Haisor: Human-aware Indoor Scene Optimization via Deep Reinforcement Learning

JIA-MU SUN, Institute of Computing Technology, CAS and University of Chinese Academy of Sciences, China
JIE YANG, Institute of Computing Technology, CAS, China
KAICHUN MO, Stanford University and NVIDIA Research, United States
YU-KUN LAI, Cardiff University, United Kingdom
LEONIDAS GUIBAS, Stanford University, United States
LIN GAO, Institute of Computing Technology, CAS and University of Chinese Academy of Sciences, China

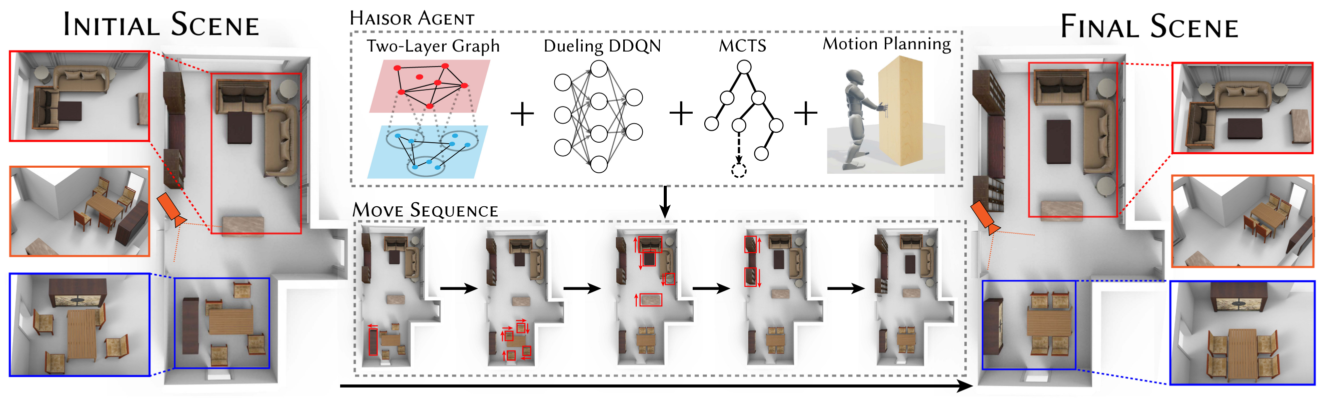


Fig. 1. Given an initial scene, our Haisor agent takes advantage of the scene graph representation, Dueling Double DQN, Monte Carlo Tree Search, and Motion Planning techniques and outputs a sequence of actions. Following these, the scene is optimized towards the goal of high rationality and human affordance. For rationality, we expect that the overall layout of the furniture is realistic, and no collisions between furniture exist. For human affordance, we expect the free space for human activity to be as large as possible, and people can manipulate movable parts of furniture without being blocked.

3D scene synthesis facilitates and benefits many real-world applications. Most scene generators focus on making indoor scenes plausible via learning from training data and leveraging extra constraints such as adjacency and symmetry. Although the generated 3D scenes are mostly plausible with visually realistic layouts, they can be functionally unsuitable for human users to navigate and interact with furniture. Our key observation is that human activity plays a critical role and sufficient free space is essential for human-scene interactions. This is exactly where many existing synthesized scenes fail—the seemingly correct layouts are often not fit for living. To tackle this, we present a human-aware optimization framework Haisor for 3D indoor scene arrangement via reinforcement learning, which aims to find an action sequence to optimize the indoor scene layout automatically. Based on the hierarchical scene graph representation, an optimal action sequence is predicted and performed via Deep Q-Learning with Monte Carlo Tree Search (MCTS), where MCTS is our key feature to search for the optimal solution in long-term sequences and large action space. Multiple human-aware rewards are designed as our core criteria of human-scene interaction, aiming to identify the next smart action by leveraging powerful reinforcement learning. Our framework is optimized end-to-end by giving the indoor scenes with part-level furniture layout including part mobility information. Furthermore, our methodology is extensible and allows utilizing different reward designs to achieve personalized indoor scene synthesis. Extensive experiments demonstrate that our approach optimizes the layout of 3D indoor scenes in a human-aware manner, which is more realistic and plausible than original state-of-the-art generator results, and our approach produces superior smart actions, outperforming alternative baselines.

CCS Concepts: • **Computing methodologies** → **Computer graphics**; **Scene understanding**; **Hierarchical representations**; *Shape modeling;*

Additional Key Words and Phrases: Scene optimization, scene synthesis, human aware, reinforcement learning, Monte Carlo search, robot simulation, imitation learning

Jia-Mu Sun and Lin Gao are also with University of Chinese Academy of Sciences.
Kaichun Mo is also with NVIDIA Research.
This work was supported by National Natural Science Foundation of China (No. 62322210), Beijing Municipal Natural Science Foundation for Distinguished Young Scholars (No. JQ21013) and Beijing Municipal Science and Technology Commission (No. Z231100005923031). Kaichun Mo and Leonidas J. Guibas were supported by the ARL grant W911NF-21-2-0104, a Vannevar Bush Faculty Fellowship, and a gift from the Adobe Corporation.
Authors' addresses: J.-M. Sun, J. Yang and L. Gao (Corresponding author), Beijing Key Laboratory of Mobile Computing and Pervasive Device, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190, China; e-mails: {sunjiamu21s, yangjie01, gaolin}@ict.ac.cn; K. Mo and L. Guibas, Computer Science Department, Stanford University, 450 Serra Mall, Stanford, CA 94305, United States; e-mails: {kaichun, guibas}@cs.stanford.edu; Y.-K. Lai, School of Computer Science and Informatics, Cardiff University, Cardiff, Wales, CF10 3AT, United Kingdom; e-mail: LaiY4@cardiff.ac.uk.

0730-0301/2024/01-ART15
https://doi.org/10.1145/3632947

ACM Reference format:
Jia-Mu Sun, Jie Yang, Kaichun Mo, Yu-Kun Lai, Leonidas Guibas, and Lin Gao. 2024. Haisor: Human-aware Indoor Scene Optimization via Deep Reinforcement Learning. *ACM Trans. Graph.* 43, 2, Article 15 (January 2024), 17 pages.
https://doi.org/10.1145/3632947

1 INTRODUCTION

In the past few years, plenty of research has emerged to demonstrate the power of automated modeling of indoor scenes. However, the automatic generation of realistic and plausible indoor scenes has not met the needs of creators. Deep generative models have received much attention in vision and graphics due to the growing popularity of applications such as VR designs, robotics, and simulation, where, traditionally, content creation involves cumbersome and tedious manual modeling pipelines. However, the generated 3D models [Paschalidou et al. 2021; Ritchie et al. 2019; Wang et al. 2019, 2018] tend to focus on visual similarity to training examples, ignoring the functionality and human-aware interaction [Blinn et al. 2021]. This significantly limits the realism and practicality of the generated models, which is crucial for 3D indoor scene synthesis.

Existing indoor scene synthesis work commonly focuses on layout generation/optimization, aiming to generate plausible layout designs. They tend to adjust the furniture locations by learning the layout patterns existing in the dataset. However, a good indoor scene layout should allow furniture to be easily interacted with, without collision and with human comfort, e.g., the door/drawer of a cabinet should be easily openable. A reasonable layout of furniture facilitates easier interaction and free movement of users. To assist in the creation of such indoor 3D scenes, furniture layouts should be designed in a way that pays attention to how people interact with the scene and access the functionality of the furniture.

Generative models of 3D indoor scene layouts have made great success in automatic creation. Such models can produce geometry-aware and visually plausible 3D indoor scenes learned from existing data, e.g., GRAINS [Li et al. 2019], ATISS [Paschalidou et al. 2021], SceneFormer [Wang et al. 2021b], Deep Priors [Wang et al. 2018], PlanIT [Wang et al. 2019], and so on. Learning-based methods fail to be functionally useful or take human interaction into consideration, since the furniture is retrieved from the database as a whole according to the bounding box, ignoring part mobility. Yu et al. [2011] first introduce automatic synthesis of furniture layout by considering in a comprehensive manner human factors via simulated annealing, but it only focuses on the accessibility of furniture in a supervised manner, instead of the interaction between humans and the functionality of furniture. Such optimization is a non-trivial task.If we regard this task as a step-by-step adjustment process, then it will be a long-term trial-and-error decision-making process and must consider the variety of relationships in each step, e.g., furniture-furniture, human-furniture. As an example, for the accessibility and functionality (openable) of a cabinet door, the front of the cabinet must be emptied for making the door openable, and the positions of the cabinet and the objects surrounding it also need to be optimized so people can access it comfortably, without sacrificing the realism and rationality of indoor scenes. The difficulty of making decision sequences arises from complex and diverse scene layouts, movable parts of furniture, and accessibility to human habitats. It leads to a large search space that it is very hard to find optimal moving sequences. Naturally, a heuristic method can be designed to achieve the target under some defined rules. However, A finite set of rules cannot cover all situations, especially in a multi-modal complex case such as indoor scenes.

To tackle the problem of scene layout optimization, our goal is to optimize the furniture arrangements for indoor scenes automatically, which accommodates human interaction and habitats. Hence, we introduce a novel human-aware optimization framework, Haisor, for 3D indoor scene arrangement via reinforcement learning, which seeks to find a sequence of actions to satisfy our criteria automatically. Figure 1 shows the overview of all the components of HAISOR. To address the complexity of indoor scenes, various optimization criteria, and potential long action sequences, our method is based on a trained Deep Q-Network along with **Monte Carlo Tree Search (MCTS)** [Chaslot et al. 2008]. The scenes are represented by a two-layer graph, which decomposes the indoor scene furniture with part mobility, including the relationships between nodes at the same level. The learned Q-Network is capable of predicting the desired smart action that accommodates the accessibility and interaction with humans. Namely, it can decide an action that earns the largest reward from the simulation environment. The human-aware reward is our core criterion for human-scene interaction during reinforcement learning. It encourages (1) free manipulation of movable parts of furniture and (2) enough space for a human to walk around. For efficient training and fast convergence, we adopt imitation learning method [Shalev-Shwartz and Ben-David 2014], using a simple heuristics agent to guide the initial learning process, and MCTS further boosts the action search for the best solution in long-term sequence prediction. Leveraging the power of reinforcement learning, our approach learns how to optimize the arrangement of scene layout in a human-aware and end-to-end fashion by giving the 3D indoor scene with part-mobility-level furniture.

Furthermore, our method can be extended by designing different actions and rewards, facilitating the creation of 3D scenes and other applications such as personalized scene customization. In addition, our method can inspire other sequential decision-making problems such as functional furniture design [Blinn et al. 2021].

In summary, our major contributions include:

— A novel optimization procedure (Haisor) via reinforcement learning for scene layout arrangement that focuses on the optimal action sequence search automatically for realism and accessibility. Additionally, the procedure can be easily extended to enable more optimization goals.
— A learnable policy network that embeds human-aware reward via Graph Convolutional Network achieves smart action prediction that accommodates the accessibility and interaction with humans, based on the scene graph representation.
— Extensive experiments and validations show that our approach outperforms alternative baselines, capable of producing realistic and plausible results via learned smart actions in a human-aware manner.

2 RELATED WORK

This research work is primarily related to *Human-aware* 3D indoor scene synthesis/optimization, which considers both furniture layout and relations between objects. It is also related to deep generative models on 3D indoor scenes that learn the distributions for novel synthesis. This section briefly reviews the recent advances in 3D indoor scene synthesis, reinforcement learning on sequential decision problems, and rearrangement planning.

2.1 3D Indoor Scene Synthesis

3D indoor scene synthesis has been investigated thoroughly, including floorplan generation [Liu et al. 2021b; Ritchie et al. 2019; Wang et al. 2019, 2018] and furniture layout arrangement [Hu et al. 2020; Nauata et al. 2020, 2021]. For comprehensive discussions on scene synthesis, please refer to survey papers Pintore et al. [2020] and Zhang et al. [2019].

