Brick-by-Brick: Combinatorial Construction with Deep Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Discovering a solution in a combinatorial space is prevalent in many real-world problems but challenging due to diverse complex constraints and the vast number of possible combinations. In a similar vein, we introduce a novel formulation, *combinatorial construction*, which requires a building agent to assemble unit primitives (i.e., LEGO bricks) sequentially – every connection between two bricks must follow a fixed rule, while no bricks mutually overlap. Typically, we provide incomplete knowledge about the desired target (i.e., 2D images) instead of exact and explicit volumetric information to the agent. Such a problem requires a comprehensive understanding of partial information and long-term planning to append a brick sequentially, which leads us to employ reinforcement learning. However, this approach has to consider variable-sized action space and the existence of a large number of invalid actions that result in an overlap between bricks. To resolve these issues, our model, dubbed *Brick-by-Brick*, adopts an action validation network that filters invalid actions to an actor-critic network. We then introduce a reinforcement-learning environment and demonstrate that our approach successfully learns to construct an unseen object conditioned on a single image or multiple views of a target object.

1 Introduction

A combinatorial space, typically characterized by discrete variables and their combinations, often induces interesting yet challenging problems such as traveling salesperson and minimum spanning tree [20, 6]. The main challenges lie in the vast number of possible combinations as well as complex constraints imposed on them. In a similar spirit, we suggest a novel problem formulation, *combinatorial construction*, that focuses on the real-world construction procedure. Given only incomplete target information (i.e., 2D images or multiple views of a target object) [25, 12], an agent sequentially assembles unit primitives (i.e., LEGO bricks). The proposed formulation is combinatorial since it engages repetitive placement of primitives, which leads to a large number of available solutions. Distinct property of our proposed formulation, however, is that the agent must build the solution incrementally by adding on to the partial solution. Specifically, a brick, which is a unit primitive of the object of interest, is placed on a discrete space by connecting to one of the previously assembled bricks. In addition, every connection between two bricks must follow a fixed rule while no bricks mutually overlap. Each assembly (i.e., action) executed by the agent is, thus, modeled as selecting one of the feasible connections to place a new brick.

The problem we introduce closely depicts how humans understand an object and adapt the acquired knowledge to a downstream task. Humans naturally analyze a 3D object by picturing its part-by-part decomposition and consequently grasp a rich semantic understanding [15, 22]. In various fields, they utilize an inherent ability to decompose objects to effectively solve challenging tasks such as object classification [17], robot grasp planning [2], and part segmentation [31, 28]. Likewise, humans

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

Table 1: Analysis of recent studies in terms of state representation, supervision, conditioning, the type of target objects, and action validation network (AVN). CE and IoU stand for cross-entropy and intersection over union with respect to volumetric comparisons.

	State	**Supervision**	**Conditioning**	**Target**	**AVN**
Hamrick et al. [11]	Image	Task-dependent	N/A	2D	✗
Bapst et al. [3]	Object/Image	Task-dependent	Object and/or image	2D	✗
Kim et al. [19]	Set	Overlap	Exact target volume	3D	✗
Thompson et al. [38]	Graph	Step-wise CE	One-hot class information	3D	✗
B^3 (ours)	Graph/Image	IoU	Image or set of images	3D	✓

exploit this ability to solve the inverse problem – combinatorial construction. Given a desired object to be constructed and no strong supervision (i.e., ordered step-by-step instructions), humans can often still manage to build a valid target object by carefully planning or, sometimes improvising, the sequence of actions. Our environment, which corresponds to the proposed problem, is designed to learn and test such behavior with only partial information of the desired target available to the agent.

Successfully constructing an object in our setup requires a comprehensive understanding of incomplete target information with the current structured state of assembled bricks and long-term planning to append each brick efficiently. These requirements, along with the absence of sequence-level supervision, incentivize us to devise a reinforcement learning (RL) approach. In this domain, however, we must carefully handle both an indefinite action space and the existence of many invalid actions when applying RL [43]. In particular, both defining an action space that varies by the number of assembled bricks and distinguishing an invalid action that results in an overlap with other existing bricks quickly become intractable as more bricks are placed. To resolve the aforementioned issues, our model, dubbed Brick-by-Brick (B^3), adopts an action validation network that filters invalid actions to an actor-critic network. In addition to the novel RL formulation, we use graph representation of the brick combination to interpret the assembling process as sequential graph generation process.

Overall, we summarize our contributions as follows:

(i) We propose a novel problem formulation, *combinatorial construction*, that closely resembles real-world object construction process that engages repetitive placement of components;

(ii) We design an RL agent for combinatorial construction, dubbed Brick-by-Brick (B^3), to effectively address both growing action space and invalid actions;

(iii) We implement the corresponding environment based on OpenAI Gym and introduce new novel evaluation scenarios that vary by their lower-dimensional target information.

2 Combinatorial Construction

To formulate the combinatorial construction problem, we start by defining a unit primitive that is used to construct a 3D object and an action space that determines where to assemble the next primitive.

As a unit primitive, we utilize a 2×4 brick, which has eight studs and their fit cavities. This design choice yields a consistently varying action space, implying that if we add one brick to the current state of brick combination, we can efficiently define the next action space. We want to emphasize that with only six 2×4 bricks, we can create *915,103,765 combinations* [9]. Accordingly, our choice of the primitive does not make our problem a trivial task; instead, every decision of where we place the next primitive can deteriorate the quality of the final result because there exists a plethora of wrong paths.

