Automated task planning using object arrangement optimization

Mincheul Kang[1], Youngsun Kwon[1] and Sung-Eui Yoon[2]
(https://sglab.kaist.ac.kr/ObjectArrangement)

***Abstract*— We present a method enabling a robot to automatically arrange objects using task and motion planning. Given an input scene consisting of cluttered objects, our method first constructs a target layout of objects as a guidance to the robot for arranging them. For constructing the layout, we use positive examples and pre-extract hierarchical, spatial and pairwise relationships between objects, to understand the user preference on arranging objects. Our method then enables a robot to arrange input objects to reach their target configurations using any task and motion planner. To efficiently arrange the objects, we also propose a priority layer that decides an order of arranging objects to take a small amount of actions. This is achieved by utilizing a dependency graph between objects. We test our method in three different scenes with varying numbers of objects, and apply our method to two well-known task and motion planners with the virtual PR2 robot. We demonstrate that we can use the robot to automatically arrange objects, and show that our priority layer reduces the total running time up to 2.15 times in those tested planners.**

I. INTRODUCTION

Robots and autonomous devices are becoming more widely available, and we are using them to a variety of tasks ranging from simple tasks, e.g., washing cars, to dangerous tasks, e.g., handling radioactive materials. Currently, a lot of manufacturing companies use industrial robots to reduce cost and boost productivity. However, domestic robots have not gained much attraction due to their incompetence in performing household chores. Recently, with the rise of artificial intelligence and advances of humanoid robots, we are getting more interest in designing and utilizing domestic robots. Intelligent domestic robots can interact with humans and offload a significant amount of works from humans.

One way that a domestic robot can be intelligent is by finding a series of feasible actions for a given task and performing those actions on its own. Many task and motion planning methods have been proposed to efficiently plan and perform various applications such as cooking food, facilitating industrial logistics, harvesting fruits, and so on. In this paper, we focus on arranging cluttered objects neatly in a room.

To perform a task such as arranging objects, robots require a specific goal or a guided plan from a user. Unfortunately, it is very inconvenient or even infeasible for a user to give specific sequences of actions to a robot. In addition, a complex case consisting of a large number of objects

[1]Mincheul Kang (mincheul.kang@kaist.ac.kr) and [1]Youngsun Kwon (youngsun.kwon@kaist.ac.kr) are with the School of Computing, and [2]Sung-eui Yoon (Corresponding author, sungeui@kaist.edu) is with the Faculty of School of Computing, KAIST at Daejeon, Korea 34141

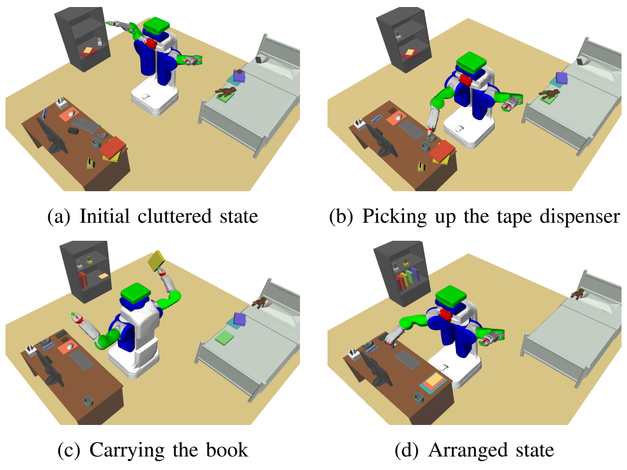

(a) Initial cluttered state (b) Picking up the tape dispenser (c) Carrying the book (d) Arranged state

Fig. 1. These figures show the process of arranging objects from an initial cluttered state (a) to an arranged state (d) by using a virtual PR2 robot and taking various actions including (b) and (c). (b) shows the robot picking up the tape dispenser, and (c) shows the robot carrying the book.

becomes the hard-to-solve problem. To address these issues, we focus on an approach that enables a robot to arrange the cluttered objects in an efficient way through automatically generating a goal from a non-specific command.

Since the problem that we target is extremely wide, we assume in this paper that a robot can recognize and grasp objects, so we can assign a grasping pose for each object. This is a reasonable assumption, because many works have been developed on object recognition [1], [2] and object grasping [3], [4] using deep learning. We therefore focus on making goal states, task sequences, and robot trajectories to arrange objects automatically.

Main contributions. In this paper, we present an integrated approach for automated planning of a robot for object arrangements. Our method consists of two main parts: generating a target layout for object arrangements, and efficiently planning a series of feasible actions to reach the target layout (Fig. 1). To find a target configuration of each object, we first optimize configurations of objects using various relationship extracted from positive examples. This process is carried out by using very fast simulated re-annealing [5] (Sec. IV). We can then use any task and motion planning to reach the target layout (Sec. V-A). We also propose a priority layer to efficiently perform complex object arrangements by utilizing the relationships of objects from the layout computation. The priority layer represents dependencies of objects as a directed graph, and determines an efficient order of arranging objects by topological sort (Sec. V-B).

To analyze our method, we test three rooms with different characteristics in terms of arranging the objects. We also apply our layout computation method to two well-known task and motion planners, and show that our method automatically arranges objects based on the computed layouts. In addition, we also show that our priority layer robustly improves the overall running performance up to 2.15 times in the two tested task and motion planners (Sec. VI).

II. RELATED WORK

In this section, we discuss prior work on object arrangements, followed by integrated task and motion planning.