Understanding the furniture arrangements such as their relations is a central topic in scene synthesis. Existing research has made considerable effort to explore approaches that can generate geometrically realistic and reasonable indoor scenes by predicting furniture placement from a semantic perspective and retrieving existing furniture for producing realistic scenes. Various solutions have been proposed by considering different settings, such as example-driven methods [Fisher et al. 2012], language-driven synthesis [Ma et al. 2018], learning-based methods [Kermani et al. 2016; Merrell et al. 2011; Zhang et al. 2022], and optimization-based methods [Henderson and Ferrari 2017; Xu et al. 2015; Yeh et al. 2012]. Recently, deep neural networks have been successfully leveraged to learn the potential distribution from large amounts of data. These works [Ritchie et al. 2019; Wang et al. 2019, 2018] introduce image-based deep generative models to model the placements and relations with the graph of furniture from an enormous amount of top-down view images of scenes. GRAINS [Li et al. 2019] is a pioneering work for scene structure modeling, leveraging hierarchical graphs to represent scenes and producing realistic furniture layouts within a square domain with border walls by **Recursive Neural Networks (RvNNs)**. Moreover, Zhang et al. [2020] presents a hybrid representation to train a scene layout generator through a combination of 2D images and 3D object placements. SceneFormer [Wang et al. 2021b] and ATISS [Paschalidou et al. 2021] predict furniture placement sequentially based on the transformer [Vaswani et al. 2017], which achieve **state-of-the-art (SoTA)** 3D scene conditional generation. Recently, LayoutEnhancer [Leimer et al. 2022] aims to generate the enhanced scene layout by combining both expert knowledge and a distribution learned from data. Besides that, SceneGraphNet [Zhou et al. 2019] models the short/long-range relations between objects based on a novel neural message-passing network, which can perform object suggestions considering the surrounding objects in a scene.

In contrast to the above scene synthesis frameworks that only consider the realism of synthesized scenes in geometry and layouts, researchers have been exploring other factors of human interaction to ensure the functional plausibility and accessibility of 3D synthesized scenes, e.g., accessibility of objects [Yu et al. 2011], human trajectory [Qi et al. 2018], and human activity [Fu et al. 2017; Liu et al. 2021a; Ma et al. 2016]. Yu et al. [2011] only regard objects as a whole to model the accessibility. Qi et al. [2018] optimize the scene layout according to the learned affordance maps with a stochastic grammar model, without considering the accessibility of furniture (especially for part mobility). Similarly, Ye et al. [2022] synthesize the indoor scene by considering the human motion. Fu et al. [2017] and Ma et al. [2016] optimize the object placement in a scene with pre-defined actions or activity-associated object relation graphs.

Instead, Haisor allows optimizing the furniture arrangements to achieve realism automatically and further enables personalizing 3D scene layout, which accommodates human manipulation with objects and their moving parts to ensure functionality and human affordance for comfortable accessibility.

2.2 Rearrangement Planning

Rearrangement Planning has been investigated in the robotic field. There is a trend to study how to rearrange objects for the goal configuration using a robot. It is NP-hard [Wilfong 1991] and an interesting challenge area for robotics research. Yuan et al. [2019, 2018] learn a non-prehensile plan for robots' arms to place the objects in the right location under some criteria. For multi-object rearrangement planning, a substantial body of works [Haustein et al. 2015; King et al. 2016, 2017; Koval et al. 2015; Song et al. 2020] has explored this problem in two stages (local path planning and global strategy searching). Bai et al. [2021] proposed a hierarchical policy to solve the non-prehensile multi-object rearrangement with deep reinforcement learning and MCTS. For complex and realistic indoor scenes, SceneMover [Wang et al. 2020] and PEARL [Wang et al. 2021a] solve the scene rearrangement planning problem via selecting pre-defined paths for movement and exploring non-prehensile action space (grasp & pick), respectively. The non-prehensile actions make the solution more tractable.

Prior works in robotics perform algorithms on a robot, which only use some simplified proxies to represent the scene, and it is difficult to describe the full complexity of the scene, e.g., detailed geometry, part mobility, and orientation. Although SceneMover and PEARL conduct the rearrangement planning in realistic scenes, the target is known or learned from the data distribution. In contrast, our method Haisor aims to optimize the furniture placement in realistic scenes with complex part geometry and mobility and further integrates the human-aware factors (affordance and manipulation with objects) for 3D scene layout customization.

2.3 Deep Reinforcement Learning

The goal of **reinforcement learning (RL)** [Sutton and Barto 2018] is to help the agent learn good policies for sequential decision problems under dynamic environments by optimizing a cumulative future reward signal in an unsupervised manner. It seeks the gradient directions from countless attempts/trials through facing different incentives from the environment, which is a process of accumulating experience. When the environment is known, it is named model-based RL, which can be solved using dynamic programming. However, most of the real-world problems are environment-unknown, namely, model-free RL, which can be divided into value-based RL and policy-based RL. Policy-based methods are usually combined with a policy gradient algorithm [Achiam et al. 2017; Agarwal et al. 2020; Chen et al. 2022; Vuong et al. 2019]. For value-based methods, Q-Learning is one of the most popular reinforcement learning algorithms. However, it has been observed that it occasionally learns unreasonably high action values, since it includes a maximization process over the estimated action values, which may bias towards overestimation.

With the success of **deep neural networks (DNN)** in the field of 2D image processing [Krizhevsky et al. 2012], reinforcement learning has seen considerable efforts that use DNNs to fit policy networks, namely, deep reinforcement learning, including DQN [Mnih et al. 2013], Ddouble-DQN [van Hasselt et al. 2016] and Dueling-DQN [Wang et al. 2016]. The deep reinforcement learning algorithms have been investigated and demonstrated their superior performance across many challenging tasks. For example, AlphaZero [Silver et al. 2017] in the games of chess and Go without human knowledge, OpenAI Five [Berner et al. 2019] in the MOBA games for multi-agents cooperation, and learning to self-navigate in complex environments [Mirowski et al. 2017]. Different from the above agents, our agent is performed in the real-world scene layout reconfiguration task. The action space and rewards are designed to optimize the scene layout in a *human-aware* fashion. Researchers have been exploring numerous approaches to make learning more efficient [Andrychowicz et al. 2016], stable [Mnih et al. 2015], and benefit from experience replay [O'Donoghue et al. 2017]. Levine et al. [2018] learn a policy to control the robots for grasping from the raw inputs of a camera. Scene Mover [Wang et al. 2020] is the most relevant work to ours, which designs a novel algorithm for scene arrangement by automatically generating a move plan. Instead, ours aims to reconfigure the furniture arrangements to realism automatically and further implicitly personalize the 3D scene layout via different reward designs.

3 OVERVIEW

Haisor seeks to find a sequence of actions that can reorganize the original scene to another one that satisfies a set of criteria. To tackle this problem, we use Deep Q-Learning with **Monte Carlo Tree Search (MCTS)** on a two-layer scene graph representation. The learned Q-Network is capable of predicting the action that earns the most rewards for the current state of the scene. Due to the highly freed action space and diverse furniture arrangement, the action sequence that leads to the optimal solution may be too long for simple searching algorithms, Hence, we proposed to utilize MCTS to boost our converge process in the long term, which plays a critical role in our framework. The rewards of the simulation environment are designed according to the criteria of human-scene interaction, which encourages the optimized furniture layout to accommodate the manipulation of object functionality and accessibility of human affordance.

3D Scene Representation. 3D indoor scenes can be decomposed into room structures (e.g., walls, doors, and windows) and arrangement of furniture, and the furniture can be further disassembled into local parts. In every step of optimization, the whole furniture is moved, but the human-scene interaction is at the part level. For example, when a person opens the door, he or she is actually grasping the handle. Based on the observation above, we construct two graphs (V, E) and (V_p, E_p) to represent the scene. The nodes V of a scene graph are furniture objects, and the nodes V_p are parts of furniture objects. Because an object consists of multiple parts, every node in V_p has a parent node in V. The edges of the graph are designed to hold information about the proximity of the nodes. We draw an edge between two nodes of the graph only if they are sufficiently close in space. The scene graph representation is discussed in Section 4.

Human-scene Interaction. Our method focuses on generating a human-friendly indoor 3D scene, taking into account human-scene interaction. We mainly consider two types of interactions:

(a) Humans can freely manipulate movable parts of the furniture in the scene.
(b) The space for humans to walk around in the scene needs to be as large as possible.

In the rest of the article, we combine these two criteria and call them **Human Affordance**. For the first type of interaction, we use a humanoid robot to imitate a human, and a motion planning module is invoked to get the possible movement for the robot to manipulate the movable parts. If no such movement exists, then we conclude that the current layout of the scene is not "comfortable." For the second type of interaction, we consider all the possible positions that a person can stand in the room. The goal is to maximize the number of these positions. We will discuss this in Section 5.

Deep Q-Network. The task of the Q-Network is to predict the Q value $Q(s, a)$ in the Bellman equation for an arbitrary scene state s and action a. In the MCTS [Chaslot et al. 2008] search process, the predicted Q values are used in the selection, expansion, and simulation steps. An accurate Q value can accelerate the search speed.

The backbone of our Q-Network is a **Graph Convolution Network (GCN)** [Defferrard et al. 2016; Kipf and Welling 2017]. We perform graph convolution twice: first is on the part-level scene graph (V_p, E_p), then the features of the part-level nodes are aggregated and fed into the convolution layer of object-level scene graph (V, E). For the output layer of the network, we adopt the dueling DQN [Wang et al. 2016] structure. The detailed network composition is described in Section 6.1. To efficiently train this network, multiple training methods such as prioritized replay buffer and imitation learning are used. These training methods are explained in Section 6.2.

Monte Carlo Tree Search. The policy provided by the learned Q-network may not be accurate all the time. MCTS is an effective method for combining the guidance of a Q-Network and a search algorithm. MCTS constructs a tree that stores the states of the scene and their Q-values. During the search process, the tree is expanded iteratively. Among all candidate nodes, the one that has the highest estimated Q-value is selected and expanded. From the state of the expanded node, a simulation is performed to provide a "real" human affordance reward. This acquired reward is then used to update all the ancestor nodes of the expanded node. Because a simulation is performed in each round, the updated Q-value is more accurate than the one predicted by the Q-network. However, with the guidance of the Q-network, the search converges more quickly. The search algorithm is described in Section 6.3.

4 SCENE REPRESENTATION

We construct two undirected scene graphs (V, E) and (V_p, E_p) to represent the scene. Below, we discuss the construction of our scene graph representation.

4.1 Scene Graph Composition

Bounding Box vs. Detailed Geometry. In real-world 3D indoor scenes, objects are associated with their detailed geometry. However, the existing scene synthesis methods such as GRAINS [Li et al. 2019] or Deep Priors [Wang et al. 2018] only consider the layout of the scene, using a bounding box to represent the whole furniture. In our setting, object parts are important, because (1) actual human-furniture interaction happens on object parts, and (2) some object placement patterns, e.g., chairs inserted under the table, cannot be modeled using object bounding boxes. Considering the fact that adding detailed geometry of object parts into our framework is too heavy, we choose to use a bounding box to represent each object part. This "hybrid" method is lightweight, and it preserves the object parts' structure, enabling the two features described above.

Nodes. In our scene graph, every part-level node is associated with an 11-dimensional data of the oriented bounding box of the corresponding part or object, denoted by $[\mathbf{p}, \mathbf{e}, \mathbf{o}, \mathbf{f}]$. $\mathbf{p}$ and $\mathbf{e}$ are 3D vectors storing the position and the extent of the bounding box. $\mathbf{o}$ is a 4D quaternion storing the orientation of the bounding box. $\mathbf{f}$ is a binary flag, indicating whether the part is movable. In object-level scene graph (V, E), the node data is represented as $[\mathbf{p}, \mathbf{e}, \mathbf{o}, \mathbf{l}]$. $\mathbf{l}$ is a one-hot vector encoding the category of the object, and its length $\mathbf{C}$ equals the number of object categories. Thus, the node data of scene graph is a $|V| \times 11$ matrix for part-level graph and a $|V| \times (10 + \mathbf{C})$ matrix for object-level graph.

