Scene Mover: Automatic Move Planning for Scene Arrangement by Deep Reinforcement Learning

HANQING WANG, Beijing Institute of Technology
WEI LIANG*, Beijing Institute of Technology
LAP-FAI YU, George Mason University

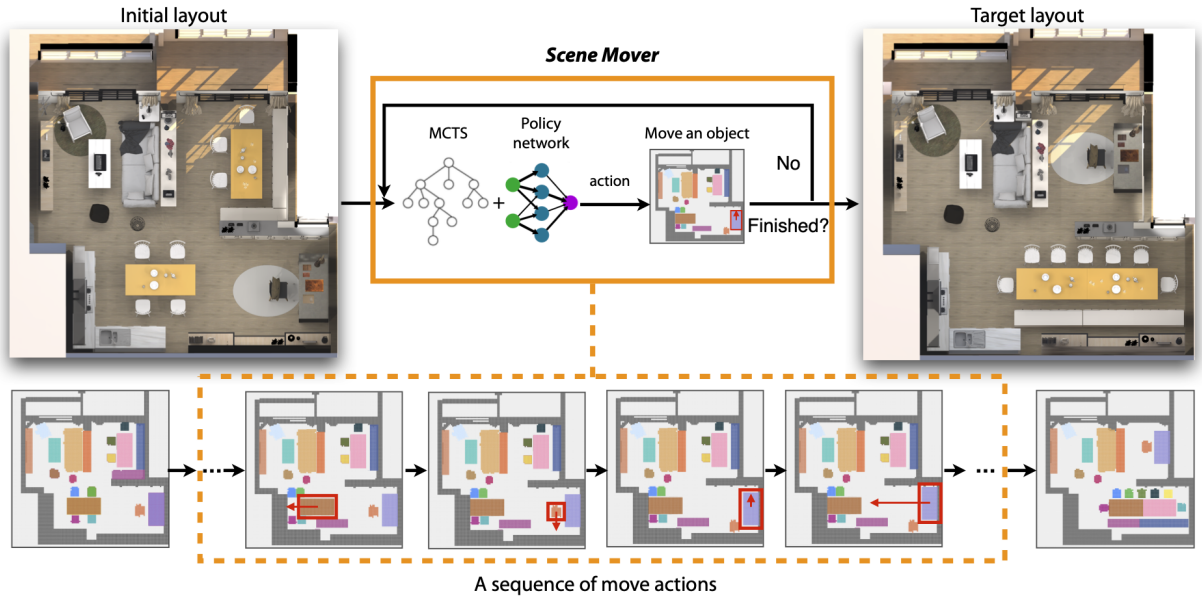


Fig. 1. Given an initial layout and a target layout, our approach automatically generates a move plan to transform the initial layout into the target layout. The move plan consists of a sequence of move actions. Each move action includes selecting an object and generating a path to move the object. Our approach, dubbed *Scene Mover*, is devised based on a Monte Carlo tree search (MCTS) driven by a novel network learned through deep reinforcement learning.

We propose a novel approach for automatically generating a move plan for scene arrangement.[1] Given a scene like an apartment with many furniture objects, to transform its layout into another layout, one would need to determine a collision-free move plan. It could be challenging to design this plan manually because the furniture objects may block the way of each other if not moved properly; and there is a large complex search space of move action sequences that grow exponentially with the number of objects. To tackle this challenge, we propose a learning-based approach to generate a move plan automatically. At the core of our approach is a Monte Carlo tree that encodes possible states of the layout, based on which a search is performed to move a furniture object appropriately in the current layout. We trained a policy neural network embedded with a LSTM module for estimating the best actions to take in the expansion step and simulation step of the Monte Carlo tree search process. Leveraging the power of deep reinforcement learning, the network learned how to make such estimations through millions of trials of moving objects. We demonstrated our approach for moving objects under different scenarios and constraints. We also evaluated our approach on synthetic and real-world layouts, comparing its performance with that of humans and other baseline approaches.

CCS Concepts: • **Computing methodologies** → **Planning for deterministic actions**;

Additional Key Words and Phrases: layout design, reinforcement learning

ACM Reference Format:
Hanqing Wang, Wei Liang, and Lap-Fai Yu. 2020. Scene Mover: Automatic Move Planning for Scene Arrangement by Deep Reinforcement Learning. *ACM Trans. Graph.* 39, 6, Article 233 (December 2020), 15 pages. https://doi.org/10.1145/3414685.3417788

*Corresponding author.
[1]Project page: https://github.com/HanqingWangAI/SceneMover

Authors' addresses: Hanqing Wang, Beijing Institute of Technology, hanqingwang@bit.edu.cn; Wei Liang, Beijing Institute of Technology, liangwei@bit.edu.cn; Lap-Fai Yu, George Mason University, craigyu@gmu.edu.

0730-0301/2020/12-ART233 $15.00
https://doi.org/10.1145/3414685.3417788

1 INTRODUCTION

People rearrange layouts from time to time. An event planner may rearrange a banquet hall for hosting different parties. A store manager may rearrange his store to give customers a refreshed look. A family may rearrange their home to accommodate the changing space needs of their kids.

To rearrange a layout, people generally take two steps: (1) designing a new layout and (2) moving all objects to their new positions. A number of approaches [Fu et al. 2017; Li et al. 2019; Wang et al. 2019a, 2018] were proposed recently for synthesizing desirable furniture layouts compatible to a room. These efforts automate the task of layout design for the first step. However, an automatic approach for moving the furniture objects in the current layout to their desired positions in the target layout is missing.

Generating a plan to move objects from an initial layout to a target layout is challenging. It involves deciding a valid sequence of move actions of many objects. People who reconfigured their home layouts may have this experience: trying to move furniture objects to desired positions, they had to go through multiple rounds of trials and errors with wasted efforts; a piece of furniture which had been moved to its desired position might need to be moved again to make way for another piece of furniture.

The difficulty of creating such a move plan arises from *sequential* and *long-term* decision-making. Making proper *sequential decisions* could be unintuitive because moving an object to its target position usually is not a one-step process. For example, to move a dining table to a new position, one may first need to move all the chairs around the table to some temporary positions. Making proper *long-term decisions* could also be tricky as the movement of an object may affect the subsequent movements of other objects. An object moved earlier on may block the way of other objects trying to get into their target positions, which could only be resolved by undoing the former movements.

Finding a feasible sequence of move actions by exhaustive search is hard due to the exponential search complexity. Suppose there are 10 objects in a scene and each object can move in 4 directions (up, down, left, and right). If the length of a move action sequence is 10, there will be 40^{10} possible sequences of move actions, making an exhaustive search difficult. One may rely on a rule-based approach to find a solution, e.g., moving bigger objects to their target positions first. However, designing a set of effective rules is difficult, which are generally abstracted from empirical knowledge gained from a lot of trials. On the other hand, it could be difficult to decide which rule to apply based on the current layout.

To address the sequential long-term decision problem, we propose the *Scene Mover* approach to yield a move plan automatically. *Scene Mover* can be considered as an agent trained by deep reinforcement learning for moving objects. Given an initial layout and a target layout, the *Scene Mover* agent generates a move plan comprising a sequence of objects selected to move and their move paths. We train a neural network that leverages the power of deep reinforcement learning to learn the mapping from the layout space to the expected accumulated rewards of move actions. Using a LSTM module, the network also models temporal causality by considering historical movement information.

To enable long-term decisions, we introduce a Monte Carlo tree search (MCTS) to capture the causality between the current and future actions. We define a sparse action space consisting of some primitive actions for shortening the move action sequence so as to improve the search efficiency.

Our approach comprises two main stages. In the first stage, it trains a deep reinforcement learning network which estimates the priors of the move actions for MCTS. In the second stage, it employs a MCTS approach which embeds the trained network to make decisions to generate a move plan iteratively. In each step of the MCTS, our approach selects an object and applies a path for moving the object. This is repeated until all objects reach their target positions.

Our approach automates the generation of a move plan for scene arrangement and facilitates the realization of scene designs, leading to potential applications such as warehouse automation and smart homes with movable furniture. It also provides insights into other sequential decision-making design problems such as reconfigurable product designs.

Major contributions of our work include:

- Proposing a novel problem of automatic move planning to transform an initial layout into a target layout, which can complement a scene synthesis algorithm for realizing a synthesized scene layout.
- Proposing a deep reinforcement learning-based Monte Carlo tree search approach for solving the move planning problem.
- Performing experiments to evaluate the effectiveness and performance of our approach, as well as comparing it with humans and other baseline approaches. We also demonstrate how our approach can be applied to move objects under different practical scenarios and constraints.

2 RELATED WORK

2.1 Scene Synthesis

There are considerable research efforts spent on synthesizing furniture arrangements and indoor scene layouts [Fisher et al. 2015; Fu et al. 2017; Li et al. 2019; Qi et al. 2018; Yu et al. 2011], as well as building layouts [Merrell et al. 2010; Peng et al. 2014; Wu et al. 2018]. Merrell et al. [2010] proposed a data-driven method to generate residential building layouts. A Bayesian network is trained based on example architectural programs to model the relationships among different rooms. Yeh et al. [2012] specified priors over which objects may occur and object spatial relationships for synthesizing scenes.

Another stream of works focused on data-driven scene synthesis. Yu et al. [2011] extracted hierarchical and spatial relationships for different furniture objects, encoding them into priors to synthesize realistic furniture arrangements. Qi et al. [2018] learned layout distributions from an indoor scene dataset, which were sampled to generate new layouts. Human contexts were also considered in the synthesis. Wang et al. [2018] used deep convolutional networks to learn priors capturing object existence and spatial relationships for generating scenes. Very recently, significant progress has been made on using deep convolutional generative model [Ritchie et al. 2019], graph-based generative model [Wang et al. 2019a], and generative recursive autoencoders [Li et al. 2019] for indoor scene synthesis.

In contrast to the scene synthesis approaches, which consider aesthetics, functionality, and general human activities to generate

reasonable scenes, our approach focuses on how to realize the synthesized scene layout. Our approach may complement existing scene synthesis approaches to fill the gap between design and realization. For example, a user may apply a scene synthesis approach to synthesize a layout for an apartment. Then he may apply our approach to generate a move plan to move furniture objects from the existing initial layout to the synthesized layout.