A. Object Arrangements

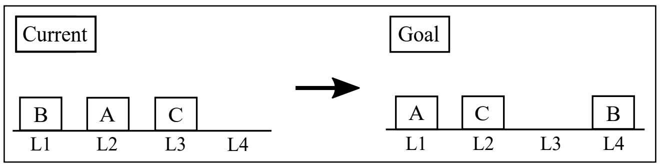
Jiang et al. [6] suggested a method for a robot to place objects in stable and semantically preferred area using a supervised learning with point cloud data. On their subsequent work, Jiang et al. [7] considered arranging objects by learning the relationship between human poses and objects so that objects are easily accessible by humans. Abdo et al. [8] proposed a method to tidy up objects on shelves and in boxes by applying user preferences. These user preferences were determined by a collaborative filtering technique based on crowd-sourced and mined data.

Our approach is inspired by the work of automatic furniture arrangements [9], which targets to synthesize a variety of indoor scenes by using the relationships of furniture for digital content creation. While this work is not designed for robotic applications, we adopt it for our problem as a goal generation and extend it by using our very fast simulated re-annealing. Furthermore, we utilize various information extracted during layout computation for our priority layer, enabling effective and efficient integration between layout computation and TMP.

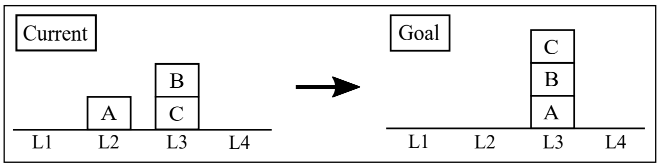
B. Task and Motion Planning

Hierarchical Planning in the Now (HPN) [10] reduces search depth and planning time by dividing the plan and execution in a large state space. Some works proposed a method using an interface layer that connects a symbolic task planner to a geometric motion planner. The work of De Silva et al. [11] used shared literals to combine two planners and control backtracking based on hierarchical task networks [12]. Srivastava et al. [13] used off-the-shelf task planners, and if an obstacle gets in the way, they simply create an additional task that removes the obstacle. Although these planners can arrange multiple movable objects accurately, they do not seek to minimize the number of actions to arrange them.

To efficiently solve this problem, Stilman et al. [14] used exhaustive backtracking search for monotone problems. Krontiris et al. [15] designed a constraint graph using minimum constraint removal paths (MCR) [16] and then got the order of moving objects by topological sort. They also proposed a higher-level incremental search algorithm to solve non-monotone problems. Our work handles a more challenging problem that requires pick-and-place actions for moving on another furniture or stacking objects. Since obtaining MCR of all objects for pick-and-place actions is time-consuming, the planning time to execute is too long. To progress planning and execution simultaneously, we greedily search the target object using the information of layout computation.



(a) Locationally intertwined case



(b) Hierarchically intertwined case

Fig. 2. These are two examples that can occur in the object arrangement. (*a*) shows that target locations of objects are intertwined. For example, L2 is the current location of the object A, and it is also the goal location of the object C. (*b*) shows that the parent-child relationships of objects are intertwined. For example, the object C is the parent of object B in the current state, while the object B is the parent of object C in the goal state.

III. OVERVIEW

In this section, we give motivations and overview of our work.

A. Motivations

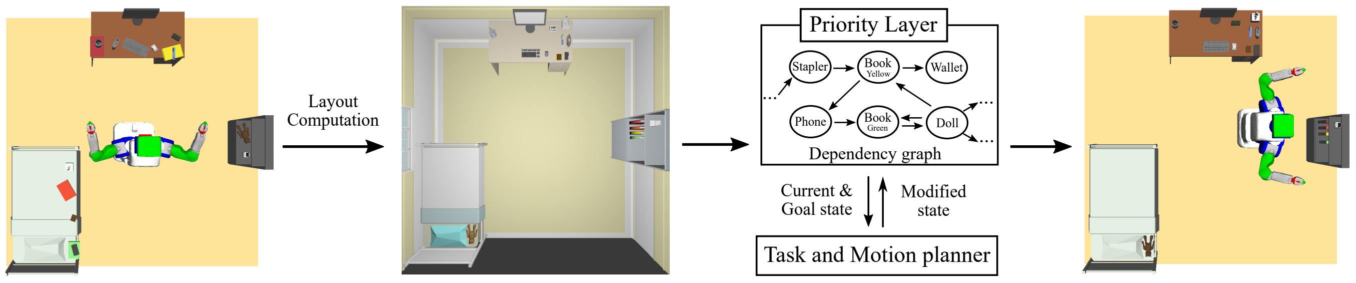
Task and Motion Planning (TMP) computes a sequence of actions to perform a given task or goal, while considering various constraints (e.g., avoiding collisions). Related to our problem of object arrangements, recent robotic applications use a TMP to automatically perform a task such as tidying up a table [17] or industrial logistics [18]. Unfortunately, these approaches assume that target configurations of objects for the task are already given or manually specified, and automatically computing such targets for TMP has been studied relatively less compared to TMP itself.

Our object arrangement tasks have many and diverse objects located in scenes. Furthermore, their states of objects can be intertwined when we consider current and goal states, as shown in Fig. 2. For handling our problem efficiently, it is desirable to find an optimal order of object arrangements, e.g., moving objects in the order of $B \rightarrow A \rightarrow C$ in Fig. 2(a). Fig. 2(b) also shows an example case containing hierarchically stacked objects, where the object A should be placed at the bottom first in the goal states.