Edges. The edges of the graph are designed to hold information of the proximity of nodes. We connect an edge between nodes v_1 and v_2 if the following condition is satisfied:

$$\|\mathbf{p}_1 - \mathbf{p}_2\|_2 \leq \frac{(\|\mathbf{e}_1\|_2 + \|\mathbf{e}_2\|_2)}{2}, \tag{1}$$

where the $\|\cdot\|_2$ means the ℓ_2 norm. Note that in the part-level graph, an edge between a pair of nodes is connected only if the parents of the two nodes are connected by an edge in the object-level graph. In our scene graph, every edge (v_1, v_2) is associated with a 3D vector $\mathbf{p}_1 - \mathbf{p}_2$. Thus, the edge data of a scene graph is an $|E| \times 3$ matrix.

2-layer Scene Graphs. We need to consider both the relationships between object parts and the relationships between objects. As a solution, the scene graph is extended to a 2-layer one. Two graphs (V, E) and (V_p, E_p) are constructed individually on the object level and part level and associated with the ownership of parts by objects. The ownership is defined as an integer a_i:

$$V_{a_i} \text{ owns } V_{p,i}$$
$$i = 1, 2, \ldots, |V_p|, a_i \in \{1, 2, \ldots, |V|\}. \tag{2}$$

4.2 Movable Parts

Movable parts are essential elements when considering human-scene interaction, while it is more complicated to model them. The two most encountered types (Figure 2) of movable parts are considered in this approach:

(1) Hinges. This type of object can rotate around an axis. The direction of the axis is denoted by unit vector $\mathbf{r}_d$, and the origin of the axis is a position $\mathbf{r}_p$. The rotation angle has an upper limit r_u and a lower limit r_b.
(2) Sliders. This type of object can move along a direction denoted by unit vector $\mathbf{s}_d$. The distance of movement has an upper limit s_u and a lower limit s_b.

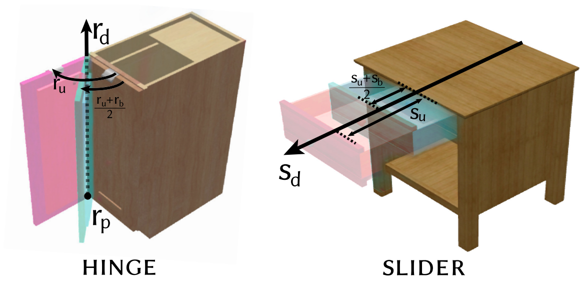


Fig. 2. Types of movable parts. We define two types of movable parts: Hinge and slider. The movable hinge parts are defined using four parameters r_p, r_d, r_u, r_b, and the movable slider parts are defined using three parameters r_d, r_u, r_b. These parameters are combined with bounding box parameters to form the data of the part-level scene graph.

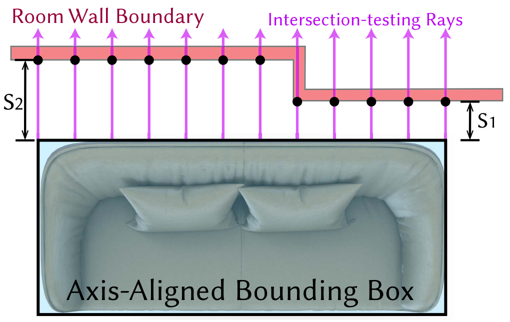


Fig. 3. Wall-object Distance Extraction. To calculate the distance between an object and the wall of the room, we adopt the ray-casting method. The calculation is based on the axis-aligned bounding box (AABB) of the object. We sample 10 points equidistantly on each face of the AABB as the origin of the ray, and the direction of the ray is perpendicular to the face and pointing at the outside of the AABB. We take the minimum of the ray intersection length as the calculated distance. In the figure, two intersection distance values s_1 and s_2 are returned by the ray intersection process, and s_1 is the true value of wall-object distance.

Directly feeding the movement data described above into the network is an option, but it is hard for the network to learn the relationship between the data and the actual movement. Instead, we decide to sample three states of the movable parts: the lower limit of movement, the upper limit of movement, and the middle of them. These three bounding boxes are added to the part-level scene graph and treated like other parts. Note that for most movable parts in our data, the lower and upper limit states are either the original state itself, or symmetric to the original state. So, we do not need to include the original bounding boxes.

4.3 Walls

Walls act as "hard constraints" in our decision sequence. Any part of the furniture must not collide with the walls. Since we do not

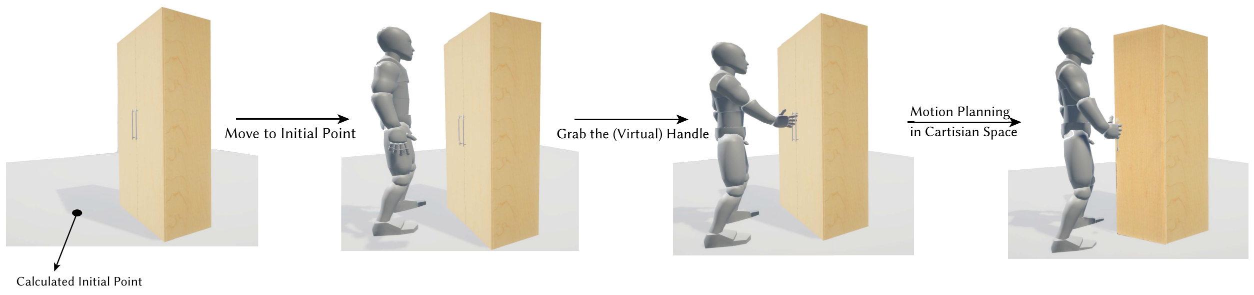


Fig. 4. Human-object Interaction. We use a motion planning pipeline to simulate human manipulating a certain movable part of the furniture. The process is divided into three steps: We estimate the initial standing point of the human using the size of furniture and human and attempt to place the human model at the initial point. If the placement succeeds, then we attempt to move the end-effector of the human to a grasping point of the movable part. If the movement succeeds, then we attempt to move the end-effector along the Cartesian path of the movable part (an arc for hinges, a line segment for sliders).

consider the ceiling and the floor, we can assume the wall is infinitely extended in the up and down directions.

There are many possible representations for walls. We can simply use 3D models or describe the inner boundary of the wall with a 2D ring. However, like in the case of movable parts, directly feeding any type of wall data into the network is not beneficial for learning the relationship between objects and walls. We observe that we need to take into account the object-wall distance to avoid collision. So, we extract the minimum distance between objects and the wall in each direction. Since we do not consider the up and down directions, for each object, we have 4-dimensional data. All the data forms a $|V| \times 4$ matrix.

Wall-object Distance Extraction. To extract the minimum wall-object distance, we adopt the ray-casting method; the brief diagram is shown in Figure 3. Due to the complicated representations of walls in the 3D-FRONT dataset [Fu et al. 2021] (e.g., walls represented by separate meshes with numerous vertices, instead of regular shapes), it is hard to calculate the distance between the wall and the bounding box of objects, and the simple 2D closed-form method is not available. Given an object bounding box and a direction, we first decide on the corresponding face. For example, if the direction is $+x$, then we select the front face of the bounding box. We then sample 10 points equidistantly along the other axis (in the example y-axis), with the center of the selected face in the middle. Then, for each point, we construct a ray. The origins of the rays are the sampled points, and the directions are the input direction. We test the interaction of these rays and the wall. The minimum distance among these intersections is our final result.

5 REINFORCEMENT LEARNING AND SIMULATION ENVIRONMENT

Our reinforcement learning agent learns the optimization policy through trial and error in a **simulation environment** that accurately represents indoor scenes. When the agent takes different actions, the environment returns rewards based on the actions taken. These rewards are calculated based on the human-scene interaction criteria described in Section 3. In this section, we will first introduce the implementation of our human-scene interaction model and then describe the reinforcement learning settings, including the detailed reward scheme.

5.1 Human-scene Interaction

Human-object interaction. We use motion planning on a humanoid robot to imitate the interaction between human and movable parts of objects. We simulate the interaction in three steps:

(a) Move to the initial point. We assume the root of the humanoid robot is in the front direction of the movable object, and the distance between them is 0.5 m. Then, we plan the path of the end effector of the robot. For now, we set the right hand as the end effector and attempt to generate a plan to move it to the grasping point ("handle") of the movable part. For hinge parts, we assume the handle lies on the opposite side of the rotation axis, and for slider parts, we assume the handle is in the middle of its front face.
(b) Attach object. If step (a) is completed, then the movable part is attached to the hand of the robot via a new joint. We observe that when grasping the handle, the palm can rotate around the main axis of the handle, which is parallel to the rotation axis in hinge objects and perpendicular to the slider axis in slider objects. The axis of this new joint is set according to the observation above.
(c) Move along a Cartesian path. When moving, the paths of hinge objects' handles are arcs, and the paths of slider objects' handles are line segments. We plan a Cartesian path for each handle, i.e., the end effector connected with it) and get the successfully planned portion of the path. Note that, in most cases, the path may not be fully planned. This is caused by various factors, including the limit of human joints, the collision between furniture, and the lack of space for the human to move.

We illustrate our procedure to produce human-object interaction information in Figure 4. Additionally, more sophisticated reward settings based on motion planning results can be added. With these extended settings, joint values of motion planning can be considered to determine the quality of human-object interaction. Please refer to the supplementary material for details.

Free space of human activity. We assume that a human occupies a volume of the shape of an elliptic cylinder (1,800mm $\times$ 750mm $\times$ 550mm) according to the dimensions of a typical human body [Panero and Zelnik 1999]. By testing all the possible positions that a person can stand, we extract the amount of free space

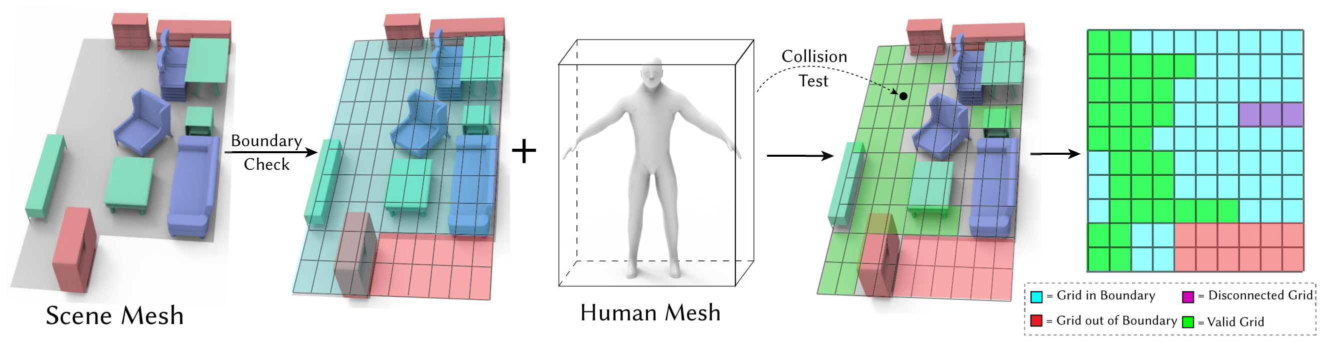


Fig. 5. Free Space for Human Activity. We use uniform grid sampling to compute the free space for human. First, we divide the 2-D bounding box of the floor into 900 grids and compute the grid inside and outside of the floor. For each grid, we attempt to place a human mesh onto its center. If the placement is free of collision, then the grid is labeled as valid. Finally, we compute the maximum connected component of the valid grids and use its size to compute the free space of human activity.

for human activity. We compute the amount of free space in two steps:

(a) Collision test. We divide the area of the bounding box of the floor into $30 \times 30 = 900$ grids and randomly sample a point in each grid. For each sampled point, we first determine whether the point lies inside the floor. If the point lies inside the floor, then we place the "human space" cylinder so the center of the cylinder coincides with the point. Then, we test whether the cylinder collides with the furniture. If the collision exists, then we rotate the cylinder along with y (up) axis by $\pi/2$ radians and test again to allow the possibility for a human to walk sideways through the space. If both tests yield collision, then the grid is not considered as free space.

