments and achieves an average coverage of the desired target shape of 75% while maintaining extremely low rates of failure (5%). We also showcase transfer to real-world RAD scenes. In the future, we want to investigate how this approach can be scaled to handle a wider range of objects.

REFERENCES

[1] E. Durmisevic, "Circular economy in construction design strategies for reversible buildings," *BAMB, Netherlands*, 2019.

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

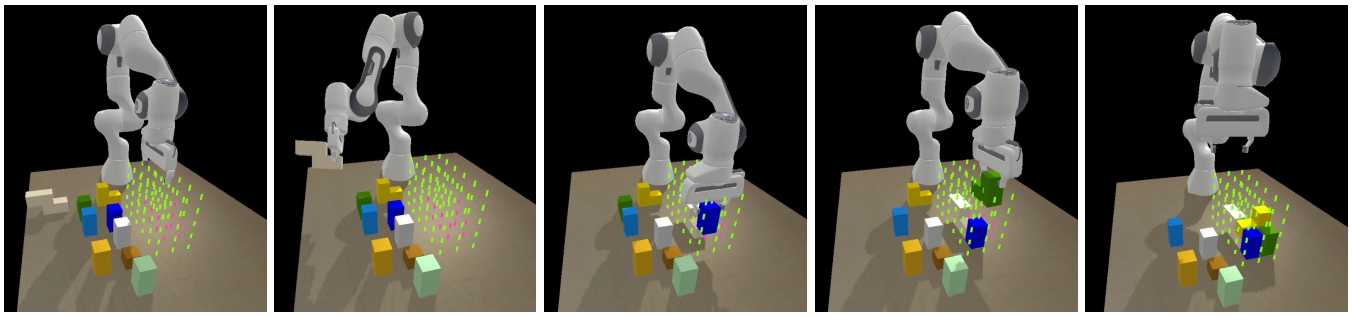

Fig. 4. Illustration of a successful RAD sequence using our proposed MILP-DQN-MCTS approach. The agent successfully the assembly successfully using in total 4 blocks and 3 different block types.

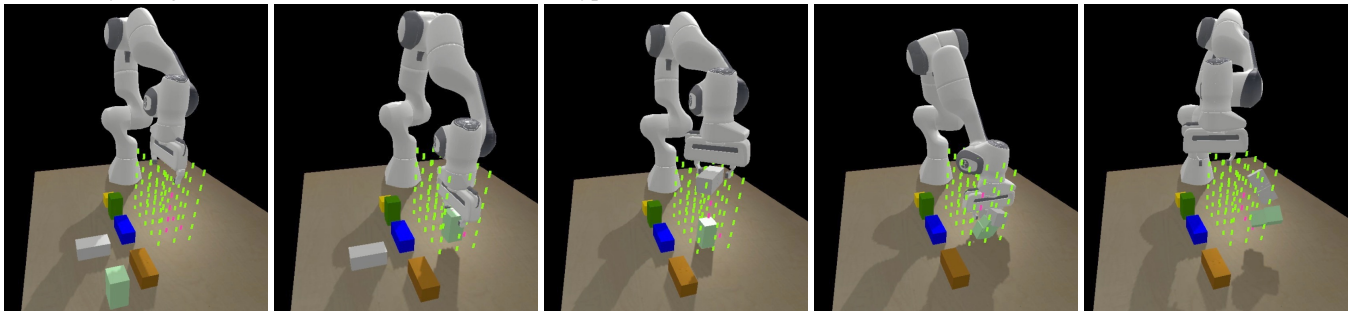

Fig. 5. Illustration of an unsuccessful RAD sequence using the heuristic agent (HEUR) introduced in Sec. V-B. As shown in the images, it is important to perform informed decisions about the assembly sequence, as the wrong sequencing can result in collisions between the block that is placed and other blocks in the scene.

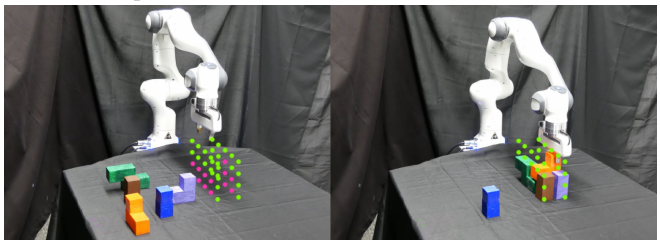

Fig. 6. Real-world RAD. Given the initial configuration (left), our proposed MILP-DQN-MCTS yields a valid assembly sequence that ends up filling all the desired target grid-cells (right).