2.2 Path Planning

Path planning is another line of research relevant to our problem. Path planning has been broadly applied for robot navigation [Gayle et al. 2005; van den Berg et al. 2009], aerial vehicles [Best et al. 2017], and agent simulation [Sud et al. 2008]. Roughly, path planning approaches are divided into three categories: roadmap, cell decomposition, and artificial potential. Please refer to a survey [Gasparetto et al. 2015] for a comprehensive review of path planning.

Roadmap approaches, such as visibility graphs [Alexopoulos and Griffin 1992; Wang et al. 2020], Voronoi diagrams [Garrido et al. 2011], Delaunay triangulation [Jan et al. 2014], map free space connectivity into a system of curves. Cell decomposition approaches [Brooks and Lozano-Perez 1985] divide the free space into small cells and use a connectivity graph to represent the adjacency relationships between cells. To search for a path in a graph constructed by a roadmap or a cell decomposition approach, a search algorithm such as A* [Hart et al. 1968] is applied.

Different from path planning, our approach generates a move plan. It computes not only a plausible path for each object, but also a global strategy of the overall move action sequence to ensure that each object reaches its target position. A different setting is multi-agent path planning. Previous works handled this setting using optimization approaches [Biswas et al. 2017]. If moving objects simultaneously is not essential, our approach could be applied to generate a sequence of movements of objects to achieve the goal.

2.3 Sequential Decision Making

In our work, we represent the problem of moving a set of objects to their target positions as a sequential decision problem, which is formulated as a Markov decision process.

Sequential decision-making refers to following a procedural approach or a step-by-step decision theory for decision-making, where the early decisions influence the subsequent available choices [Frankish and Ramsey 2014]. Sequential decision problems are commonly formulated as Markov decision processes. Such problems represent a wide range of real-world tasks such as robot control [Kober et al. 2013], game playing [Silver et al. 2016; Wiering and Van Otterlo 2012], and military planning [Aberdeen et al. 2004]. In particular, using sequential decision methods to play games, e.g., computer chess [Campbell et al. 2002], Go [Silver et al. 2017], and Atari 2600 [Hausknecht and Stone 2015], is an appealing application.

2.4 Rearrangement Planning

Robotics researchers have been studying the rearrangement planning problem. In the robotics field, considerable efforts have been spent on accomplishing tabletop rearrangement tasks using a robotic arm with nonprehensile actions like pushing. Comparing to actions like grasping or picking up objects [Labbé et al. 2020], nonprehensile actions are easier to execute. However, collisions with static objects need to be avoided when applying nonprehensile actions for rearrangement, causing additional challenges.

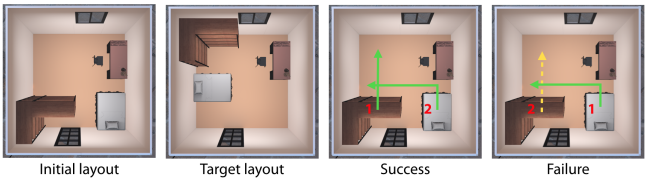


Fig. 2. Different move action sequences may succeed or fail to realize the target layout from the initial layout. The red numbers show the moving order of the objects. The yellow dashed arrow refers to a blocked path.

A rearrangement task generally comes with a certain target configuration of the objects. In this case, some works [Yuan et al. 2019, 2018] focused on planning the specific motions of the robotic arm to rearrange objects. For example, a scene had only one moveable cube and the goal was to find a collision-free move path to push the cube to its target position. An alternative rearrangement setting is based on object clustering [Song et al. 2019], where each object has a certain target region that it should move to.

Some works [Haustein et al. 2019, 2015; King et al. 2016, 2017; Koval et al. 2015; Song and Boularias 2019] targeted at multi-object rearrangement planning. They solved the problem in two stages, namely, a local path planning stage and a global strategy searching stage. In the local path planning stage, they used a RRT-based path searching alogrithm to find a valid path. In the global strategy stage, they employed heuristic tree searches.

Prior works in robotics focused on realizing the algorithm on a real robot, simplifying the rearrangement task complexity as possible to facilitate the adoption of a global strategy. For instance, the objects are often assumed to have regular shapes (e.g., cubes, cylinders). The poses of the objects are usually not considered because it is hard to orient objects via nonprehensile actions. The scenes usually have few or no obstacles and boundaries. In contrast, our approach aims to rearrange furniture objects in realistic indoor scenes with substantial complexity, comprising many irregularly-shaped objects placed densely as well as irregular obstacles and boundaries (e.g., walls, pillars). As conventional search-based algorithms could hardly cope with such scene complexities, we propose a novel neural network and adopt deep reinforcement learning to learn a planning policy by trial and error.

2.5 Deep Reinforcement Learning

Reinforcement learning is a strategy for solving the sequential decision problem. It is an experience-driven learning approach, in which an agent learns through trial-and-error interactions in a dynamic environment. With the significant progress of deep learning, reinforcement learning also benefits from incorporating deep learning, leading to deep reinforcement learning (DRL) approaches.

Anthony et al. [2017] presented Expert Iteration (EXIT), a reinforcement learning algorithm which decomposed the sequential decision problem into separate planning and generalization tasks. EXIT showed good performance in training a neural network to play the board game Hex. Silver et al. [2017] applied AlphaZero to

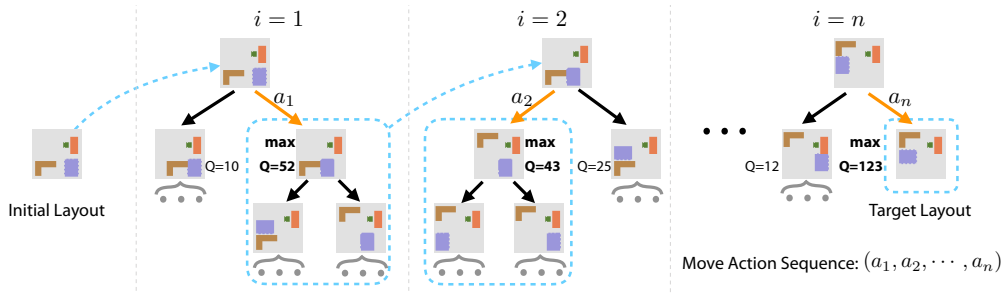


Fig. 3. Overall search process of *Scene Mover*. A sequence of search trees are created to produce a sequence of move actions. At each iteration, the search tree grows and gets updated through performing many rounds as shown in Fig. 4.

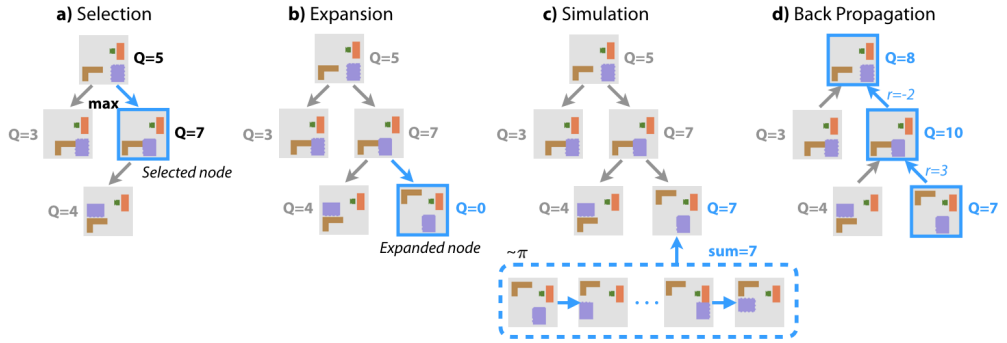


Fig. 4. Four steps in one round of search tree update.

the games of chess, shogi, and Go without any additional domain knowledge except the game rules, demonstrating that a general-purpose reinforcement learning algorithm could achieve superior performance across many challenging domains. Different from the games, our scene rearrangement task is a real-world task. We design representation and action space to facilitate move planning. Our method also takes the characteristic of sequential decision making into account. We introduce the LSTM layer into the Q-network so as to leverage historical move information.

Levine et al. [2018] proposed to learn control policies for robots directly from camera inputs in real world. Mnih et al. [2015] introduced Deep Q-Network (DQN), which stabilized the training of Q-function approximation with deep neural networks. The algorithm performed well on 49 Atari games. Andrychowicz et al. [2016] cast the design of an optimization algorithm as a learning problem, allowing the algorithm to learn to perform more efficiently. O'Donoghue et al. [2016] proposed to combine policy gradient with off-policy Q-learning (PGQ) to benefit from experience replay.

In our work, we leverage deep reinforcement learning techniques to train a neural network for estimating the long-term reward for each proposed move action. We apply the network to search for a move plan to achieve scene rearrangement automatically.

3 OVERVIEW

Our goal is to find a feasible move plan following which all objects in the initial layout can be moved to their target positions in the target layout. The move plan consists of a sequence of move actions, where each action represents selecting one object in the current layout and taking a path to move the object to a new position.

In the move plan, all actions are coherent, that is, selecting and moving one object will affect another action in the future. This is a long-term decision problem. For example, as shown in Fig. 2, the bed moved earlier on may block the way of the bookshelf moved later, causing failure in realizing the target layout.

To deal with such a long-term decision problem and to accomplish the moving task efficiently, we propose a Monte Carlo tree search approach embedded with a neural network to search for a feasible move plan. In this section, we give an overview of our framework.

3.1 Monte Carlo Tree Search

Due to the enormous search space of move plans, an efficient searching strategy is crucial. We adopt the Monte Carlo tree search (MCTS) framework to solve the search problem as shown in Fig. 3. The main idea is to construct a sequence of search trees to create a sequence of move actions iteratively. Each search tree corresponds to one move action decision.

Tree Structure. Fig. 4(a) shows an example search tree. A search tree is a rooted tree. Each node represents a layout after taking a sequence of move actions from the root node. The root node refers to the current layout. A node and its child nodes are connected by edges. By executing an action (i.e., selecting an object and moving it along a certain path), the layout represented by the parent node is transformed into the layout represented by the corresponding child node. Each node stores a Q-value, which refers to the expected future accumulated rewards for achieving the target layout starting from the layout represented by that node.