(b) Extract the maximal connected component. After the collision test, multiple regions of free space can exist, but the regions are blocked from each other by furniture. In a real-world scene, only one of the regions is accessible via the door of the room. Since we do not consider doors, the region with the maximal area is considered the accessible one. To extract the maximal connected component, we use a **breadth-first search (BFS)** algorithm, which returns the area of the maximal connected component.

We illustrate our procedure to produce free-space information in Figure 5.

5.2 Settings of Reinforcement Learning

To train the Q-Network described in Section 3, we employ reinforcement learning techniques. For each scene in the training dataset, we create a virtual environment. The agent explores the environment using the ϵ-greedy policy, and each action taken by the agent results in an immediate reward. Once the agent stops optimizing, it is given a final reward that indicates the overall "quality" of the optimized scene. The states and rewards are stored in a replay buffer, from which actual training data is sampled. We provide details of the settings in this section.

Action Space. The action taken by the agent can be written as a pair (v, d), where v is a node in the object-level graph (V, E), and d is a move action type (or direction).

There are four types of actions $t_{+x}, t_{-x}, t_{+z}, t_{-z}$. Every action refers to moving the object towards a direction denoted by its subscript for 0.1 m, and they can be indexed by $0, 1, 2, 3$. We encode all actions in a scene into a number $a \in \mathbb{N}$. If the action is to move object i towards direction d, then we have

$$a = i \times 4 + d. \tag{3}$$

Rewards. When the agent decides to take a specific action, the state of the simulation environment is changed according to the action, and an immediate reward is given to the agent when every step of action is taken. Additionally, a final reward is given when the agent decides to stop the optimization. An immediate reward is a form of **reward shaping**, since the final reward is too sparse for the agent to learn. The components of the reward are described below.

(1) Collision. To obtain the collision number between two objects, we compute **IoU (Intersection over Union)** for every object part and compute the average of these IoU scores and multiply the score by -80. When an action is performed, the "discounted" difference between the collision score before the action and the collision score after the action is given as an immediate reward. The collision score at the final state is given as the final reward.
(2) Motion planning. For every movable part, we first determine whether the robot can move to the initial point. If it fails, then a reward of -5 is given. If it succeeds, then a reward of $p \times 5 - 5$ is given, where p represents the length of the successfully planned robot path length divided by the length of the desired path fed into the motion planning module (the length of the actually planned path is always less than the desired path) of the movable object. This reward is only given as the final reward.
(3) Free space. This reward is defined as

$$R_{fs} = 15 \frac{A_{free}}{A_{floor} - \sum A_i}. \tag{4}$$

A_{free} denotes the free space computed by the BFS algorithm. A_{floor} denotes the area of the floor, and A_i denotes the area of the 2D bounding box of furniture i. The "discounted" difference between the free space reward before and after an action is given as an immediate reward, and the free space reward at the final state is given as the final reward.

(4) First-time collision elimination. A reward of 20 is given immediately when the number of collisions changes from positive to zero for the first time.

(5) Rationality. A collision-free and high human affordance scene may be irrational. For example, when two nightstands are placed in the corners of a room instead of placed adjacent to the bed, the collision and human affordance metric may be high, but the scene is not rational in the sense of indoor scene layout. We thus train a regressor based on the ResNet18 [He et al. 2016] network backbone on the data of 3D-FRONT [Fu et al. 2021]. The training data format is a 22-dimension top-down image. The positive examples are original 3D-FRONT scenes, the negative examples are randomly perturbed 3D-FRONT scenes. The details of this network are described in Section 7.3. The network outputs the possibility of the scene being rational, and we give the rationality reward when the optimizing process concludes:

$$R_{ra} = 100 \times p_{real}, \tag{5}$$

where p_{real} is the possibility output by the network. This reward is only given as the final reward.

Stopping. Unlike the episodic task in Wang et al. [2020], our reinforcement learning task is a continuous task without a goal or stopping state. The only "episodic" task in our settings is to eliminate all collisions between furniture, thus, we give the first-time collision elimination reward to encourage the reduction of collisions. To tackle the stopping problem, we feed the average feature of all the nodes into an MLP and output a single Q-value indicating the reward obtained by immediately stopping the optimization. To fit the stopping action into our action-simulation pipeline, we add a "virtual object" after the actual objects, extending our action space dimension from $4 \times |V|$ to $4 \times (|V| + 1)$. The Q-value of all the four actions of the virtual object equals to the predicted Q-value. Once the virtual object is selected by the agent, we stop the simulation, concluding the optimization process.

6 THE ARCHITECTURE OF HAISOR AGENT

Our HAISOR agent consists of two parts: Q-Network and Monte Carlo Tree Search. We show an overview of our method in Figure 6.

6.1 Q-network

The goal of the Q-network is to predict the Q-value $Q(s, a)$ for state s and action a. For any state s, the optimal action predicted by the network is $\max_{a'} Q(s, a')$.

Network Structure. The Q-network is a Dueling Double Deep Q-Network based on a **Graph Convolution Network (GCN)**. The input of the network is a scene graph representation $\{(V, E), (V_p, E_p)\}$, and the output of the network is the estimated Q-value $\hat{Q}(s, a)$ for every possible action a.

The first building block of our network is GCN. For an arbitrary input graph (V, E), the GCN performs multiple message-passing passes on the graph, and every message-passing pass can be described as

$$V_i^{(k+1)} = \frac{1}{N_i} \sum_{(i,j)\in E} f^{(k)}([V_i^{(k)}; V_j^{(k)}; E_{ij}]), \tag{6}$$

where $V_i^{(k)}$ is the feature of the ith node after k rounds of message passing, N_i is the number of neighbors of the ith node, E_{ij} is the feature attached to the edge (i, j), and $f^{(k)}$ is a learnable encoder.

We first perform several rounds of message passing on the part-level scene graph (V_p, E_p) and obtain a matrix of data $V_p^{(k)}$ of shape $|V_p| \times L_{feature}$, where $L_{feature}$ is the length of feature vector output.

The second step is to aggregate the features of part-level nodes that are owned by the same object. This can be denoted by

$$V_i = \frac{1}{\sum_j \delta(a_j, i)} \sum_j \delta(a_j, i) V_{p,j}, \tag{7}$$

where $\delta(i, j)$ is 1 when $i = j$, and 0 otherwise.

After the aggregation of part-level node features, we then concatenate object-level feature V_i and the distance of object i to the walls in all four directions to the corresponding row of the feature matrix, changing shape of the feature matrix to $|V| \times (L_{feature} + 11 + 4)$. Then, we perform another round of message passing on this feature matrix and edges of object-level graph E. The obtained data $V^{(k)}$ is fed into two networks

$$Q_s = f_{state}\left(\frac{1}{|V|}\sum_j V_j^{(k)}\right), \tag{8}$$

$$Q_i' = f_{action}\left(V_i^{(k)}\right), \tag{9}$$

where f_{state} and f_{action} are **Multi-Layer Perceptrons (MLPs)**. We use this two-branch network structure after Dueling Q-Network [Wang et al. 2016]. Q_s is a scalar representing the Q-value of the state, and Q_i' is a 4-dimensional vector representing the Q-value of every action performed on object i.

The following equation is used to produce the final estimated Q-value $\hat{Q}(s, a)$:

$$\hat{Q}(s, a) = Q_s + \left(Q_{i(a)}'[d(a)] - \frac{1}{4 \times |V|} \sum_j \sum_{k=0}^{3} Q_j'[k]\right), \tag{10}$$

where $i(a) = \lfloor (a \div 4) \rfloor$, $d(a) = a \mod 4$, $x[n]$ is to obtain the data of vector x on index n.

Double Q-network. According to DQN [van Hasselt et al. 2016], using a single Q-network in both action selection and action evaluation may cause the overestimation of Q-values. To solve this problem, we use two networks of the same structure $Q_1(s, a), Q_2(s, a)$. When training, the target Q-value is computed using

$$y = r + Q_1(s, \max_{a'}(Q_2(s, a'))), \tag{11}$$

where y is the target Q-value, and r is the immediate reward given by taking the action.

Wall Constraints. Walls act as hard constraints, but our Q-network is not capable of handling the constraints directly, so we set the Q-value of the invalid actions that will produce new collisions between objects and walls to $-\infty$ in the output layer of our Q-network.

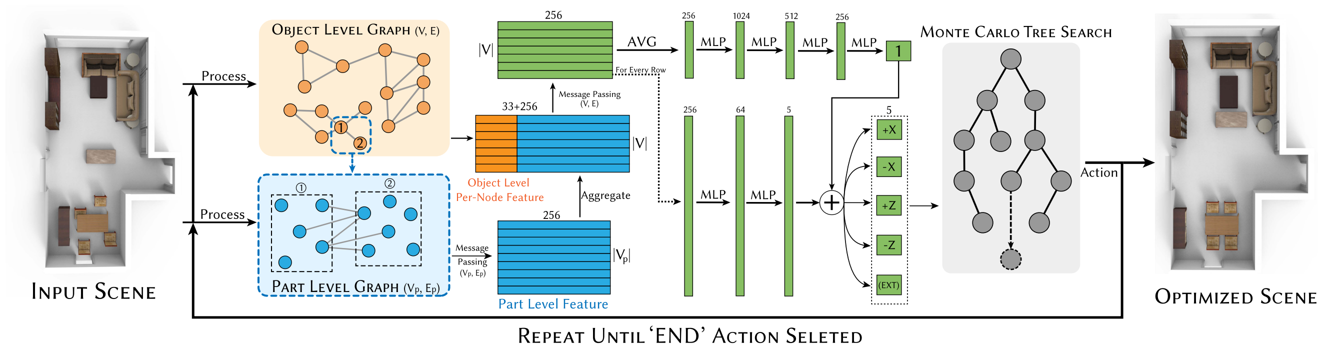


Fig. 6. Overview of HAISOR Agent. We use a two-layer scene graph as our representation. The Q-network is based on a message-passing graph network (GCN) on the two-layer graph. In the network, the part-level node feature is fed into a GCN on the part-level graph, aggregated in the object-level graph, and concatenated with the object-level feature. This block of feature is fed into a GCN on the object-level graph and produces the final per-object feature. This feature is the input of two branches of the MLP (Multi-Layer Perceptron) network, following the structure of Dueling Q-network [Wang et al. 2016]. The Q-value is finally used in the searching process in Monte Carlo Tree Search and outputs the final action.

6.2 Training Methodology

To optimize the training of our network, we adopt the prioritized replay buffer and imitation learning techniques and further extend the imitation learning to a version that can be bootstrapped by learning the policies of a simple heuristics agent at the beginning of training.

Prioritized Replay Buffer. In the training process of a deep Q-network, a set of data (s_t, a, s_{t+1}, r_t) is needed, where s_t, s_{t+1} denotes the states in time t and $t+1$, a is the action taken, and r_t is the immediate reward. During exploration, every step produces a set of data, and the data is stored in a buffer called Replay Buffer. Since our rationality and human affordance reward is sparse, only a small amount of data in the buffer is needed to correctly learn the Q-value. Thus, we associate every set of data with a priority $p = |y - Q(s_t, a)|$. This priority makes the data that the current Q-network predicts incorrectly to be sampled more frequently [Schaul et al. 2016]. The sampling algorithm is further controlled by parameters α and β.