B. Overview of Our Approach

To automatically arrange objects, we propose a method that integrates layout computation and TMP. Fig. 3 shows an overview of our approach. Our approach is divided into two stages.

Our layout computation first constructs a target layout of objects contained in the environment. As its learning phase, our work extracts various object relationships, such as



(a) Initial state (b) Arranged scene (c) Planning & Execution (d) Goal state

Fig. 3. This shows an overview of our approach. Our method first constructs an arranged scene as the target layout (b) from the initial state (a) using various relationships extracted from user's positive examples. At runtime, we use our priority layer, considering the computed layout and various relationship between objects defined in a dependency graph, for a robot to arrange objects by using a task and motion planner (TMP). The priority layer communicate with TMP (c). We repeat to use TMP with our priority layer until all the objects are arrived at their goal state (d).

spatial, hierarchical and pairwise relationship, from positive examples to a given environment. By using the extracted relationship and considering the reachability of a robot to objects, our work optimizes configurations of objects into arranged configurations. We also propose to use very fast simulated re-annealing for efficiently optimizing those configurations.

Second, using the target layout, our work plans feasible actions of the robot to arrange objects using an existing TMP. We propose a priority layer that utilizes extracted relationships between objects, and enables efficient object arrangement, resulting in effective integration between layout computation and TMP. While the layout computation constructs the target configurations of objects, the priority layer decides an order of arranging objects by avoiding obstacles and redundant actions, to take the minimum number of actions. For this purpose, we define prerequisites of placing objects and represent them in a dependency graph. Our priority layer determines an order of placing objects using topological sorting with greedy search on the dependency graph.

According to the decision of the priority layer, TMP plans the movement of a robot to arrive at the goal state. After executing the plan, the priority layer updates the state received from TMP. Our work repeats this process until all objects are arranged.

IV. Automatic Layout Computation

In this section, we explain how to compute arranged configurations of objects from the input cluttered environment, followed by feeding them to TMP for enabling a robot to arrange them.

Our work optimizes layout configurations of objects based on user's positive examples. Our work assumes that users provide such positive examples in the 3D scene that consists of 3D objects. Note that advances of 3D modeling tools, e.g., 3ds Max, and widely available 3D objects, e.g., Google 3D warehouse, it is relatively easy even for novice to construct such 3D scenes reflecting his or her rooms. As a result, we choose to utilize such positive examples for considering user's preference.

A. Extraction

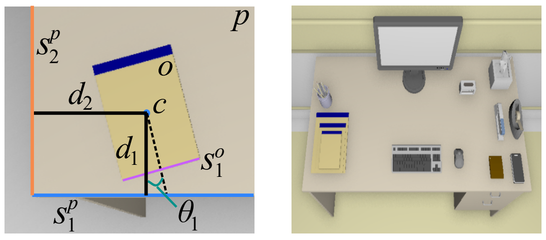
The extraction process enables us to obtain various relationships that serve as the basis for optimizing configurations of objects. Specifically, our work extracts hierarchical and pairwise relationships from positive examples. The hierarchical relationship is defined as a parent-child relation of two objects, where a parent object can support a child object; e.g., the parent of a keyboard can be a desk. We also derive a relative spatial transformation from the parent to the child. This relative spatial transformation can be well represented by 2D translation and 1D angle of an object given its parent.

Fig. 4(a) represents an example of hierarchical relationship between a desk and a notebook located on the desk. For computing a relative configuration of an object o, given its parent object p, we first identify the closest surface, s_1^o, of the object o from its parent object p. We then also find another surface, s_1^p, of the parent object p, from s_1^o of the object o. Once we compute such a pair of surfaces, we then compute the relative distance d_1 and angle θ_1; examples of them are also shown in Fig. 4(a). In addition, we pick another side, s_2^p, between two or multiple sides connected to the surface s_1^o as the closest side to the object o. We then also compute another relative distance, d_2, between s_2^p and the object o. Fig. 4(b) shows an arrangement result computed by considering these relative translation and angle.

Our work also extracts similar information between various pairs of objects, as the pairwise relationship, from positive examples. When these additional information are extracted and considered during our optimization process, we can get more appropriate results for our arrangement application (Fig. 4(b)). One may think that we also need to encode the relative spatial location along the height, but we found that it is unnecessary for our applications, since objects are stacked together for our application and thus their height can be implicitly encoded.

B. Layout Optimization

We use an optimization process to compute target, arranged configurations of objects using information obtained from the extraction process. Our optimization is based on very fast simulated re-annealing (VFSR) [5], which is



(a) Hierarchical-spatial relationship (b) Considering 2D translation and 1D angle

Fig. 4. (a) We extract two distances (d_1, d_2) and one angle θ_1 between the parent p and its child o objects. (b) shows a result of our arragement computed by considering these relative 2D translations and 1D angle.

analogous to physical annealing process, to compute layout candidates that have low cost values given our cost function (Eq. 3, 4). Intuitively speaking, the cost function simply measures the distance between the current configuration of an object and its reference configuration extracted from positive examples, indicating a cluttered level of the object from the positive examples.

VFSR is one of the powerful simulated annealing approaches, which has been known to compute the global minimum, under the assumption that its cooling stage is slow enough. Among many alternatives, we choose VFSR because it has been shown to be exponentially faster convergence than others (e.g., fast simulated annealing [19] and genetic algorithms [20]).