[3] H. M. Salkin and C. A. De Kluyver, "The knapsack problem: a survey," *Naval Research Logistics Quarterly*, 1975.

[4] J. B. Hamrick, V. Bapst, and A. Sanchez-Gonzalez et al., "Combining q-learning and search with amortized value estimates," in *ICLR*, 2019.

[5] N. Funk, G. Chalvatzaki, B. Belousov, and J. Peters, "Learn2assemble with structured representations and search for robotic architectural construction," in *CoRL*, 2021.

[6] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *IJCAI*, 2015.

[7] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *Workshops AAAI*, 2010.

[8] D. Silver, J. Schrittwieser, and K. Simonyan et al., "Mastering the game of go without human knowledge," *nature*, 2017.

[9] N. Vahrenkamp, M. Do, T. Asfour, and R. Dillmann, "Integrated grasp and motion planning," in *ICRA*, 2010.

[10] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *JAIR*, 2003.

[11] T. Ren, G. Chalvatzaki, and J. Peters, "Extended tree search for robot task and motion planning," *arXiv:2103.05456*, 2021.

[12] V. N. Hartmann, A. Orthey, D. Driess, O. S. Oguz, and M. Toussaint, "Long-horizon multi-robot rearrangement planning for construction assembly," *arXiv:2106.02489*, 2021.

[13] C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Learning to rank for synthesizing planning heuristics," in *IJCAI*, 2016.

[14] D. Driess, J.-S. Ha, and M. Toussaint, "Learning to solve sequential physical reasoning problems from a scene image," *IJRR*, 2021.

[15] M. Noseworthy, I. Brand, C. Moses, S. Castro, L. Kaelbling, T. Lozano-Perez, and N. Roy, "Active Learning of Abstract Plan Feasibility," in *RSS*, 2021.

[16] T. Silver, R. Chitnis, A. Curtis, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Planning with learned object importance in large problem instances using graph neural networks," in *AAAI*, 2021.

[17] G. Fasano, "A modeling-based approach for non-standard packing problems," in *Optimized packings with applications*. Springer, 2015.

[18] ——, *Solving non-standard packing problems by global optimization and heuristics*. Springer, 2014.

[19] L. Junqueira, R. Morabito, D. S. Yamashita, and H. H. Yanasse, "Optimization models for the three-dimensional container loading problem with practical constraints," in *Modeling and Optimization in Space Engineering*, 2012.

[20] F. Wang and K. Hauser, "Robot packing with known items and nondeterministic arrival order," *IEEE T-ASE*, 2020.

[21] P. W. Battaglia, J. B. Hamrick, and V. Bapst et al., "Relational inductive biases, deep learning, and graph networks," *arXiv:1806.01261*, 2018.

[22] M. Janner, S. Levine, and W. T. Freeman et al., "Reasoning about physical interactions with object-oriented prediction and planning," *arXiv:1812.10972*, 2018.

[23] V. Bapst, A. Sanchez-Gonzalez, and C. Doersch et al., "Structured agents for physical construction," in *ICML*, 2019.

[24] A. Vaswani, N. Shazeer, N. Parmar, and J. Uszkoreit et al., "Attention is all you need," in *NeurIPS*, 2017.

[25] P. Veličković, G. Cucurull, and A. Casanova et al., "Graph attention networks," *arXiv:1710.10903*, 2017.

[26] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *ICLR*, 2018.

[27] S. Stevšić, S. Christen, and O. Hilliges, "Learning to assemble: Estimating 6d poses for robotic object-object manipulation," *IEEE RA-L*, 2020.

[28] R. Li, A. Jabri, T. Darrell, and P. Agrawal, "Towards practical multi-object manipulation using relational reinforcement learning," in *ICRA*, 2020.

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

[30] K. Zakka, A. Zeng, J. Lee, and S. Song, "Form2fit: Learning shape priors for generalizable assembly from disassembly," in *ICRA*, 2020.

[31] F. Furrer, M. Wermelinger, H. Yoshida, F. Gramazio, M. Kohler, R. Siegwart, and M. Hutter, "Autonomous robotic stone stacking with online next best object target pose planning," in *ICRA*, 2017.

[32] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, 2014.

[33] Gurobi Optimization, LLC, "Gurobi Optimizer," 2022.

[34] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016–2021.
[35] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, and J. Peters, "Mushroomrl: Simplifying reinforcement learning research," 2021.