Imitation Learning. The main problem for our network training lies in the slow speed of the initial exploration of the ϵ-greedy agent. Overall, our environment has sparse rewards, which means positive rewards can only be obtained in a few states. If we use the ϵ-greedy strategy, then the agent will spend a lot of time searching for positive rewards. Thus, we consider generating a sequence of actions known as demonstration data, which can achieve a subset of our goals, e.g., collision elimination. Since the demonstration data is only used in the early stage of training, it can be suboptimal. This training strategy is called imitation training [Hester et al. 2018]. The demonstration data is considered in the training by adding a loss term

$$L_d = \max_{a'}(Q(s,a') + l(a',a_d)) - Q(s,a_d) \tag{12}$$

$$l(a,a_d) = \begin{cases} 0, a = a_d \\ c, a \neq a_d \end{cases},$$

where a_d is the action in demonstration data, and c is a positive constant. This loss term forces the Q-value of a_d to be higher than other actions by c.

The demonstration data can be obtained by multiple methods. The most straightforward method is human annotation. However, we can produce the demonstration data by a simple heuristic method consisting of several rules:

(1) Rule 1 (Object Collision Elimination). If a collision between two objects exists, then select one of them randomly, and move the selected object away from the other. The movement direction is the axis of the maximum component of the vector that points from the other object to the moving object.
(2) Rule 2 (Wall Collision Elimination). If a collision between an object and the wall exists, then move the object toward the center of the scene. The movement direction is determined using the same method as in Rule 1.
(3) Rule 3 (Movable Part space). An object with movable parts has an extra space in front of it, with a length of 0.3 m, and the same width and height as the object. This is to roughly simulate the space needed for the movable part to be manipulated.

6.3 Monte Carlo Tree Search Algorithm

Monte Carlo Tree Search is a heuristic algorithm widely used in AI agents of board games. In our setting, the algorithm constructs a tree, the nodes of which are states of the environment. Each node is associated with a value, representing the expectation of the reward gained by selecting the best actions starting from this state. To decide an action for a certain state of the environment, the algorithm performs multiple iterations. Every iteration is divided into four steps:

(1) Selection. Starting from the root node (current state), the agent selects the child node with the largest value of **UCT (Upper Confidence bounds applied to Trees)** formula [Kocsis and Szepesvári 2006]:

$$R + C \times \sqrt{\frac{\ln N}{n}}, \tag{13}$$

where R is the associated reward of the node, N and n denote the count of visits of the parent node and the child node. C is a constant. The UCT formula aims to incorporate exploration and exploitation of the agent.

The selection process is repeated until it reaches a leaf node, and the leaf node is then selected.

(2) Expansion. The agent expands the selected leaf node by taking the action with the highest reward predicted by the Q-Network. Taking the action yields a new leaf node that is the child of the selected node.
(3) Simulation. The agent copies the state of the expanded leaf node to the simulation environment. Then the agent starts simulating S steps. The decision of each step is based only on the Q-network. S is a constant. The final reward after S steps is denoted by R_f.
(4) Back-propagation. The agent uses R_f to update the associated reward of the nodes in the "chain of parent elements" of the expanded leaf node.

7 EXPERIMENTS AND APPLICATIONS

In this section, we perform extensive evaluations on our HAISOR for scene optimization and the extension of HAISOR. Further, we compare it with the random agent, heuristics agent (see the rules in Section 6.2) and Sync2Gen-Opt agent [Yang et al. 2021] as three baselines in terms of visualization, quantitative metrics, and a perceptual study, illustrating our excellent performance. Some key designs are further validated via an ablation study.

7.1 Dataset Preparation

Training Data. To our knowledge, 3D indoor scene datasets with movable parts are unavailable. The 3D-FRONT [Fu et al. 2021] is a currently available 3D indoor scene dataset that is created by professional designers. We take the scenes generated by the underlying generators (see the next section) and replace the 3D-FUTURE [Fu et al. 2021] meshes with PartNet [Mo et al. 2019] and PartNet-mobility [Xiang et al. 2020] meshes. For each 3D-FUTURE mesh, we retrieve top-10 meshes with the smallest Chamfer Distance [Barrow et al. 1977] from PartNet. The 3D-FUTURE mesh in the scene is randomly replaced with top-10 retrieved meshes.

Note that this simple mesh-replacement approach may produce a large number of scenes that contain objects with inappropriate movable parts. For example, nearly half of the scenes have cabinet doors that cannot be opened. Since most indoor scene generation pipelines choose furniture using the scales of the bounding box of furniture, our replaced furniture can simulate the generated scenes by such methods.

7.2 Baselines

Scene Generators. To test our optimization framework, we utilize two SoTA scene generators: Sync2Gen [Yang et al. 2021] and ATISS [Paschalidou et al. 2021]. They are both trained on the 3D-FRONT dataset and generate layouts with the 3D box representation. Following these two works, we then retrieve meshes based on the sizes of generated boxes from the PartNet dataset. The generated scenes are then fed into different scene optimization algorithms to be evaluated. Considering the scale of the scenes to be optimized, the maximum optimization step count of the random, heuristic, and HAISOR agents in the following experiments are limited to 80 steps.

In our experiments, we perform the comparison with three agents, including the random agent, heuristics agent, and Sync2Gen-Opt agent:

(1) Random Agent: The agent randomly selects an action to perform from the action space.
(2) Heuristic agent: Our goal is to optimize the scene to make it satisfy multiple criteria (e.g., no collision, more free space) simultaneously. Instead of our HAISOR agent, we can use an agent driven by some simple heuristics that are identical to the rules described in Section 6.2.
(3) Sync2Gen-Opt agent [2021]: The scene generation framework consists of two steps: It first generates an initial prediction of scene layout by a Variational Autoencoder and then optimizes the prediction by L-BFGS [Liu and Nocedal 1989], a Quasi-Newton method. The target function is defined by Bayesian theory, and it optimizes the translation and the existence of the furniture objects. The "generate-optimize" process of Sync2Gen is similar to our setting. To better compare with theirs, we only use the VAE part of Sync2Gen (denoted Sync2Gen-VAE) to generate an initial scene layout. However, we modify the prior of their Bayesian target function to take human-free space of activity and collision between objects into account and delete the part of object existence (i.e., prohibit the agent from adding or removing objects); the modified version of the agent is denoted as Sync2Gen-Opt.
(4) Simulated Annealing Agent. Qi et al.'s pipeline [Qi et al. 2018] also leverages the layout optimization problem using a set of criteria, but the optimization algorithm is simulated annealing. We do not fully reproduce their method, because they optimize the scene from a totally random initial state. Instead, we use the criteria of our setting (trained regressor, human affordance, etc.) and substitute the RL+MCTS agent with simulated annealing.

Besides, to demonstrate the changes relative to the originally generated scenes, the metrics measured on the originally generated scenes are also reported.

7.3 Metrics

To evaluate the performance quantitatively, we adopt three kinds of metrics: accuracy, collision, and human affordance [Qi et al. 2018].

Accuracy. To measure the overall realism of the optimized scenes, a ResNet18 [He et al. 2016] is trained to regress the score for each top-down rendered image of the generated scene. The training data of the network consists of top-down renderings of randomly perturbed 3D-FRONT scenes and scores between 0 and 1. The scores are assigned as follows: First, we define a constant distance d_{max} for every type of room, which indicates the maximum distance of furniture from its original position. The distance is 0.35 m for bedrooms and 1.00 m for living rooms. Second, we pick a random value between 0 and 1, denoted d_{rand}. Third, we move every furniture in a random direction along the ground for a distance of $d_{rand} \cdot d_{max}$. The score of the scene is then assigned as $1 - d_{rand}$.

The rendered images have 2+C channels of three types: (a) Floor channel: takes the value of 255 when inside the floor; (b) Layout channel: takes the value of 255 when inside any furniture object; (c) Category channels: a channel for each object type and takes the value of 255 when inside any furniture object of that category. Any other pixel is assigned the value of 0.

Note that both the reward given by the simulation environment and the "accuracy" reported by experiments below make use of this network, but we use different sets of 3D-FRONT scene data to train two versions of the network in practice. We randomly split the 3D-FRONT scenes into two datasets with equal sizes and train the two networks of the same architecture separately.

Collision. The collision metric measures the collisions between different objects and between objects and the wall. It is calculated as $N_{coll} = 3 \times N_{wall} + N_{obj}$, where N_{wall} is the number of collisions between objects and the wall, and N_{obj} is the number of collisions between different objects. Note that N_{wall} and N_{obj} are both collision numbers, which means any small collision between objects will count as 1 collision.

Human Affordance. According to the discussion of human-scene interaction in Section 3, we evaluate two metrics associated with human affordance: (1) Movable Manipulation, defined as the percentage p described in Section 5.1, and (2) Free Space, defined as the free space of human activity divided by available space, which is exactly $R_{fs} \div 15$ in Section 5.1. For the detailed calculation of the two metrics, please refer to Section 5.1. The overall metric of human affordance can be regarded as the average of these two individual metrics.

7.4 Evaluations & Comparisons

The comparison of scene optimization. We perform optimization on the generated scenes by every baseline, namely, Sync2Gen [Yang et al. 2021] and ATISS [Paschalidou et al. 2021]. In Table 1, we show the numerical results of different baselines on scene optimization of generated scenes. According to the generated scene, we can see that both ATISS and Sync2Gen-VAE commonly perform worse in the living rooms than in bedrooms, which indicates the generation in living room is more challenging than in bedrooms. After the optimization, our method performs the best on all three metrics, indicating that Haisor is able to reconfigure the furniture placement to satisfy the designed criteria. In Figure 7, some optimized visual results are presented for different baselines. For the generated results of living rooms, more collisions between furniture still exist and more movable parts cannot be manipulated. We observe that our method captures the "functional region" of a set of furniture better, and our method can achieve multiple goals (e.g., solving collisions, extending free space, and finding space for movable parts) simultaneously, while the heuristic agent is only capable of solving collisions, Sync2Gen-Opt agent struggles to strike a balance between multiple goals, and the simulated annealing agent tends to overly separate the objects apart and fails to combine furniture into meaningful regions.

Furthermore, in our optimization results, we observed some smart strategies. One of the examples is shown in Figure 8. In this example, the initial scene is a living room, in which six chairs are placed together, causing a lot of collisions. The agent first moves two chairs of the same size to the center of the room, forming a region of "chairs surrounding a table" and subsequently moves four other chairs, forming the second region of "chairs surrounding a table." Finally, the agent makes some additional moves that increase the human affordance of the scene. We observe that our agent can learn the key feature for a scene to be realistic (in this case, the "chair-table" region) and derive the corresponding actions. Examples of more strategies learned to improve rationality and human affordance can be found in the supplementary material. Additionally, we include another example in Figure 9. The initial scene is overall rational, with the chair-table region and the table-sofa region placed together correctly. However, there are three objects with movable parts: two hinges of cabinets and one slider of a table. There is not enough space for humans to manipulate them comfortably. The results in Figure 9 demonstrate that our pipeline is able to optimize the scene to get more space for movable parts while preserving the rationality of the scene.

Perceptual Study. In addition to quantitative and qualitative results, we conduct a perceptual study to further evaluate how realistic and viable the optimized results of our method and two baseline methods (Sync2Gen-Opt and Heuristics agent) are. To fairly compare the results, we randomly select 50 living rooms and 50 bedrooms generated by Sync2Gen-VAE and ATISS to be optimized. We render the scenes in a top-down view and ask each participant to rank each algorithm according to two criteria (Rationality and Human Affordance): (a) the overall performance of the optimized rooms; (b) whether the participants will enjoy themselves when living in the scene. In every round of the survey, the participants are presented with two scenes for each method, including the input scene before optimization (without informing participants which is which) and rank the five results according to the above criteria. All the participants were local volunteers known to be reliable. The results of this perceptual study are reported in Table 2, and our method is the most preferred among the five results (the input scene and three optimized results). Further, we can see that the input scene gets a high score due to users' preference for the overall rationality of the scene, and Sync2Gen-Opt and heuristic agents may move furniture for long distances and make the scene look messy, although they may have higher metrics of the collision or human affordance.

Ablation Study. The performance of our method relies heavily on various components of rewards used in the simulation environment, the training methodology (i.e., imitation learning), and the Monte Carlo Tree Search. We perform ablation studies to demonstrate the key designs of our method.