To effectively compute configurations of arranged objects, we design our problem of object arrangement to be suitable for the VFSR algorithm. We regard a cluttered level of a scene as temperature, i.e., a cost of the scene; we give a higher cost to the scene as more objects are cluttered away from positive examples. Our goal is then to find a state for each object that minimizes the temperature. Our state, X, includes 2D positions and angle, i.e., $[x, y, \theta]$, for our problem. At an iteration i of our optimization process, the temperature associated with each state parameter is given by:

$$T_i^{X^j} = T_0^{X^j} e^{c i^{1/D}}, \tag{1}$$

where T_0 is an initial temperature, which is set as an initial cluttered level by our cost function, and X^j is the j-th element of the state space X, D is the dimension of the state space X, which is 3 in our case. The control parameter c is the Boltzmann's constant in the above equation.

VFSR has advantages of processing different temperatures for each state space and specifying a finite range. These advantages can be effectively utilized for our arrangement by moving and rotating objects within their parent objects, to efficiently reach to the optimized configurations. To move and rotate an object, we generate a random variable, δX^j, using the distribution of VFSR:

$$\delta X^j = sgn(u^{X^j} - \frac{1}{2}) T^{X^j} [(1 + \frac{1}{T^{X^j}})^{|2u^{X^j} - 1|} - 1]; \quad u^{X^j} \in U[0,1], \tag{2}$$

where u^{X^j} is from the uniform distribution. The calculated δX^j belongs to $[-1, 1]$, and the amount of change in the j-th state space, ΔX^j, is then constrained.

Specifically, the configuration of an object is updated by translation, $(x_i, y_i) \rightarrow (x_i + \Delta x, y_i + \Delta y)$ or rotation, $(\theta^i) \rightarrow (\theta_i + \Delta\theta)$, where the range of Δx and Δy are constrained by the size of the parent object, and the range of $\Delta\theta$ belongs to $[-\pi, +\pi]$.

As a minor optimization for hierarchical positioning of objects, we can reduce the search space of VFSR within independent sets of objects, where each set contains objects that have the parent-child relationship. This is possible because placing objects are constrained within their parent objects determined from the hierarchical relationship. Specifically, for each set, we assign a different, local temperature depending on the number and cluttered level of objects in the set, instead of using a global temperature for all the objects in a scene. As a result, we can improve the convergence by reducing iterations on nearly arranged sets and thus allocating more time on significantly cluttered sets. We found that this simple optimization improves 1.44 times on average on our layout computation (Sec.VI-A).

Our cost minimization process based on VFSR works as follows: Most objects in our applications are placed on top of other objects, e.g., desk, bookshelf and notebook. Therefore, we first check whether an object, o, in our target layout has its parent object, p, and if so, we place the object o on top of its parent object p by utilizing the extracted hierarchical relationship. Given that hierarchical positioning, we continue to move or rotate the object o until the cost evaluated from our cost function becomes small enough.

Cost calculation. In the extraction process, we extracted various information, d_1, d_2, θ_1 with specific surfaces, and pairwise information. The overall cost function is defined to be the sum of the following cost terms. A hierarchical cost, $C_h(o)$, for an object o given its parent object p is defined as:

$$C_h(o) = \min_{\psi_i \in \Psi} (w_h^{d_1} |d_1^{\psi_i} - \overline{d_1^{\psi_i}}| + w_h^{d_2} |d_2^{\psi_i} - \overline{d_2^{\psi_i}}| + w_h^{\theta_1} |\theta_1^{\psi_i} - \overline{\theta_1^{\psi_i}}|), \tag{3}$$

where $w_h^{\cdot}$ is a weight for each element term, ψ_i represents each positive example from its positive example set, Ψ; e.g., d_1 is computed in the current configuration, while $\overline{d_1^{\psi_i}}$ indicates d_1 value extracted from the i-th positive example from Ψ. A pairwise cost, C_{pair}, for a pair of two interacting objects, (o_{pair_1}, o_{pair_2}), is defined as:

$$C_{pair}((o_{pair_1}, o_{pair_2})) = \min_{\psi_i \in \Psi} (w_{pair}^{d} |d - \overline{d^{\psi_i}}| + w_{pair}^{\theta} |\theta - \overline{\theta^{\psi_i}}|), \tag{4}$$

where d and θ are the L2 distance and angle between two interacting objects (o_{pair_1}, o_{pair_2}), respectively; $\overline{d^{\psi_i}}$ and $\overline{\theta^{\psi_i}}$ are extracted from the i-th positive example from Ψ.

Our work also considers a constraint of reachability to reflect physical limitations of a robot (e.g., gripper size and arm length). To consider reachability, we utilize an unreachable region as a space where a robot cannot pick and place objects, and determine the region by considering the kinematics of robot and the configurations of objects. We then disallow configurations of objects that are in the unreachable area, while they reduce the overall cost.

V. Task and Motion Planning

In this section, we explain how to use the computed layout on TMP using the priority layer.

A. Representing the Layout as Symbolic Forms

Our work uses existing TMP planners such as HPN [10] and the combined TMP approach [13]. Since these planners are domain-dependent, they define states of a domain in a symbolic form through PDDL (Planning Domain Definition Language) [21] format. PDDL consists of a domain and a problem. A domain describes predicates and a set of robot actions, and a problem sets the initial state and the goal state specifications for objects.

