Sampling-Based Planning for Retrieving Near-Cylindrical Objects in Cluttered Scenes Using Hierarchical Graphs

Hao Tian, Chaoyang Song, *Senior Member, IEEE*, Changbo Wang, Xinyu Zhang, and Jia Pan, *Senior Member, IEEE*

***Abstract*—We present an incremental sampling-based task and motion planner for retrieving near-cylindrical objects, like bottle, in cluttered scenes, which computes a plan for removing obstacles to generate a collision-free motion of a robot to retrieve the target object. Our proposed planner uses a two-level hierarchy, including the first-level roadmap for the target object motion and the second-level retrieval graph for the entire robot motion, to aid in deciding the order and trajectory of object removal. We use an incremental expansion strategy to update the roadmap and retrieval graph from the collisions between the target object, the robot, and the obstacles, in order to optimize the object removal sequence. The performance of our method is highlighted in several benchmark scenes, including a fixed robotic arm in a cluttered scene with known obstacle locations and a scene, where locations of some objects or even the target object are unknown due to occlusions. Our method can also efficiently solve the high-dimensional planning problem of object retrieval using a mobile manipulator and be combined with the symbolic planner to plan complex multistep tasks. We deploy our method to a physical robot and integrate it with nonprehensile actions to improve operational efficiency. Compared to the state-of-the-art approaches, our method reduces task and motion planning time up to 24.6% with a higher success rate, and still provides a near-optimal plan.**

***Index Terms*—Manipulation planning, motion and path planning, object retrieval, task planning.**

Manuscript received 24 December 2021; revised 20 April 2022; accepted 16 June 2022. Date of publication 22 August 2022; date of current version 8 February 2023. This work was supported in part by the HKSAR Research Grants Council (RGC) General Research Fund (GRF) HKU under Grant 11202119 and Grant 11207818, and in part by the Innovation and Technology Commission of the HKSAR Government under the InnoHK initiative. The work of Chaoyang Song supported by the Shenzhen Long-term Support Program for Higher Education. This aritcle was recommended for publication by Associate Editor J. Kober and Editor W. Burgard upon evaluation of the reviewers' comments. *(Corresponding author: Jia Pan.)*

Hao Tian and Jia Pan are with the Department of Computer Science, University of Hong Kong, Hong Kong, and also with the Centre for Garment Production Limited, Hong Kong (e-mail: nicktian.ecnu@gmail.com; panjia1983@gmail.com).

Chaoyang Song is with the Department of Mechanical and Energy Engineering, Southern University of Science and Technology, Shenzhen 518055, China (e-mail: songcy@ieee.org).

Changbo Wang is with the School of Computer Science and Technology, East China Normal University, Shanghai 200050, China (e-mail: cbwang@sei.ecnu.edu.cn).

Xinyu Zhang is with the School of Software Engineering, East China Normal University, Shanghai 200050, China (e-mail: xyzhang@sei.ecnu.edu.cn).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TRO.2022.3191596.

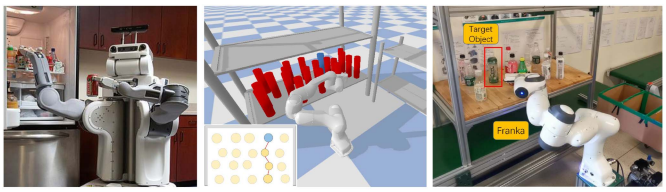
Digital Object Identifier 10.1109/TRO.2022.3191596

NOMENCLATURE

$\mathcal{O}$	The set of all objects (including the target object).
$\mathcal{O}_{\text{col}}$	The set of objects colliding with the robot.
O_g	The target object.
$\mathcal{M}$	The roadmap of the target object.
$\mathbf{E}$	The set of roadmap edges.
$\mathbf{N}$	The set of retrieval graph nodes.
$\mathbf{H}$	The set of retrieval graph edges.
$\mathcal{L}$	The exploration limit.
Π	The set of robot motions.
$\mathbf{x}_s$	The start configuration point of the target object.
$\mathbf{x}$	Roadmap node.
$\mathbf{n}_g$	The goal node of the retrieval graph containing $\mathbf{x}_g$.
$\mathbf{g}$	The set of preset grasps.
r_b	The radius of footprint circle of the object.
r_f	The thickness of the gripper finger.
$\mathcal{O}_r$	The sequence of objects to be removed for retrieving the target object.
$\mathcal{O}_{\text{new}}$	The set of new objects is found in the occlusion scenario.
O_i	Object i.
$\mathbf{X}$	The set of roadmap nodes.
$\mathcal{G}$	The retrieval graph.
$\mathbf{N}_r$	The sequence of retrieval graph nodes along the best path.
$\mathcal{R}$	The object removal sequence.
$\mathcal{K}$	The sampling limit.
T	The number of roadmap samples in each iteration.
$\mathbf{x}_g$	The goal configuration point of the target object.
$\mathbf{n}_s$	The start node of the retrieval graph containing $\mathbf{x}_s$
$\mathbf{n}$	Retrieval graph node.
$\mathbf{g}_j$	Grasp j.
r_g	The radius of the target object.
r_{base}	The half of the gripper base length.

I. INTRODUCTION

OBJECT retrieval in cluttered scenes is a challenging task and motion planning (TAMP) problem, e.g., the robot retrieves an item from the refrigerator in the kitchen or the shelves in the warehouse, as shown in Fig. 1(a). From the task perspective, the robot must determine which objects to remove in what order, even in challenging situations where the robot only has partial observation to the cluttered objects due to occlusions.



(a) (b) (c)

Fig. 1. (a) Retrieving the target object from the refrigerator in the kitchen [1]. (b) We propose a sampling-based TAMP method for the near-cylindrical object retrieval, where the object is represented by a compact cylinder. The robot must remove and relocate some red obstacles before retrieving the blue target object. (c) A physical experiment using a 7-DOF robot with our method. The target object is marked with a red box. In order to retrieve the target, the robot has to remove two objects that block the robotic motions.

From the motion perspective, the robot must determine how to manipulate objects in an optimal manner avoiding the collisions between the robot, the target object, and the obstacles. In this way, object retrieval in cluttered scenes inherits challenges from both motion and task planning.

Object retrieval can be formulated as a complex optimization problem whose objectives include minimizing the number of obstacles to be removed, optimizing the order of removing these obstacles, and computing the low-level collision-free trajectories accomplishing the manipulation and movement of different objects in a most efficient manner. Its exact solution must consider the complex spatial relations among the robot and all objects in the scene, and thus is of high computational complexity exponential to the number of objects in the scene. As a tractable solution, some recent works [2], [3] first preconstruct a graph to encode the complex spatial relations for all objects in the scene, though it needs to check the collisions between all object pairs and thus is time-consuming. Then, object retrieval task is approximately solved by planning over the graph. The graph will be updated during planning because invalid edges due to robot–obstacle collisions will be deleted and thus frequent replanning is necessary. If some critical edges important for the connectivity from the robot to the target are removed, the algorithm would fail to find a solution. A more tractable alternative is to greedily compute the minimum set of obstacles that block the robot from reaching and retrieving the target, i.e. by solving the minimum constraint removal (MCR) problem [4]. However, the robot may collide with more objects when actually removing one obstacle, resulting in another new retrieval task with the obstacle to be removed as the target. In this way, the object retrieval problem can be recursively decomposed into a hierarchy of object retrieval subproblems, but such greedy task decomposition may provide a performance significantly worse than the exact solution, making appropriate heuristics necessary.

In this work, we propose a sampling-based TAMP algorithm to retrieve the target object in a cluttered scene, as shown in Fig. 1(b) and (c). Our algorithm mainly handles near-cylindrical objects in cluttered scenes and provides a tractable and efficient way to minimize the number of obstacles to be removed for clearing the way and retrieving the target object. In task planning stage, we use an incremental sampling-based method to construct a hierarchy consisting of the roadmap and the retrieval graph to describe the obstacles that block the way to reach the target object. The task planning results are a sequence of obstacles to be removed from the scene. In motion planning stage, we perform feasibility checking, including grasp planning and motion planning for pick-and-place actions, to determine whether the obstacles in the task plan can be manipulated by the robot. If the motion planning fails due to robot–obstacle collisions, the collision obstacles will be given as feedback to the task planner and a new task plan will be computed.

Our framework mainly focuses on generating the sequence of removal obstacles with valid pick-and-place actions. Although nonprehensile actions such as pushing can sometimes be more efficient than prehensile pick-and-place actions, the cluttered scenes may not leave enough space for nonprehensile actions to relocate objects. Nevertheless, our framework is general and if the removal object and its goal position can be reached by nonprehensile actions, it can also leverage nonprehensile actions for accomplishing object retrieval.

Our contributions include the following.

1) We propose a two-level hierarchy that records target object–obstacle collisions and robot–obstacle collisions to aid in determining the order and the subtasks of object removal in cluttered scenes.
2) We propose a task planner that uses an incremental expansion strategy with interleaved updates of the roadmap and the retrieval graph to find a sequence of removal objects, which can update the infeasible sequence by collecting failures from motion planning.
3) We show the probabilistically completeness of our incremental strategy to find removal object sequence and polynomial time complexity of roadmap expansion and retrieval graph update.
4) We extend our algorithm to handle complex tasks, such as the case when the target object is unknown due to occlusions, mobile manipulation, and the combination with symbolic planners. Our algorithm can be easily integrated with nonprehensile actions such as pushing to further improve operational efficiency.

The rest of this article is organized as follows. In Section II, we provide a review of the related work. In Section III, we formulate the object retrieval problem. In Section IV, we first introduce the overview of our algorithm and then describe how to construct the roadmap and retrieval graph to help compute the task plan. Section V describes the extension of our algorithm to handle complex tasks. Section VI describes the implementation details and experimental results of our algorithm on different tasks. We demonstrate the performance of our method in both simulation and physical environments, from simple retrieval tasks to complex mobile manipulation.

II. Related Work

In this section, we give a brief overview of prior work related to object retrieval in cluttered scenes.

A. Manipulation Planning Among Movable Obstacles

Manipulation planning among movable obstacles (MAMO) using prehensile actions has been studied extensively in robotics [5], [6], which has been proved to be NP-hard even for simple

TABLE I
PLANNING PERFORMANCE OF OUR PLANNER. WE TEST SCENARIOS WITH DIFFERENT NUMBERS OF OBJECTS, AND WE TEST EXPLORATION LIMIT $\mathcal{L}$ WITH VARIOUS INITIAL VALUES

Robot	Measure	$\lvert\mathcal{O}\rvert$=12			$\lvert\mathcal{O}\rvert$=16			$\lvert\mathcal{O}\rvert$=20		
		$\mathcal{L}$=1	$\mathcal{L}=\lvert\mathbf{n}_g.\text{NodeCol}\rvert$	$\mathcal{L}$=$\lvert\mathcal{O}\rvert$	$\mathcal{L}$=1	$\mathcal{L}=\lvert\mathbf{n}_g.\text{NodeCol}\rvert$	$\mathcal{L}$=$\lvert\mathcal{O}\rvert$	$\mathcal{L}$=1	$\mathcal{L}=\lvert\mathbf{n}_g.\text{NodeCol}\rvert$	$\mathcal{L}$=$\lvert\mathcal{O}\rvert$
UR10	$\lvert\mathcal{O}_r\rvert$	2.14(0.34)	2.35(0.47)	2.36(1.08)	2.81(0.52)	3.15(0.53)	3.35(1.0)	3.28(0.58)	3.57(0.62)	4.25(0.82)
	Time (s)	10.54(5.89)	9.85(4.91)	8.95(5.74)	20.50(6.71)	16.25(6.0)	21.43(7.50)	21.73(7.13)	17.63(7.18)	15.73(7.38)
	# Iteration	1.78(0.67)	1.5(0.5)	1(0)	3.18(0.72)	1.76(0.97)	1(0)	3.28(0.83)	1.92(0.45)	1(0)
	Success Rate (%)	100	100	100	95	100	100	95	95	100

The numbers in parentheses are standard deviations. The number of removed objects including the target object.

cases with cubic obstacles or when all the obstacles are fixed and observable [7]–[9]. Most previous works aimed at designing a complete planner or improving its efficiency for some special cases of MAMO [10]–[12]. A few probabilistically complete solutions are available [6], [10], [13], but they are limited to 2-D/3-D navigation due to their explicit representation or discretization of the robot's configuration space [10], [13]. When the obstacles are removable, there exist ad-hoc solutions for manipulation planning in cluttered scenes [14]–[17], but they do not consider the optimality in terms of the number of objects to be removed and the computational complexity. Some recent works deal with object rearrangement problems [18]–[21]. In some cases, each object needs to be moved only once (i.e., monotone), and in other cases, some objects may need to be moved multiple times (i.e., nonmonotonic).

Some recent works [2], [3], [22] solve the MAMO problem by preconstructing a traversability graph to encode the complex spatial relations for all objects in the scene. A shortest path on the graph from the robot node to the target node provides the smallest set of objects that needs to be removed. The graph construction step is time-consuming due to the collision checking between all object pairs, especially when the number of objects is large. The graph structure will be updated during planning when edges invalidated by robot–obstacle collisions are deleted from the graph and replanning over the updated graph is performed. However, replanning does not consider important information like which specific obstacle blocks robot's motion. The planner may search multiple times to find the successful path, so it takes a lot of time to perform motion planning. The representation and usage of such details are one main contribution of our method. In particular, we use incremental sampling to explore the cluttered scene and construct a retrieval graph that represents the collision states of the target object, rather than a graph with all obstacles involved. In this way, the computational complexity of our algorithm is independent with the number of objects in the scene. During planning, the robot–obstacle collisions would be added into the retrieval graph as new nodes, rather than simply deleting invalid edges. With such collision information encoded, replanning on the retrieval graph is able to minimize the number of removal objects more efficiently.