With our specific choice of unit primitive, we can define an action space for determining the next action and evaluating the future states. However, since every assembly step gradually expands an action space, naïve approaches to defining a growing action space are not appropriate for our problem; an action space with redundant actions [43] is not applicable due to varying action space, and an action sampling approach [13] is also not suitable due to nominal or invalid actions. Thus, we define a *successive* action space composed of a two-step decision: (i) choosing a *pivot brick* and (ii) choosing an *offset* from the pivot brick.

Before explaining a pivot brick, we first assume a simplified assembly scenario that follows an Eulerian path[1] – new brick is always placed by connecting to the last assembled brick. This enables us to define a finite action space, though most objects are infeasible to assemble with the Eulerian path. To broaden search space by generalizing the Eulerian path, one of previously assembled bricks is chosen as a pivot brick. Then, B^3 decides an offset from the pivot brick, which describes how the next brick is placed relative to the pivot. Because of homogeneous brick type, the number of possible offsets is finite – for 2×4 bricks, there exist a maximum of 92 possible offsets.

As described earlier, our agent must consider an *invalid action* during assembly due to the disallowance of overlap between bricks. Identifying the validity of actions from the current brick combination becomes intractable as more bricks are placed; the complexity of this process is $\mathcal{O}(|A_{\text{off}}|t^2)$, where A_{off} is an action space for offsets and t is the cardinality of assembled bricks at a given step. Such expensive overhead for validation naturally leads us to adopt an action validation network that learns to identify invalid actions.

As described in Section 1 and this section, our problem has interesting but challenging characteristics derived from the assumptions on discrete placement, a connectivity rule, disallowance of overlap, and ultimately invalid actions. We, therefore, present a comparison to other existing studies in terms of state, supervision (or a reward function), conditioning, the type of target objects, and combinatorial construction, as shown in Table 1. Compared to the previous works [11, 3, 19, 38], our method B^3 construct an object in 3D with the presence of invalid actions and incomplete target descriptions; see Section 5 for a detailed description.

3 Brick-by-Brick

In this section, we briefly introduce the definition of RL and the corresponding framework for combinatorial construction, where an agent places a brick sequentially. We then explain the details of our model, B^3, that learns to select appropriate actions given only partial information of the desired target so that the assembled 3D object resemble the target. Moreover, to cope with enormous number of invalid actions and decision of such invalid actions, we propose an action validation network.

Definitions In a standard RL framework, there exists an agent that interacts with the environment by iteratively making decisions given an observation of the environment. This follows general decision making procedure of a Markov decision process (MDP), where a transition function satisfies the Markov property, i.e., $p(s_{t+1}|s_0, s_1, \ldots, s_t, a_t) = p(s_{t+1}|s_t, a_t)$, where s_t and a_t are a state and an action at timestep t, respectively.

In our problem setting, we only consider a finite horizon MDP formally defined as a tuple of (S, A, P, R, γ), where $S = \{s_t\}$ is a set of states, $A = \{a_t\}$ is a set of actions, $R : S \times A \to \mathbb{R}$ is a reward function, $P : S \times A \to S$ is a transition function, and $\gamma \in [0, 1)$ is a discount factor. The goal of the agent is to learn a policy $\pi(a_t|s_t)$ that maximizes the expected future cumulative reward. We introduce our detailed MDP formulation for combinatorial construction in the following sections.

3.1 Problem Formulation

Given target information $\mathcal{T}$, the agent aims to construct a 3D target object $\mathbf{T}$ by assembling bricks sequentially, one brick for each t-th step. Each t-th brick is represented by its pose $(\mathbf{x}_t, d_t)$, where $\mathbf{x}_t \in \mathbb{Z}^3$ is the center coordinate of the brick in 3D space and $d_t \in \{0, 1\}$ denotes one of two possible directions, meaning that its longer axis is aligned along either x axis or y axis in 3D space.

Target information. As described earlier, target information $\mathcal{T}$, which is a single binary image or a set of three binary images from different views of a target object, is given as partial target information. In practice, obtaining (incomplete) partial information is easier than accessing a 3D target. The goal of our task, thus, is to create a sequence of unit bricks by inferring a target object from abstract information in a combinatorial manner.

State. Each t-th state s_t of the MDP is represented by a tuple of a directed graph G_t composed of t bricks and target information $\mathcal{T}$, i.e., $s_t = (G_t, \mathcal{T})$. The graph is defined as $G_t = (V_t, E_t)$,

[1] It is a path that visits all the vertices without revisiting the edges visited before.