To utilize those planners, we also define a domain as well as a problem for our task of arranging objects through PDDL. Our object arrangement can be solved by three basic actions: *pick*, *place*, and *move*; a robot can pick up or place an object and move it to the goal position of the object. We define the domain using these three actions according to each planner. In addition, we define a problem based on the domain for a robot to arrange objects.

The problem is constructed simply by expressing the automatically arranged layout computed by VFSR as the goal state. Fortunately, the arranged layout already contains all the information for making a symbol to define the goal state. For an example, we define a symbol $At(o,\ p,\ \mathbf{T}_o^p)$, to denote the position of an object o relatively to its parent p through a 4 by 4 transformation $\mathbf{T}$. This symbol is used to specify the object o both in initial and goal states.

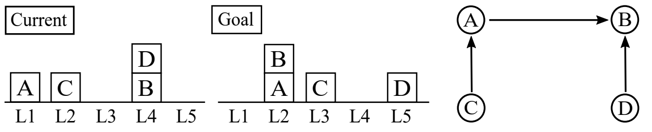
B. Priority layer

We now explain how to arrange objects efficiently using our priority layer. Our work is based on the approach of Krontiris et al. [15] that uses the minimum constraint removal paths (MCR) [16] for constructing a directed graph. On the other hand, our approach predicts the possible constraints using the information of layout computation and then performs a task using TMP according to the predicted result. This is a greedy way to overcome the drawbacks of many planning time to execute until obtaining paths for all objects.

Our priority layer sets prerequisites for moving objects in our application. A prerequisite of an object o to reach its goal is that its obstacle o_{obs} located in a trajectory from o to its goal must be moved to another location or its target location before moving the object o. In the example of Fig. 5(a), an object C is located at the target location of the object A. To avoid any unnecessary actions, we must move the object C to another location before moving the object A.

We encode these prerequisites between objects as a dependency graph and use our priority layer to determine an order of arranging objects through topological sort on the graph.

Making a dependency graph. Our priority layer represents the relationship of objects as a dependency graph



(a) No circular dependency case (b) Dependency graph

Fig. 5. (a) is an example that has no circular dependency. (b) shows the dependency graph of the (a).

$G_t = (V_t, E_t)$, where t is a time step at which TMP is executed, V_t is a set of nodes, which is composed of objects, i.e., $V_t = \{o_1, o_2, \cdots, o_n\}$ with the current state Q_t and the goal state Q_Γ. E_t is a set of directed edges, each of which indicates a prerequisite between two objects. Specifically, a directed edge e indicates geometric constraints (e.g., having collisions) or hierarchical constraints (e.g., having the parent-child relationship) between two objects. Let us first explain on creating edges for geometric constraints. Suppose that we attempt to move an object o to its target configuration. In this case, if we have a collision against another object o_{obs}, we have a dependency between two objects and thus create an edge, $(o_{obs} \rightarrow o)$, indicating intuitively that we have to move o_{obs} to another location first before moving o to its target configuration. In the example of Fig. 5(a), moving the object A to its target configuration, $L2$, causes a collision with the object, C, which is located on $L2$ at the current state. As a result, we have to move C to another location, preferably, its target configuration, $L3$, and we thus create an edge $(C \rightarrow A)$, which is shown in Fig. 5(b).

We now explain how to create edges for the hierarchical relationship between two objects: a parent, p, and its child, c. Fortunately, this hierarchical relationship can be easily obtained from our layout computation. However, a special care is given for considering the relationship between ones in the current state and goal state. In the example of Fig. 5(a), the object D is on the object B in the current state, and the object B is on the object A in the goal state. In the current state, the child object D must be moved first before moving its parent object B, and we thus create an edge $(D \rightarrow B)$. On the other hand, for the goal state, the parent object A must reach the target configuration first before its child object B, and we thus create an edge $(A \rightarrow B)$ (Fig. 5(b)).

Topological sort. We use topological sort in order to effectively compute an order of arranging objects from the constructed dependency graph, G_t. Since topological sort lists nodes by considering directions of the directed edges, we can construct a sequence of arranging objects while considering dependency and thus avoiding future collisions. Overall, we perform the following steps until the robot reaches the final goal state computed by our layout computation.

In our method, we collect nodes that do not have any incoming edges in the dependency graph G_t into a list L_t, which is in line 3 of Alg. 1. Note that those nodes without having any incoming edges mean that they do not have any dependency and thus can be processed earlier than others. Among those possible candidates in the list, we select a node, i.e., an object, which has a minimum traveling distance

Algorithm 1: Topological sort for arranging objects.

```
Input: $R^{loc}$, $Q_0$, $Q_\Gamma$, robot's location, current and goal
       states.
1  $G_0 \leftarrow MakeGraph(Q_0, Q_\Gamma)$
2  while $Q_t \neq Q_\Gamma$ do
3      $L_t \leftarrow GetNoIncomingEdge(G_t)$
4      if $L_t$ is empty then
5          $o_{inter} \leftarrow CutCircularDependency(G_t)$
6          $o_{tar} \leftarrow FindIntermediateLocation(o_{inter})$
7      else
8          $o_{tar} \leftarrow FindLowcostNode(L_t, R_t^{loc})$
9      $Q_{t+1}, R_{t+1}^{loc} \leftarrow DeliverTMP(o_{tar})$
10     $G_{t+1} \leftarrow UpdateGraph(Q_{t+1})$
```

for a robot to reach the object and move it to its target configuration.