Search Tree Update. To decide on a move action, a search tree grows and gets updated for many rounds to produce a "final tree"

based on which the decision is made. A round of update involves four steps as shown in Fig. 4:

(a) **Selection**: In this step, a node which if expanded may lead the search towards finding a feasible move plan to reach the target layout, is selected to expand the search tree. The selection is performed by searching the child node with the maximum Q-value starting from the root node. In Fig. 4(a), the node highlighted in blue, which has the maximum Q-value, is selected. We discuss details of the selection in Sec. 6.
(b) **Expansion**: If the selected node does not represent the target layout, a legal action is chosen according to some policy and is applied to create a child node (referred as "expanded node"). In Fig. 4(b), the node highlighted in blue is the expanded node. The action taken is to move an object towards its up until an obstacle is reached. We provide details of the types of move actions in Sec. 4.2.
(c) **Simulation**: Based on the expanded node, a simulation is performed. During the simulation, a sequence of move actions are chosen according to some policy and are performed to transform the layout until a terminal layout is reached (e.g., reaching the target layout or the maximum number of simulation steps allowed). A simulation process is depicted by the dashed box in Fig. 4(c). After the simulation, the Q-value of the expanded node is updated according to the accumulated rewards of the move actions applied during the simulation.
(d) **Back Propagation**: Q-value updates are back-propagated from the expanded node towards the root node. All the ancestor nodes along the path from the expanded node to the root node (highlighted in blue in Fig. 4(d)) have their Q-values updated with the rewards associated with the edges (which denote move actions taken).

Move Action Decision. By performing these four steps, the tree grows and is updated in each round. A certain number of rounds are applied to grow the tree into a final tree, based on which a move action decision is made to modify the current layout by a step (e.g., moving an object towards its right).

Overall Search Process. Fig. 3 shows the overall search process to generate a sequence of move actions. Starting from the initial layout, at each iteration a search tree is grown into a final search tree. Out of the actions associated with the final search tree's root node (referring to the current layout), the action that leads to the child node with the maximum Q-value is chosen to update the current layout. In addition, the subtree rooted at that child node is used as the initial search tree for the next iteration. As depicted in Fig. 3, in the i-th iteration, a_i is the move action chosen. The subtree in the blue dashed box is used as the initial tree in the $i+1$-th iteration. The iterative process continues until the target layout is reached. This process results in a sequence of move actions $(a_1, a_2, \cdots, a_n)$ as the output for moving the furniture objects step by step.

3.2 Reinforcement Learning of Q-Network

The performance of MCTS depends on the policy used in the expansion and simulation steps. In the expansion step, the policy guides the search to explore move actions which are likely to lead to the target layout. In the simulation step, the policy helps to choose actions for evolving the layout represented by the expanded node for a number of times so as to obtain a good estimate of the expanded node's Q-value, which is back-propagated to update the Q-values of its ancestor nodes.

Our approach trains a neural network to facilitate the search process. The goal is to maximize the expected return, i.e., cumulative discounted reward, in the task of transforming an initial layout into a target layout. To achieve such training, we apply the unsupervised learning strategy of reinforcement learning through trials and errors. On the other hand, we leverage the powerful function approximation properties of deep neural networks to regress the policy function. We discuss the training process in Sec. 5.1.

After training, given the current layout, the Q-network can be employed to predict the Q-value of the layout resulting from applying each move action on the current layout. The Q-network is employed to choose move actions in such a way in the expansion and simulation steps, i.e., choosing a move action to create the expanded node in the expansion step and choosing move actions to evolve the layout represented by the expanded node in the simulation step. As the network is trained by many trials on many different layouts, compared to a simple rule-based policy, it is general, scalable and can tackle complex scene arrangement scenarios.

4 FORMULATION

4.1 Problem Definition

Given an initial layout l_I and a target layout l_T, the goal of our approach is to generate a feasible move plan, i.e., generating a move action sequence $A = (a_1, a_2, ..., a_n)$, following which l_I is transformed into l_T. $a_i \in \mathcal{A}$ and $\mathcal{A}$ represents the move action space.

Formally, the goal of our approach is to maximize the accumulated reward for transforming the initial layout l_I into the target layout l_T by a sequence of move actions:

$$\arg\max_{A} \sum_{t=1}^{\|A\|} r(l_t, a_t), \tag{1}$$

where $l_1 = l_I$ and $l_{\|A\|+1} = l_T$. $r(l, a) : (\mathcal{L}, \mathcal{A}) \rightarrow \mathbb{R}$ is a reward function. l_t is the root layout of the search tree at the t-th move action decision state. $l_t = e(l_{t-1}, a_{t-1})$, where $e(l, a) : (\mathcal{L}, \mathcal{A}) \rightarrow \mathcal{L}$ is an environment simulator. Given a layout l_{t-1} and an action a_{t-1}, the simulator returns the next layout l_t obtained after performing the action a_{t-1}.

Layout Representation. A layout l is represented as a series of matrices. A 3D scene is first projected onto the floor plane from a top-down view. Then we discretize the projection into a $M \times N$ grid. The objects in the layout are denoted as $O = \{o_i\}$. The ith object is represented as a $M \times N$ matrix, whose element stores the occupancy of the object, i.e., whether the object is projected onto the corresponding cell (Fig. 5).

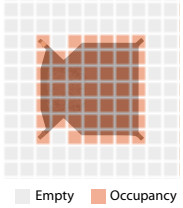


Fig. 5. Discretization of an object in the layout.

4.2 Reinforcement Learning

Move Action. A move action is represented by a pair $a =< o_i, p >$, $o_i \in O, p = (< x_1, y_1, \phi_1 >, < x_2, y_2, \phi_2 >, \cdots)$. o_i is the object chosen to move. p is the move path which is an ordered sequence of tuples. Each tuple consists of the x, y coordinates and the orientation ϕ, meaning that the center of the object o_i is moved to (x, y) with an orientation of ϕ. We discretize the orientation of an object into 24 bins with an interval of 15°.

Path Types: We define five types of paths to move the chosen object. That is, $p \in \{p_1, p_2, \cdots, p_5\}$. The first four types of paths refer to straight paths defined as moving an object straightly along the up (p_1), down (p_2), left (p_3), or right (p_4) direction until the moved object reaches an obstacle (i.e., another object or the wall). Comparing to moving an object cell by cell, moving an object following one of these paths greatly shortens the move action sequence in general.

The fifth type of path (p_5) refers to a collision-free path searched from the object's current position to the the object's target position. Such a path is searched through an A^* search algorithm [Hart et al. 1968]. At each step along this path, the object is able to move to any of its four adjacent cells as long as there is no collision with any obstacle. Note that we also allow the object to dodge obstacles by adjusting its orientation along the path. The fifth type of path does not exist if there is no such collision-free path.

Using the five types of paths allows our approach to balance between search complexity and movement flexibility. First, the larger the number of types of paths we have, the higher the complexity of the action space $\mathcal{A}$ is and the larger the breadth of the search tree is, which increases the search complexity, although using more types of paths would allow objects to move in the layout more flexibly. Second, the fine-graininess of an object's movement resulting from following a path affects the length of the move action sequence. The more fine-grained the movement is, the deeper the search tree is, and a longer move action sequence needs to be formed to reach the target layout, although using more fine-grained movements would allow objects to move in the layout more flexibly. To this end, we use the five types of paths aforementioned to balance between search complexity and flexibility of the movements allowed.

Rewards. At moment t, the *Scene Mover* agent observes the layout l_t and selects an action a_t, which is passed to the environment. An environment simulator $e(l, a) : (\mathcal{L}, \mathcal{A}) \rightarrow \mathcal{L}$ and a reward function $r(l, a) : (\mathcal{L}, \mathcal{A}) \rightarrow \mathbb{R}$ are executed to obtain the next layout l_{t+1} and reward r_t. We define 7 types of rewards whose values are shown in Table 1. An action results in a reward as per the following:

- ***Base***: Each legal action results in a base negative reward (penalty), which regularizes the move action sequence's length.
- ***Repetition***: The action results in a layout that appeared before.
- ***First-arrival***: By taking the action, an object arrives at its target position for the first time.
- ***Multi-arrival***: By taking the action, an object arrives at its target position again (not for the first time).
- ***First-leave***: By taking the action, an object which is currently at its target position leaves for the first time.
- ***Multi-leave***: By taking the action, an object currently at its target position leaves again. Note that this object has arrived at and then left its target position before. A negative reward (penalty) is imposed to discourage such repeated leaving.
- ***Success***: After taking the action, all objects have arrived at their target positions, hence the target layout is achieved.

Table 1. Rewards used.

	Reward		Reward
Base	-1	First-arrival	4
Repetition	-2	Multi-arrival	2
Success	50	First-leave	-4
		Multi-leave	-2

Note that if an action results in a state that meets multiple criteria (e.g., Base and Repetition), the resulting reward is the sum of reward of each criterion.

5 Q-NETWORK

To facilitate expansion and simulation of the MCTS process, our approach trains a network which approximates the Q-function, known as Q-network. With the Q-network, the *Scene Mover* agent determines which move action it should take to expand the selected node and how to perform simulation starting from the expanded node.

Our approach aims to learn a policy based on which move actions are chosen to maximize the expected future accumulated rewards for achieving the target layout. The optimal Q-function $Q_p^*(l, a)$ is defined as:

$$Q_p^*(l, a) = \max_{\pi} \mathbb{E}[R_t | l_t = l, a_t = a, \pi], \tag{2}$$

where π is a policy. The subscript p denotes that this Q-function is approximated by the Q-network as we describe as follows.

R_t is the future accumulated rewards at move action decision state t. Our approach makes a common assumption that the future rewards are decayed by a factor over time steps. The accumulated reward is defined as $R_t = \sum_{i=t}^{\tau} \gamma^{i-t} r_i$, where γ is the decay factor set as 0.95 empirically in our approach to favor learning convergence. With the layouts being further from the current move decision state t, their rewards discount more in computing the accumulated reward R_t. r_i is the reward at move decision state i, which is defined in Table 1. τ is the total number of move decision states.