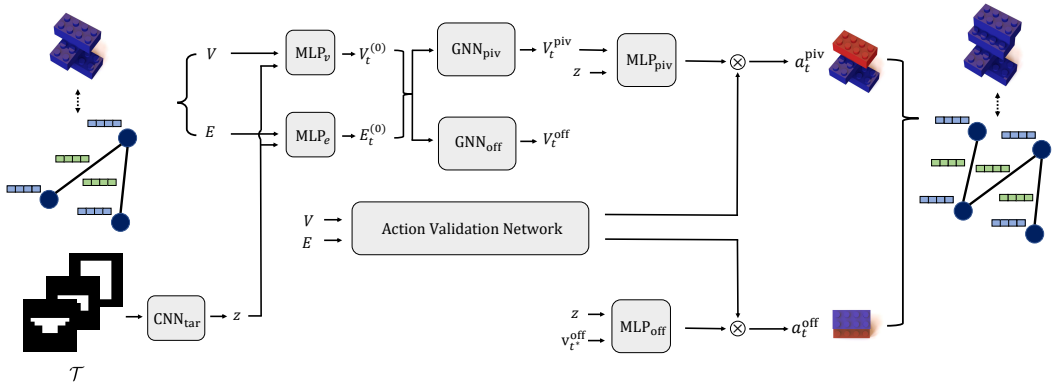


Figure 1: An overview of our proposed method B^3. State input $s_t = (G_t, \mathcal{T})$ is embedded and passed through GNNs and MLPs to predict action a_t that consists of pivot brick indicator a_t^{piv} and offset a_t^{off}. The red brick in both a_t^{piv} and offset a_t^{off} indicates the chosen brick.

where $V_t = \{\mathbf{v}_i\}_{i=1}^t$ is a set of t bricks, i.e., $\mathbf{v}_i = (\mathbf{x}_i, d_i) \in \mathbb{Z}^4$, and $E_t = \{\mathbf{e}_{ij}\}_{i,j=1}^t$ is a set of the offset vectors between two connected nodes, i.e., $\mathbf{e}_{ij} = (\mathbf{x}_i - \mathbf{x}_j, d_i \oplus d_j) \in \mathbb{Z}^4$. Note that nodes are connected by edges according to sequential actions and relative offsets in pose are used for edge features in order to induce translational and orientational equivariance. Since all the edges are bi-directional, we omit the arrows when displaying graphs.

Action. As noted earlier, we define a successive action space of choosing the pivot brick first and the corresponding offset next. Formally, with t bricks assembled, we define an action $a_t = (a_t^{\text{piv}}, a_t^{\text{off}})$ where a_t^{piv} is to select a pivot brick and a_t^{off} is to select an offset with respect to the pivot brick. The pose of the next brick is then $(\mathbf{x}^{\text{piv}} + \Delta\mathbf{x}, (d^{\text{piv}} + \Delta d) \mod 2)$ where $(\mathbf{x}^{\text{piv}}, d^{\text{piv}})$ and $(\Delta\mathbf{x}, \Delta d)$ are determined by a_t^{piv} and a_t^{off}, respectively. In choosing actions a_t^{piv} and a_t^{off}, we exclude invalid actions: (i) choosing a pivot brick near which no additional brick can be placed and (ii) choosing an offset for the next brick that overlaps with existing bricks. By the decision on invalid actions using the action validation network, we mask out all the probabilities of invalid actions, re-normalize the distributions over actions a_t, and then sample one of valid actions.

Transition Function. Given state s_t and action a_t, our transition function $p(s_{t+1}|s_t, a_t)$ is designed to determine the next state s_{t+1} by deterministically updating s_t based on a_t. The node of the new brick $\mathbf{v}_{t+1}$ is created so that $V_{t+1} = V_t \cup \{\mathbf{v}_{t+1}\}$. The edges between the new brick and existing bricks in physical contact via studs are created so that $E_{t+1} = E_t \cup \{\mathbf{e}_{(i)(t+1)}\}_{i\in\mathcal{N}} \cup \{\mathbf{e}_{(t+1)(i)}\}_{i\in\mathcal{N}}$ where $\mathcal{N}$ denotes the set of bricks in direct contact with the new brick $\mathbf{v}_{t+1}$. As a result, the graph in the state s_{t+1} is updated to $G_{t+1} = (V_{t+1}, E_{t+1})$.

Reward Function. In contrast to the tasks where rigorous evaluations are available, an appropriate countermeasure to quantify the object assembled by combinatorial construction is difficult to seek out, especially, in the context of graph generative model [26]. To mitigate such an issue, we exploit the property of a voxel representation. More concretely, given a desired object, we first create voxels in a closed space and determine the occupancy of voxels with a target object, after normalizing it to the bottom center of voxels. Then, we transform the combination of currently assembled bricks into the occupancy of the voxels defined with the target object and measure the overlap between them:

$$\Delta\text{IoU}(\mathbf{C}_t, \mathbf{T}) = \frac{\text{vol}(\mathbf{C}_t \cap \mathbf{T})}{\text{vol}(\mathbf{C}_t \cup \mathbf{T})} - \frac{\text{vol}(\mathbf{C}_{t-1} \cap \mathbf{T})}{\text{vol}(\mathbf{C}_{t-1} \cup \mathbf{T})}, \tag{1}$$

where $\mathbf{C}_t$, $\mathbf{C}_{t-1}$, and $\mathbf{T}$ are the occupied voxels at timestep t, timestep $t-1$, and a desired target, respectively. In addition, $\text{vol}(\cdot)$ is a function that measures a volume. The step-wise reward function is then ΔIoU if the new brick overlaps at least 50% with the occupied voxels of target object else -1. Consequently, our agent will learn the ordering and placement of the bricks to construct the target object, without explicit supervision as it tries to maximize Eq. (1).