We compare our full method with eight ablated versions of our method, where we remove a certain design of our method for each ablated version and train the network on the same settings as our full method. Note that we have included the w/o GCN ablated version, which replaces the GCN in the Q-network with a plain MLP. The MLP is simply fed with the concatenated feature as input, and the dimension of the output is $4 \times V_{max}$, where V_{max} is the maximum number of furniture in a room. But, the first $4 \times |V|$ elements of the output are used as the predicted Q-value, where $|V|$ is the real number of furniture of a certain scene. The rest elements are simply discarded in our experiments. We have also included a "Pure MCTS" ablated version of the agent,

Table 1. Comparisons with Other Optimization Methods

Generators	Baselines	Metrics (**Bedroom**)				Metrics (**Living Room**)			
		Accuracy(%, ↑)	Collision(↓)	Free Space(↑)	Movable Manipulation(↑)	Accuracy(%,↑)	Collision(↓)	Free Space(↑)	Movable Manipulation(↑)
Sync2Gen-VAE [2021]	Original (Input)	94.80	5.20	0.40	0.19	70.30	1.08	0.64	0.25
	Random	54.17	5.26	0.29	0.13	52.44	1.09	0.44	0.23
	Heuristic	59.49	5.08	0.32	0.16	69.13	0.53	0.61	0.31
	Sync2Gen-Opt [2021]	64.27	4.30	0.52	0.18	72.57	0.20	**0.73**	0.19
	Simulated Annealing [2021]	44.61	**3.77**	0.45	0.11	67.36	0.39	0.61	0.21
	Ours	**95.16**	4.12	**0.60**	**0.25**	**81.65**	**0.29**	**0.73**	**0.45**
ATISS [2021]	Original (Input)	70.20	1.40	0.31	0.14	69.94	2.38	0.49	0.29
	Random	44.86	1.66	0.23	0.14	45.19	1.99	0.35	0.30
	Heuristic	62.19	**0.49**	0.28	0.21	70.78	1.19	0.41	0.24
	Sync2Gen-Opt [2021]	68.10	0.79	0.41	0.18	68.32	0.81	0.64	0.34
	Simulated Annealing [2021]	64.27	1.14	0.39	0.28	57.17	0.40	0.38	0.19
	Ours	**78.44**	0.59	**0.50**	**0.36**	**84.69**	**0.37**	**0.72**	**0.42**

We show the accuracy, collision, and human affordance metrics of the original scenes generated by two different pipelines, and the optimization results of a random agent, heuristic agent, Sync2Gen-Opt agent, and our HAISOR agent. In the table, ↑ means higher is better, and ↓ means lower is better. We can see from the table that our method performs the best.

Table 2. Perceptual Study Results on 3D Indoor Scene Optimization

Gen.	Metrics	**Bedroom**		**Living Room**	
		R.(↑)	H.A.(↑)	R.(↑)	H.A.(↑)
S [2021]	Input	1.77	2.03	1.54	1.75
	Heuristic	1.43	1.67	2.60	2.49
	Sync2Gen-Opt [2021]	2.37	2.37	1.60	1.55
	Simulated Anneal [2021]	1.30	1.13	1.60	1.49
	Ours	**3.13**	**2.80**	**2.64**	**2.72**
A [2021]	Input	1.91	2.29	1.96	1.93
	Heuristic	1.88	1.79	2.25	1.60
	Sync2Gen-Opt [2021]	2.38	2.05	1.71	1.96
	Simulated Anneal [2021]	0.97	0.94	1.11	1.67
	Ours	**2.85**	**2.91**	**2.96**	**2.82**

We show the average ranking scores (from 3 (the best) to 0 (the worst)) on the rationality and Human Affordance of the scenes. The results are calculated based on 250 trials. We see that our method achieves the best on all metrics. The Generator "S" stands for Sync2Gen-VAE [Yang et al. 2021], "A" stands for ATISS [Paschalidou et al. 2021]. "R." stands for Rationality, and "H.A." stands for Human Affordance.

which makes decisions solely based on the search results of MCTS without the Q-Network prior.

The quantitative metrics are reported in Table 3. Statistics from the table clearly indicate that our full model outperforms all the ablated versions overall. The ablated methods achieve similar performance to our full method on some metrics (e.g., collision or human affordance), but our method is capable of considering all the aspects simultaneously. The qualitative results of all ablated versions and ours are presented in Figure 10. It is very clear that the results without imitation learning are totally worse than our full model under the same coverage steps. The agent without MCTS fails to move the chair under the table. When the free space and human affordance components are removed from our model, the furniture is moved to be highly scattered, which results in not enough space for humans to manipulate the cabinet beside the dining table and chairs. Without the collision reward, there are still some unresolved slight collisions, since they have almost no effect on rationality and human affordance. Without the rationality reward, the agent behaves similarly to the heuristics agent, which only tends to resolve collision. If the network is replaced by a plain MLP, then the agent is only capable of capturing part of the relationships between objects but fails to perform as well as the agent with GCN.

7.5 Extension of HAISOR to Scene Personalization

In addition to the rearrangement of the scene layout, our method can be extended for scene customization by adding more components to the reward settings. The motivation of our extension is to solve some common issues of general generative models. There are two key observations: (1) some of the generative models still predict the unreasonable orientation of objects. For example, it is unrealistic that a chair faces the wall. (2) Some objects are duplicated and lie in the same location, such as two dining chairs surrounding the dining table are predicted at similar locations.

Based on the observation above, we perform two types of extensions: orientation adjustment and object removal. In general, we extend the action space for each object by adding an extra action below:

(1) Orientation Adjustment: When the agent performs this action, the rotation angle θ along y-axis of the selected object is added by $\pi/2$, and if $\theta > 2\pi$, θ is subtracted by 2π. Additionally, a reward of -10 is given to avoid this action to be performed repeatedly.

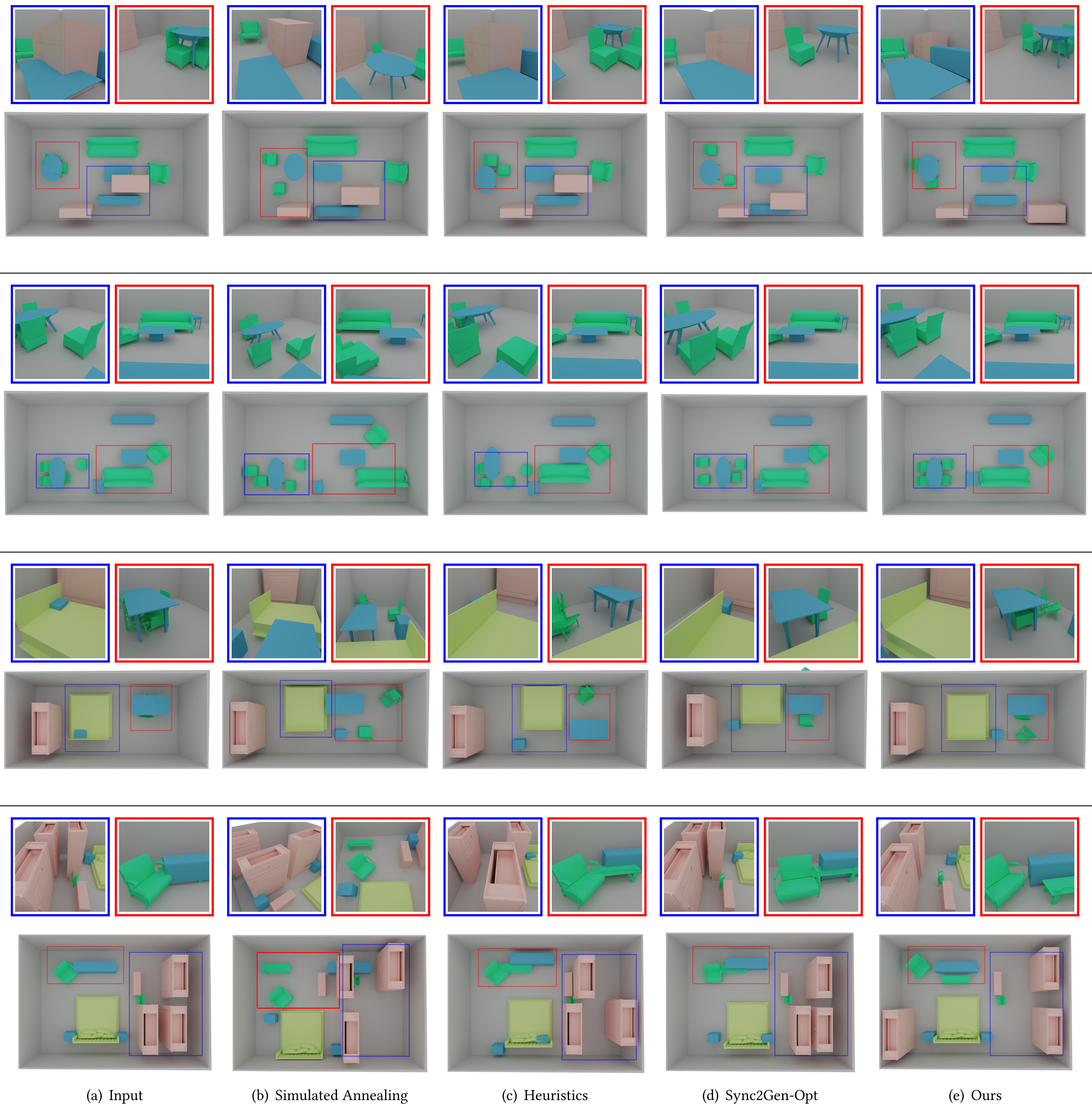


Fig. 7. Comparison of Scene Optimization. We show the comparison of optimization results of our method, random agent, heuristic agent, and Sync2gen-Opt [Yang et al. 2021]. From the results, we can see that our method better considers the rationality of the room, the collision between furniture, and human affordance of the room simultaneously. For each scene, the first row: the zoom-in view of two particular regions, the corresponding area is labeled by a colored rectangle in the figures of the second row; the second row: top-down view of the whole scene. For each row, left to right: Input scene, results of random optimization, results of heuristics optimization, results of Sync2Gen-Opt optimization, and results of our method.

(2) Object Removal: When the agent performs this action, the selected object will be removed from the scene. Additionally, a reward of $-25 \times S$ is given to avoid emptying all objects in the scene. S represents the summation of dimensions along three axes of the axis-aligned bounding box of the selected object.

Figures 11 and 12 show two examples of the extensions of our method. In Figure 11, one dining chair around the dining table does not face the table, while three other chairs do. This is frequently seen in the generation results of the SoTA scene generative models. The predicted orientation is often aligned with the x or z axes, so

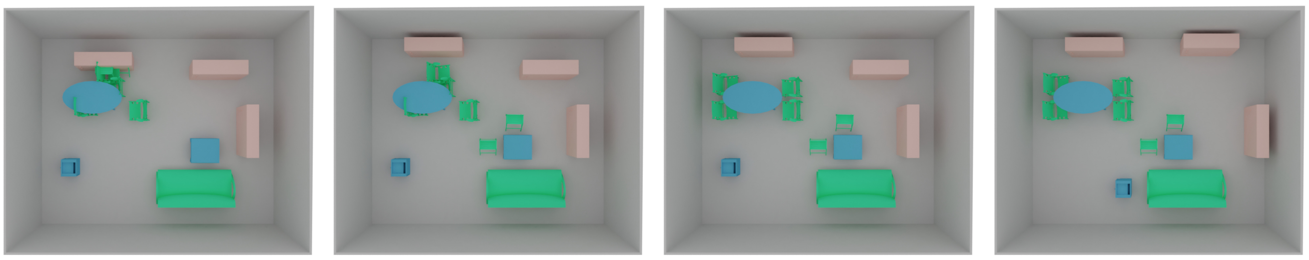

(a) Initial Scene (b) Restore 1st Region (c) Restore 2nd Region (d) Final Adjustment

Fig. 8. An Example of Learned Smart Strategy on More Complex Scenes. (a) The initial scene with multiple collisions. (b) The agent moves the coffee table towards the left direction and moves two stools around the coffee table, forming the first region of table-chairs. (c) The agent moves four chairs toward the dining table forming the second region. (d) The agent moves cabinets towards the wall and moves the stand towards the sofa to increase the human affordance scores.