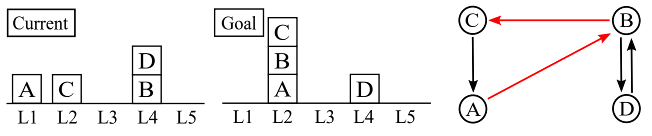
For each chosen node, o_{tar}, we need to know its target position for computing its estimated traveling distance and for passing it to TMP. Fortunately, this information is readily available thanks to our layout computation. We then create a problem for the selected target node o_{tar} and send it to TMP for making the object reach its target configuration (line 9 of Alg. 1).

After executing the task for placing o_{tar} at its target configuration, we accept the modified state Q_{t+1} and then update the graph G_{t+1} based on the modified state Q_{t+1} (line 10 of Alg. 1). Since moving an object makes only local changes to its node and its dependent nodes in the graph, updating the dependency graph takes only a minor portion, less than 1%, of the total running time.

Handling circular dependency. There is a special case requiring additional treatment for effective arrangement: the case involving the circular dependency. In this case with two or multiple objects having the circular dependency, e.g., swapping two objects, we must find an intermediate target position for an object before we finally move it to the final target position. We can easily identify such cases based on our dependency graph, even if many objects are intertwined deeply, e.g., a cluttered desk with many stacked objects in the tested bedroom scene.

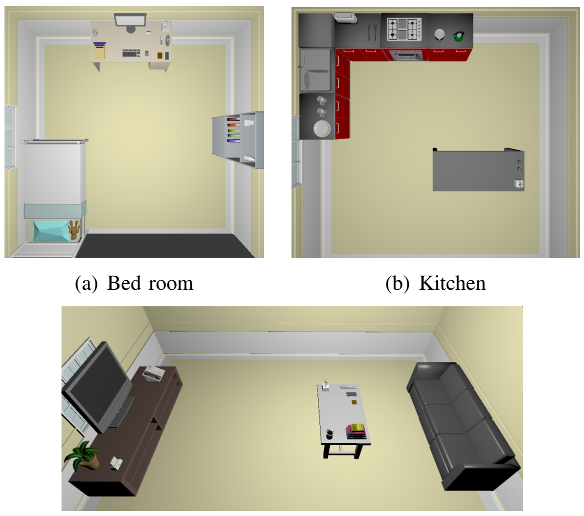
It is important not only to solve a circular dependency, but also not to make it. To avoid making such circular dependency, the intermediate position of an object o should not be located at the target configuration of objects that are dependent on the object o. Considering these conditions, we randomly select the intermediate position without collision.

To address a circular dependency, we find a sequence of actions to remove the circular dependency. We can remove such edges involved in the circular dependency by moving objects, o_{inter} (line 6 of Alg. 1), to intermediate locations. Fig 6(a) shows a case involving the circular dependency. We, however, do not remove edges created by hierarchical relationships in the goal state, since we do not recompute our arranged layout that is already computed by considering user preference. We show such edges in the red arrows in



(a) Circular dependency case (b) Dependency graph

Fig. 6. (a) is an example that has circular dependency. (b) shows the dependency graph of (a). Arrows represent edges, while red arrows indicates unremovable edges due to the final, fixed layout.



(a) Bed room (b) Kitchen

(c) Living room

Fig. 7. These figures show target, arranged layouts computed by our method for three different types of rooms.

Fig. 6(b).

VI. Experiments and Results

We test our method and others on a machine that has 3.60GHz Intel i7-3820 CPU and 16GB RAM. We use the PR2 robot for arranging objects within OpenRAVE simulator and Trajopt [22], which is an optimization-based planner, for manipulating the robot. We use two well-known task planners to show a wide applicability of our method.

To test the feasibility of our method for object arrangements, we use various 3D models obtained from Google 3D warehouse and Fisher et al. [23]. Overall, we use 55 different 3D models to place furniture and objects in three different rooms: kitchen, living room, and bedroom (Fig. 7). We construct these three test scenes that have different characteristics in terms of arranging the objects.

We design the kitchen and living room scenes to analyze effects on the hierarchically stacked objects. The living room scene has deep hierarchical relationships, e.g., four books are stacked on the table, while the kitchen scene has no hierarchical relationships between objects. The bed room scenes are used for comparing the performance depending on the number of objects.

A. *Results of layout computation*

We compute an object layout based on information extracted from positive examples. Our work uses five positive examples for each scene, while having more objects and their

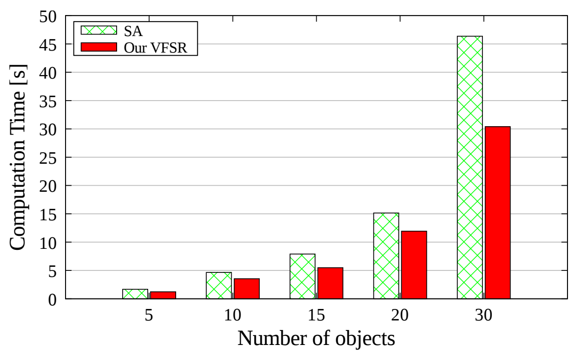


Fig. 8. This figure shows the running time taken for layout computation, as a function of the number of objects. Our VFSR approach shows 1.44 times on average and up to 1.52 times performance improvement on average over simulated annealing (SA).