3.2 Sequential Construction

In this section, we describe how we process G_t and $\mathcal{T}$ with neural networks. The overall pipeline of our model is illustrated in Figure 1.

Node and Target Embeddings. Given a state $s_t = (G_t, \mathcal{T})$ where $G_t = (V_t, E_t)$, we first use a CNN to extract features $\mathbf{z}$ from the target: $\mathbf{z} = \text{CNN}_{\text{tar}}(\mathcal{T})$. If the partial information is given as a set of images, the feature $\mathbf{z}$ is obtained by first applying CNN to each image separately and then concatenating to a single vector.

For node and edge features, an MLP embeds them to higher-dimensional features with $\mathbf{z}$ extracted from the target:

$$\mathbf{v}_i^{(0)} = \text{MLP}_v([\mathbf{v}_i, \mathbf{z}]) \quad \text{and} \quad \mathbf{e}_{ij}^{(0)} = \text{MLP}_e([\mathbf{e}_{ij}, \mathbf{z}]), \tag{2}$$

for all $i, j \in \{1, \ldots, t\}$, where $[,]$ and $^{(0)}$ denote concatenation and the first layer, respectively.

CNN_{tar} and Eq. (2) can be viewed as pre-processing inputs to feed in a graph neural network (GNN). Inspired by [4], we apply a variant of Graph Networks (GN) in which the global graph feature is omitted. At the ℓ-th layer of GNNs, we first update edge features, and then aggregate message vectors for each node, and finally update node features:

$$\mathbf{e}_{ij}^{(\ell+1)} = \text{MLP}_e^{(\ell)}\left([\mathbf{v}_i^{(\ell)}, \mathbf{v}_j^{(\ell)}, \mathbf{e}_{ij}^{(\ell)}]\right), \tag{3}$$

$$\mathbf{m}_i^{(\ell)} = \sum_{j \in \text{NN}(\mathbf{v}_i)} \text{aggregate}\left(\mathbf{e}_{ij}^{(\ell+1)}\right), \tag{4}$$

$$\mathbf{v}_i^{(\ell+1)} = \text{MLP}_v^{(\ell)}\left([\mathbf{v}_i^{(\ell)}, \mathbf{m}_i^{(\ell)}]\right), \tag{5}$$

where $\text{NN}(\mathbf{v}_i)$ is the neighborhood nodes of $\mathbf{v}_i$ and $\text{aggregate}(\cdot)$ is the aggregation function that computes a message for each node by aggregating features its neighboring nodes. Note that MLP_v, MLP_e, $\text{MLP}_v^{(\ell)}$, and $\text{MLP}_e^{(\ell)}$ have their own learnable parameters.

Action Selection. In order to obtain richer representations that are useful for predicting a_t, we employ two separate GNNs: GNN_{piv} and GNN_{off}, with L layers in total, to produce sets of node embeddings, V_t^{piv} and V_t^{off} for selecting a pivot and relative displacement:

$$V_t^{\text{piv}} = \text{GNN}_{\text{piv}}(V_t^{(0)}, E_t^{(0)}) \quad \text{and} \quad V_t^{\text{off}} = \text{GNN}_{\text{off}}(V_t^{(0)}, E_t^{(0)}), \tag{6}$$

where $V_t^{(0)} = \{\mathbf{v}_i^{(0)}\}_{i=1}^t$ and $E_t^{(0)} = \{\mathbf{e}_{ij}^{(0)}\}_{i,j=1}^t$ are the sets of node and edge features obtained by Eq. (2). Each layer of both GNNs updates node and edge features using Eq. (3), Eq. (4), and Eq. (5). Finally, the set of node and edge features, V_t^{piv} and V_t^{off} along with a target feature $\mathbf{z}$ are used to compute a prediction for the next action a_t^{piv} and a_t^{off}:

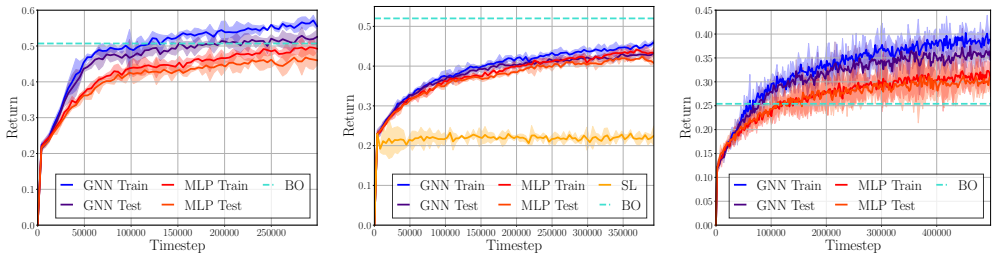
$$p\big(a_t^{\text{piv}}\big) = \sigma\Big(\text{MLP}_{\text{piv}}\big([V_t^{\text{piv}}, \mathbf{z}]\big)\Big) \quad \text{and} \quad p\big(a_t^{\text{off}}\big) = \sigma\Big(\text{MLP}_{\text{off}}\big([\mathbf{v}_{i^*}^{off}, \mathbf{z}]\big)\Big), \tag{7}$$

where σ is a softmax function and $\mathbf{v}_{i^*}^{off}$ is the node feature selected by the index i^* of a_t^{piv}.