According to the Bellman equation [Sutton and Barto 2011], if the optimal value $Q_p^*(l', a')$ of the layout l' at the next state is known for all possible action a', the optimal strategy for achieving Eq. (2) is to select the action a' maximizing the expected value of $r + \gamma Q_p^*(l', a')$ [Mnih et al. 2015]. Here r is the reward of executing action a. Eq. (2) can be adapted as:

$$Q_p^*(l, a) = r + \gamma \max_{a'} Q_p^*(l', a'). \tag{3}$$

We use a convolutional network (referred as the Q-network) with weights θ to approximate the optimal Q-function $Q_p^*(l, a)$. The output of the network is a vector which contains the same number of entries as the number of move actions in the move action space $\mathcal{A}$. Recall that each move action $a =< o_i, p >$ contains an object o_i and a path p which denotes how it should move (Sec. 4.2). Each entry of the Q-network's output vector contains the approximation of

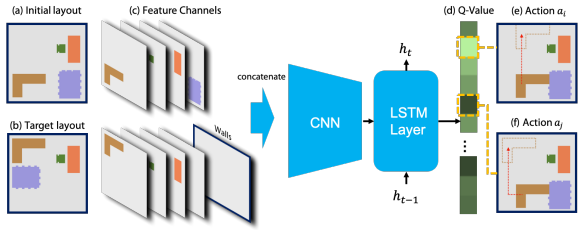


Fig. 6. Our network structure. (a-b) show the initial and target layouts. (c) The input of the Q-network includes the feature channels extracted from (a-b). Furniture objects are highlighted in different colors. (d) The output is a vector with each entry storing the predicted Q-value of the layout resulting from taking an action. (e-f) show two examples of move actions.

the optimal Q-function $Q_p^*(l, a)$ for each move action (i.e., selecting an object and moving it by one of the five types of paths). Such an approximation is also referred as the Q-value.

5.1 Network Structure

Input and Output. At time step t, the input of the network includes an initial layout, a target layout, and the hidden state of the LSTM layer from the last time step h_{t-1} as depicted in Fig. 6.

Both the initial layout l_I and target layout l_T are represented by a series of feature channels of $M \times N$ matrices. Each feature channel is used to encode the location and appearance of an object. For a certain channel, each entry of the matrix corresponds to a cell in the discretized, grid-representation of the layout discussed in Sec. 4.. The entry stores a 0 if the cell location is unoccupied in the layout; or a 1 if the cell location coincides with the object that this channel represents. Suppose there are K objects in the scene. Then there are $2K$ channels to represent the intial layout and the target layout. An additional channel is used for encoding the unmovable objects (e.g., walls). So there are $2K + 1$ channels in total.

The output of the network is the hidden h_t, a vector of length $K|\mathcal{A}|$, with each entry representing the predicted Q-value of taking a move action, i.e., moving an object by a path.

Structure. The backbone of the Q-network is a convolutional network whose purpose is to regress the optimal Q-function. To help make long-term decisions, we employ a LSTM layer which helps to embed the historical information. The structure is shown in Fig. 6. Given the input described above, for the initial and the target layout, we first apply a CNN to encode them as feature maps. The feature maps are flattened to a vector. At last, an LSTM layer is appended to the concatenated vector as the regressor. Using LSTM units as the hidden layer helps model temporal causality.

Loss. The weights of the network are updated iteratively. For iteration i, the network with weights $\theta^{(i)}$ is trained by minimizing the following loss function:

$$Loss^{(i)} = \mathbb{E}_{l, a\sim\rho(\cdot)}[(y^{(i)} - Q_p(l, a; \theta^{(i)}))^2], \tag{4}$$

where $y^{(i)} = r + \gamma \max_{a'} Q_p(l', a'; \theta^{(i-1)})$ is the update target and ρ is the distribution over actions a.

5.2 Training

Training Data Synthesis. Taking advantage of reinforcement learning, our approach trains the network with random synthetic layouts. Fig. 7 shows an example of a synthetic layout. A synthetic layout is synthesized by two steps: obstacle synthesis and object synthesis.

Obstacles: Given an empty layout, our approach grows obstacles following the Bernoulli distribution. Specifically, an obstacle starts to grow from a cell located at the grid boundary following a random initial direction. At each step it grows one cell along the current direction. It chooses a new direction to grow with a probability of α and terminates the growth with a probability of β. In our approach, we use $\alpha = 0.5$ and $\beta = 0.2$.

Objects: Each object is represented as a rectangle with a random size. Using different sizes of rectangles trains the convolutional network to handle shape variations. Note that in the testing experiments the objects can be of non-rectangular shapes.

The initial and target positions for each object are synthesized randomly. Our approach uses collision detection during the synthesis to avoid overlapping between objects. There are more than 10k layouts synthesized automatically for training the Q-network.

Training Details. The dimensions of the grids of the training layouts are 64×64. We design the network for at most 25 objects. The input channels are shuffled during training to enforce the network to learn object-specific correspondences. The length of the output vector is assigned as 125 (each of the 25 objects has 5 types of move). If there are less than 25 objects, the addtional channels are filling with zeros. To handle more objects, the length of the output vector can be increased accordingly.

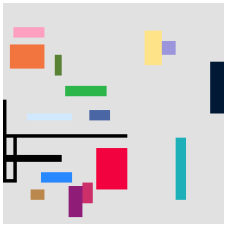


Fig. 7. A synthetic layout.

We adopt ϵ-greedy strategy during training where $\epsilon = 0.1 + 0.9 * exp(-0.0001 * i)$ for iteration i. It has ϵ probability to do exploration which is conducted through a heuristic search method. The batch size is 64. ADAM solver is used for training with a learning rate of 0.0001. We implemented *Scene Mover* using Python. The training of the Q-network was implemented using TensorFlow. The training took about one day to finish on a machine equipped with an Intel Core i7-5930K CPU and an NVIDIA TITAN GPU with 12GB RAM.

6 MOVE ACTION SEQUENCE SEARCH

Our approach leverages MCTS incorporated with the trained policy network to find the move action sequence for a given pair of initial and target layouts. We introduce the details of the search process.

As described in Sec. 3.1 and Fig. 3, a sequence of search trees are created to produce a sequence of move actions. To explain how a search tree is grown based on which the move action decision is made, we take the t-th iteration as an example.

In this case, a search tree is grown for making the t-th move action decision. The root node of this tree represents the layout l_t. Each node of this tree represents a layout l and stores a Q-value $Q_m(l)$ that refers to the current estimated Q-value of layout l. A number of rounds (200 rounds in our experiments) are run to grow and update this tree by adding more nodes and refining the estimated Q-values $Q_m(\cdot)$. Each round includes four steps detailed below:

Selection. Our approach first selects a tree node with the maximum estimated Q-value to expand. Formally, the selection of the tree node

Table 2. Results of different synthetic layouts.

Layout	No. of Objects (total/need to move)	Size (m × m)	No. of Actions
Living Room	15/10	10.0 × 10.0	30
Gym	10/6	10.0 × 18.5	6
Library	25/18	18.0 × 15.5	79
Restaurant	25/25	20.0 × 20.0	25

representing layout $\hat{l}$ is given by:

$$\begin{aligned} \hat{l} &= \arg\max_{l} \widetilde{Q}(l), \\ \widetilde{Q}(l) &= Q_m(l) + u(l), \end{aligned} \tag{5}$$

where the Q-function $Q_m(l)$ is normalized to $[0, 1]$ by $\frac{Q_m(l)-Q_{\min}+1}{Q_{\max}-Q_{\min}+1}$; $Q_{\min}$ and $Q_{\max}$ are the minimum and the maximum estimated Q-values in the current search tree respectively. $u(l)$ is a term for trading off the breadth and depth of the search, which is defined as:

$$u(l) \propto \sqrt{\frac{log(N(l_p))}{1 + N(l)}}. \tag{6}$$

where $N(l)$ and $N(l_p)$ are the numbers of times of visits of the node representing layout l, and of the node's parent representing layout l_p, respectively. This formula is derived from UCB1 [Auer et al. 2002] which aims to balance exploration and exploitation.

Expansion. After the selection, our approach takes a legal action to expand the selected node representing layout $\hat{l}$ to create a new node (expanded node). More specifically, using the layout $\hat{l}$ represented by the selected node as input, the Q-network is inquired to estimate the Q-value $Q_p()$ of each hypothetical layout created from layout $\hat{l}$ by following each of the possible move actions. The move action $\hat{a}$ used to create the expanded node is sampled from the distribution $p(\hat{l}, a)$, which depends on the Q-value $Q_p(l)$ estimated by the Q-network,

$$\hat{a} \sim p(\hat{l}, a), \quad p(\hat{l}, a) = \text{softmax}\, Q_p(\hat{l}, a). \tag{7}$$

Then we get the expanded node which represents a layout $l^{(0)}$, where $l^{(0)} = e(\hat{l}, \hat{a})$. We use 0 in the superscript as it is used as the initial layout in the simulation step.

Simulation. A simulation is performed from the expanded node $l^{(0)}$. During the simulation, a sequence of move actions $a^{(0)}, a^{(1)}, \cdots$ are performed. Each action $a^{(i)}$ is sampled from the distribution depending on the Q-values estimated by the Q-network:

$$a^{(i)} \sim p(l^{(i)}, a).$$

The simulation stops if the layout reaches the target layout (i.e., $l^{(i)} = l_T$) or the simulation reaches a maximum number of steps allowed (20 steps allowed in our experiment). The simulation returns an accumulated reward to the expanded node:

$$Q_m(l^{(0)}) = \sum_{i=0}^{D} \gamma^i r(l^{(i)}, a^{(i)}), \tag{8}$$

where D is the number of simulation steps.

Back Propagation. After the simulation, the current estimated Q-values $Q_m()$ stored at the nodes are updated from the expanded node via its ancestor nodes to the root node recursively. For a node representing layout l, its estimated Q-value $Q_m(l)$ is updated as:

$$\begin{aligned} a^* &= \arg\max_{a} Q_m(e(l, a)), \\ Q_m(l) &= \gamma Q_m(e(l, a^*) + r(l, a^*), \\ N(l) &= N(l) + 1. \end{aligned} \tag{9}$$

Move Action Decision. After a certain number of rounds, the growing of the search tree at the t-th iteration completes. The best action a_t^* of the root node is chosen as the move action to apply:

$$a_t^* = \max_{a} Q_m(e(l_t, a)). \tag{10}$$

We include the pseudocode of an iteration of *Scene Mover*'s MCTS in the supplementary material.