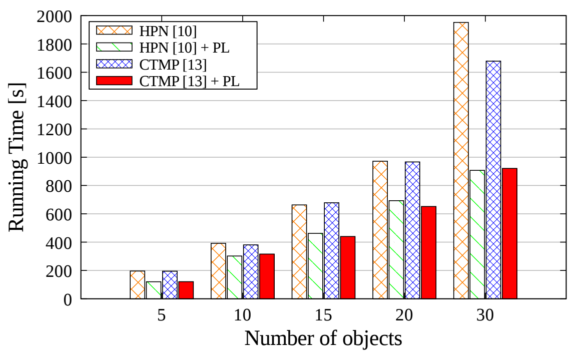


Fig. 9. This figure shows the overall running time when we use different task and motion planners for our method w/ and w/o using our priority layer. We can see that the running time of two existing TMP is reduced robustly by applying our priority layer (PL).

diverse positions in examples can bring better results; the used positive examples for the bedroom scene are shown in the accompanying video. The habits of arranging objects vary across different persons, and a user can reflect his or her preference based on those positive examples (Sec. IV-A).

To show benefits of our VFSR used for layout computation, we also implement and test a standard simulated annealing (SA) [24] on the bedroom scene. We measure how long these two different layout computation methods take to compute an object arrangement layout as a function of the number of objects. Intuitively, the relationships of objects become complicated further, as the number of objects increases. Therefore, finding optimized configurations in such complex situations requires many random rotation as well as translation of objects for the tested optimized methods. As a result, layout computation takes more running time as the number of objects increases (Fig. 8). Nonetheless, our VFSR shows 1.38 times on average (1.52 up to) performance improvement over SA in the tested bedroom scene.

B. *Analysis of the priority layer*

Once we compute target layouts of objects, we can use any existing task and motion planner to maneuver a robot to arrange objects to their target layouts. To show a wide applicability of our approach, we test two well-known TMP planners: Hierarchical Planning in the Now (HPN) [10] and Combined Task and Motion Planner via interface layer (CTMP) [13]. Our priority layer helps these planners do work efficiently by selecting the target object in consideration of prerequisites in our application.

To verify the efficiency of our priority layer, we measure the running time for arranging objects by the PR2 robot coupled with these planners. We also evaluate the number of actions made by PR2 to reach the goal and the total distance traveled by PR2.

Fig. 9 shows the running time of the tested planners in the bedroom scene that has varying numbers of objects. The running time includes the processing time on planning, execution time of actions, and computation time on the priority layer. As the number of objects increases, the running time of HPN and CTMP increases. Given this general trend, we are able to observe meaningful performance improvement, 1.64 times on average and up to 2.15 times, by using our priority layer in both tested HPN and CTMP task planners. This improvement is achieved mainly by avoiding inefficient and redundant actions thanks to our priority layer, and thus reducing the number of actions. Note that our computed target layouts enable a robot to arrange objects, and our priority layer accelerates its planning time by utilizing various information available from our computed layouts through the dependency graph.

Table I shows various statistics, the number of actions, traveled distance, and runtime breakdown of running different TMP methods with and without our priority layer. Across all the tested scenes, using the priority layer shows up to 2.15 times and 1.82 times performance improvement in terms of the overall running time compared to not using the priority layer, measured with two tested planners. Let us look at results of living room and kitchen scenes in Table I. Note that these two scenes have the same number of objects. However, it takes 1.5 times more actions for the living room than that of the kitchen, when using existing planners without using our priority layer. This is mainly because the living room has some objects that need to be stacked for arrangements (e.g., four books stacked), while we do not have such objects in the kitchen scene. When this hierarchical relationship is not carefully handled, it can cause unnecessary actions in the process of moving and stacking objects. On the other hand, when these planners are used together with our priority, they can effectively reduce the number of actions, resulting in higher performance, thanks to considering dependency between objects encoded in our dependency graph.

Limitations. Our layout computation based on VFSR is faster than the standard simulated annealing, but it is not real-time (e.g., less than 1 sec. or so). While a faster approach is desirable, users can reduce the hassle of manually giving goal states for objects in scenes even with the current method. When our method encounters new objects that are not included in the positive examples, however, users need to create additional examples with those objects.

VII. Conclusion

In this paper, we have introduced a novel integration method between layout computation and existing TMP, to

TABLE I

WE MEASURE VARIOUS STATISTICS, AVERAGED OUT FROM TEN INDEPENDENT TESTS, OF DIFFERENT METHODS FOR EACH SCENE. THE NUMBER IN PARENTHESES IS THE NUMBER OF OBJECTS.