Most previous MAMO works focused on using prehensile actions, which require a firm grasp of an object and thus are difficult to achieve when objects are close to each other. Some recent works leveraged nonprehensile actions to rearrange objects for reaching the target object [14], [17], [23]–[25]. For instance, [14] proposed a planning framework that utilizes nonprehensile actions to grasp an object in the clutter. To determine the set of obstacles to be removed given a planned trajectory, this method first computes the shortest distances between the obstacles and the end-effector when the robot follows the planning result, and then all obstacles collide with the end-effector will be removed. This strategy is greedy and thus may take redundant or even incorrect removal actions. In this way, rearrangement can be accomplished via simple nonprehensile actions (e.g., pushing [26], [27]), but it is difficult to predict where and how much an object will move after or during pushing due to the contacts and frictions between the gripper, objects, and table. Such difficulty can be handled via reinforcement learning or physics based simulation [28]–[30].

B. Task and Motion Planning

MAMO can also be solved as a special case of the TAMP [16], [31], [32]. Many TAMP approaches employ high-level symbolic planning along with low-level motion planning for computing a sequence of continuous robot motions. However, TAMP frameworks could be inefficient for manipulation tasks in cluttered scenarios. First, the task planner may search in a huge state space, especially when the number of objects in the clutter is large. Second, in a dense scene, the low-level motion planning of robot motions could fail frequently, the task planners have to iterate over symbolic plans in a brute force manner whenever the motion planning fails [33]. To resolve this issue, some previous works introduced geometric backtracking, in which the planner will search for different geometric instances until a solution is found [34], [35]. To reduce the search space, they proposed to maintain the interval constraints over placement parameters. Hierarchical task networks (HTNs) have also been used for symbolic search where HTNs backtracking is applied over choices made by the geometric module, allowing more freedom to the geometric planning [36], [37]. Besides sampling-based motion planners, nonlinear optimization has also been used for solving TAMP problems by identifying geometric parameters on the geometric level where all geometric parameters are considered simultaneously without backtracking [38], [39]. Learning techniques [40]–[42] have been applied to the TAMP framework to accelerate inference and replace the handcrafted search heuristics. Their goal is to guide the search procedure toward a more promising task plan and reduce the number of motion planning problems to be solved. Recently, [43] and [44] proposed a classifier to evaluate the feasibility of motion planning generated by discrete decisions in the task domain. However, the limitation of their methods is that only one single action is predicted, but the combinatorial complexity of TAMP especially arises from action sequences and it is not easy to use such a classifier to predict the action sequence in TAMP.

TAMP combining a symbolic task planner suffers from the huge planning graph in a densely cluttered scene, which leads to the high complexity of computing the removal and manipulation actions of all possible objects. Our method is not symbolic but our method could help symbolic task planners focus on the important objects and reduce the search space.

C. Minimum Constraint Removal

Determining the smallest set of obstacles to be removed for the target retrieval can also boil down to the MCR problem, which minimizes the number of violated constraints or obstacles to be removed along a planned path [4], [45], [46]. The problem has been proven to be NP-hard in discrete cases and some greedy and asymptotically optimal algorithms were proposed in [4], which however does not consider the detailed interaction between the robot and the obstacles when actually executing the removal operation. Our method provides a tractable way to generate obstacle removal sequences with intermediate pick-and-place actions. We use a two-level hierarchical graph to collect collision feedback when the robot fails to grasp the object and efficiently replan from failure to minimize the number of removal obstacles. The roadmap graph is the first level of our two-level hierarchical graph. The roadmap expansion incrementally grows the roadmap and collect target object–obstacle collisions to find initial object retrieval sequences. If the initial retrieval sequence obtained from the roadmap graph is valid, which means the robot can successfully retrieve all objects in the sequence, the task is completed. However, if there are robot–obstacle collisions when the robot grasps the object, we use the second retrieval graph to collect these collision feedbacks and efficiently replan from failures.

III. Problem Formulation

Suppose there are m near-cylindrical objects in a cluttered scene (see the example scenarios shown in Fig. 1), including one target object O_g and $m-1$ removable obstacles $\mathcal{O} = \{O_1, \ldots, O_{m-1}, O_g\}$, and there are no collisions between these objects. Our goal is to retrieve the target object O_g while avoiding any collisions. However, the objects located in the front of the clutter will obstruct the robot to retrieve objects behind in the cluttered scene, which means that the robot cannot move over the object to grasp other objects. Thus, to accomplish the task, the robot needs to determine first which objects need to remove and then an optimal removal order for manipulating them one by one. Formally, we denote $\mathcal{O}_r$ as sequence of objects to be removed from the cluttered scene. Our goal is to retrieve the target object O_g, while minimizing the number of objects in $\mathcal{O}_r$

$$
\begin{aligned}
&\min \quad |\mathcal{O}_r| \\
&s.t. W(O_i, \mathrm{g}_j) \cap (\mathcal{O}/\mathcal{O}^i_{\text{removed}}) = \emptyset \qquad \forall O_i \in \mathcal{O}_r \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \exists \mathrm{g}_j \in \mathbf{g} \\
&\mathcal{O}_r \subseteq \mathcal{O} \\
&O_g \in \mathcal{O}_r \\
&\mathcal{O}^i_{\text{removed}} = \{O_1, \ldots, O_{i-1}\} \subseteq \mathcal{O}_r
\end{aligned} \tag{1}
$$

where $W(O_i, \mathrm{g}_j)$ is the swept volume of the object O_i when it is being grasped by grasp g_j and moved along with the manipulation motion of the robot, $\mathbf{g}$ represents the preset grasps, $\mathcal{O}^i_{\text{removed}} = \{O_1, \ldots, O_{i-1}\}$ contains objects that are previously removed before O_i. Here, we have several assumptions as follows:

1) We assume the objects around the target object prevent the robot from generating collision-free motions to reach and retrieve the target, which is the key challenge for our task.
2) We assume the robot cannot perform overhand grasping due to the restriction of the shelves.
3) We assume all sampling object poses are reachable by the robot.
4) We assume that the objects in $\mathcal{O}_r$ removed before O_i will not affect the manipulation of O_i, i.e., each object's interaction feasibility is monotonic within the constraint relationships of objects in the cluttered scene. As a result, we can exclude objects $\mathcal{O}^i_{\text{removed}}$ from checking collisions with $W(O_i, \mathrm{g}_j)$. This is reasonable in practice, for instance by placing these removed objects in an area far from the cluttered scene and the released free space will increase the success rate of feasible interaction.

The optimization problem described in (1) is very complicated and is NP-hard. Solving this problem requires not only optimizing the order of object retrieval, but also considering robot constraints and computing the robot grasp poses and motions. Thus, we simplify the problem by only considering near-cylindrical objects and also leveraging the objects' monotonic interaction feasibility to decouple the problem into two stages as shown in Fig. 2: 1) find an ordering sequence of remove objects in task planning stage; 2) find a feasible manipulation of each object in the sequence in motion planning stage.

We denote a pose of the target object in an open space outside the cluttered scene as the start configuration $\mathbf{x}_s$ and the initial pose of the target object inside the cluttered scene as the goal configuration $\mathbf{x}_g$. At first glance, this nomenclature may look a bit strange because the target object is being retrieved from the goal $\mathbf{x}_g$ to the start point $\mathbf{x}_s$. Our choice lies in the fact that when removing objects that block the way, the end-effector of the robot will start from objects on the periphery (i.e., near $\mathbf{x}_s$) and finally reach the target object hiding inside the cluttered scene (i.e., at $\mathbf{x}_g$). Setting $\mathbf{x}_s$ as the start point enables the motion planner to return a path from the scene periphery to the target object's position, from which we can conveniently generate a object removal sequence. In addition, the exchange of the start and the goal configurations will only reverse the path direction without changing the path's geometry.

IV. Incremental Sampling-Based TAMP for Object Retrieval

Our sampling-based TAMP framework is shown in Fig. 2, with the left and right parts for task planning and motion planning, respectively. In particular, our task planner uses an incremental sampling strategy to construct a roadmap $\mathcal{M}(\mathbf{X}, \mathbf{E})$ for the target object attached to the robot's end-effector, where $\mathbf{X}$ and $\mathbf{E}$ are the sets of roadmap nodes and edges, respectively. By searching on $\mathcal{M}$, we can find a path

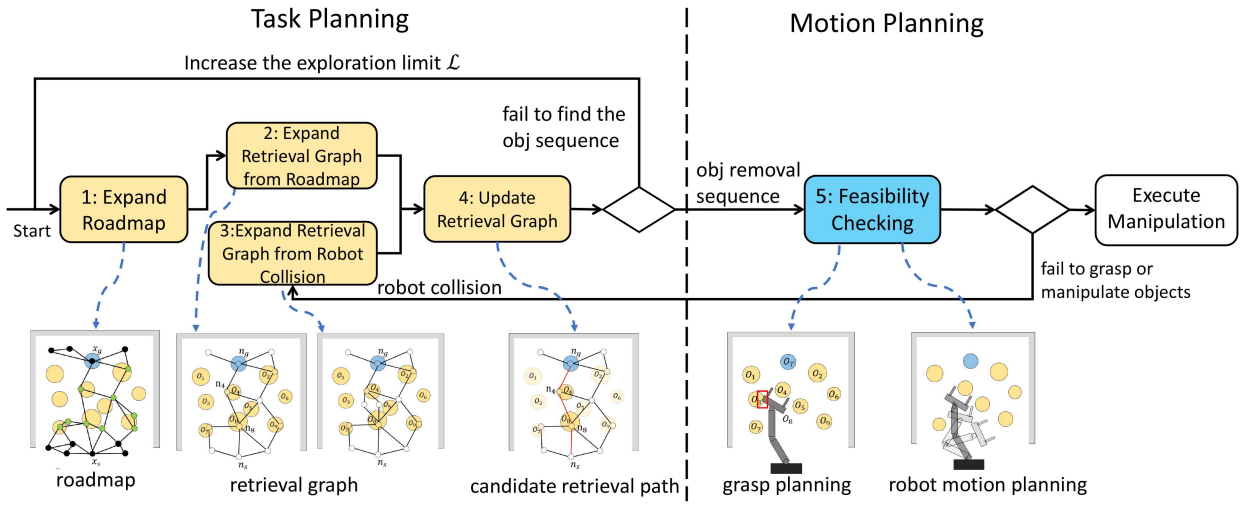


Fig. 2. Pipeline of our algorithm. Task planning: We use an interleaved update strategy to incrementally (1) expand the roadmap and (2) update its corresponding retrieval graph. Based on the retrieval graph, the task plan is computed by finding a retrieval path reaching the target with an allowed number of collision obstacles to be removed (4). Motion planning: We check whether the robot can compute collision-free motions to reach and manipulate these candidate removal objects (5). If motion planning fails, the robot collisions will be feedback to the task planner to update the task plan (3).

connecting $\mathbf{x}_s$ and $\mathbf{x}_g$ with the minimum number of collision objects and can generate a sequence of objects to be removed.

However, just removing this sequence of objects is not sufficient for successful target object retrieval, because the aforementioned planning does not consider the robot components besides the end-effector. Thus, motion planning follows to check if the plan is executable by the robot. If not, the collision events involving the robot must be collected and feedback to the task planner for updating the sequence of objects to be removed. To enable geometric reasoning of task planning in the discrete decision space of objects, our task planner constructs a retrieval graph $\mathcal{G}(\mathbf{N}, \mathbf{H})$ over the roadmap $\mathcal{M}$, where $\mathbf{N}$ and $\mathbf{H}$ are the sets of retrieval graph nodes and edges, to record the objects that collide with the entire robot and the order in which collisions occur. By searching a path from the start node to the goal node on $\mathcal{G}$, we can obtain another sequence of objects to be removed, including objects colliding with the entire robot and the attached target object, as the task plan. The details of this TAMP solver are discussed below.

A. Overview: Incremental Sampling-Based TAMP Using Interleaved Updates

As shown in Algorithm 1, our sampling-based planner uses an incremental expansion strategy with interleaved updates. In particular, we first initialize the roadmap $\mathcal{M}$ with two isolated nodes $\{\mathbf{x}_s, \mathbf{x}_g\}$, which are the start and goal configurations of the target object. Then, we perform an incremental expansion that interleaves the update of $\mathcal{M}$ and $\mathcal{G}$ to find the object removal sequence as well as check whether the robot could remove objects in the sequence and retrieve the target object. This incremental procedure repeats until the target object O_g is retrieved. This process includes several main subroutines:

1) *Roadmap Expansion:* Our task planner expands $\mathcal{M}$ by incrementally sampling in the configuration space of the target object attached to the robot's end-effector to explore the cluttered environment around the target object. Since a collision-free path is not possible without removing some obstacles, the expansion procedure will also explore the

Algorithm 1: PLANNINGWITHINTERLEAVEDUPDATES.