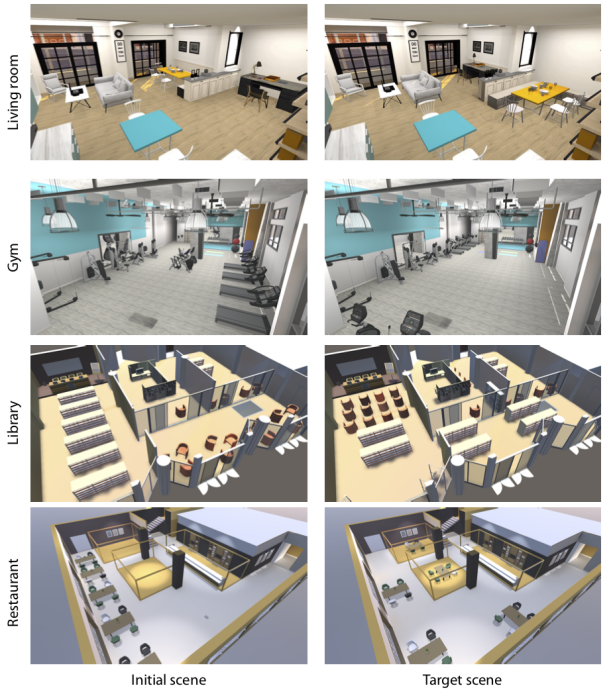


Fig. 8. Synthetic scenes. Please refer to our supplementary video for the generated move action sequences.

7 EXPERIMENTS

7.1 Synthetic Scenes

We tested *Scene Mover* on four synthetic scenes, consisting of a living room, a gym, a library, and a restaurant. Both the initial and the target scenes are shown in Fig. 8. Each scene has a different layout furnished with various furniture. Some details are reported in Table 2. Note that not all objects change their locations between the initial and the target scene. For example, the library has 25 furniture objects in total, of which 18 objects change their locations.

In the generated move plans, we observe some interesting actions. Take the living room shown in Fig. 9 as an example. To make way for the dining table, as shown in Fig. 9(b), *Scene Mover* moved the obstacles (square table, chairs, desk, etc.), highlighted by a yellow dashed bounding box, to the lower-left corner. After that, it moved

Fig. 9. "Smart moves" performed by *Scene Mover* in the living room example. (a) Initial layout. (b) To make way for the dining table to reach its target position, *Scene Mover* moved the obstacles, highlighted by a yellow dashed bounding box, to the lower-left corner, and then moved the bench to the lower-right corner. (c) *Scene Mover* moved the dining table to its target position. (d) Layout after moving the dining table. (e) Target layout. The areas to be swapped are high-lighted by the pink and green box in (a) and (e).

Fig. 10. Adding furniture to an existing layout. (a) A refrigerator (in cyan) needed to move to its target position in the kitchen. (b) The intermediate layout before the refrigerator was moved to its target position. Note that *Scene Mover* moved some existing furniture objects (in red dashed boxes) to make way for the incoming object.

the bench to the lower-right corner and then moved the dining table to its target position as shown in Fig. 9(c).

7.2 Adding Furniture to an Existing Layout

People buy new furniture to refurnish their homes sometimes. *Scene Mover* can be employed to add new furniture to an existing layout. Fig. 10 shows an example. A new refrigerator needs to be moved from the entrance to its target position in the kitchen. Though all other objects stay at their original positions in the new layout, they could be temporarily moved to give way to the refrigerator.

In this example, *Scene Mover* moved the refrigerator to its target position through 11 move actions. Specifically, 6 move actions were performed to make way for the refrigerator, e.g., moving 3 chairs and the dining table away from their original positions (highlighted by red dashed bounding boxes in Fig. 10(b)). After the refrigerator had reached its target position, each moved object were moved back to its original position.

7.3 Furnishing a Vacant House

For moving a house, the *Scene Mover* agent is capable of generating a move plan automatically to furnish a vacant house. Fig. 11 shows such a scenario, where a van has unloaded all furniture in the yard waiting to be moved inside the house.

To tackle this situation, we extend the range of the initial layout by setting a boundary to cover both the house and the yard region as shown in Fig. 11(a). There are 15 furniture objects in total. The

Fig. 11. Furnishing a vacant house. (a) Initial layout with many furniture objects in the front yard waiting to be moved. (b) Target layout where the objects have been moved to their target positions by *Scene Mover*.

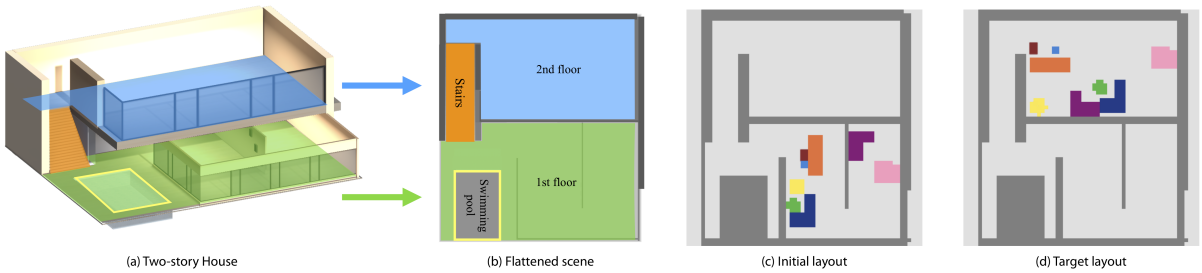
Scene Mover agent took 35 move actions to move all furniture into the house to realize the target layout as shown in Fig. 11(b).

In this special scenario, the *Scene Mover* agent needs to consider object placement and movement both outside and inside the house, which may have conflicts with each other. Intuitively, it would be convenient to move the objects near the entrance inside the house first so as not to obstruct other objects' from entering the house. On the other hand, it is important to consider the target positions of such objects in case placing them early would obstruct the other incoming objects. For example, moving the white bench (which is far away from the entrance) to its target position (bottom-right corner of the house) late might fail, as it could be blocked by other objects (e.g., the dining table) near the entrance. We observed that *Scene Mover* tended to first move the objects (e.g., armchair, desk) whose target positions were at the room corners, which would not obstruct the other incoming objects.

7.4 Terrain Constraints

When arranging furniture in a house we may have to consider some terrain constraints. Take the layout in Fig. 12(a) as an example, which shows a two-story house with a swimming pool in the yard. There are two terrain constraints: (1) furniture may locate on different floors and the stairway is the only passageway connecting the two floors; (2) the swimming pool is not passable.

To handle the terrain constraints, we represented the problem as a restricted move planning problem and solved it by the *Scene Mover* agent. The initial and target layouts were preprocessed by two steps. First, the two floors were flattened into one plane as



(a) Two-story House (b) Flattened scene (c) Initial layout (d) Target layout

Fig. 12. Terrain constraints. (a) A two-story house with a swimming pool in the yard. (b) The house is flattened into one plane. The orange region refers to the stairway. The gray region enclosed by the yellow rectangle refers to the swimming pool. (c) Initial layout. (d) Target layout.

Table 3. The average travel length, number of steps and travel length per step of our approach with and without the travel length constraint.

Agent	Ave. TL	Ave. # of Steps	Ave. TL/Step
Scene Mover	588.4	**15.0**	39.4
Scene Mover (constraint)	**573.2**	15.8	**36.2**

shown in Fig. 12(b). The two floors were separated by unmovable walls and the stairway (orange region) connected the two floors. Second, the unreachable region was bounded by unmovable walls and no path could cross them. In Fig. 12(b), the pool is enclosed by a yellow rectangle which represents the bounding walls added.

7.5 Travel Length Constraint

We can extend our approach to handle different practical scenarios by incorporating additional terms in the reward scheme of the MCTS. For example, we may prompt *Scene Mover* to generate a move plan that involves less movement effort characterized by a short travel length by adding a term to the reward as a soft constraint.

Specifically, since a layout is represented as a $N \times M$ grid, we estimate the distance between two cells by their Manhattan distance. The travel length is defined as the accumulated distance along the path connecting the cells. In the t-th action, assume that the travel length in the single move a_t is L_t. We add a term $-\lambda \log L_t$ to penalize the travel length, where λ is a trade-off parameter to balance the travel length's penalty (i.e., a negative reward) with the existing rewards. The goal in Eq. 1 can be extended as:

$$\arg\max_{A} \sum_{t=1}^{\|A\|} r(l_t, a_t) - \lambda \log L_t. \tag{11}$$

In our test, we set $\lambda = 0.5$. It is worth noting that we applied this soft constraint term in MCTS instead of in training the Q-network. We trained the Q-network to estimate Q-values for the fundamental task, i.e. realizing the target layout. The trained Q-network can then be used modularly. It can cooperate with specific constraints incorporated in the MCTS framework, such as the travel length constraint in this example, to handle specific considerations as needed. This modular design provides flexibility and convenience for extending our approach to handle different considerations.

To examine the effectiveness of the travel length constraint, we performed an evaluation to compare the travel lengths of the move plans generated by *Scene Mover* with and without using this constraint. We randomly generated 20 layouts as the testing set where each layout contained 13 objects.

Table 3 shows that with the travel length constraint, the average travel length decreased by 15.2 and the average travel length per step decreased by 3.2, while the average number of steps increased by 0.8. The decrease in the average travel length indicates that the constraint term prompts *Scene Mover* to choose some plausible actions with shorter travel lengths. The objective of using fewer steps and achieving a shorter travel length could be contradictory sometimes, which caused the number of steps to increase as the average travel length decreased after adding the travel length constraint. The objective can be controlled through the trade-off parameter λ, i.e., a larger λ results in move plans with shorter travel lengths.

8 EVALUATION ON SYNTHETIC LAYOUTS

In this section, we validate the effectiveness of our approach by comparing it against other agents and conducting an ablation study. We investigate the performance of *Scene Mover* and other agents under different difficulty levels. To facilitate quantitative analysis, we experimented on synthetic layouts, whose difficulty level can be controlled by adjusting the number of the objects in a scene. We validate our approach on real-world layouts in Sec. 9. All the results on synthetic layouts and real-world layouts in the quantitative evaluations were produced using one trained *Scene Mover*. It is worth noting that all the results of the compared approaches come from the best of multiple runs.

8.1 Data

The difficulty of generating a move plan generally increases with the number of objects present in the layout. In this experiment, we designed 4 difficulty levels corresponding to layouts with 5, 9, 13, and 17 objects, respectively. Fig. 13 shows some example layouts.

More specifically, for each difficulty level, we randomly synthesized 40 layouts containing a fixed number of objects of that level. Each object was a rectangle with random dimensions. The initial and target positions of the objects were also randomized. In total, we synthesized 160 layouts (4 levels $\times$ 40 layouts). We used half of the layouts in each level for training and the rest for testing.