Planner		HPN [10]		CTMP [13]	
Priority layer (PL)		w/o	w/	w/o	w/
Living Room (10)	Planning time [s]	684.52	336.84	594.02	321.88
	Execution time [s]	122.68	71.79	124.43	72.38
	PL time [s]	0.00	0.18	0.00	0.16
	Actions	84	48	83	48
	Distances [m]	57.59	45.00	58.61	45.38
Kitchen (10)	Planning time [s]	347.59	254.40	336.44	259.10
	Execution time [s]	86.30	72.70	86.78	72.37
	PL time [s]	0.00	0.15	0.00	0.16
	Actions	55	48	55	48
	Distances [m]	50.41	43.02	52.70	44.73
Bed Room (10)	Planning time [s]	305.46	234.41	300.47	250.85
	Execution time [s]	86.24	67.37	79.84	64.90
	PL time [s]	0.00	0.19	0.00	0.17
	Actions	51	44	50	44
	Distances [m]	61.31	51.80	62.46	52.96
Bed Room (20)	Planning time [s]	796.61	559.44	791.14	519.73
	Execution time [s]	174.87	132.82	175.58	131.66
	PL time [s]	0.00	0.81	0.00	0.73
	Actions	116	88	119	88
	Distances [m]	125.95	113.74	125.94	109.17
Bed Room (30)	Planning time [s]	1629.68	693.01	1366.79	722.58
	Execution time [s]	322.44	211.87	311.63	196.34
	Layer time [s]	0.00	2.28	0.00	2.16
	Actions	214	132	216	132
	Distances [m]	170.25	101.10	162.68	102.86

automatically arrange cluttered objects. We constructed a target layout automatically by considering various relationships between objects that also reflect the user preference on object arrangement. To arrange objects in the computed layout, our method plans a series of feasible actions using the well-known task and motion planner such as HPN and CTMP. We also proposed a priority layer to arrange objects efficiently for deciding the order of arranging objects using our dependency graph and topological sort, while considering hierarchical relationships.

There are many future directions in automatically arranging objects using a robot. First, we would like to design a real-time layout computation method using a recent deep learning approach. In the current work, we have tested the usability of our method in the simulation environment. We believe that this can be easily translated to a real robot with various sensors, thanks to the mature level of our tested simulation environment. Nonetheless, it requires additional techniques, such as balancing a humanoid robot while maneuvering objects, and handling various uncertainty caused by sensors. Fortunately, these topics have been well studied, and would like to extend our approach to work well with these techniques.

ACKNOWLEDGMENT

We would like to thank SGVRLab members and anonymous reviewers for constructive comments. This research was supported by MI/KEIT (10070171) and NRF/MSIT (No.2011-0030079).

REFERENCES

[1] Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, Min Lu, and Jianwei Wan, "3d object recognition in cluttered scenes with local surface features: a survey", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2270–2287, 2014.

[2] Daniel Maturana and Sebastian Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition", in *IROS*. IEEE, 2015, pp. 922–928.

[3] Joseph Redmon and Anelia Angelova, "Real-time grasp detection using convolutional neural networks", in *ICRA*. IEEE, 2015, pp. 1316–1322.

[4] Di Guo, Tao Kong, Fuchun Sun, and Huaping Liu, "Object discovery and grasp detection with a shared convolutional neural network", in *ICRA*. IEEE, 2016, pp. 2038–2043.

[5] Lester Ingber, "Very fast simulated re-annealing", *Mathematical and computer modelling*, 1989.

[6] Yun Jiang, Marcus Lim, Changxi Zheng, and Ashutosh Saxena, "Learning to place new objects in a scene", *The International Journal of Robotics Research*, 2012.

[7] Yun Jiang, Marcus Lim, and Ashutosh Saxena, "Learning object arrangements in 3d scenes using human context", in *International Conference on Machine Learning*. 2012, ACM.

[8] Nichola Abdo, Cyrill Stachniss, Luciano Spinello, and Wolfram Burgard, "Robot, organize my shelves! tidying up objects by predicting user preferences", in *ICRA*. IEEE, 2015.

[9] Lap Fai Yu, Sai Kit Yeung, Chi Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley J Osher, "Make it home: automatic optimization of furniture arrangement", *ACM Transactions on Graphics*, 2011.

[10] Leslie Pack Kaelbling and Tomás Lozano-Pérez, "Hierarchical task and motion planning in the now", in *ICRA*. IEEE, 2011.

[11] Lavindra de Silva, Amit Kumar Pandey, and Rachid Alami, "An interface for interleaved symbolic-geometric planning and backtracking", in *IROS*. IEEE, 2013.

[12] Kutluhan Erol, James Hendler, and Dana S Nau, "Htn planning: Complexity and expressivity", in *AAAI*, 1994.

[13] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer", in *ICRA*. IEEE, 2014.

[14] Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour, "Manipulation planning among movable obstacles", in *ICRA*. IEEE, 2007, pp. 3327–3332.

[15] Athanasios Krontiris and Kostas E Bekris, "Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner", in *ICRA*. IEEE, 2016, pp. 3924–3931.

[16] Kris K Hauser, "Minimum constraint displacement motion planning.", in *Robotics: Science and Systems*, 2013.

[17] Christian Dornhege and Andreas Hertle, "Integrated symbolic planning in the tidyup-robot project.", in *AAAI Spring Symposium: Designing Intelligent Robots*, 2013.

[18] Mikkel Rath Pedersen and Volker Krüger, "Automated planning of industrial logistics on a skill-equipped robot", in *Workshop on Task Planning for Intelligent Robots in Service and Manufacturing*, 2015.

[19] Harold Szu and Ralph Hartley, "Fast simulated annealing", *Physics letters A*, 1987.

[20] Lester Ingber and Bruce Rosen, "Genetic algorithms and very fast simulated reannealing: A comparison", *Mathematical and computer modelling*, 1992.

[21] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins, "Pddl-the planning domain definition language", 1998.

[22] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization.", in *Robotics: science and systems*, 2013.

[23] Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan, "Example-based synthesis of 3d object arrangements", *ACM Transactions on Graphics*, 2012.

[24] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al., "Optimization by simulated annealing", *science*, 1983.