Input: target object start configuration $\mathbf{x}_s$, target object goal configuration $\mathbf{x}_g$, objects $\mathcal{O}$, target object O_g, robot rob, preset grasps $\mathbf{g}$
Output: object removal sequence $\mathcal{R} = \{\mathcal{O}_r, \mathbf{N}_r\}$, the set of robot motions Π

/*Initialize the roadmap and retrieval graph*/
$\mathcal{M}(\mathbf{X}, \mathbf{E}) \leftarrow (\{\mathbf{x}_s, \mathbf{x}_g\}, \emptyset)$
$\mathcal{G}(\mathbf{N}, \mathbf{H}) \leftarrow (\emptyset, \emptyset)$
/*Incremental expansion with interleaved updates*/
$\mathcal{L} = 0$ ▷ exploration limit
while O_g is not retrieved **do**
  /*Roadmap expansion*/
  $\mathcal{M} \leftarrow$ EXPANDROADMAP$(\mathbf{x}_s, \mathbf{x}_g, \mathcal{M}, \mathcal{L})$
  /*Retrieval graph expansion and update*/
  $\mathcal{G}, \{\mathbf{n}_{\text{new}}\} \leftarrow$ EXPANDGRAPHFROMROADMAP$(\mathcal{G}, \mathcal{M})$
  for $\forall \mathbf{n}'_{\text{new}} \in \{\mathbf{n}_{\text{new}}\}$ **do**
    $\mathcal{G}, \mathcal{R} \leftarrow$ UPDATERETRIEVALGRAPH$(\mathbf{n}'_{\text{new}}, \mathcal{G})$
  end for
  /*Feasibility checking*/
  $\Pi \leftarrow$ FEASIBILITYCHECKING$(\mathcal{G}, O_g, rob, \mathcal{O}, \mathcal{R}, \mathcal{L}, \mathbf{g})$
  $\mathcal{L} \mathrel{+}= 1$
  if Π is not Empty **then**
    break
  end if
end while
return $\mathcal{R}, \Pi$

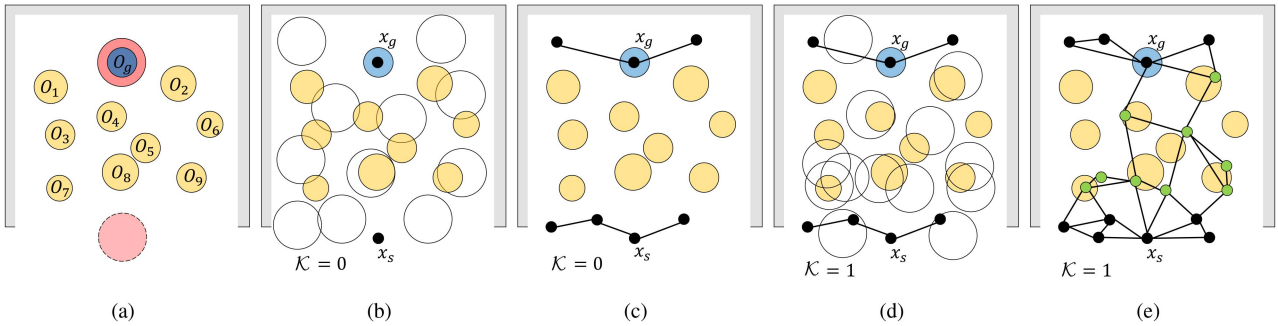


Fig. 3. Example of roadmap expansion. (a) The initial setup of all objects in the scene. The yellow disks are obstacles and the blue disk is the target object O_g. The red disk represents the footprint of the target object plus the gripper geometry, where the solid disk is for the goal configuration $\mathbf{x}_g$ in the cluttered region and the dashed disk is for an assumed start configuration $\mathbf{x}_s$ in the open space. The entire scene is bounded using shelves on three sides. (b)–(c) initialize the roadmap by generating samples with the sampling limit $\mathcal{K} = 0$, where the black circles represent the geometry shapes for these samples and the black dots represent the collision-free samples. (d)–(e) incrementally expand roadmap by sampling with the sampling limit $\mathcal{K} = 1$. The green dots are the samples that collide with one object.

in-collision region in the configuration space by allowing each sample to have collisions with a limited number of obstacles. Details about the expansion are given in Section IV-B.

2) *Retrieval Graph Expansion and Update:* The retrieval graph $\mathcal{G}$ is constructed over the roadmap $\mathcal{M}$ and will be expanded in two cases: when the roadmap expands and during the feasibility checking. After the roadmap expansion, we expand $\mathcal{G}$ by clustering the roadmap nodes that collide with the same set of obstacles and setting the clusters as $\mathcal{G}$ nodes. Then we collect sets of obstacles as the target object–obstacle collisions and add them to $\mathcal{G}$ nodes. $\mathcal{G}$ also expands during feasibility checking. We create new nodes for the robot-obstacle collisions that include the obstacles colliding with the robot and connect new nodes to $\mathcal{G}$ nodes where the feasibility checking fails. After adding new nodes, we update the retrieval graph to compute the object removal sequence $\mathcal{R} = \{\mathcal{O}_r, \mathbf{N}_r\}$, where $\mathcal{O}_r$ is the sequence of objects to be removed and $\mathbf{N}_r$ is the sequence of retrieval graph nodes along the best path from the start node to the goal node. The details of $\mathcal{G}$ and how to update $\mathcal{G}$ are described in Section IV-C.

3) *Feasibility Checking:* After obtaining the object removal sequence $\mathcal{R}$, our motion planner performs feasibility checking, which includes grasp planning and motion planning, to check if objects in $\mathcal{O}_r$ could be removed sequentially by the robot. If the checking fails and the robot collides with some other obstacles, we collect these robot–obstacle collisions into the new retrieval graph node and add it to $\mathcal{G}$. The task planner follows to update $\mathcal{G}$ to search for a new task plan in terms of a new object removal sequence. Details about the feasibility checking are given in Section IV-D.

The incremental expansion is controlled with an exploration limit $\mathcal{L}$, which allows each sample in the roadmap to collide with at most $\mathcal{L}$ objects. In particular, if the task planner cannot find a path connecting $\mathbf{x}_s$ and $\mathbf{x}_g$ on the current graph, it will increase $\mathcal{L}$ by one to explore more regions in the configuration space corresponding to more cluttered areas in the workspace. The increment of $\mathcal{L}$ continues until a path connecting $\mathbf{x}_s$ and $\mathbf{x}_g$ is found. We find it is important to start from low $\mathcal{L}$ (e.g., $\mathcal{L} = 0$) because if we start from large $\mathcal{L}$, the relatively open regions that are reachable with low $\mathcal{L}$ will be undersampled, which will negatively affect the task planner's ability to explore the cluttered regions requiring higher $\mathcal{L}$.

B. Roadmap Expansion

We use an example scene consisting of 2-D disk objects as illustrated in Fig. 3 to explain the roadmap expansion. We use 2-D disks to represent the objects in the scene because we use 3-D cylinders to represent the near-cylindrical objects. The size of the cylinder of the target object is determined by the target object plus the gripper geometry to reflect the fact that the object can move only being attached to a gripper, as shown in Fig. 3(a). This cylinder representation of the target object is conservative, which can find the potential collision samples, where the gripper collides with the obstacles when manipulating the object. We construct the roadmap $\mathcal{M}$ by sampling x and y coordinates of the target object within the cluttered space, which is a subset of R^2, as shown in Fig. 3(b). The sampling process is controlled by the sampling limit $\mathcal{K}$, which restricts each generated sample to collide with at most $\mathcal{K}$ objects. We gradually increase $\mathcal{K}$ to balance the number of samples in different regions of the configuration space where the target object collides with different numbers of objects. In particular, we initialize $\mathcal{K}$ to be 0 and update $\mathcal{K}$ after every T_{raise} steps using an incremental raising strategy as proposed in [4]: $\mathcal{K} \mathrel{+}= \dfrac{\mathcal{L}}{(T - T_{\text{cur}})/(T_{\text{raise}})}$, where T is the total number of sampling iterations and T_{cur} is the current sampling iteration index. For instance, Fig. 3(c) shows a roadmap with $\mathcal{K} = 0$. By increasing $\mathcal{K}$ to be greater than 0, we can explore the in-collision regions in the configuration space, as shown in Fig. 3(d) and (e), where the green nodes are samples that collide with a single object. Since the roadmap nodes can collide with the objects, we denote $\text{Col}(\mathbf{x}_i)$ and $\text{EdgeCol}(\mathbf{x}_{i-1}, \mathbf{x}_i)$ as the sets of objects colliding with the target object when it is at node $\mathbf{x}_i$ and moves along the edge between nodes $\mathbf{x}_{i-1}$ and $\mathbf{x}_i$, respectively. These two sets can be computed efficiently using collision checking and continuous collision checking routines.

Algorithm 2: EXPANDROADMAP.

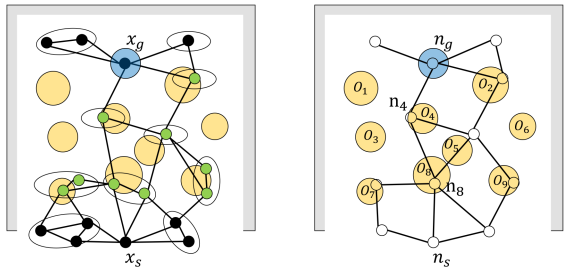
Input: target object start configuration $\mathbf{x}_s$, target object goal configuration $\mathbf{x}_g$, roadmap $\mathcal{M}$, exploration limit $\mathcal{L}$
Output: expanded roadmap $\mathcal{M}$

1: $\mathcal{K} \leftarrow 0$ ▷ roadmap sampling limit
2: **for** $T_{\text{cur}} \leftarrow 1 \ldots T$ **do**
3: $\mathbf{x}_{\text{rand}} \leftarrow$ SAMPLE()
4: $\mathbf{x}_{\text{nearest}} \leftarrow$ NEAREST($\mathcal{M}, \mathbf{x}_{\text{rand}}, \mathcal{K}$)
5: $\mathbf{x}_{\text{new}} \leftarrow$ EXTEND($\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{rand}}, \delta, \mathcal{K}$) ▷ maximum distance δ of the new edge
6: $\mathcal{M}.\mathbf{X} \leftarrow \mathcal{M}.\mathbf{X} \cup \{\mathbf{x}_{\text{new}}\}$
7: $\mathbf{X}_{\text{near}} \leftarrow$ NEAR($\mathcal{M}, \mathbf{x}_{\text{new}}$)
8: **for** $\forall \mathbf{x}_i \in \mathbf{X}_{\text{near}}$ **do**
9: **if** $|\text{EdgeCol}(\mathbf{x}_{\text{new}}, \mathbf{x}_i)| = |\text{Col}(\mathbf{x}_{\text{new}}) \cup \text{Col}(\mathbf{x}_i)|$ and $d(\mathbf{x}_{\text{new}}, \mathbf{x}_i) < \delta$ **then**
10: $\mathcal{M}.\mathbf{E} \leftarrow \mathcal{M}.\mathbf{E} \cup \{(\mathbf{x}_i, \mathbf{x}_{\text{new}})\}$
11: **end if**
12: **end for**
13: Every T_{raise} steps, raise $\mathcal{K}$ ▷ adjust roadmap sampling limit
14: **if** $\mathcal{K} \geqslant \mathcal{L}$ **then**
15: $\mathcal{K} \leftarrow \mathcal{L} - 1$
16: **end if**
17: **end for**
18: **return** $\mathcal{M}$

Algorithm 2 describes how the roadmap is expanded. We first generate a random sample $\mathbf{x}_{\text{rand}}$ and find the nearest node $\mathbf{x}_{\text{nearest}}$ that satisfies the sampling limit $\mathcal{K}$ (Line 3–4). We generate samples and compute the nearest neighbor in the (x, y) coordinate system without considering the orientation of samples, because we aim to compute, which obstacles collide with the target object when the target object is retrieved from the clutter, rather than computing accurate motions of the target object. The conservative cylinder representation contains different orientations of the target object when performing collision detection, so we only need to consider the position of samples. We use an RRT-like operation $Extend(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{rand}}, \delta, \mathcal{K})$ to extend an edge from $\mathbf{x}_{\text{nearest}}$ to a node $\mathbf{x}_{\text{new}}$ in the direction of $\mathbf{x}_{\text{rand}}$ (Line 5–6). The length of the new edge is limited by the maximum distance δ, i.e.,

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{nearest}} + \min\left(\frac{\delta}{d(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{rand}})}, 1\right)(\mathbf{x}_{\text{rand}} - \mathbf{x}_{\text{nearest}}).$$

We require the number of objects colliding with the target object along the edge connecting $\mathbf{x}_{\text{nearest}}$ and $\mathbf{x}_{\text{new}}$ to be no more than $\mathcal{K}$, i.e., $|\text{EdgeCol}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})| \leq \mathcal{K}$. If this requirement is not met, we perform bisection search on the edge connecting $\mathbf{x}_{\text{new}}$ and $\mathbf{x}_{\text{nearest}}$ until it is satisfied. Finally, we connect $\mathbf{x}_{\text{new}}$ to its k nearest neighbor nodes $\mathbf{X}_{\text{near}}$ (Line 7–12). For each neighbor node $\mathbf{x}_i \in \mathbf{X}_{\text{near}}$, we check if the edge distance between $\mathbf{x}_i$ and $\mathbf{x}_{\text{new}}$ is shorter than δ and whether $|\text{Col}(\mathbf{x}_{\text{new}}) \cup \text{Col}(\mathbf{x}_i)| = |\text{EdgeCol}(\mathbf{x}_{\text{new}}, \mathbf{x}_i)|$. If the latter condition holds, it implies that the edge would neither directly pass through other objects nor collide with objects that do not belong to the collision set of these two vertex nodes (Line 9). If the edge connecting $\mathbf{x}_{\text{new}}$ to $\mathbf{x}_i$ passes both checks above, the edge will be added to $\mathcal{M}.E$ (Line 10).