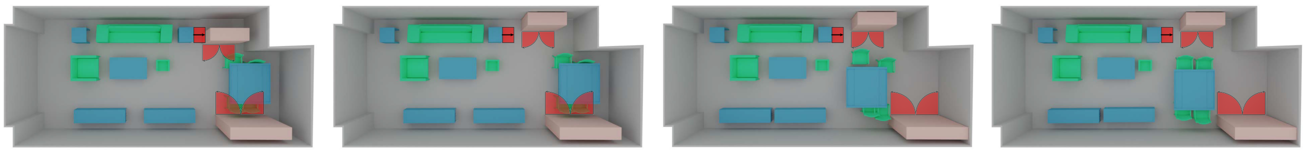

(a) Initial Scene (b) Get more space for the drawer (c) Get more space for the hinge (d) Final Adjustment

Fig. 9. An Example of Optimizing Human Affordance while Preserving Scene Rationality.The layout of the initial scene is generally rational, but two objects with movable parts: a drawer and a hinge cannot be properly manipulated by humans. Here, we show our agent is capable of optimizing this type of scene without losing the original layout of the scene.

Table 3. Ablation Study

Generators	Baselines	Metrics (**Bedroom**)				Metrics (**Living Room**)			
		Accuracy(%,↑)	Collision(↓)	Free Space(↑)	Movable Manipulation(↑)	Accuracy(%,↑)	Collision(↓)	Free Space(↑)	Movable Manipulation(↑)
Sync2Gen-VAE [2021]	Original (Input)	94.80	5.20	0.40	0.19	70.30	1.08	0.64	0.25
	w/o Imitation Learning	75.38	4.50	0.42	0.22	68.07	0.70	0.55	0.28
	w/o MCTS	70.62	5.10	0.25	0.20	66.45	0.97	0.60	0.35
	Pure MCTS	62.01	4.30	0.37	0.23	60.52	1.00	0.71	0.38
	w/o free space	82.13	**3.78**	0.41	0.19	62.38	0.60	0.54	0.44
	w/o manipulation	88.92	3.90	0.45	0.18	74.31	0.55	0.66	0.28
	w/o GCN	73.01	4.22	0.49	0.23	71.83	0.50	0.59	0.33
	w/o collision	91.18	4.34	0.53	0.20	58.11	0.91	0.63	0.21
	w/o rationality	75.31	3.99	0.51	0.17	48.00	0.31	0.69	0.27
	Ours	**95.16**	4.12	**0.60**	**0.25**	**81.65**	**0.29**	**0.73**	**0.47**

We show the accuracy, collision, and human affordance metrics of the original scenes and the optimization results of our full HAISOR agent and the ablated versions, each of which is trained without a certain element. We can see from the table that our full method performs the best.

we only need to adjust the orientation by $\pi/2, \pi, 3\pi/2$ radians or rotate the object by $\pi/2$ for 1,2,3 times. In Figure 12, two dining tables are predicted in exactly the same position. This is very common in the current generative models based on sequential generation, such as some frameworks based on transformers. To optimize this scene, we only need to remove one of the two tables to make the indoor scene more realistic.

The two examples above are extensions of our optimization method. Since our method is based on a simulation environment, Reinforcement Learning, and MCTS, HAISOR is able to achieve various optimization actions and goals. By simply adding some personalized actions or rewards, our method can optimize the indoor scene to the different targets that satisfy individual needs.

8 CONCLUSIONS, LIMITATIONS, AND FUTURE WORK

In this article, we have proposed a Reinforcement Learning-based method for solving the 3D indoor scene optimization problem. By iteratively predicting actions to move furniture in the scene, the HAISOR agent is capable of reconfiguring the layout of a 3D indoor scene, resolving collisions between objects, and making enough space for human-scene interaction. The agent consists of two parts: a Q-Network-based Reinforcement Learning agent and a Monto

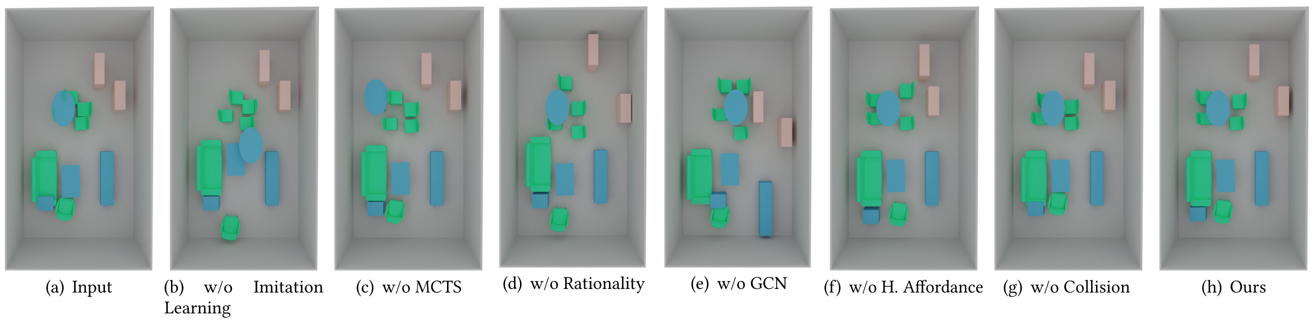


Fig. 10. Ablation Study on Some Components of Our Agent. We perform ablation studies that, respectively, remove various components of our method. In (b), the version without imitation learning, the agent fails to optimize. In (c), the version without MCTS, the agent fails to move the chair and table together. In (d), the version without rationality, the agent behaves like a heuristics agent, which mainly moves the object away from each other. In (e), the version without GCN, the agent can briefly capture some relationships between objects but performs worse than the version with GCN. In (f), the version without Human Affordance rewards (both free space and movable manipulation), there is no space between the cabinet and the dining chairs and table. And in (g), the version without collision, the last bit of collision between the armchair and sofa is not solved. In contrast, our full method does not have any of these problems.

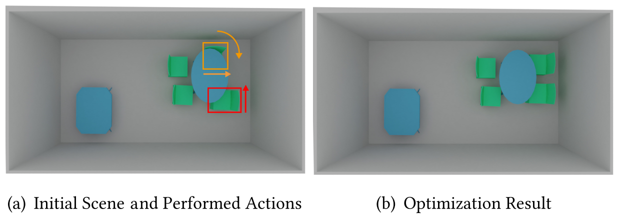


Fig. 11. Extending Our Method: Adding Orientation Adjustment. Figure (a) shows the initial scene, in which one of the chairs is not facing towards the dining table. To optimize this, we adjust both the position and orientation of the chairs. Figure (b) shows the optimization result of the extended method. The orientation of the chair is now correct.

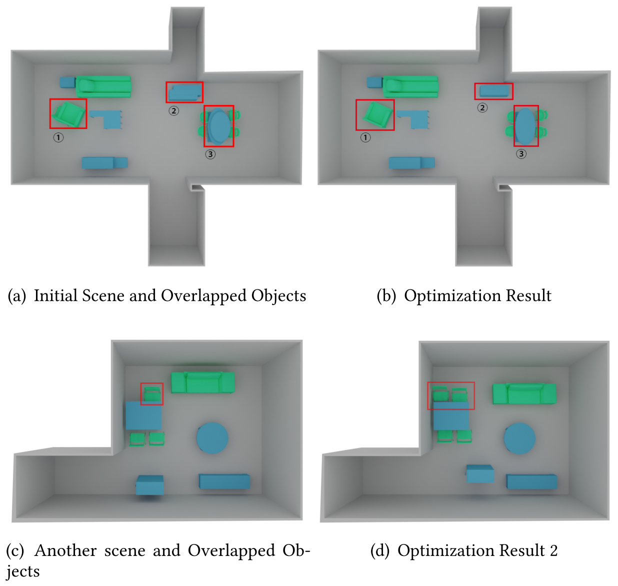
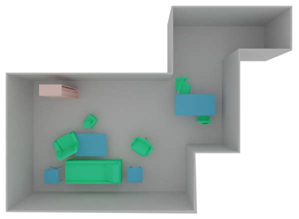
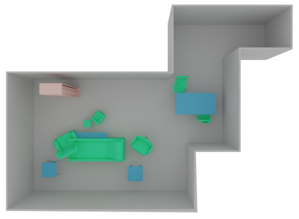

①
②
③
(a) Initial Scene and Overlapped Objects
(b) Optimization Result
(c) Another scene and Overlapped Objects
(d) Optimization Result 2

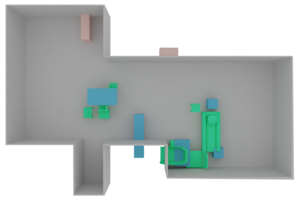
Fig. 12. Extending Our Method: Object Removal. Figure (a) shows the initial scene, in which three objects are overlapped with another object that is of identical function and size: (1) two chairs overlap with each other, (2) two cabinets overlap with each other, (3) two dining tables overlap with each other. To optimize this, we remove the smaller overlapped objects (due to our reward setting, removing the smaller objects gains more reward). Figure (b) shows the optimization result of the extended method. The collisions caused by overlapping are now resolved. Figure (c) shows another scene with overlapped objects. Removing one overlapped chair is not optimal, since it makes the "table-chair" structure lack one chair. It is not reasonable in the realistic scene. Figure (d) shows the optimization result of our agent. The chair is correctly moved to the suitable position, instead of being removed.

Carlo Tree Search agent, where the Q-value generated by the RL agent aids the searching process of the MCTS agent. Experiments on bedrooms and living rooms in 3D-FRONT demonstrate that the HAISOR agent can achieve superior performance on scene rearrangement and generate reasonable action sequences that efficiently optimize scenes towards the goal of rationality and human affordance in a human-aware manner.

Limitations & Future Works. There are some limitations of our HAISOR: (a) Our method is difficult to train on mixed-type datasets. Since there is a different distribution (e.g., table-chair regions for living rooms, bed-nightstand pairs for bedrooms) of furniture layouts in the different types of rooms, our network is very hard to capture all furniture distribution using only a network. For each type of dataset, we trained our HAISOR on the different types of rooms separately. (b) Our HAISOR agent can not work perfectly on the large-scale optimization, i.e., the diverse and various furniture layouts in the large scenes, the large action space. Due to the discrete actions in our network, the object only is moved by a small step (0.1 m) at each iteration. If the initial scene has numerous objects to be optimized, then it is not tolerable that the agent takes more time to search for the optimal action sequence. (c) Trapped in local minima. Since the action history is not considered in our framework, our agent may hesitate when moving a certain object. For example, one table may be moved back and forth multiple times in a few cases. (d) Missing the other structure within the indoor scene, e.g., doors and windows, which usually play an important role in the rationality and functionality of the 3D indoor scene. In Figure 13, some failure cases of our HAISOR are presented. Figures (a) and (b) show the results of trying to apply the HAISOR agent trained on bedrooms to living rooms, and the optimization fails because the agent can not capture the features of object layout in living rooms. Figures (c) and (d) show the results of trying to


(a) Initial Scene 1


(b) Optimization Result 1 (Using Q-Network Trained on Bedrooms)


(c) Initial Scene 2 (Large and Messy Scene)

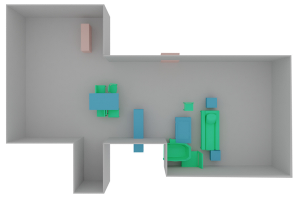

(d) Optimization Result 2

Fig. 13. Failure Cases. Figure (a) shows a typical living room to be optimized, and figure (b) shows the result that uses the Q-network trained on bedrooms. The agent fails to capture the relationship between the sofa, table, and chairand finishes the optimization prematurely, causing unresolved collisions. Figure (c) shows a large and "messy" living room, with objects outside the wall and multiple objects colliding with each other. Figure (d) shows the optimization result. The agent uses 80 steps, which is the maximum step count we have set and optimized for some of the objects such as chairs and tables, but more objects are not optimized. We observe that our agent has limited performance when asked to optimize this kind of scene.

optimize a large and complex scene with HAISOR agent. The optimization fails because the agent reaches a maximum of 80 steps.