8.2 Comparison

Human Agent. To investigate human performance in achieving the move planning task, we developed a Scene Mover mobile game to collect data of human participants. In the game, an initial layout and the target positions of all objects were shown to participants. The participants moved all objects to their target positions through selecting an object and taking one of the 5 given paths, which were consistent with the ones in *Scene Mover* and other agents so as to compare. The participants were asked to finish the task using as few steps as possible. There was a maximum of 100 move actions allowed and the participants would be ranked for each difficulty level by the number of the used move actions.

Before starting the game, each participant went through a familiarization process, where the participant learned about the mission (moving all objects to the target positions), the rules (selecting an object and applying one of the 5 paths), and the game control (clicking for selecting an object and swiping for choosing a path to apply). The participant would pass the game from the easiest level (5-objects layout) to the most difficult level (17-objects layout), one by one. In each level, one layout was randomly assigned to the participant to solve. The participant could access the next level only after he had passed the current level.

Heuristics Agent. The Heuristics agent was devised based on the naive rule-based heuristic search method. We designed 3 intuitive heuristic rules as follows.

Rule 1: If an object can be moved to its target position through the p_5 path (i.e., a collision-free path to the target position exists), select the object and move it along the p_5 path.

Rule 2: If multiple objects can be moved to their target positions through the p_5 path, randomly select one of them to move.

Rule 3: If no object can be moved to its target position through the p_5 path, randomly select an object and one of the 4 straight paths (i.e., p_1, p_2, p_3, or p_4) to move.

Note that although more heuristic rules could be included, it could be difficult to decide which rule to apply based on the current layout due to the complication of sequential decision-making.

MCTS Agent. MCTS was used as the backbone in the prior robotics works[King et al. 2017; Labbé et al. 2020]. Following these works, our MCTS agent was implemented based on a classic MCTS framework without a Q-network akin to [Chaslot et al. 2008]. The heuristic rules used by the Heuristics Agent were applied to enhance the expansion and simulation steps.

The MCTS agent followed a similar search process as that of the *Scene Mover* agent and used the same settings: a sequence of search trees were created to generate a move action sequence. At each iteration, an MCTS search tree grew and got updated for 200 rounds to form a final tree based on which a move action was determined. At each round, the simulation stopped if the simulated layout reached the target layout or if a limit of 20 simulation steps were reached.

8.3 Results and Analysis

Metrics. We used a match score mechanism to evaluate the agents' performances in generating move plans. There were 12 matches totally (3 compared agents × 4 difficulty levels). Each match comprised 20 layouts of the same difficulty level and was played by two agents, e.g., *Scene Mover* vs. Human.

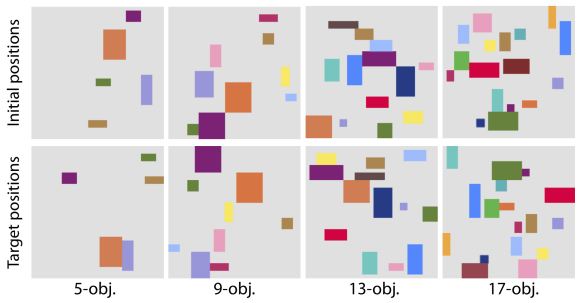


Fig. 13. Examples of the four difficulty levels.

In a layout, if an agent outplayed another agent, it received one point. If two agents played even, it was a draw and no agent would receive a point. The win and draw conditions in a layout were : (1) if both agents succeeded in generating a feasible move plan, the agent whose move plan had fewer move actions won; (2) If both agents succeeded and their move plans were with the same number of move actions, it was a draw; (3) If neither of the two agents succeeded, i.e., not reaching the target layout within 100 move actions allowed, it was a draw. Table 4 shows the scores of the 12 matches between the *Scene Mover* agent and the other three agents.

We calculate the success rates for each difficulty level. If an agent used up the maximum number of 100 move actions without reaching the target layout, this would be regarded as a failure. Table 5 shows the success rates of the four agents for different difficulty levels.

Comparison with Human Agent. From the Scene Mover game, we collected 1,191 playing records from 362 participants. Most participants were university students. We observed that some participants gave up at a certain difficulty level, which might happen as they found the level too difficult. 42 participants passed all four difficulty levels, who were probably more skilled than others who gave up in a low difficulty level. To make our comparison strict, only the data of these 42 skilled participants were used in calculating the scores. The median length of the move action sequences created by them was compared with the *Scene Mover* agent's result.

As Table 4 shows, the *Scene Mover* agent achieves better performance on most layouts comparing to human participants because human participants generally made some "trial" move actions (e.g., redundant or unnecessary move actions) to come up with a move plan, resulting in a longer action sequence.

In the easier levels of 5-obj and 9-obj, there are 4 layouts where human participants were able to find more optimal solutions than *Scene Mover*. Nevertheless, for the more difficult levels (13-obj.), *Scene Mover* outplayed the human agent in all layouts as the situation became much more complicated for human. For the most difficult level (17-obj.), the human agent still won in some layouts, which was reasonable as we compared *Scene Mover* with only the skilled participants rather than all participants. The boxplot in Fig. 14 shows that *Scene Mover* achieved lower medians compared to the human agent over all difficulty levels.

We also investigate the success rates in Table 5. Note that the success rates were calculated over all 362 participants. It shows that

Table 4. Comparing *Scene Mover* with other agents in different difficulty levels. Each entry shows a score in the format of A:B, where A and B refer to the numbers of wins of *Scene Mover* and the compared agent, respectively. We exclude those layouts where the two agents played even in calculating the scores.

Agent	5-obj.	9-obj.	13-obj.	17-obj.	Total
Scene Mover vs. Human	17:3	18:1	20:0	17:3	72:7
Scene Mover vs. Heuristics	12:4	19:1	20:0	18:0	69:5
Scene Mover vs. MCTS	3:5	16:1	17:0	17:1	53:7

Table 5. Success rates of different agents in different difficulty levels.

Agent	5-obj.	9-obj.	13-obj.	17-obj.
Human	100%	54%	43%	12%
Heuristics	100%	90%	50%	15%
MCTS	100%	100%	95%	70%
Scene Mover	100%	100%	100%	90%

with an increased level of difficulty the move planning task becomes more challenging for ordinary humans. We note that even though some skilled human participants could finish the difficult levels, it usually took them a significant amount of mental efforts to figure out a feasible move plan. As *Scene Mover* could generate a move plan automatically, it could help humans create move plans especially for layouts with many objects.

Comparison with Heuristics Agent. In the matches with the Heuristics agent, the *Scene Mover* agent won all 4 matches and achieved a total score of 69:5 as shown in Table 4. We observe that in the level with the fewest objects (5-obj.), the Heuristics agent and *Scene Mover* agent played 4 draws, whereas they played 1 draw in the 9-obj. level. Table 5 shows that for the 5-obj level, the Heuristics agent could find move plans for all layouts. However, the success rate decreased as the difficulty level increases.

The boxplot in Fig. 14 shows that *Scene Mover* attained much lower medians compared to the Heuristics agent in the levels of 9-obj., 13-obj., and 17-obj.. Note that in the 17-obj. level, the Heuristics agent frequently failed to solve the task within the limit of 100 actions (15% success rate) so the median shown was 100.

The results show that the heuristic approach could handle the move planning tasks well in low difficulty levels with a smaller number of objects. However, solving more difficult moving tasks might require defining a more sophisticated set of rules. The results also support that *Scene Mover* is capable of finding a feasible move action sequence, and the advantage becomes more apparent in tackling more difficult levels with more objects.

Comparison with MCTS Agent. The *Scene Mover* agent outplayed the MCTS agent overall in the matches of the 3 difficulty levels (9-obj., 13-obj., and 17-obj.). It achieves a total score of 53:7 as shown in Table 4. In the 5-obj. level, the MCTS agent and the *Scene Mover* agent performed similarly in terms of the match score. This is also supported by the statistics of the lengths of move action sequences shown in Fig. 14, where both the *Scene Mover* agent and the MCTS agent had a median of 5 move actions.

As the difficulty increased, the *Scene Mover* agent showed advantages. In the 9-obj. level, the score of 16:1 (*Scene Mover* vs. MCTS) reveals a marked difference of the two agents' performances. The medians of the *Scene Mover* and MCTS agents are 9 and 12 move

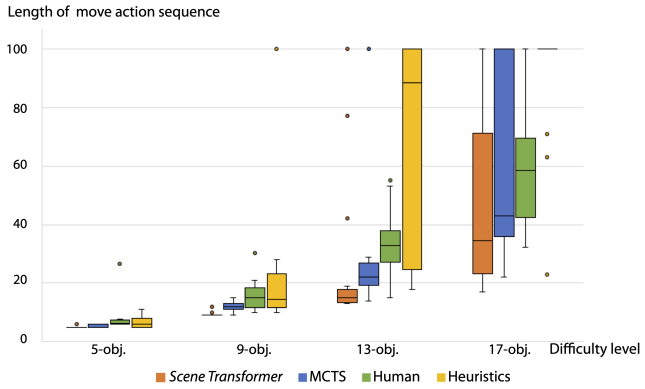


Fig. 14. A boxplot showing the statistics of the length of move action sequences computed by the *Scene Mover*, MCTS, Heuristics, and Human agent over the 4 difficulty levels. The black horizontal lines, bottom edges, and top edges of the boxes show the median length of the move action sequence, 25th percentiles, and 75th percentiles, respectively. The whiskers extend to the farthest data points not taken as outliers. Dots denote outliers.

actions. In the level of 13-obj. and 15-obj., the *Scene Mover* agent still maintained its lead in the matches. The scores of the *Scene Mover* vs. MCTS agent are 17:0 and 17:1, respectively. The medians of the *Scene Mover* agent and the MCTS agent are 14 and 22 in the 13-obj. level, while the medians are 35 and 43 in the 17-obj. level.

Table 5 shows that the success rates (100%) of the *Scene Mover* agent equate those of the MCTS agent in the 5-obj. and 9-obj. levels. The MCTS seems to enable it to escape from a local minimum in some situations with a random action selection strategy. In the most difficult 17-obj. level, the *Scene Mover* agent with a 90% success rate outperforms the MCTS agent with a 70% success rate.

We believe the Q-network of the *Scene Mover* agent contributed to its more superior performance. During a search, the Q-network facilitates the expansion and simulation steps of the MCTS process by making good estimates of the optimal move action to take, leading to a more efficient search.

8.4 Ablation Study

The performance of *Scene Mover* depends a lot on the expansion and simulation steps, which are facilitated by using a Q-network. We study the effectiveness of the designed Q-network realized by a LSTM module in our framework.