(a) (b)

Fig. 4. Retrieval graph construction via clustering. The clusters are illustrated as ellipses. $\mathbf{n}_s$ and $\mathbf{n}_g$ are retrieval graph nodes that contain the target object start configuration $\mathbf{x}_s$ and goal configuration $\mathbf{x}_g$, respectively, where $\mathbf{n}_s$.ClusterNodes $= \{\mathbf{x}_s\}$ and $\mathbf{n}_g$.ClusterNodes $= \{\mathbf{x}_g\}$; $\mathbf{n}_4$ and $\mathbf{n}_8$ are clustered from the roadmap nodes that collide with O_4 and O_8, respectively, where $\mathbf{n}_4$.NodeCol $= \{O_4\}$ and $\mathbf{n}_8$.NodeCol $= \{O_8\}$.

The roadmap expansion process continues until we can find a path connecting $\mathbf{x}_s$ and $\mathbf{x}_g$ on the resulting roadmap $\mathcal{M}$. When traversing this path, we can collect a sequence of objects colliding with the target object, which will make the candidate task planning result that includes the object removal sequence.

C. Retrieval Graph Expansion and Update

1) Retrieval Graph: Since we only take into account the robot's end-effector when operating the roadmap $\mathcal{M}$, the robot may collide with some more obstacles when actually executing the removal of the object sequence computed above. After being aware of such new robot–obstacle collisions, the task planner should update the plan accordingly. One straightforward solution is to inform each roadmap node in $\mathcal{M}.\mathbf{X}$, which extra objects need to be removed to make itself collision-free. However, it is not efficient to store such information in all the nodes because nearby nodes in the roadmap usually share the same set of robot–obstacle collisions and thus there exists huge storage redundancy if we did that, especially for complex scenarios with a large roadmap.

To reduce such redundancy and improve graph search efficiency, we propose a higher-level data structure called the retrieval graph $\mathcal{G}$, which is constructed by clustering the roadmap nodes that are connected with each other and collide with the same set of obstacles, as shown in Fig. 4. Each cluster is treated as a retrieval graph node and added to $\mathcal{G}.\mathbf{N}$, with nodes $\mathbf{n}_s$ and $\mathbf{n}_g$ for the start and goal of the retrieval graph, respectively. The retrieval graph node $\mathbf{n}$ is a tuple of four items $\mathbf{n} = \langle \text{ClusterNodes}, \text{NodeCol}, \text{PathCol}, \text{BestNeighbor} \rangle$. ClusterNodes is the set of roadmap nodes that are clustered into $\mathbf{n}$.NodeCol is the set of obstacles colliding with the target object or the robot, which is shared for all roadmap nodes in ClusterNodes. For each path connecting the start node $\mathbf{n}_s$ and $\mathbf{n}$, we can define the path's set of collision obstacles as the union of NodeCol sets for all nodes on that path. PathCol contains the minimum collision obstacle set from $\mathbf{n}_s$ to $\mathbf{n}$ along the path

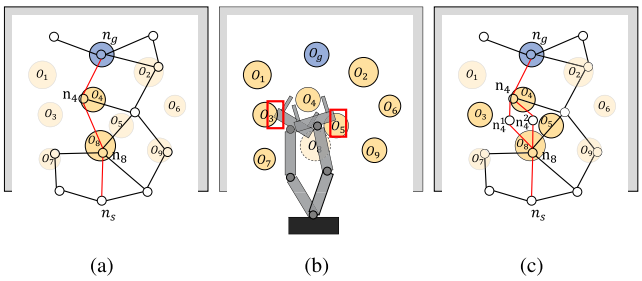


Fig. 5. Retrieval graph expansion by collecting feasibility checking failures. (a) The red line represents the best path with the minimum number of collision obstacles from the start node to the goal node. (b) Assume the object O_8 has been removed, but the feasibility checking of O_4 fails because the robot collides with O_3 and O_5. (c) New retrieval graph nodes $\mathbf{n}_4^1$ and $\mathbf{n}_4^2$ are created with the robot-obstacle collision sets $\{O_3\}$ and $\{O_5\}$, and new nodes are added to the retrieval graph. The red lines are new candidate paths from the start node to the goal node.

computed by the greedy search. The topology of this path is maintained in BestNeighbor, which is the label of $\mathbf{n}$'s neighbor node in the path connecting $\mathbf{n}$ and $\mathbf{n}_s$. More discussion about these PathCol and BestNeighbor are available in Section IV-C3.

The edges in the retrieval graph are added based on the connections between the roadmap nodes, where if two roadmap nodes are connected with each other and clustered in two retrieval graph nodes, we connect these two retrieval graph nodes and add this edge to $\mathcal{G}.\mathbf{H}$. By adding clusters to $\mathcal{G}.\mathbf{N}$ and adding new edges to $\mathcal{G}.\mathbf{H}$, we obtain the retrieval graph $\mathcal{G}$.

2) Retrieval Graph Expansion: Since the retrieval graph $\mathcal{G}$ is constructed over the roadmap $\mathcal{M}$, we need to expand $\mathcal{G}$ based on the new $\mathcal{M}$ after the roadmap expansion. Our task planner calls EXPANDGRAPHFROMROADMAP in Algorithm 1 to cluster the roadmap nodes using the same clustering operation as introduced above and obtain new clustering nodes set $\{\mathbf{n}_{\text{new}}\}$. $\mathcal{G}$ is then expanded by adding these new clustering nodes to $\mathcal{G}.\mathbf{N}$ and their connected edges to $\mathcal{G}.\mathbf{H}$.

The retrieval graph nodes clustered from $\mathcal{M}$ only record the set of obstacles colliding with the target object and the robot's end-effector, but $\mathcal{G}$ also needs to maintain the set of obstacles colliding with the entire robot. As a result, besides expanding the retrieval graph $\mathcal{G}$ from the clustering roadmap nodes, our task planner expands $\mathcal{G}$ when new collisions between the robot and the obstacles are reported by the motion planner, i.e., when the feasibility checking fails. For example, suppose we find a path from $\mathbf{n}_s$ to $\mathbf{n}_g$ that go through $\mathbf{n}_8$ and $\mathbf{n}_4$, where $\mathbf{n}_8$.NodeCol $= \{O_8\}$ and $\mathbf{n}_4$.NodeCol $= \{O_4\}$, as shown in Fig. 5(a) with the red line. Suppose O_8 has been removed and the robot tries to manipulate O_4. Here, the feasibility checking of O_4 fails because two grasp poses of the robot collide with O_3 and O_5, respectively, as shown in Fig. 5(b). To collect these robot–obstacle collisions, we create two new retrieval graph nodes, $\mathbf{n}_4^1$ and $\mathbf{n}_4^2$, and assign $\mathbf{n}_4^1$.NodeCol $= \{O_3\}$ and $\mathbf{n}_4^2$.NodeCol $= \{O_5\}$. Since the above failure occurs when we check the feasibility of removing objects contained in $\mathbf{n}_4$.NodeCol after removing objects in $\mathbf{n}_8$.NodeCol, it implies that the edge between $\mathbf{n}_8$ and $\mathbf{n}_4$ is not feasible and it is necessary to remove the objects colliding with the robot after removing the objects in $\mathbf{n}_8$.NodeCol and before removing the objects in $\mathbf{n}_4$.NodeCol. Therefore, we delete the invalid edge between $\mathbf{n}_8$ and $\mathbf{n}_4$ and then add new graph nodes $\mathbf{n}_4^1$ and $\mathbf{n}_4^2$ to $\mathcal{G}$ by connecting them with $\mathbf{n}_8$ and $\mathbf{n}_4$, as shown in Fig. 5(c). In this way, we obtain two updated removal sequences when the robot traversing from $\mathbf{n}_8$ and $\mathbf{n}_4$: $\langle O_8, O_3, O_4\rangle$ and $\langle O_8, O_5, O_4\rangle$.

3) Retrieval Graph Update: According to NodeCol of each node, we can find a path with removal obstacles from the start to the goal on $\mathcal{G}$. However, some connected retrieval graph nodes may have their NodeCol sets overlapped and thus directly searching the path according to the size of NodeCol of each node cannot minimize the number of removed obstacles. For example, suppose a node's NodeCol is $\{O_1\}$ and its neighbor node's NodeCol is $\{O_1, O_2\}$. If we search path according to the NodeCol size of each node, the path collision size between these two nodes is 3, but the actual size is 2. To avoid this issue, we adopt the coverage set of path defined in [4]. We compute the collision set of a path by accumulating NodeCol sets of all nodes along the path. We define PathCol of a node $\mathbf{n}$ as the minimum collision set of the path from the start node $\mathbf{n}_s$ to $\mathbf{n}$ computed by the greedy search. Formally, we set PathCol of start node $\mathbf{n}_s$ to be equal to its NodeCol, i.e., $\mathbf{n}_s$.PathCol $= \mathbf{n}_s$.NodeCol. The PathCol of node $\mathbf{n}$ in $\mathcal{G}$ is computed recursively as follows:

$$\mathbf{n}.\text{PathCol} = \mathbf{n}.\text{NodeCol} \cup \mathbf{n}_{\text{best}}.\text{PathCol} \tag{2}$$

$$\mathbf{n}_{\text{best}} = \underset{\mathbf{n}_i \in \mathbf{N}_{\text{neighbor}}}{\arg\min} \{|\mathbf{n}.\text{NodeCol} \cup \mathbf{n}_i.\text{PathCol}|\} \tag{3}$$

where $\mathbf{N}_{\text{neighbor}}$ is the set of neighbor nodes of $\mathbf{n}$, $\mathbf{n}_{\text{best}}$ is the best neighbor node of $\mathbf{n}$, which minimizes the path collision set from the start node to $\mathbf{n}$ through $\mathbf{n}_{\text{best}}$. This operation is feasible because we restrict the roadmap edge from colliding with objects that are not contained in NodeCol of the two vertex nodes, and the retrieval graph edge is equivalent to the roadmap edge.

For computing the best path with the minimum PathCol set to $\mathbf{n}_g$, we adopt the greedy search algorithm proposed in [4], as described in Algorithm 3. In particular, we first compute the best path with the smallest path collision set among all the paths connecting the start node $\mathbf{n}_s$ and the new node $\mathbf{n}_{\text{new}}$ (Line 1–7). For each neighbor node $\mathbf{n}_{\text{neighbor}}$ of $\mathbf{n}_{\text{new}}$, we compute the collision set of the path from the start node $\mathbf{n}_s$ to $\mathbf{n}_{\text{new}}$ through $\mathbf{n}_{\text{neighbor}}$ (Line 2). We compute the best neighbor node with the minimum PathCol size and set its label to $\mathbf{n}_{\text{new}}$.BestNeighbor (Line 3–6). Since the new node added to $\mathcal{G}$ may provide a better path to reach its neighbors, we continue to check which nodes are affected by the new node and update nodes using a greedy search strategy (Line 8–19). In this greedy search, we store the candidate nodes in an ascending order priority queue Q_n where the priority is according to the size of PathCol (Line 8). In each iteration, we pop up the node $\mathbf{n}$ with the minimum PathCol size (Line 10), and then we use the same strategy as above to check whether the best path from the start node to $\mathbf{n}$ could provide a better path with a smaller path collision set from $\mathbf{n}_s$ to its neighbor $\mathbf{n}_{\text{neighbor}}$ (Line 11–18). The updated $\mathbf{n}_{\text{neighbor}}$ will be added to Q_n. When Q_n is empty or the goal node is updated, the algorithm terminates.

The solution computed by the greedy algorithm is optimal on $\mathcal{G}$ according to Theorem IV.2 in Section IV-E. We set the label $\mathbf{n}$.BestNeighbor to the best neighbor node $\mathbf{n}_{\text{best}}$ of $\mathbf{n}$, so

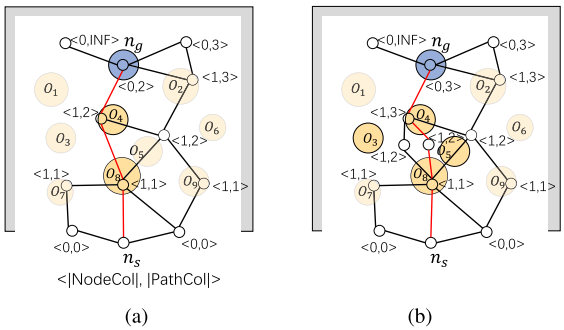


Fig. 6. $\langle|\text{NodeCol}|, |\text{PathCol}|\rangle$ for the retrieval graph nodes in Fig. 5. The red lines are the best paths from the start node to the goal node. The nodes that are not on any paths connecting $\mathbf{n}_s$ and $\mathbf{n}_g$ will not be updated, so the PathCol set of these nodes is empty and |PathCol| of these nodes is assigned INF.

tracing the label BestNeighbor from $\mathbf{n}$ to the start node, we can find the best path with the smallest path collision set from $\mathbf{n}_s$ to $\mathbf{n}$. Fig. 6 shows the NodeCol and PathCol of each node for the two retrieval graphs in Fig. 5, with the red lines as the best paths. There may exist multiple paths with PathCol of the same size, e.g., as shown in Fig. 6(b), the PathCol $\{O_8, O_5, O_4\}$ of one path is of the same size as another path whose PathCol is $\{O_8, O_3, O_4\}$. We break ties among these paths by accumulating the Euclidean distance between the consecutive obstacles along the path and choose $\mathbf{n}_{\text{best}}$ from the path with the shortest distance. In this way, the distance of the path containing $\{O_8, O_5, O_4\}$ is shorter than the path containing $\{O_8, O_3, O_4\}$.