In this work, we mainly focus on optimizing the arrangement of one relatively small 3D indoor scene by moving objects. Our work can be improved by following future directions: (1) To address the optimization problem in a large and complex indoor scene with other personalized goals, a continuous action space is a suitable choice to strengthen the power of reinforcement learning rather than discrete DQN. (2) Our work could be extended to the whole house with multiple rooms and take more relations into our network instead of a single room for some future interesting applications. (3) More factors (e.g., furniture styles, room doors, and windows) are not involved in our work; more labeling in the public dataset is required to allow our approach to create more realistic rooms. (4) We can extend the optimization target to a single 3D geometric object for modeling and design, or the macro placement in chip design, like the task in Mirhoseini et al. [2021]. (5) We currently rely on a user study to provide holistic views of the results, and it is worth developing new objective holistic metrics to facilitate evaluation.

REFERENCES

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained policy optimization. In *34th International Conference on Machine Learning*, Vol. 70. 22–31.

Alekh Agarwal, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. 2020. Optimality and approximation with policy gradient methods in Markov decision processes. In *Conference on Learning Theory*, Vol. 125. 64–66.

Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Conference on Advances in Neural Information Processing Systems*. 3981–3989.

Fan Bai, Fei Meng, Jianbang Liu, Jiankun Wang, and Max Q.-H. Meng. 2021. Hierarchical policy for non-prehensile multi-object rearrangement with deep reinforcement learning and Monte Carlo tree search. *CoRR* abs/2109.08973 (2021).

Harry G. Barrow, Jay M. Tenenbaum, Robert C. Bolles, and Helen C. Wolf. 1977. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Image Understanding Workshop*. 21–27.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. 2019. Dota 2 with large scale deep reinforcement learning. *CoRR* abs/1912.06680 (2019).

Bryce Blinn, Alexander Ding, Daniel Ritchie, R. Kenny Jones, Srinath Sridhar, and Manolis Savva. 2021. Learning body-aware 3D shape generative models. *CoRR* abs/2112.07022 (2021).

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-Carlo tree search: A new framework for game AI. In *4th Artificial Intelligence and Interactive Digital Entertainment Conference*. 216–217.

Cheng Chen, Weinan Zhang, and Yong Yu. 2022. Efficient policy evaluation by matrix sketching. *Front. Comput. Sci.* 5 (2022).

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Conference on Advances in Neural Information Processing Systems*. 3837–3845.

Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. 2012. Example-based synthesis of 3D object arrangements. *ACM Trans. Graph.* 31, 6 (2012), 135:1–135:11.

Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 2021. 3d-future: 3d furniture shape with texture. *International Journal of Computer Vision* 129, 12 (2021), 3313–3337.

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. Graph.* 36, 6 (2017), 201:1–201:13.

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 International Conference on Robotics and Automation*. 3075–3082.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 770–778.

Paul Henderson and Vittorio Ferrari. 2017. A generative model of 3D object layouts in apartments. *CoRR* abs/1711.10939 (2017).

Todd Hester, Matej Vecerík, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John P. Agapiou, Joel Z. Leibo, and Audrunas Gruslys. 2018. Deep Q-learning from demonstrations. In *32nd AAAI Conference on Artificial Intelligence*. 3223–3230.

Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. 2020. Graph2Plan: Learning floorplan generation from layout graphs. *ACM Trans. Graph.* 39, 4 (2020), 118:1–118:14.

Z. Sadeghipour Kermani, Zicheng Liao, Ping Tan, and H. Zhang. 2016. Learning 3D scene synthesis from annotated RGB-D images. In *Computer Graphics Forum*. 197–206.

Jennifer E. King, Marco Cognetti, and Siddhartha S. Srinivasa. 2016. Rearrangement planning using object-centric and robot-centric action spaces. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3940–3947.

Jennifer E. King, Vinitha Ranganeni, and Siddhartha S. Srinivasa. 2017. Unobservable Monte Carlo planning for nonprehensile rearrangement tasks. In *IEEE International Conference on Robotics and Automation*. 4681–4688.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations*.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based Monte-Carlo planning. In *17th European Conference on Machine Learning*. 282–293.

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 *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2678–2685.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Conference on Advances in Neural Information Processing Systems*. 1106–1114.

Kurt Leimer, Paul Guerrero, Tomer Weiss, and Przemyslaw Musialski. 2022. LayoutEnhancer: Generating good indoor layouts from imperfect data. In *ACM SIGGRAPH Asia Conference (SA'22)*. 27:1–27:8.

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. *Int. J. Robot. Res.* 37, 4-5 (2018), 421–436.

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. Graph.* 38, 2 (2019), 12:1–12:16.

Dong C. Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Math. Program.* 45, 1–3 (1989), 503–528.

Jingjing Liu, Wei Liane, Bing Ning, and Ting Mao. 2021a. Work surface arrangement optimization driven by human activity. In *IEEE Conference on Virtual Reality and 3D User Interfaces.* 270–278.

Lijuan Liu, Yin Yang, Yi Yuan, Tianjia Shao, He Wang, and Kun Zhou. 2021b. In-game residential home planning via visual context-aware global relation learning. In *35th AAAI Conference on Artificial Intelligence.* 336–343.

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), 173:1–173:13.

Rui Ma, Akshay Gadi Patil, Matthew Fisher, Manyi Li, Sören Pirk, Binh-Son Hua, Sai-Kit Yeung, Xin Tong, Leonidas Guibas, and Hao Zhang. 2018. Language-driven synthesis of 3D scenes from scene databases. *ACM Trans. Graph.* 37, 6 (2018), 212:1–212:16.

Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. 2011. Interactive furniture layout using interior design guidelines. *ACM Trans. Graph.* 30, 4 (2011), 87:1–87:10.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim M. Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, Will Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (2021), 207–212.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. 2017. Learning to navigate in complex environments. In *5th International Conference on Learning Representations.*

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with deep reinforcement learning. *CoRR* abs/1312.5602 (2013).

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. 2019. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 909–918.

Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. 2020. House-GAN: Relational generative adversarial networks for graph-constrained house layout generation. In *16th European Conference on Computer Vision.* 162–177.

Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang, Hang Chu, Chin-Yi Cheng, and Yasutaka Furukawa. 2021. House-GAN++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 13632–13641.

Brendan O'Donoghue, Rémi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. 2017. Combining policy gradient and Q-learning. In *5th International Conference on Learning Representations.*

Julius Panero and Martin Zelnik. 1999. *Human Dimension & Interior Space: A Source Book of Design Reference Standards.* Whitney Library of Design, New York, NY.

Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. 2021. ATISS: Autoregressive transformers for indoor scene synthesis. In *Conference on Advances in Neural Information Processing Systems.* 12013–12026.

Giovanni Pintore, Claudio Mura, Fabio Ganovelli, Lizeth Fuentes-Perez, Renato Pajarola, and Enrico Gobbetti. 2020. State-of-the-art in automatic 3D reconstruction of structured indoor environments. In *Computer Graphics Forum.* 667–699.

Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song-Chun Zhu. 2018. Human-centric indoor scene synthesis using stochastic grammar. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 5899–5908.

Daniel Ritchie, Kai Wang, and Yu-An Lin. 2019. Fast and flexible indoor scene synthesis via deep convolutional generative models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 6182–6190.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized experience replay. In *4th International Conference on Learning Representations.*

Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354–359.

Haoran Song, Joshua A. Haustein, Weihao Yuan, Kaiyu Hang, Michael Yu Wang, Danica Kragic, and Johannes A. Stork. 2020. Multi-object rearrangement with Monte Carlo tree search: A case study on planar nonprehensile sorting. In *IEEE/RSJ International Conference on Intelligent Robots and Systems.* 9433–9440.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* Bradford Books.

Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *30th AAAI Conference on Artificial Intelligence.* 2094–2100.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Conference on Advances in Neural Information Processing Systems.* 5998-6008.

Quan Ho Vuong, Yiming Zhang, and Keith W. Ross. 2019. Supervised policy update for deep reinforcement learning. In *7th International Conference on Learning Representations.*

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 (2020), 233:1–233:15.

Hanqing Wang, Zan Wang, Wei Liang, and Lap-Fai Yu. 2021a. PEARL: Parallelized expert-assisted reinforcement learning for scene rearrangement planning. *CoRR* abs/2105.04088 (2021).

Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel X. Chang, and Daniel Ritchie. 2019. PlanIT: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Trans. Graph.* 38, 4 (2019), 132:1–132:15.

Kai Wang, Manolis Savva, Angel X. Chang, and Daniel Ritchie. 2018. Deep convolutional priors for indoor scene synthesis. *ACM Trans. Graph.* 37, 4 (2018), 70:1–70:14.

Xinpeng Wang, Chandan Yeshwanth, and Matthias Nießner. 2021b. SceneFormer: Indoor scene generation with transformers. In *International Conference on 3D Vision.* 106–115.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *33rd International Conference on Machine Learning*, Vol. 48. 1995–2003.

Gordon Wilfong. 1991. Motion planning in the presence of movable obstacles. *Ann. Math. Artif. Intell.* 1 (1991), 131–150.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. 2020. SAPIEN: A SimulAted part-based interactive ENvironment. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 11094–11104.

Wenzhuo Xu, Bin Wang, and Dong-Ming Yan. 2015. Wall grid structure for interior scene synthesis. *Comput. Graph.* 46 (2015), 231–243.

Haitao Yang, Zaiwei Zhang, Siming Yan, Haibin Huang, Chongyang Ma, Yi Zheng, Chandrajit Bajaj, and Qixing Huang. 2021. Scene synthesis via uncertainty-driven attribute synchronization. In *IEEE/CVF International Conference on Computer Vision.* 5610–5620.

Sifan Ye, Yixing Wang, Jiaman Li, Dennis Park, C. Karen Liu, Huazhe Xu, and Jiajun Wu. 2022. Scene synthesis from human motion. In *ACM SIGGRAPH Asia Conference (SA'22).* 26:1–26:9.

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. Graph.* 31, 4 (2012), 56:1–56:11.

Lap Fai Yu, Sai Kit Yeung, Chi Keung Tang, Demetri Terzopoulos, Tony F. Chan, and Stanley J. Osher. 2011. Make it home: Automatic optimization of furniture arrangement. *ACM Trans. Graph.* 30, 4 (2011), 86:1–86:12.

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. *Robot. Auton. Syst.* 119 (2019), 119–134.

Weihao Yuan, Johannes A. Stork, Danica Kragic, Michael Y. Wang, and Kaiyu Hang. 2018. Rearrangement with nonprehensile manipulation using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation.* 270–277.

Song-Hai Zhang, Shao-Kui Zhang, Yuan Liang, and Peter Hall. 2019. A survey of 3D indoor scene synthesis. *J. Comput. Sci. Technol.* 3 (2019), 594–608.

Song-Hai Zhang, Shao-Kui Zhang, Wei-Yu Xie, Cheng-Yang Luo, Yongliang Yang, and Hongbo Fu. 2022. Fast 3D indoor scene synthesis by learning spatial relation priors of objects. *IEEE Trans. Visualiz. Comput. Graph.* 28, 9 (2022), 3082–3092.

Zaiwei Zhang, Zhenpei Yang, Chongyang Ma, Linjie Luo, Alexander Huth, Etienne Vouga, and Qixing Huang. 2020. Deep generative modeling for scene synthesis via hybrid representations. *ACM Trans. Graph.* 39, 2 (2020), 17:1–17:21.

Yang Zhou, Zachary While, and Evangelos Kalogerakis. 2019. SceneGraphNet: Neural message passing for 3D indoor scene augmentation. In *IEEE/CVF International Conference on Computer Vision.* 7383–7391.

Received 6 November 2022; revised 24 July 2023; accepted 12 October 2023