We compared 4 policies, namely, Random, Heuristics, CNN, and LSTM. We first tested each policy on completing the move task. For each policy, in each step, we randomly selected one action to perform (Random), applied heuristic rules to select an action (Heuristics), selected the action with the maximum Q-value estimated by CNN (CNN), or selected the action with the maximum Q-value estimated by LSTM (LSTM). If the move tasks could be completed well with one policy, applying such a policy would help to improve the performance in the expansion and simulation steps of the MCTS and hence improve the performance of the solution search.

The results are shown in Table 6. The CNN agent fails to finish any layout since it can only act according to the current state without the ability of making long-term decisions. The Random agent performs poorly like the CNN agent. The Heuristics agent and the LSTM

Table 6. Success rates of different agents. Rows 3-4 refer to the neural networks-based agents. Rows 5-8 refer to the MCTS-based agents.

Agent	5-obj.	9-obj.	13-obj.	17-obj.
Random	5%	0%	0%	0%
Heuristics	100%	90%	50%	15%
CNN	0%	0%	0%	0%
LSTM	100%	70%	35%	20%
MCTS + Random	100%	45%	15%	0%
MCTS + Heuristics	100%	100%	95%	70%
MCTS + CNN	30%	0%	25%	5%
MCTS + LSTM (*Scene Mover*)	100%	100%	100%	90%

agent achieve 100% success rate in the 5-obj. level. In the 9-obj. and 13-obj. levels, the Heuristics agent outpeforms the LSTM agent, but in the 17-obj. level, the LSTM agent achieves a better success rate. Due to the complexity of the layouts in the 17-obj. level, the agent has to make long-term decisions to complete the move task. The higher success rate of the LSTM agent could be explained by its better ability to make long-term decisions.

Additionally, we compared the performances of the MCTS framework embedded with each of the policies. Table 6 shows the results. MCTS + CNN performs worse than MCTS + Random where the actions in the expansion and simulation steps are randomly sampled. In the 5-obj. level, MCTS + CNN only achieves 30% success rate while MCTS + Random achieves 100%. The CNN network seems to provide priors even worse than random sampling as it fails to correlate the states between time steps. MCTS + LSTM (*Scene Mover*) outperforms all other agents especially in 13-obj. and 17-obj. levels. Note that although the LSTM agent performs worse than the Heuristics agent in policy comparison in the 13-obj. level, the *Scene Mover* agent achieves better success rate. It shows that the Q-network with a LSTM layer learns good priors in training.

9 EVALUATION ON REAL-WORLD LAYOUTS

We tested our approach on real-world layouts for solving scene arrangement problems. As humans may use prior knowledge of real-world layouts to facilitate move planning, we compare the performance of humans and *Scene Mover* on real-world layouts.

9.1 Real-World Layouts

We selected 20 indoor scenes from the ScanNet dataset [Dai et al. 2017] as the testing set, covering a variety of 3D reconstructed scenes with rich annotations. The number of objects in a scene was 11 on average, ranging from 6 to 19. The objects in the scenes are normal furniture with general shapes like rectangle, circle, triangle, "T", "L", etc.. Some novel shapes in the scenes are the combinations of the basic shapes. The shapes vary by scale and aspect ratio. Fig. 15(a) shows two example scenes. As the objects of the raw 3D scanned scenes were segmented and annotated, we obtained initial layouts using the segmentation and annotation as shown in Fig. 15(b). For each initial layout, we designed a target layout with the same objects as shown in Fig. 15(c).

To collect human manipulation data for analysis, we designed a mobile game akin to the Scene Mover game discussed in Sec. 8.2. The player needed to change an initial layout to its target layout via a series of manipulations. We designed a user-friendly interface to facilitate object manipulation so that the player could focus on the move task itself. The player could drag an object anywhere as long as the object does not hit any obstacle on its trajectory. The player could also rotate an object on the 2D plane. We also designed a snap mechanism to help the player: if an dragged object was close to its target position and if there was a collision-free path to the target position, the object would snap to its target position.

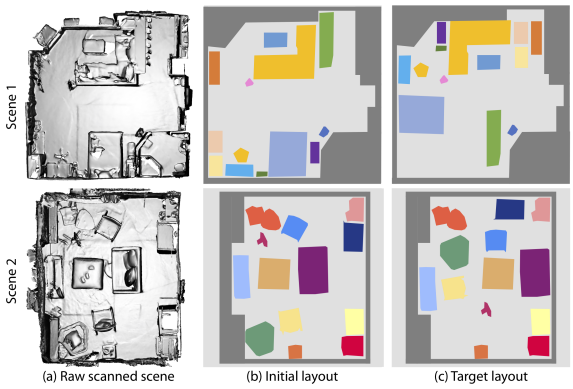


Fig. 15. Real-world layouts. Different objects are shown in different colors. The unmovable objects (e.g., walls) are shown in gray. The supplementary video contains the move action sequence generated for each layout.

At the beginning of the game, the player got familiar with the manipulation control and goal through a tutorial. Before each play, the 3D scene of the layout was shown to the user. We collected 389 playing records from 84 participants in total. Refer to our supplementary material for more details about the data.

9.2 Comparison

We compared the performances of *Scene Mover* and human participants using three metrics: success rate, average number of steps, and average travel length. Success is defined using the same metric as in Sec. 8.3, i.e., reaching the target layout within 100 move actions. As a player could drag and rotate objects separately, for a fair comparison on the number of steps, a continuous series of manipulations on the same object was counted as one step. For each scene, we calculated the median number of steps and the median travel length from the participants' results. The overall average number of steps was calculated by averaging the number of steps medians of the 20 scenes used. The average travel length was calculated similarly.

Table 7 shows the results. *Scene Mover* attained a success rate of 100% while the participants attained a success rate of 83.7%. *Scene Mover* used 13 steps on average to complete the moving, which is less than the participants' average steps of 15.7. *Scene Mover* tended to give better global solutions. *Scene Mover* used a much shorter average travel length of 201.4 while the participants used an average travel length of 681.5. The travel length results could be explained from two perspectives. First, using more steps generally resulted in a longer travel. Second, when the participants moved the objects in a scene with obstacles, they rarely used the shortest path, resulting in a longer travel. In the supplementary material, we show more details about the participants' failure cases.

The results show that our approach can generate move plans for real-world layouts and can work with general shapes. Though the

Table 7. Performances of human participants and *Scene Mover*.

	Success Rates	No. of Steps	Travel Length
Humans	83.7%	15.7	681.5
Scene Mover	100.0%	13.0	201.4

objects used in training were rectangular, they varied in length and width. The combination of rectangular objects constructed many different shapes in the hidden layers of the CNNs, which probably helped the CNNs generalize to irregular shapes.

Scene Mover may complement 3D reconstruction algorithms [Wang et al. 2019b], scene synthesis algorithms, and robotics techniques to realize move tasks in the real world. For example, one may reconstruct a real-world scene as an initial layout, followed by a scene synthesis algorithm to rearrange the furniture to create a target layout. *Scene Mover* can be applied to generate a move action sequence via which a robot moves all objects, step by step, to their target positions.

10 CONCLUSION

In this paper, we address a novel problem of automatic move planning for scene arrangement. We propose the *Scene Mover* agent, which uses a deep reinforcement learning-based MCTS approach to find a feasible move action sequence. By automatically generating a move plan, the *Scene Mover* agent could complement scene synthesis algorithms for realizing a synthesized layout design. Experiments on both synthetic and real-world layouts show that *Scene Mover* can generate move plans to tackle arrangement scenarios automatically. To discover the policies picked up by our approach, we summarize the moving strategies by analyzing the results of some cases in our experiments and providing an empirical explanation.

Limitations and Future Work. Fig. 16 shows an example that *Scene Mover* failed. In this case, only the objects in green and purple can be directly moved to their target position at the beginning. By taking those actions, *Scene Mover* gets good short-term rewards but also be trapped in a difficult situation, where the objects on the top (i.e., the objects in blue, orange, and yellow) are blocked. We show the intermediate layouts at the 90th iteration where *Scene Mover* falls into a local minimum. If the green object is not moved to make way for the orange object, the orange object can never reach its target position. Such a corner case shares similarity with constrained "puzzle" problems, which might be better solved if our network is trained with more "puzzle" scenarios. Some strategies might be helpful for jumping out of the local minima, e.g., introducing randomness at the expansion steps in growing the search tree. More sophisticated reinforcement learning scheme such as Advantage Actor-Critic (A2C) [Haarnoja et al. 2018; Mnih et al. 2016] could also help enhance the network's performance, exploring the state space more thoroughly to avoid getting trapped in poor local minima [Shen et al. 2018]. Our current framework only models situations where objects have no overlap in the vertical direction. In future work, we can modify the representation to store the object's height instead of occupancy in the grid of a channel. The overlap problem in the vertical direction can be solved by modifying the collision detection mechanism without changing other modules.

In our experiments, we trained *Scene Mover* to move a maximum of 25 objects, which is sufficient to illustrate our core idea. To train this model, the maximum number of objects in the training scenes

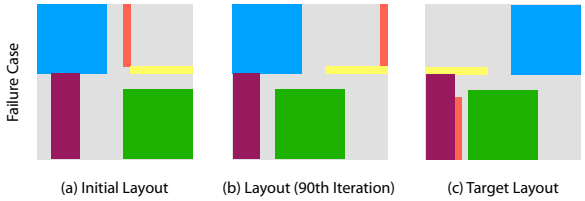


Fig. 16. A failed "puzzle" case of *Scene Mover*. When the object in green is moved to its target position at the beginning, the object in orange is blocked and many steps of action are needed to make way for it.

is 25. We also tried to train the model with fewer objects, i.e., at most 17, 13, and 9 objects in the scene. In this experiment, *Scene Mover* shows limited generalizability in terms of object count. We observed an obvious performance drop on the model trained with at most 9 objects while the models trained with 17 and 13 objects have marginal performance drop. It means that the model fails to capture the essential factor of complex layouts when it is trained in relatively simple scenes. If needed, *Scene Mover* could be trained to handle more objects by increasing the number of outputs of its network. In future work, It would be interesting to investigate *Scene Mover*'s performance in handling large-scale scenarios with many objects (e.g., hundreds of objects) and obstacles to shed light upon applications such as warehouse automation, urban design, and urban planning. More efforts can be given to the interpretability of the strategy that the network learned.