Action Validation. As described earlier, we predict an invalid action using a surrogate for confirming the validity of a given action. By exhibiting an available information we have, i.e., graph representation of the bricks assembled so far, we train a graph neural network, of which the head corresponds to the level of validity. The structures of both pivot and offset validation networks are identical to the networks described in (7), but the last activation is a sigmoid function and no target feature $\mathbf{z}$ is used. Importantly, these networks can be pre-trained by the ground-truth validity of actions, which are obtained by randomly-assembled objects, and moreover such pre-trained networks can be applied in training an actor-critic network, *without re-training*.

Training. We adopt the proximal policy optimization (PPO) algorithm [35], which is one of the state-of-the-art on-policy algorithms. In particular, we optimize the clipped surrogate objective over parameters $\boldsymbol{\theta}$:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}\left[\min(r_t(\boldsymbol{\theta})\hat{A}_t, \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right], \tag{8}$$



(a) MNIST (b) Randomly-Assembled (c) ModelNet

Figure 2: Episode return curves vs. timesteps in different setups. The curves measured by training and test episodes are reported by repeating 3 times with different seeds.

where $r_t(\boldsymbol{\theta})$ is a probability ratio between the previous and updated policy, clip is a clipping function between the second and the third arguments, and $\hat{A}_t$ is an advantage function [34]. To calculate the advantage of a state s_t, our model employs a value network:

$$\text{MLP}_{\text{val}}\left([\mu(V_t^{\text{piv}}), \mu(V_t^{\text{off}}), \mathbf{z}]\right), \tag{9}$$

where $\mu(\cdot)$ is a global average function over instances in a given set.

4 Experimental Results

As shown in Table 1, the prior studies do not evaluate their models in a scenario conditioned on the abstract expression of target objects. In a spirit of generalization in machine learning, we show the possibility of an image-conditioned 3D object construction scheme and the effectiveness of our model B^3.

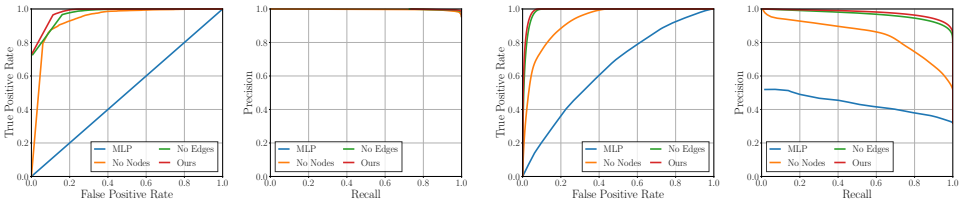
For evaluation metric, we measure the episode return or IoU between the constructed object and the desired target at the end of each episode:

$$\text{IoU}(\mathbf{C}_N, \mathbf{T}) = \frac{\text{vol}(\mathbf{C}_N \cap \mathbf{T})}{\text{vol}(\mathbf{C}_N \cup \mathbf{T})}, \tag{10}$$

where N is the total number of bricks and $\mathbf{T}$ is the voxel representation of the target object. The maximum number of bricks to be placed depends on $\mathcal{T}$ and is pre-defined. After exhausting a brick budget, we terminate the episode and compute the final IoU.

To show the effectiveness of our method, we compare B^3 to MLP-based model where all GNNs are replaced to MLPs and Bayesian optimization-based approach (BO) [5] that sequentially optimizes the step-wise reward function in terms of IoU to search for an optimal construction sequence. In addition, we compare B^3 to supervised learning methodm, trained with cross entropy loss between ground truth sequence in Randomly-Assembled benchmark specifically. Since sequence supervision is used, performance of supervised learning method is only measured on test dataset. As presented in Table 1, BO uses exact volumetric information for both training and test instances since it cannot assemble an object with only partial information. For each benchmark, the episode returns of BO model are averaged over both training and test datasets. In addition, the computation time for each construction step of BO is limited to 3 seconds. Details can be found in supplementary materials.

Action Validation Network. We test our action validation network by creating training and test datasets. The training dataset is composed of 200,000 brick combinations and their ground-truth action masks, and the test dataset is composed of 30,000 brick combinations and their ground-truth action masks. Importantly, the range of the size of brick combination in the training dataset is $[1, 20]$, and the range in the test dataset is $[1, 30]$. Even though the test dataset contains larger size of brick combination than the training dataset, the performance of action validation network in terms of precision and recall is satisfactory by predicting a combination within unseen range, as presented in Figure 3 and Table 2. In addition, our graph neural network outperforms other baselines including MLP and graph neural networks either without node feature or without edge feature.



(a) Pivot (b) Offset

Figure 3: ROC and PR curves for the action validation network. Left and right panels of each figure present ROC and PR curves, respectively. All the results are measured using test dataset of randomly-assembled objects.

Table 2: Results on predicting invalid actions by an action validation network. Thresholds for deciding either valid or invalid actions are set to 0.5.