The PathCol of the goal node $\mathbf{n}_g$ represents the set of obstacles colliding with the robot and the target object when moving along the best path from $\mathbf{n}_s$ to $\mathbf{n}_g$, from which we can generate $\mathcal{O}_r$, the sequence of objects to be removed. The objects in $\mathcal{O}_r$ are sorted according to the order of corresponding nodes in the best path. For instance, as shown in Fig. 6(a), we obtain the sequence $\mathcal{O}_r = \langle O_8, O_4, O_g\rangle$, which means the robot will try to manipulate O_8 before grasping O_4. Furthermore, every object in $\mathcal{O}_r$ is unique, which is ensured by the set operation mentioned above. For an object contained in NodeCol of multiple nodes, we only store the first one in the path, because if the object was manipulated successfully, it will be removed from the scene and will not influence the other nodes. The object removal sequence $\mathcal{R}$, which includes $\mathcal{O}_r$, the object sequence to be removed and $\mathbf{N}_r$, the retrieval graph node sequence, will be returned as the task planning result (Line 20).

D. Feasibility Checking of the Object Removal Sequence

After updating the retrieval graph $\mathcal{G}$ and obtaining the new object removal sequence $\mathcal{R}$, the motion planner follows to check whether objects in $\mathcal{O}_r$ could be grasped and manipulated by the robot. We perform feasibility checking for the objects according to their order in $\mathcal{O}_r$ (Algorithm 1, Line 15). For each object $O_i \in \mathcal{O}_r$, we first check whether O_i could be grasped by computing inverse kinematics (IK) with the preset grasps $\mathbf{g}$, which include grasp poses around the object, and then we perform motion planning to compute collision-free motions for the robot to reach and manipulate the object. If O_i cannot be manipulated by the robot without collisions, we compute the grasp pose with the minimum number of collision obstacles and collect the set of collision obstacles $\mathcal{O}_{\text{col}}$. Then, we create new node $\mathbf{n}_{\text{new}}$ with $\mathcal{O}_{\text{col}}$ and add $\mathbf{n}_{\text{new}}$ to expand $\mathcal{G}$ as introduced in Section IV-C2. After adding the new node, the task planner calls UPDATERETRIEVALGRAPH (see Algorithm 3) to find the new best path with the new object removal sequence. We repeat the feasibility checking and search for new object removal sequences until no sequence satisfies $|\mathcal{O}_r| \leq \mathcal{L}$, i.e., the total number of objects in $\mathcal{O}_r$ is no more than $\mathcal{L}$. When the target object O_g is retrieved, the task is accomplished.

We require the path reaching the target with $|\mathcal{O}_r| \leq \mathcal{L}$ in the incremental expansion process, because it can prevent the task planner from obtaining a suboptimal solution to the object removal order. For example, suppose there is only one path reaching the target with $\mathcal{O}_r = \langle O_1, O_2\rangle$ when $\mathcal{L} = 2$, and when computing grasp poses for the robot with O_1 and O_2, the robot will collide with O_3 and O_4, respectively. If we collect these collision obstacles and directly consider the new sequence $\mathcal{O}_r = \langle O_3, O_1, O_4, O_2\rangle$, we may get a feasible task planning result by removing four obstacles or the feasibility checking of O_3 and O_4 fails again and we need to collect new collision obstacles into $\mathcal{O}_r$. In this way, the eventual number of obstacles to be removed would be greater or equal to 4. However, this solution may be suboptimal because a better path with $|\mathcal{O}_r| = 3$ may exist. To avoid missing the optimal solution, we only consider paths with $|\mathcal{O}_r| \leq \mathcal{L}$ in each iteration. By increasing $\mathcal{L}$ in the subsequent iterations, our task planner can explore more new paths that satisfy $\mathcal{L}$ and eventually find the optimal result.

Algorithm 3: UPDATERETRIEVALGRAPH.

Input: new retrieval graph node $\mathbf{n}_{\text{new}}$, retrieval graph $\mathcal{G}$
Output: updated retrieval graph $\mathcal{G}$, object removal sequence $\mathcal{R} = \{\mathcal{O}_r, \mathbf{N}_r\}$

1: **for** $\forall \mathbf{n}_{\text{neighbor}} \in$ NEIGHBOR$(\mathbf{n}_{\text{new}})$ **do**
2: $S \leftarrow \mathbf{n}_{\text{neighbor}}$.PathCol $\cup$ $\mathbf{n}_{\text{new}}$.NodeCol
3: **if** $|\mathbf{n}_{\text{new}}$.PathCol$| = \emptyset$ **or** $|S| < |\mathbf{n}_{\text{new}}$.PathCol$|$ **then**
4: $\mathbf{n}_{\text{new}}$.PathCol $\leftarrow S$
5: $\mathbf{n}_{\text{new}}$.BestNeighbor $\leftarrow \mathbf{n}_{\text{neighbor}}$
6: **end if**
7: **end for**
8: $Q_n \leftarrow$ PRIORITYQUEUE$(\mathbf{n}_{\text{new}})$
9: **repeat**
10: $\mathbf{n} \leftarrow Q_n$.pop()
11: **for** $\forall \mathbf{n}_{\text{neighbor}} \in$ NEIGHBOR$(\mathbf{n})$ **do**
12: $S \leftarrow \mathbf{n}$.PathCol $\cup$ $\mathbf{n}_{\text{neighbor}}$.NodeCol
13: **if** $|\mathbf{n}_{\text{neighbor}}$.PathCol$| = \emptyset$ **or** $|S| < |\mathbf{n}_{\text{neighbor}}$.PathCol$|$ **then**
14: $\mathbf{n}_{\text{neighbor}}$.PathCol $\leftarrow S$
15: $\mathbf{n}_{\text{neighbor}}$.BestNeighbor $\leftarrow \mathbf{n}$
16: Q_n.add$(\mathbf{n}_{\text{neighbor}})$
17: **end if**
18: **end for**
19: **until** Q_n is Empty or $\mathbf{n}_{\text{neighbor}} = \mathcal{G}.\mathbf{n}_g$
20: $\mathcal{R} \leftarrow$ EXTRACTOBJREMOVALSEQUENCE$(\mathbf{n}_g)$
21: **return** $\mathcal{G}, \mathcal{R}$

E. Analysis

Theorem IV.1: Our incremental sampling-based TAMP is probabilistically complete.

Proof: Suppose $\mathcal{L} \geq |\mathcal{O}|$, i.e. when performing sampling-based motion planning for the target object in its configuration space $\mathscr{C}$, we allow the target object to collide with other objects when constructing a roadmap and finding a path to connect $\mathbf{x}_s$ and $\mathbf{x}_g$. This sampling-based motion planner is probabilistically complete. The retrieval graph over the roadmap records the best path from $\mathbf{x}_s$ to $\mathbf{x}_g$ with $\mathcal{O}_r$, the object sequence to be removed. Removing objects in $\mathcal{O}_r$ ensures the robot end-effector is collision-free when moving along the path to retrieve the target object in $\mathscr{C} \setminus \mathcal{O}_r$. Because we iteratively add new collision obstacles to $\mathcal{O}_r$ from motion planning failures, the worst case is all objects are contained in $\mathcal{O}_r$ (i.e., removing all obstacles). Moreover, the node of the retrieval graph contains the minimum collision obstacle set along the path from $\mathbf{x}_s$ to $\mathbf{x}_g$ and $\mathcal{O}_r$ is an ordering sequence, the worst case is our sampling-based planner explore all different retrieval paths with different orders from $\mathbf{x}_s$ to $\mathbf{x}_g$. Since the Bi-RRT used in robot motion planning is probabilistic complete [47], given sufficient time, the probability of retrieving the target object approaches one. ■

Theorem IV.2: The solution computed by the greedy algorithm is optimal on the retrieval graph $\mathcal{G}$ if there exists a path from $\mathbf{n}_s$ to $\mathbf{n}_g$ with the minimum collision set $S_g^\star$.

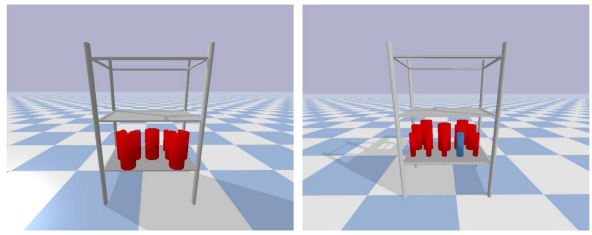
Proof: For any path $\langle \mathbf{n}_s, \mathbf{n}_1, \ldots, \mathbf{n}_t \rangle$ in $\mathcal{G}$ starting from $\mathbf{n}_s$, the size of the path collision set is $|\mathbf{n}_s.\text{NodeCol}| + \sum_{i=1}^{t} |\mathbf{n}_i.\text{NodeCol} \setminus \mathbf{n}_{i-1}.\text{NodeCol}|$. Suppose there exists a path $\langle \mathbf{n}_s, \mathbf{n}_1, \ldots, \mathbf{n}_g \rangle$ in $\mathcal{G}$ from $\mathbf{n}_s$ to $\mathbf{n}_g$ with the minimum collision set $S_g^\star$, where $|S_g^\star| = |\mathbf{n}_s.\text{NodeCol}| + \sum_{i=1}^{g} |\mathbf{n}_i.\text{NodeCol} \setminus \mathbf{n}_{i-1}.\text{NodeCol}|$. By the Lemma A.1, which is proposed in [4], the size of the path collision set computed by the greedy search algorithm is no more than $|S_g^\star|$, i.e., $|\mathbf{n}_g.\text{PathCol}| \leq |\mathbf{n}_s.\text{NodeCol}| + \sum_{i=1}^{g} |\mathbf{n}_i.\text{NodeCol} \setminus \mathbf{n}_{i-1}.\text{NodeCol}| = |S_g^\star|$. By the Theorem A.1, which is proposed in [4], since $|S_g^\star|$ is optimal on $\mathcal{G}$ and $|\mathbf{n}_g.\text{PathCol}| \geq |S_g^\star|$, so $|\mathbf{n}_g.\text{PathCol}| = |S_g^\star|$. Thus, the solution computed by the greedy algorithm is optimal on $\mathcal{G}$. ■

Theorem IV.3: Algorithm 2 has $O(T_{\text{total}} \log T_{\text{total}})$ time complexity.

Proof: Algorithm 2 uses a sampling based method to construct the roadmap. Each sampling iteration generates T samples and our incremental algorithm takes at most $\mathcal{L}$ iterations, so the total number of samples is $T_{\text{total}} = \mathcal{L} \cdot T$. For each sample, finding nearest neighbors takes $\log T_{\text{total}}$ time. Thus, the time complexity of constructing roadmap is $O(T_{\text{total}} \log T_{\text{total}})$.

Theorem IV.4: Algorithm 3 has $O(|\mathcal{G}.\mathbf{N}| \log |\mathcal{G}.\mathbf{N}|)$ time complexity.

Proof: For finding the best path with the smallest path collision set to the target object, we adopt the greedy search [4] on the retrieval graph in Algorithm 3. The greedy search runs in time $O(|\mathcal{G}.\mathbf{N}|)$ on the retrieval graph $\mathcal{G}$. We use the candidate nodes in an ascending order priority queue, which is implemented as a Fibonacci heap. Extracting the minimum PathCol size node takes $O(\log |\mathcal{G}.\mathbf{N}|)$ time. Thus, the time complexity is $O(|\mathcal{G}.\mathbf{N}| \log |\mathcal{G}.\mathbf{N}|)$. ■



(a) (b)

Fig. 7. An example scene of occlusion experiment. (a) Front view of the cluttered environment. Three large objects hide other objects. (b) Another view from the backend side.

V. Extension for Complex Tasks

A. Occlusion and Unobservable Target Pose

Our TAMP algorithm can be extended to handle the realistic scenario where there exist occlusions between objects and thus the robot cannot fully observe the entire scene or even does not know the position of the target object. For instance, as shown in Fig. 7, there are three large objects at the front of the clutter, and other objects, including the target object (blue), are unobservable to the robot. Thanks to the incremental strategy of our algorithm, it can be conveniently extended to handle the objects that were not observable to the robot in the beginning and become observable after some other objects are removed. The extended algorithm is shown in Algorithm 4 and Fig. 8 illustrates how it works for the scenario of Fig. 7. In particular, Fig. 8(a) shows an initial setup where the robot can only observe O_1, O_2, and O_3, while its view of other objects is obstructed by obstacles between them and the robot. For example, O_g is occluded by O_6 and O_6 is occluded by O_2. If the target object is unobservable to the robot, we first find an object farthest from the start point and assume a temporal node $\mathbf{x}_{\text{temp}}$ behind this object (Line 3–5). We denote $\mathbf{x}_{\text{temp}}$ as the goal node $\mathbf{x}_g$ and execute our algorithm (Algorithm 1, Line 9). As shown in Fig. 8(b)–(d), suppose that we find a path from the start node $\mathbf{n}_s$ to the goal node $\mathbf{n}_g$ with the obstacle O_2, and O_2 can be successfully removed by the robot. After removing O_2, we can find a new object O_6 behind O_2. We then add O_6 to the set of objects $\mathcal{O}$ (Line 12–14) and set a new temporal node as the goal node to execute our algorithm. If the target object is observable, we set the target object pose as the goal node [Line 6, Fig. 8(e)] and compute a retrieval plan for the target object. We repeat this procedure until the target object is retrieved.