In this work we focus on generating a feasible move plan for transforming an initial layout into a target layout. Note that such feasible move plans are generally not unique and *Scene Mover* just finds one solution upon each run. In future work it would bring practical benefits to regularize and qualify the generated move plans further. For instance, additional constraints such as average load and number of turns could be incorporated into the MCTS framework in a similar fashion as the travel length constraint (Sec. 7.5).

We devise a move planning approach to automate scene rearrangement. Future works may integrate the high-level move planning approach with object-level, human-centric action space considerations [Ma et al. 2016; Savva et al. 2016] for executing the plan.

It would also be interesting to devise deep reinforcement learning-based approaches to tackle other graphics and design problems. For example, such approaches could be applied for assembly-based modeling: given an initial 3D object and a target 3D object comprising the same set of components (e.g., Lego bricks), one may train a reinforcement learning agent to generate a disassembly and assembly plan to transform the initial object into the target object. Such an approach may bring new insights for re-configurable design. Another avenue for future extension is to incorporate *Scene Mover* into robots for physically realizing procedurally-generated designs.

ACKNOWLEGEMENT

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61972038. Thanks for the great support from Sifan Hou and Zan Wang in developing the *Scene Mover* Micro-App. Thanks for the help from Hongfei Yu on the demonstrations.

REFERENCES

Douglas Aberdeen, Sylvie Thiébaux, and Lin Zhang. 2004. Decision-theoretic military operations planning. In *International Conf. on Automated Planning and Scheduling*.

Christos Alexopoulos and Paul M Griffin. 1992. Path planning for a mobile robot. *IEEE Trans. on systems, man, and cybernetics* 22, 2 (1992), 318–322.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *NIPS*.

Thomas Anthony, Zheng Tian, and David Barber. 2017. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2-3 (2002), 235–256.

Andrew Best, Sahil Narang, Daniel Barber, and Dinesh Manocha. 2017. Autonovi: Autonomous vehicle planning with dynamic maneuvers and traffic constraints. In *International Conference on Intelligent Robots and Systems*. IEEE, 2629–2636.

Sumana Biswas, Sreenatha G Anavatti, and Matthew A Garratt. 2017. Obstacle avoidance for multi-agent path planning based on vectorized particle swarm optimization. In *Intelligent and Evolutionary Systems*. Springer, 61–74.

Rodney A Brooks and Tomas Lozano-Perez. 1985. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Trans. on Systems, Man, and Cybernetics* 2 (1985), 224–233.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. 2002. Deep blue. *Artificial Intelligence* 134, 1-2 (2002), 57–83.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-Carlo Tree Search: A New Framework for Game AI.. In *AIIDE*.

Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. 2017. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In *IEEE CVPR*.

Matthew Fisher, Manolis Savva, Yangyan Li, Pat Hanrahan, and Matthias Nießner. 2015. Activity-centric Scene Synthesis for Functional 3D Scene Modeling. *ACM Trans. on Graphics* 34, 6 (2015).

Keith Frankish and William M Ramsey. 2014. *The Cambridge handbook of artificial intelligence.* Cambridge University Press.

Qiang Fu, Xiaowu Chen, Xiaotian Wang, Sijia Wen, Bin Zhou, and Hongbo Fu. 2017. Adaptive synthesis of indoor scenes via activity-associated object relation graphs. *Acm Trans. on Graphics* 36, 6 (2017), 201.

Santiago Garrido, Luis Moreno, and Pedro U Lima. 2011. Robot formation motion planning using fast marching. *Robotics and Autonomous Systems* 59, 9 (2011).

Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. 2015. Path planning and trajectory planning algorithms: A general overview. In *Motion and Operation Planning of Robotic Systems*. Springer, 3–27.

Russell Gayle, Paul Segars, Ming C. Lin, and Dinesh Manocha. 2005. Path Planning for Deformable Robots in Complex Environments. In *Robotics: Science and Systems*.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290* (2018).

Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

Matthew Hausknecht and Peter Stone. 2015. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527* 7, 1 (2015).

Joshua A Haustein, Isac Arnekvist, Johannes Stork, Kaiyu Hang, and Danica Kragic. 2019. Learning Manipulation States and Actions for Efficient Non-prehensile Rearrangement Planning. *arXiv preprint arXiv:1901.03557* (2019).

Joshua A Haustein, Jennifer King, Siddhartha S Srinivasa, and Tamim Asfour. 2015. Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states. In *IEEE ICRA*.

Gene Eu Jan, Chi-Chia Sun, Wei Chun Tsai, and Ting-Hsiang Lin. 2014. An $O(nlogn)$ Shortest Path Algorithm Based on Delaunay Triangulation. *IEEE/ASME Trans. On Mechatronics* 19, 2 (2014), 660–666.

Jennifer E King, Marco Cognetti, and Siddhartha S Srinivasa. 2016. Rearrangement planning using object-centric and robot-centric action spaces. In *IEEE ICRA*.

Jennifer E King, Vinitha Ranganeni, and Siddhartha S Srinivasa. 2017. Unobservable monte carlo planning for nonprehensile rearrangement tasks. In *IEEE ICRA*.

Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32, 11 (2013), 1238–1274.

Michael C Koval, Jennifer E King, Nancy S Pollard, and Siddhartha S Srinivasa. 2015. Robust trajectory selection for rearrangement planning as a multi-armed bandit problem. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Yann Labbé, Sergey Zagoruyko, Igor Kalevatykh, Ivan Laptev, Justin Carpentier, Mathieu Aubry, and Josef Sivic. 2020. Monte-Carlo Tree Search for Efficient Visually Guided Rearrangement Planning. *IEEE Robotics and Automation Letters* (2020).

Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. 2018. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research* 37, 4-5 (2018).

Manyi Li, Akshay Gadi Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, Daniel Cohen Or, and Hao Zhang. 2019. GRAINS: Generative Recursive Autoencoders for INdoor Scenes. *ACM Trans. on Graphics* 38 (2019).

Rui Ma, Honghua Li, Changqing Zou, Zicheng Liao, Xin Tong, and Hao Zhang. 2016. Action-driven 3D indoor scene evolution. *ACM Trans. Graph.* 35, 6 (2016), 173–1.

Paul Merrell, Eric Schkufza, and Vladlen Koltun. 2010. Computer-generated residential building layouts. In *ACM Trans. on Graphics*, Vol. 29. ACM, 181.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *ICML*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

Brendan O'Donoghue, Rémi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. 2016. PGQ: Combing Policy Gradient and Q. *arXiv preprint arXiv:1611.01626* (2016).

Chi-Han Peng, Yong-Liang Yang, and Peter Wonka. 2014. Computing layouts with deformable templates. *ACM Trans. on Graphics* 33, 4 (2014), 99.

Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song Chun Zhu. 2018. Human-centric Indoor Scene Synthesis Using Stochastic Grammar. In *IEEE CVPR*.

Daniel Ritchie, Kai Wang, and Yu-an Lin. 2019. Fast and Flexible Indoor Scene Synthesis via Deep Convolutional Generative Models. In *IEEE CVPR*.

Manolis Savva, Angel X Chang, Pat Hanrahan, Matthew Fisher, and Matthias Nießner. 2016. PiGraphs: learning interaction snapshots from observations. *ACM Trans. on Graphics* 35, 4 (2016), 139.

Yelong Shen, Jianshu Chen, Posen Huang, Yuqing Guo, and Jianfeng Gao. 2018. ReinforceWalk: Learning to Walk in Graph with Monte Carlo Tree Search. *arXiv: Artificial Intelligence* (2018).

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354.

Changkyu Song and Abdeslam Boularias. 2019. Object Rearrangement with Nested Nonprehensile Manipulation Actions. *arXiv preprint arXiv:1905.07505* (2019).

Haoran Song, Joshua A Haustein, Weihao Yuan, Kaiyu Hang, Michael Yu Wang, Danica Kragic, and Johannes A Stork. 2019. Multi-Object Rearrangement with Monte Carlo Tree Search: A Case Study on Planar Nonprehensile Sorting. *arXiv preprint arXiv:1912.07024* (2019).

Avneesh Sud, Erik Andersen, Sean Curtis, Ming C Lin, and Dinesh Manocha. 2008. Real-time path planning in dynamic virtual environments using multiagent navigation graphs. *IEEE Trans. on Visualization and Computer Graphics* 14, 3 (2008), 526–538.

Richard S Sutton and Andrew G Barto. 2011. Reinforcement learning: An introduction. (2011).

Jur van den Berg, Jack Snoeyink, Ming Lin, and Dinesh Manocha. 2009. Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In *Robotics: Science and Systems*.

Hanqing Wang, Wenguan Wang, Tianmin Shu, Wei Liang, and Jianbing Shen. 2020. Active Visual Information Gathering for Vision-Language Navigation. In *ECCV*.

Hanqing Wang, Jiaolong Yang, Wei Liang, and Xin Tong. 2019b. Deep Single-View 3D Object Reconstruction with Visual Hull Embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Kai Wang, Yu-an Lin, Ben Weissmann, Manolis Savva, Angel X. Chang, and Daniel Ritchie. 2019a. PlanIT: Planning and Instantiating Indoor Scenes with Relation Graph and Spatial Prior Networks. *ACM Trans. on Graphics* 38, 4 (2019).

Kai Wang, Manolis Savva, Angel X Chang, and Daniel Ritchie. 2018. Deep convolutional priors for indoor scene synthesis. *ACM Trans. on Graphics* 37, 4 (2018), 70.

Marco Wiering and Martijn Van Otterlo. 2012. Reinforcement learning. *Adaptation, Learning, and Optimization* 12 (2012), 51.

Wenming Wu, Lubin Fan, Ligang Liu, and Peter Wonka. 2018. MIQP-based layout design for building interiors. In *Computer Graphics Forum*, Vol. 37. 511–521.

Yi-Ting Yeh, Lingfeng Yang, Matthew Watson, Noah D Goodman, and Pat Hanrahan. 2012. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Trans. on Graphics* 31, 4 (2012), 56.

Lap-Fai Yu, Sai Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F. Chan, and Stanley Osher. 2011. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph.* 30, 4 (2011), 86.

Weihao Yuan, Kaiyu Hang, Danica Kragic, Michael Y Wang, and Johannes A Stork. 2019. End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer. *Robotics and Autonomous Systems* 119 (2019).

Weihao Yuan, Johannes A Stork, Danica Kragic, Michael Y Wang, and Kaiyu Hang. 2018. Rearrangement with nonprehensile manipulation using deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation*.