	Pivot				**Offset**			
	Training		Test		Training		Test	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
MLP	0.9618	**1.0000**	0.9557	**1.0000**	0.5614	0.1410	0.5130	0.1398
No Node	0.9874	0.9895	0.9804	0.9869	0.8261	0.7518	0.7931	0.7344
No Edge	0.9947	0.9986	0.9850	0.9948	0.9199	**0.9736**	0.8897	**0.9672**
Ours	**0.9976**	0.9987	**0.9909**	0.9944	**0.9408**	0.9709	**0.9125**	0.9661

MNIST Construction. In each episode, an agent is provided with an image from the MNIST dataset and is provided to create a 3D object resembling the digit target. Similar to [32], we binarize the MNIST dataset to convert a real-valued number to either 0 or 1, for brevity of the calculation of IoU. To create a 3D target object with a 2D MNIST image, we first rescale an image to half of the original size and then expanding an image along channel dimension, in order to assemble with 2×4 bricks, i.e., an image of size 28×28 is transformed to a 3D object of size $14 \times 14 \times 4$. Furthermore, we limit possible offset candidates to 6 different types of which the values according to the channel dimension are fixed to the same value. Training and test datasets are established by choosing one of the ten classes in the binarized MNIST and splitting images from that class. In particular, 500 images from one of available classes are chosen, further divided into 400 samples for a training dataset and 100 samples for a test dataset.

Due to a space limit, we only report the average reward performance on class 0 in Figure 2(a). Results for other classes are available in the supplementary material. The gap between training and test on episode returns are marginal, which implies that our model B^3 generalizes to unseen targets well. In addition, both our training and test performance shows better performance compared to both BO and MLP-based model. We provide visualization of constructed objects for test dataset of class 0 and 9 in Figure 4. Qualitative results on other classes are also provided in supplementary materials. In general, our agent successfully constructs objects of unseen instances. This can be understood as our agent catches distinctive details in the target information and reflect in a construction process.

Randomly-Assembled Construction. Contrary to MNIST based experiments, this task focuses on building objects that require more than one image to fully understand the structure. Accordingly, the agent must construct an object with three 14×14 images from different viewpoints initially given as the target information. Objects in this experiment are generated by artificially connecting bricks by random choice. The total number of bricks is also chosen randomly between 10 to 15. For available offset types, we only utilize connection types that occupy four or more studs and only allow a new brick to be placed on top of the pivot brick so that the resulting target becomes more distinguishable. In this scheme, the total number of offsets are 16. For computation of target feature $\mathbf{z}$, we separately pass through CNN_{tar} and then concatenate them in channel dimension. Finally, we sample images from 800 instances for training while 200 instances for testing.

The result is shown in Figure 2(b). Our model achieved comparable return when compared to MLP-based model while slightly lower return when compared to BO. This is because the number of bricks

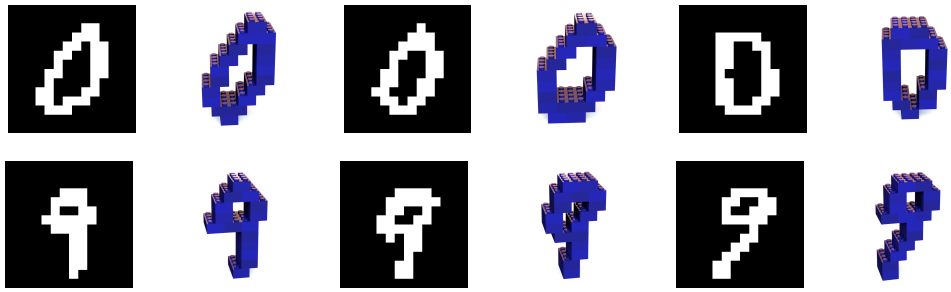
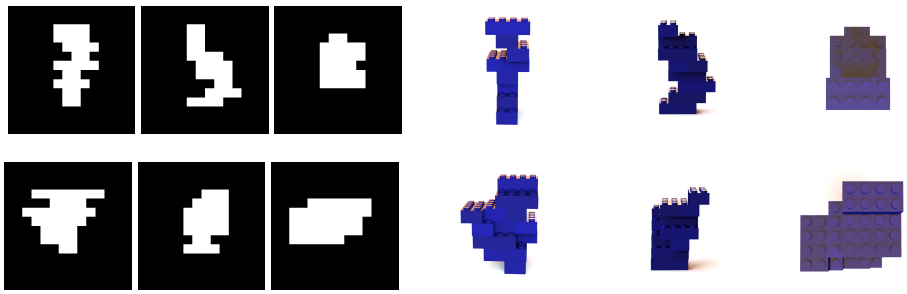


Figure 4: Qualitative results on MNIST Construction. Our model is separately trained on each class, and target images are unseen while training.



(a) Target images (b) Constructed object from three viewpoints

Figure 5: Qualitative results on arbitrary shape construction. Targets are obtained from a test dataset.

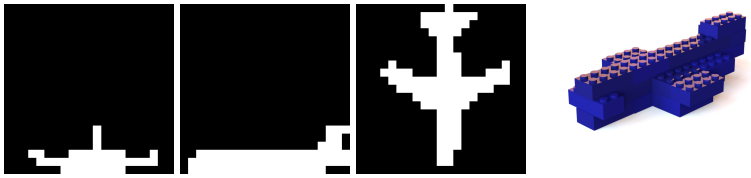
used are relatively small compared to other test suites. However, we observe that our agent is still capable of associating the target object in 3D space from multiple images as illustrated in Figure 5. Our model learns to assemble bricks in a way that images of resulting object matches the initially given images fairly well. Furthermore, the model trained with supervised learning method does not generalize to unseen images of test dataset. This clearly demonstrates effectiveness of applying reinforcement learning compared to sequence-level supervision.