B. Mobile Manipulation

Our method can also be extended to handle the high-dimensional planning problem of using a mobile manipulator. Mobility can increase the robot's manipulation capability by allowing it to reach an object from different directions. For instance, suppose a set of objects are placed on a table and the target is in the middle of the group, as shown in Fig. 9. If the base of PR2 robot is fixed, it can only remove objects from one side and have to remove at least two obstacles. If the base of PR2

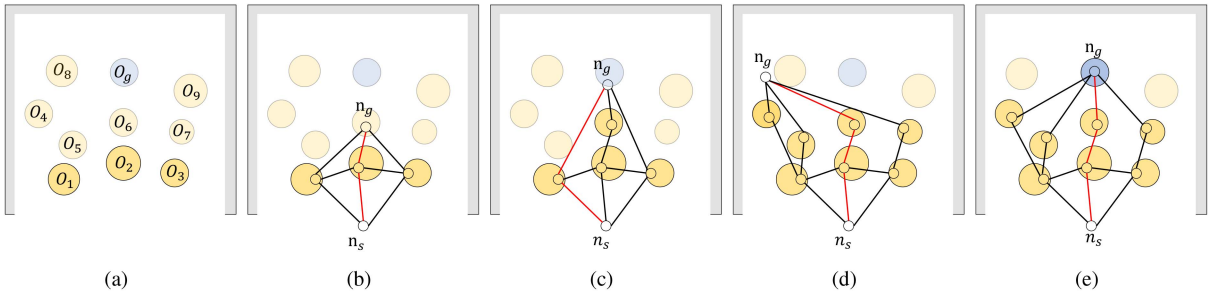


Fig. 8. Example of occlusion experiment. (a) The initial setup where only O_1, O_2, and O_3 can be observed and other objects are hidden, including the target object. (b) We choose a node $\mathbf{x}_{\text{temp}}$ behind the farthest object, and create a retrieval graph node contain $\mathbf{x}_{\text{temp}}$ and set it as the temporal goal node $\mathbf{n}_g$. After performing our TAMP algorithm, we can find paths to $\mathbf{n}_g$. (c)–(d) If new objects are found, we add them to the object set and update $\mathbf{n}_g$, then continue to perform our TAMP algorithm with the new $\mathbf{n}_g$. (e) If the target object is found, we replace the temporal node with the target object pose and set it as the goal node $\mathbf{n}_g$, then continue to perform our TAMP algorithm with the new $\mathbf{n}_g$.

Algorithm 4: OCCLUSIONPLANNER.

Input: target object start point $\mathbf{x}_s$, objects set $\mathcal{O}$, target object O_g, robot rob, grasps $\mathbf{g}$
Output: object removal sequence $\mathcal{R}$, robot motion set Π

$\mathcal{R}, \Pi \leftarrow \emptyset, \emptyset$
while O_g is not retrieved **do**
  if O_g is not detected **then**
    compute $\mathbf{x}_{\text{temp}}$ based on the farthest reachable object from $\mathcal{O}$
    $\mathbf{x}_g = \mathbf{x}_{\text{temp}}$
  else
    $\mathbf{x}_g = O_g.pose$
  end if
  $\mathcal{R}', \Pi' \leftarrow$ PLANNINGWITHINTERLEAVEDUPDATES$(\mathbf{x}_s, \mathbf{x}_g, \mathcal{O}, O_g, rob, \mathbf{g})$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}', \Pi \leftarrow \Pi \cup \Pi'$
  if find new objects $\mathcal{O}_{\text{new}}$ **then**
    $\mathcal{O} \leftarrow \mathcal{O} \cup \mathcal{O}_{\text{new}}$
  end if
end while
return $\mathcal{R}, \Pi$

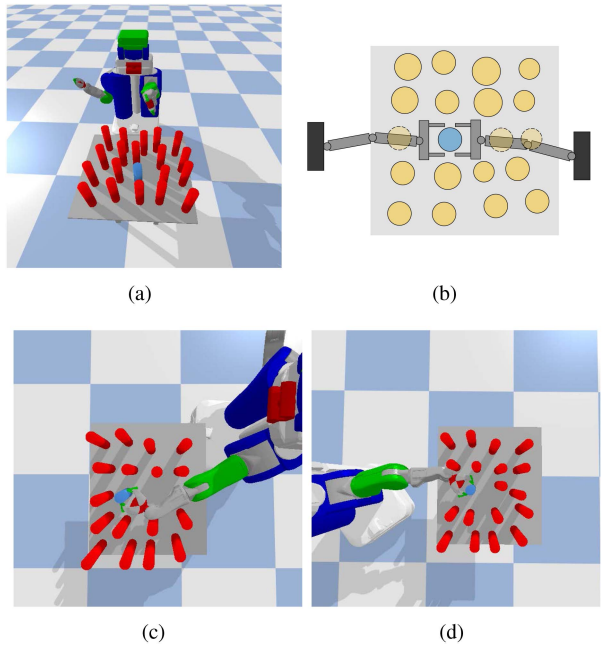


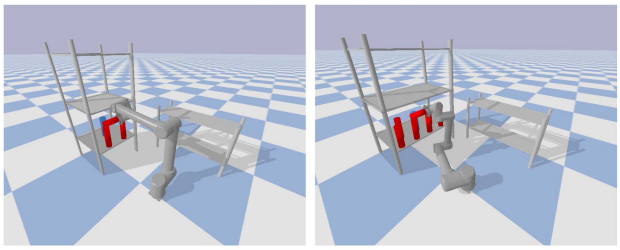
Fig. 9. Example scene of mobile manipulation with the PR2 robot with one arm locked. (a) 20 objects located on the table and the target object in the center. (b) Bird's-eye view of the environment. The target object can be retrieved from all directions with different number of obstacles to be removed. (c) If the robot tries to retrieve the target object without moving the base, it needs to remove at least two obstacles before retrieving the target object. (d) If the robot can move its base, it can move to another side of the clutter and only remove one obstacle.

can move, it can reach the target from any direction and choose a path with a smaller number of objects to be removed.

For handling mobile manipulation, we enlarge the workspace to cover the free space around the cluttered scene and generate samples in the corresponding configuration space. All samples outside the cluttered region are collision-free and can be connected to the start point with collision-free paths. In this way, the paths from all directions to the target can be investigated.

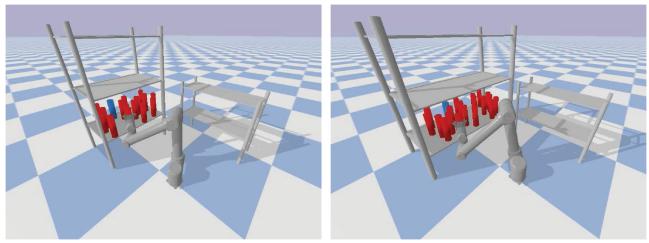
C. PDDL Planning

Our task planner can be incorporated with the PDDLStream planning [32] to complete more complex tasks. In particular, the candidate object removal sequence generated by our task planner could help the general PDDLStream planning focus on a small set of objects in the cluttered scene and thus significantly improve its performance in the special task of target retrieval. PDDLStream is an extension of the standard AI planning PDDL (Planning Domain Definition Language) [48], which incorporates sampling procedures (called streams) to combine task planning and motion planning. PDDLStream plans the task by iteratively increasing the maximum number of subtasks. In each iteration, it finds an optimistic plan and evaluates all stream instances. For example, for the target retrieval task, PDDLStream tries to pick and manipulate the target object directly at first. If the direct manipulation fails, it will increase the maximum number of subtasks and try an alternative task plan to remove an obstacle and then manipulate the target. If the task plan still fails, it will continue to increase the subtask number and try new task plans with more obstacles being removed. For each subtask, PDDLStream evaluates stream instances with all removable obstacles. For example, for the subtask "pick an obstacle," the planner will



(a) (b)

Fig. 10. Object is placed on top of two objects in front of the target object. If the robot wants to remove front objects, it needs to remove the top object in advance. (a) Experiment with 4 objects. (b) Experiment with 8 objects.



(a) (b)

Fig. 11. Retrieving a target object (blue) from a cluttered scene with many obstacles. (a) Experiment with UR10 robot and 16 objects. (b) Experiment with UR10 robot and 20 objects.

try to pick all obstacles to find a feasible one by sampling grasp pose and performing manipulation motion with each object. As the number of subtask increases, each subtask needs to evaluate all stream instances and thus the computational complexity increases rapidly when the number of removable objects is large.

In this extension, we incorporate our task planner with PDDLStream by replacing the routine of grasp planning and robot motion planning with PDDLStream planning. This extension has two advantages. First, our planner can help the general PDDLStream focus on a small range of objects in the cluttered scene. Second, PDDLStream planning can enhance our algorithm capability to handle a more complex environment, such as an object is placed on top of other objects, as shown in Fig. 10. In this experiment, we use a predicate `(Clear ?o)` to describe whether there is an object placed on top of the object o. If no, `(Clear ?o)` is true, otherwise false. Suppose the robot wants to remove objects located in front of the target object. Then, it needs to reason out a plan that removes the object placed on top of the front objects and sets the predicate `(Clear ?o)` of these objects to be true. Only after such a plan is executed can the robot remove these front objects. The pick and place action schemas used in our experiment are as follows:

```
  (:action pick
    :parameters (?o ?p ?g ?q)
    :precondition (and (Clear ?o) (At-
Pose ?o ?p)
        (HandEmpty) (AtConf ?q))
    :effect (and (AtGrasp ?o ?g) (not (At-
Pose ?o ?p))
        (not (Clear ?o)) (not (Han-
dEmpty)))
  )
  (:action place
    :parameters (?o1 ?o2 ?p ?g ?q)
    :precondition (and (not (Clear ?o1))
(Clear ?o2)
         (AtGrasp ?o1 ?g) (AtConf ?q)
    :effect (and (AtPose ?o1 ?p) (not (At-
Grasp ?o1 ?g))
        (on ?o1 ?o2) (Clear ?o1)
(not (Clear ?o2))
        (HandEmpty))
  )
```

Here, an action consists of a set of parameters `:parameters`, the conjunctive boolean preconditions `:precondition` that must be satisfied for the action to be executed, and the conjunctive Boolean effects `:effect` that describe the changes of the state after executing the action. `and, or, not` are Boolean operators. Instead of performing PDDLStream with all objects, we first perform our task planner to compute candidate object removal sequences. Then, we treat the removal objects as movable objects and other objects as fixed obstacles and perform PDDLStream planning. In this way, we can significantly reduce the search space and consequently the computational complexity by combining our task planner specific for the target retrievial with and the general PDDLStream.

VI. Experiments

In this section, we present some details of the implementation, experiment environments, and the performance of our algorithm on different tasks.

A. Implementation

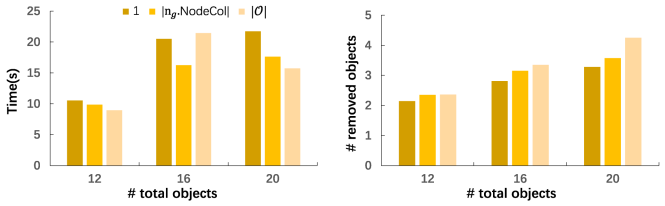
We have implemented our algorithm on a PC running Linux. The performance is measured on a PC with a 2.9 GHz Intel Core i7 CPU and 16 GB memory. We use Bi-directional RRT (Bi-RRT) to compute the motion planning for the robot. In our algorithm, we set $T = 1000$ and $T_{\text{raise}} = 100$. When comparing different planners, we limit the maximum planning time to 1 min and consider a planning instance unsuccessful if the time is running out. The target object only moves when the robot's end-effector grasps it, so the radius of the footprint circle of the target object r_b in our simulations is set according to the end-effector size and the target object size $r_b = r_g + r_f$, where r_g is the radius of the target object and r_f is the thickness of the gripper finger. The radius of the footprint circle with different grippers is given in Appendix A. We build the simulation environment and compute collision detection using PyBullet [49].

B. Experiments With Our Planner

We test our planner with multiple scenarios with different numbers (12, 16, 20, respectively) of objects to investigate the planning performance. As shown in Figs. 1 and 11, we use cylinders to represent the objects, where the radius and height of cylinder are randomly sampled between [3.5 cm, 4.0 cm]

TABLE II
PLANNING RESULT OF OCCLUSION EXPERIMENTS. WE USE A 6-DOF UR10 ROBOT TO TEST OUR ALGORITHM

Measure	$\|\mathcal{O}\|$=12	$\|\mathcal{O}\|$=16	$\|\mathcal{O}\|$=20
$\|\mathcal{O}_r\|$	2.84	4.02	4.86
Time (s)	18.85	27.45	36.01
# Iteration	3.14	5.2	6.56
Success Rate (%)	95	90	90



(a) (b)

Fig. 12. Comparison between different initial values of $\mathcal{L}$ with UR10 robot. (a) The planning time. (b) The number of removed objects (including the target object).