ModelNet Construction. Similar to Artificial Object Construction, the agent is given 3 images of a realistic target from ModelNet dataset [40] where the maximum budget of bricks are below 60. Typically, we chose airplane, table, and monitors from the dataset specifically and use offset types that occupy four or more studs and allow a new brick to be placed above and below the pivot brick. This task is the most challenging due to excessive search space compared to other tasks, and assesses the agent's ability to generate realistic target.

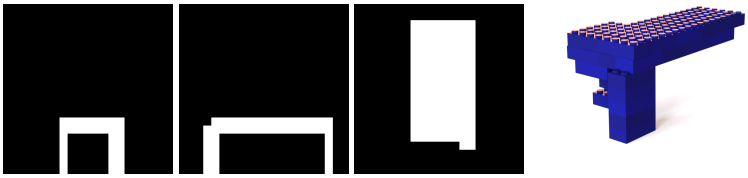
Due to a space limit, curve for only airplane class is provided in Figure 2(c). Despite the difficulty raised from the large search space and long sequence, the result demonstrate that B^3 is capable of learning the construction process of real-world objects. BO with limited budget achieves lower return compared to B^3 since search space is too big to explore with limited computation. By comparing the constructed object to images of the desired target in Figure 6, it can be observed that B^3 generally captures overall shape of the target. Though, our model tends to struggle to catch fine-grained details such as wings of the airplane or legs of the table. More visualization of instances and constructed objects, including examples from the other class, are available in the supplementary materials.

5 Related Work

3D Object Generation. Following the studies on 2D object generation, e.g., [8], 3D object generation is often achieved in holistic manner [39, 1, 14, 30]. Though promising, they generate the object in a single feed-forward operation which limits exploitation of intermediate structures. Compared to these holistic methods, [19] proposes an approach to tackle a combinatorial assembly problem



(a) Airplane



(b) Table

Figure 6: Results on ModelNet Construction. Targets are obtained from the test dataset. The first three panels and the last panel of each figure show the target images and the construed example.

but by using Bayesian optimization [5], not a learning-based method. Unlike [19], [38] applies a graph-structured generative model in combinatorial 3D object generation task, by training to match a ground-truth sequence of LEGO bricks.

Graph-based Reinforcement Learning. A common technique for creating a graph is to use an autoregressive model such as recurrent neural networks [42, 24], adversarial network [7], and variational autoencoder [33]. Moreover, several recent studies have explored generating novel molecules in autoregressive manner using variational autoencoder [18] and flow-based autoregressive model [41]. In addition, [42] adopts reinforcement learning to generate unseen molecules under certain constraints. [36, 37] extend this idea of generating molecules with reinforcement learning such that generated molecules are placed in the Cartesian coordinate. Key difference to our work, is that we are sequentially generating 3D shapes which have much larger search space. Furthermore, [3] shows that an RL agent can learn physical construction in 2D space. This work utilizes rich visual information as well as a graph-based representation, in order to define an agent state.

Image-Conditioned Reinforcement Learning. [10] proposes a method to synthesizing program for 2D image data when either unconditional or conditional scenario is assumed. This paper generates an image by sequentially conducting an action in a MuJoCo environment. [29] proposes a goal-conditioned RL approach of which the goal is provided by visual information. [16] suggests a method to paint a palette with stokes where a target image is conditionally given, by utilizing an RL algorithm.

Brick Assembly Optimization. This problem satisfying pre-defined constraints is a longstanding topic in computer graphics. [23] tackles LEGO brick layout optimization by genetic algorithm. Similarly, [27] solves building sculptures safely with LEGO brick by stability aware refinement. [44] proposes method for generating component-based building instructions that is safe based on segmentation models. [21] tackles similar problem of brick assembly from images with octree voxel-based model.

6 Conclusion

In this work, we have proposed a novel problem formulation, *combinatorial construction*, which asks the agent to construct an object sequentially. We adopt reinforcement learning by defining a state as graph-structured representation to express assembled bricks and their connections. In addition, we develop our algorithm with a successive action space that does not depend on the number of bricks already constructed and a reward function that measures overlap between the target and the current state. Through extensive experiments, we demonstrate that our method can construct objects in various construction scenarios and provide detailed analysis of our action validation network.

References

[1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3D point clouds. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 40–49, 2018.

[2] J. Aleotti and S. Caselli. Part-based robot grasp planning from human demonstration. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 4554–4560, 2011.

[3] V. Bapst, A. Sanchez-Gonzalez, C. Doersch, K. Stachenfeld, P. Kohli, P. Battaglia, and J. Hamrick. Structured agents for physical construction. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 464–474, 2019.

[4] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[5] E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

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

[7] N. De Cao and T. Kipf. MolGAN: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[8] A. Dosovitskiy, J. T. Springenberg, M. Tatarchenko, and T. Brox. Learning to generate chairs, tables and cars with convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):692–705, 2016.

[9] S. Eilers. The LEGO counting problem. *The American Mathematical Monthly*, 123(5):415–426, 2016.

[10] Y. Ganin, T. Kulkarni, I. Babuschkin, S. M. A. Eslami, and O. Vinyals. Synthesizing programs for images using reinforced adversarial learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1666–1675, 2018.

[11] J. B. Hamrick, K. R. Allen, V. Bapst, T. Zhu, K. R. McKee, J. B. Tenenbaum, and P. W. Battaglia. Relational inductive bias for physical construction in humans and machines. In *Proceedings of the Annual Conference of the Cognitive Science Society (CogSci)*, pages 1773–1778, 2018.

[12] W. Han, S. Xiang, C. Liu, R. Wang, and C. Feng. SPARE3D: A dataset for SPAtial REasoning on three-view line drawings. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14690–14699, 2020.

[13] J. He, M. Ostendorf, X. He, J. Chen, J. Gao, L. Li, and L. Deng. Deep reinforcement learning with a combinatorial action space for predicting popular Reddit threads. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1838–1848, 2016.

[14] P. Henzler, N. Mitra, and T. Ritschel. Escaping Plato's Cave using adversarial training: 3D shape from unstructured 2D image collections. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 9984–9993, 2019.

[15] D. D. Hoffman and W. A. Richards. Parts of recognition. *Cognition*, 18(1-3):65–96, 1984.

[16] Z. Huang, W. Heng, and S. Zhou. Learning to paint with model-based deep reinforcement learning. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 8709–8718, 2019.

[17] D. Huber, A. Kapuria, R. Donamukkala, and M. Hebert. Parts-based 3D object classification. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 82–89, 2004.

[18] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2323–2332, 2018.

[19] J. Kim, H. Chung, J. Lee, M. Cho, and J. Park. Combinatorial 3D shape generation via sequential assembly. In *NeurIPS Workshop on Machine Learning for Engineering Modeling, Simulation, and Design (ML4Eng)*, 2020.

[20] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 6 edition, 2018.

[21] T. Kozaki, H. Tedenuma, and T. Maekawa. Automatic generation of LEGO building instructions from multiple photographic images of real objects. *Computer-Aided Design*, 70:13–22, 2016.

[22] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the Annual Conference of the Cognitive Science Society (CogSci)*, pages 2568–2573, 2011.

[23] S. Lee, J. Kim, J. W. Kim, and B-R. Moon. Finding an optimal LEGO® brick layout of voxelized 3D object using a genetic algorithm. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1215–1222, 2015.

[24] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. W. Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[25] Y. Li, K. Mo, L. Shao, M. Sung, and L. J. Guibas. Learning 3D part assembly from a single image. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 664–682, 2020.

[26] R. Liao, Y. Li, Y. Song, S. Wang, C. Nash, W. L. Hamilton, D. Duvenaud, R. Urtasun, and R. Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

[27] S-J. Luo, Y. Yue, C-K. Huang, Y-H. Chung, S. Imai, T. Nishita, and B-Y. Chen. Legolization: Optimizing LEGO designs. *ACM Transactions on Graphics*, 34(6):222:1–222:12, 2015.

[28] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 909–918, 2019.

[29] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 9191–9200, 2018.

[30] C. Nash, Y. Ganin, S. M. A. Eslami, and P. W. Battaglia. PolyGen: An autoregressive generative model of 3D meshes. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 7220–7229, 2020.

[31] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017.

[32] R. R. Salakhutdinov and I. Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 872–879, 2008.

[33] B. Samanta, A. De, G. Jana, V. Gómez, P. Chattaraj, N. Ganguly, and M. Gomez-Rodriguez. NeVAE: A deep generative model for molecular graphs. *Journal of Machine Learning Research*, 21(114):1–33, 2020.

[34] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[36] G. Simm and J. M. Hernández-Lobato. A generative model for molecular distance geometry. *arXiv preprint arXiv:1909.11459*, 2019.

[37] G. Simm, R. Pinsler, and J. M. Hernández-Lobato. Reinforcement learning for molecular design guided by quantum mechanics. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 8959–8969, 2020.

[38] R. Thompson, G. Elahe, T. DeVries, and G. W. Taylor. Building LEGO using deep generative models of graphs. In *NeurIPS Workshop on Machine Learning for Engineering Modeling, Simulation, and Design (ML4Eng)*, 2020.

[39] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 82–90, 2016.

[40] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.

[41] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3D point cloud generation with continuous normalizing flows. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 4541–4550, 2019.

[42] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 5708–5717, 2018.

[43] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 3562–3573, 2018.

[44] Man Zhang, Yuki Igarashi, Yoshihiro Kanamori, and Jun Mitani. Component-based building instructions for block assembly. *Computer-Aided Design and Applications*, 14(3):293–300, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Our contribution summarized in abstract matches the content in Section 2 and 3.
   (b) Did you describe the limitations of your work? [Yes] Due to the space limit, we provide briefly in Section 4. More detailed analysis is provided in supplementary materials.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Potential negative societal impacts are provided in supplementary materials.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We have read the ethics review guideline.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include instructions required for both data preparation and model setup in supplementary materials.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Details are presented in Section 4.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Results in Figure 2 are averaged over 3 random seeds.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The type of resources is provided in supplementary materials.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We cite original paper that the data is first proposed.

(b) Did you mention the license of the assets? [Yes] We will explicitly mention the license of the assets in the supplementary materials.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We will submit our environments in submission time.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] No such personally identificable information or offensive content exsit in data we use.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]