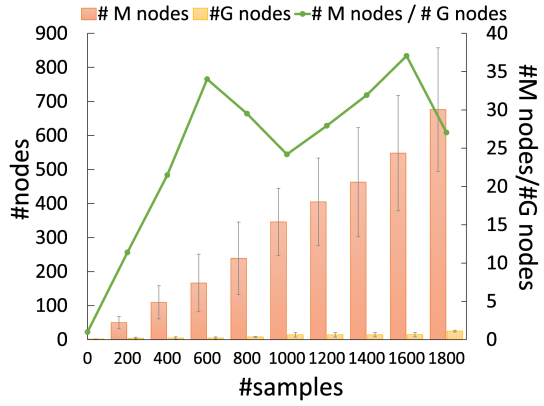
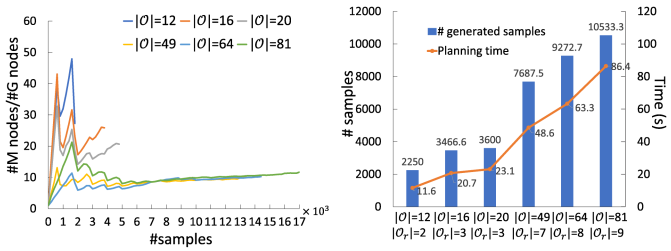


Fig. 13. Number of nodes in the roadmap $\mathcal{M}$ and the retrieval graph $\mathcal{G}$. We show the changes of node numbers and the ratio between the numbers of $\mathcal{M}$ nodes and $\mathcal{G}$ nodes during sampling iterations. This experiment uses a UR10 robot to retrieve the target from 12 objects.



(a) (b)

Fig. 14. (a) Ratios between the numbers of $\mathcal{M}$ nodes and $\mathcal{G}$ nodes for different experiments during sampling iterations. (b) The average number of samples and planning time for experiments with different number of objects. The x-axis is the number of objects in the cluttered scene and the number of objects that need to be removed to retrieve the target object.

and [24 cm, 29 cm], respectively. Objects are randomly and uniformly placed on the table without any collisions. The size of the cabinet changes from 0.9 m $\times$ 0.6 m $\times$ 0.5 m to 1.2 m $\times$ 0.7 m $\times$ 0.5 m for three scenarios. We use a Universal Robot 10 (UR10) with 6 DoFs to accomplish the task and the parallel gripper set on UR10 is a DH AG-95 gripper. The robot cannot reach the target without removing the blocked objects. For each task, we measure the number of removed objects, the total planning time, the number of iterations, and the success rate. We generate 20 instances of object distribution and run 20 times for each instance to collect quantitative performance measurements that are summarized in Table I.

We found the initial value of the exploration limit $\mathcal{L}$ will affect the algorithm performance. Fig. 12 shows the comparison between different $\mathcal{L}$ values. Among the three ways to set the initial values ($\mathcal{L} = 1$, $\mathcal{L} = |\mathbf{n}_g.\text{NodeCol}|$ and $\mathcal{L} = |\mathcal{O}|$), the planner starting with $\mathcal{L} = 1$ can compute a path with the fewest average number of collision obstacles because it uses more samples with fewer collision obstacles to explore the scene. However, this also increases the planning time due to the more iterations before finding a feasible path. For the choice $\mathcal{L} = |\mathbf{n}_g.\text{NodeCol}|$, we use the size of NodeCol of the goal node as the initial value. This requires reachability checking for the target object before planning, i.e., computing the minimum number of collision obstacles when the robot directly grasps the target object, which is a tighter lower bound of the number of objects eventually removed than $\mathcal{L} = 1$. Thus, starting from $\mathcal{L} = |\mathbf{n}_g.\text{NodeCol}|$, we can reduce iteration number and save the planning time, but the average number of collision obstacles to be removed is larger than $\mathcal{L} = 1$ because the regions reachable with a small number of collision obstacles are not sufficiently sampled. The choice $\mathcal{L} = |\mathcal{O}|$ means all paths to the goal node can be treated as valid, and we plan a motion without considering obstacles. This method does not have the incremental iteration step. Instead, it directly finds a path to the goal and then checks the feasibility of the collision obstacles along the path. This method takes the least time to find the path to the target object in most cases, but it may take more time to replan in some cases due to unexpected robot-obstacle collisions. This method removes more objects than the other two choices.

Our task planner uses the retrieval graph $\mathcal{G}$ to accelerate update efficiency by clustering the roadmap nodes and reducing the node number. We show the benefits of using the retrieval graph by comparing node numbers of $\mathcal{M}$ and $\mathcal{G}$ during the iterations in Fig. 13, where the experiment is to retrieve the target from 12 objects using a UR10 robot. We demonstrate the ratio of the node number of $\mathcal{M}$ to the node number of $\mathcal{G}$ during the iterations. We can find that the ratio goes up quickly in the early stage, which means that the collision regions of the target object can be represented with fewer retrieval graph nodes. After generating more samples and performing feasibility checking, we need more retrieval graph nodes to collect the robot's grasp and motion planning failures. We further demonstrate the ratios in multiple scenarios with different numbers of objects in Fig. 14(a). We find that the complex scenario with a large number of objects needs more retrieval graph nodes than simpler scenarios. Fig. 14(b) also shows the number of samples generated to find the object retrieval sequence and the planning times in different situations.

TABLE III
PLANNING RESULT OF MOBILE MANIPULATION TASK. WE USE A PR2 ROBOT AND TEST OUR ALGORITHM WITH DIFFERENT EXPLORATION LIMIT VALUES $\mathcal{L}$

Measure	$\mathcal{L}$=1	$\mathcal{L} = \lvert\mathbf{n}_g.\text{NodeCol}\rvert$	$\mathcal{L} = \lvert\mathcal{O}\rvert$
$\lvert\mathcal{O}_r\rvert$	2.22	2.33	3.45
Time (s)	44.23	42.57	49.25
# Iteration	1.12	1.22	1
Success Rate (%)	90	90	95

C. Occlusion and Unobservable Target Pose

In this experiment, we test our occlusion planner with three scenarios that include 12, 16, and 20 objects, respectively. Each object is occluded by the object in front of it and the target object pose is unobservable to the robot. We use a 6-DOF UR10 robot arm with a suction gripper to manipulate objects and the results are summarized in Table II. In each iteration, our planner finds paths that satisfy the exploration limit and then tries to remove objects along the path to find new objects that were occluded. The procedure of trying to grasp and manipulate objects that are not in the final object removal sequence would take extra time cost, and thus the occlusion planner is slower than our basic TAMP algorithm working in a known scene.

D. Experiment With Mobile Manipulation

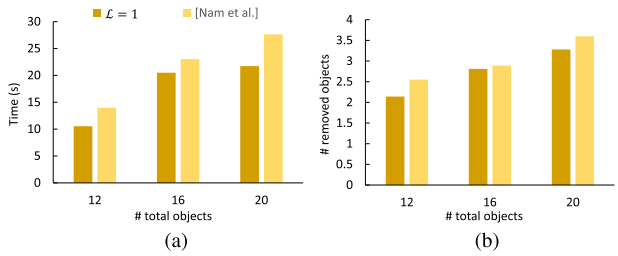
In this experiment, we test our planner with the mobile manipulation task, with the results summarized in Table III. As shown in Fig. 9(a), we randomly place 20 objects on the table, including the target object, and use a PR2 robot to retrieve the target object. We initially put the robot on one side of the table, lock its right arm and only allow the left arm to manipulate objects. The PR2 robot can manipulate objects from any direction around the table. If the robot tries to retrieve the target from the initial position, it needs to remove at least two objects, as shown in Fig. 9(b) and (c). If the robot relocates to another side of the table, it can retrieve the target object by removing only one object, as shown in Fig. 9(d).

E. Experiment With PDDLStream

In this experiment, we incorporate our planner with PDDLStream planning to complete complex tasks. As shown in Fig. 10, an object is placed on top of two objects located in front of the target object. We use a UR10 robot and test in two scenarios that contain 4 and 8 objects, respectively. The additional objects in Fig. 10(b) other than those in Fig. 10(a) shall not affect the robot's final action sequence. The optimal manipulation strategy for both scenarios is to remove the object placed on top of two objects and then remove one of the two objects in front of the target. Compared with the original PDDLStream method, our planner first computes candidate object removal sequences and then uses them as input of PDDLStream, which helps PDDLStream focus on a small range of objects even though the number of objects in Fig. 10(b) is more than Fig. 10(a). Thus, the planning time of our method increases more slowly than the original PDDLStream when the number of objects in the scene increases, with the results summarized in Table IV.

TABLE IV
COMPARISON RESULT WITH PDDLSTREAM. WE TEST TWO SCENES WITH DIFFERENT NUMBERS OF OBJECTS AND MEASURE AVERAGE PLANNING TIME

	$\lvert\mathcal{O}\rvert$ = 4	$\lvert\mathcal{O}\rvert$ = 8
Our Method+ PDDLStream	25.31 (s)	28.82 (s)
PDDLStream	33.65 (s)	>120 (s)



(a) (b)

Fig. 15. Comparison with the state-of-the-art algorithm. (a) Average time. (b) The number of removed objects (including the target object).

F. Comparisons

We compare our method with the state-of-the-art method [3]. The method proposed in [3] constructed a traversability graph to represent the movable paths of objects in the cluttered environment. They construct the graph by choosing the largest object and then move the largest object between all objects to see if there is a collision-free path between two object poses. If the path exists, the planner connects two related objects and adds an edge to the graph. After preconstructing the graph, they search in the graph to find a feasible path to the target. Compared to this method, we do not need to handle all objects in the scenario and check collision detection between all object pairs, which is a highly time-consuming process, especially when the number of objects is large. We compare our planner with the experiments of Case I in [3]. We set benchmark scenes with the same object numbers and a robot arm with the same degree of freedom. We use a 6-DOF robot and test on three scenes with 12, 16, 20 objects, respectively. We set the initial value of the exploration limit $\mathcal{L} = 1$ and the results are shown in Fig. 15. Our planner takes shorter planning time than [3] with a time reduction of 24.6%, 10.9%, and 21.3% on three scenes respectively. Our planner also manipulates fewer objects than [3]. In particular, the average numbers of removed objects in the three benchmark scenes are reduced by 16.1%, 3%, and 8.8%, respectively. Our algorithm also provides a higher success rate because our planner can quickly collect failure from motion planning and feedback it to the task planner.

Furthermore, to compare the performance of our method with [3] in handling a large number of objects in cluttered scenes, we test on the scene with 100 objects. In this task, the robot needs to remove at least 10 objects to retrieve the target object. Table V shows the results. Our method takes a longer task planning time than [3] since the cost of constructing the roadmap by sampling and updating the retrieval graph in our method is more than the cost of constructing the traversability

TABLE V
COMPARISON BETWEEN OUR METHOD ($\mathcal{L} = 1$) AND THE STATE-OF-THE-ART METHOD [3] IN HANDLING THE LARGE NUMBER OF OBJECTS. WE USE A 6-DOF UR10 ROBOT AND $|\mathcal{O}| = 100$.

	[Nam *et al.*]	Our Method ($\mathcal{L}$ =1)
Task Planning Time (s)	9.71 (0.58)	40.95 (4.72)
Motion Planning Time (s)	126.40 (31.44)	78.54 (6.68)
Total Time (s)	136.10 (31.52)	119.49 (9.52)
# Planning Sequences	5.05 (1.57)	1.30 (0.47)
# Removed Objects	10.95 (0.88)	10.45 (0.51)

We test on the scene with the minimum $|\mathcal{O}_r| = 10$. We run each algorithm 10 times and report the mean (std) of the performance.

TABLE VI
COMPARISON BETWEEN OUR METHOD ($\mathcal{L} = 1$) AND THE BRUTE-FORCE SEARCH METHOD IN BOTH THE NUMBER OF OBJECTS TO BE REMOVED AND THE TIME REQUIRED

Min $\lvert\mathcal{O}_r\rvert$	Measure	Brute−force Search	Our Method ($\mathcal{L}$=1)
2	$\lvert\mathcal{O}_r\rvert$	2 (0)	2.25 (0.45)
	Time (s)	4.34 (0.11)	4.11 (0.65)
3	$\lvert\mathcal{O}_r\rvert$	3 (0)	3.18 (0.38)
	Time (s)	18.39 (2.54)	14.71 (2.26)
4	$\lvert\mathcal{O}_r\rvert$	4 (0)	4.27 (0.44)
	Time (s)	145.56 (7.44)	20.84 (5.57)

We use a 6-DOF UR10 robot and $|\mathcal{O}| = 16$. We test on three scenes with the minimum $|\mathcal{O}_r| = 2$, 3, and 4. For each scene, we run each algorithm 20 times and report the mean (std) of the performance.

graph and search paths in [3]. However, [3] takes a longer motion planning time than our method. The motion planning in [3] is to check whether the objects in the searched path could be manipulated by the robot. If an object cannot be manipulated due to robot-obstacle collisions, the edge that connects to the object in the path will be deleted from the graph and then replan a new path over the updated graph. However, replanning does not consider important information like which specific obstacle blocks robot's motion. The planner needs to replan multiple times to find the successful path. Thus, the number of planning sequences of [3] is larger than our method and they takes a lot of time to perform motion planning to check whether sequences are feasible. Our planner uses the retrieval graph to collect collision feedback and efficiently replan from failures, so we have fewer replanning times, shorter motion planning time, and manipulates fewer objects than [3].

To investigate the quality of our result, we compare our method with a brute-force search method. The brute-force search object retrieval will check all the possible combinations of removing obstacles, including different subsets of obstacles and in different orders. Although very time-consuming, it can provide the optimal solution for object retrieval. In this experiment, we use a 6-DOF UR10 robot to retrieve the target object from three scenes all with 16 objects. But these scenes have different numbers of minimum objects to be removed for successful retrieval (2, 3, 4, respectively). For our method, since it is randomized, we run it 20 times for each scene to collect performance measurements, and the mean and standard variance for the numbers of retrieval objects and planning time are shown in Table VI. We can see that our result is close to the optimal solution in terms of the number of retrieval objects, but our planning time is significantly shorter than the brute-force search.

G. Implementation With Physical Robot

We validate our planner with a real-world example using a physical robot. As shown in Fig. 1(b), we test our planner with a 7-DoF Franka Panda robot arm. We place 12 bottles on a shelf, and the positions of bottles are the same as distribution in the simulation. In order to retrieve the target bottle, the robot has to remove two bottles that block robot manipulation motion.

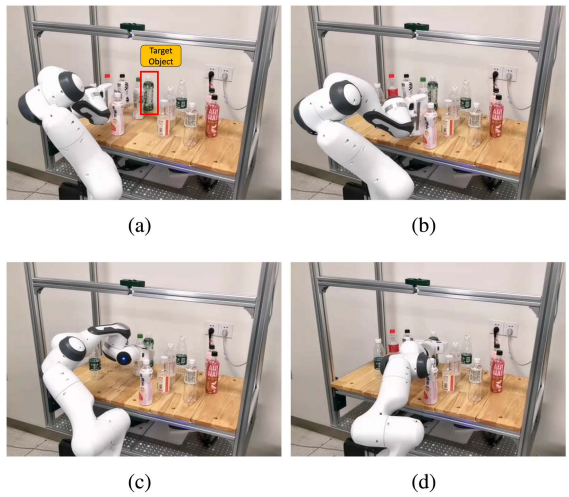


Fig. 16. Snapshot of the target object retrieval process with non-prehensile and prehensile actions. (a), (b) Pushing the bottle to a collision-free position using non-prehensile action. (c) Picking the bottle in front of the target object as there does not have valid goal positions for pushing. (d) Retrieving the target object using prehensile manipulation.

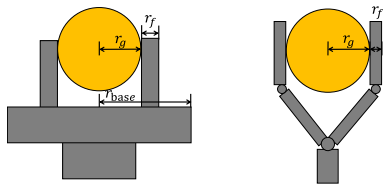


Fig. 17. Radius of the bounding cylinder of the target object with different grippers. (a) For the gripper with a large base, we set half of the base length r_{base} as the radius of bounding cylinder r_b. (b) For the gripper without a large base, r_b is equal to the radius of the target r_g plus finger thickness r_f.

We integrate our algorithm with a nonprehensile action (pushing) to further improve operational efficiency, as shown in Fig. 16. After computing the sequence of objects to be removed, we collect collision-free sampling poses that locate around removal objects and have no collision with the swept volume of the robot motions. If these poses can be reached with the nonprehensile action, we can use this non-prehensile action to

replace pick-and-place action. Please watch the supplementary video for the manipulation process.

VII. Conclusion

We present a sampling-based TAMP solution to retrieve the target object occluded by many other objects. Our method incrementally constructs a roadmap and a retrieval graph that represents the manipulation collision states. Our solution can efficiently determine the object removal order and plan a series of collision-free manipulation motions for the robot. Our algorithm can handle both known and occlusion scenes and mobile manipulation. We demonstrated our experimental results both in simulation and physical environments. Compared to the state-of-the-art, we observed up to 24.6% performance improvement in terms of TAMP time. Moreover, our method is probabilistically complete.

Our algorithm has a few limitations. First, we assume the objects are near-cylindrical. We use cylinders to represent these objects and prespecified grasp poses for these models. Second, we assume the gripper could grasp the object firmly. For future work, we would like to extend our algorithm to handle objects with various geometric shapes. It is also challenging and interesting to handle dynamic environments. Moreover, we would like to investigate parallel and GPU-based algorithms to achieve real-time performance.

Appendix A

In this appendix, we copy some important Lemma and theorem as well as their proofs in [4] using our terminology to make our paper self-contained.

Lemma A.1: Let $P = \langle \mathbf{n}_0, \mathbf{n}_1, \ldots, \mathbf{n}_t \rangle$ be any path in $\mathcal{G}$ starting at $\mathbf{n}_0 = \mathbf{n}_s$. Then the size of the subset at $\mathbf{n}_t$ computed by the greedy algorithm is no more than

$$|\mathbf{n}_0.\text{NodeCol}| + \sum_{i=1}^{t} |\mathbf{n}_i.\text{NodeCol} \setminus \mathbf{n}_{i-1}.\text{NodeCol}|. \quad (4)$$

Proof: Let $S_0, \ldots, S_t$ denote PathCol obtained at $\mathbf{n}_0, \ldots, \mathbf{n}_t$ by greedy search. Define the partial sums $k_0 = |\mathbf{n}_0.\text{NodeCol}|, k_1 = k_0 + |\mathbf{n}_1.\text{NodeCol} \setminus \mathbf{n}_0.\text{NodeCol}|, \ldots, k_t = k_{t-1} + |\mathbf{n}_t.\text{NodeCol} \setminus \mathbf{n}_{t-1}.\text{NodeCol}|$. We can prove that $|S_i| \leq k_i$ by induction on i. The base case with $i = 0$ is true by definition. Now consider the subset S_{i-1} reached at $\mathbf{n}_{i-1}$ and make the inductive assumption $|S_{i-1}| \leq k_{i-1}$. Because S_{i-1} also contains $\mathbf{n}_{i-1}.\text{NodeCol}$, the search would add precisely $|\mathbf{n}_i.\text{NodeCol} \setminus \mathbf{n}_{i-1}.\text{NodeCol}|$ elements to S_{i-1} if it took the candidate edge from $\mathbf{n}_{i-1}$ to $\mathbf{n}_i$. The greedy search has the option of traversing the edge $\mathbf{n}_{i-1} \to \mathbf{n}_i$, and will not choose an edge into $\mathbf{n}_i$ that produces a larger subset. So, we have $S_i \leq |S_{i-1}| + |\mathbf{n}_i.\text{NodeCol} \setminus \mathbf{n}_{i-1}.\text{NodeCol}| \leq k_{i-1} + |\mathbf{n}_i.\text{NodeCol} \setminus \mathbf{n}_{i-1}.\text{NodeCol}| = k_i$ as desired. By induction this inequality holds for all i. ■

Theorem A.1: The solution computed by the greedy algorithm is optimal on the retrieval graph $\mathcal{G}$ if there exists a path P from $\mathbf{n}_s$ to $\mathbf{n}_t$ with PathCol cover $S^\star$ such that each obstacle in $S^\star$ enters into P at most once. Here, "entering" means that for any $i \in S^\star$, the set of vertices along P for which i lies in their cover form a connected subsequence.

Proof: Number the vertices of such a path $P = \langle \mathbf{n}_0, \ldots, \mathbf{n}_t \rangle$ on the retrieval graph $\mathcal{G}$. The size of the covers of each prefix of P form a nondecreasing sequence $\langle k_0, \ldots, k_t \rangle$ for which $k_0 = |\mathbf{n}_s.\text{NodeCol}|$, where $\mathbf{n}_s = \mathbf{n}_0$, and $k_t = |S^\star|$. The single entry assumption shows that $k_{i+1} - k_i = |\mathbf{n}_{i+1}.\text{NodeCol} \setminus \mathbf{n}_i.\text{NodeCol}|$. Now consider the S_t obtained at $\mathbf{n}_t$ by greedy search. By the Lemma A.1, $|S_t| \leq |\mathbf{n}_0.\text{NodeCol}| + \sum_{i=1}^{t} |\mathbf{n}_i.\text{NodeCol} \setminus \mathbf{n}_{i-1}.\text{NodeCol}| = k_t = |S^\star|$. Since $|S^\star|$ is optimal, $S_t \geq |S^\star|$, and hence $|S_t| = |S^\star|$ as desired. ■

Appendix B

In this appendix, we give a brief introduction of the radius of the bounding circle for different types of grippers. As shown in Fig. 17, some grippers like Franka Panda have a large base, which affects object movements when the object is grasped by the gripper. Thus, we set half of the base length r_{base} as the radius of bounding circle r_b for this case. For the gripper without a large base, the radius of the target object is $r_b = r_g + r_f$, where r_g is the radius of the target object and r_f is the thickness of the gripper finger.

References

[1] M. Geoffrey, "Thirsty? there's a robot for that," [Online]. Available: https://www.stuff.co.nz/technology/8741807/Thirsty-Theres-a-robot-for-that

[2] C. Nam, J. Lee, S. H. Cheong, B. Y. Cho, and C. Kim, "Fast and resilient manipulation planning for target retrieval in clutter," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 3777–3783.

[3] C. Nam, S. H. Cheong, J. Lee, D. H. Kim, and C. Kim, "Fast and resilient manipulation planning for object retrieval in cluttered and confined environments," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1539–1552, Oct. 2021.

[4] K. Hauser, "The minimum constraint removal problem with three robotics applications," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 5–17, 2014.

[5] D. Nieuwenhuisen, A. F. van der Stappen, and M. H. Overmars, "An Effective Framework for Path Planning Amidst Movable Obstacles," in *Algorithmic Foundation of Robotics VII*. Berlin, Germany: Springer, 2008, pp. 87–102.

[6] J. Van Den Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha, "Path planning among movable obstacles: A probabilistically complete approach," in *Algorithmic Foundation of Robotics VIII*. Berlin, Germany: Springer, 2009, pp. 599–614.

[7] G. Wilfong, "Motion planning in the presence of movable obstacles," *Ann. Math. Artif. Intell.*, vol. 3, no. 1, pp. 131–150, 1991.

[8] E. D. Demaine, M. L. Demaine, and J. O'Rourke, "Pushpush and push-1 are np-hard in 2D," in *Proc. Annu. Can. Conf. Comput. Geom.*, 2000, pp. 211–219.

[9] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2007, pp. 3327–3332.

[10] P. C. Chen and Y. K. Hwang, "Practical path planning among movable obstacles," Sandia National Labs., Albuquerque, NM (USA), Tech. Rep. SAND-90-2383C, 1990.

[11] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *Int. J. Humanoid Robot.*, vol. 2, no. 4, pp. 479–503, 2005.

[12] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," *Int. J. Robot. Res.*, vol. 27, no. 11–12, pp. 1295–1307, 2008.

[13] O. Ben-Shahar and E. Rivlin, "To push or not to push: On the rearrangement of movable objects by a mobile robot," *IEEE Trans. Syst., Man, Cybern., Part B. (Cybern.)*, vol. 28, no. 5, pp. 667–679, Oct. 1998.

[14] M. R. Dogar and S.S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Auton. Robots*, vol. 33, no. 3, pp. 217–236, 2012.

[15] J. A. Haustein, J. King, S.S. Srinivasa, and T. Asfour, "Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 3075–3082.
[16] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2014, pp. 639–646.
[17] M. Moll et al., "Randomized physics-based motion planning for grasping in cluttered and uncertain environments," *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 712–719, Apr. 2018.
[18] R. Wang, Y. Miao, and K. E. Bekris, "Efficient and high-quality prehensile rearrangement in cluttered and confined spaces," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 1968–1975.
[19] R. Wang, K. Gao, D. Nakhimovich, J. Yu, and K. E. Bekris, "Uniform object rearrangement: From complete monotone primitives to efficient non-monotone informed search," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 6621–6627.
[20] A. H. Qureshi, A. Mousavian, C. Paxton, M. Yip, and D. Fox, "NeRP: Neural rearrangement planning for unknown objects," in *Proc. Robot.: SCi. Syst.*, 2021.
[21] J. Lee, Y. Cho, C. Nam, J. Park, and C. Kim, "Efficient obstacle rearrangement for object manipulation tasks in cluttered environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2019, pp. 183–189.
[22] J. Ahn, J. Lee, S. H. Cheong, C. Kim, and C. Nam, "An integrated approach for determining objects to be relocated and their goal positions inside clutter for object retrieval," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 6408–6414.
[23] R. Papallas and M. R. Dogar, "Non-prehensile manipulation in clutter with human-in-the-loop," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 6723–6729.
[24] J. E. King, V. Ranganeni, and S.S. Srinivasa, "Unobservable monte carlo planning for nonprehensile rearrangement tasks," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 4681–4688.
[25] J. Lee, C. Nam, J. Park, and C. Kim, "Tree search-based task and motion planning with prehensile and non-prehensile manipulation for obstacle rearrangement in clutter," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 8516–8522.
[26] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push planning for object placement on cluttered table surfaces," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 4627–4632.
[27] B. Huang, S. D. Han, A. Boularias, and J. Yu, "Dipn: Deep interaction prediction network with application to clutter removal," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 4694–4701.
[28] W. Yuan, J. A. Stork, D. Kragic, M. Y. Wang, and K. Hang, "Rearrangement with nonprehensile manipulation using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 270–277.
[29] M. S. Saleem and M. Likhachev, "Planning with selective physics-based simulation for manipulation among movable objects," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 6752–6758.
[30] F. Paus, T. Huang, and T. Asfour, "Predicting pushing action effects on spatial object relations by learning internal prediction models," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 10584–10590.
[31] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: An Efficient Heuristic for Task and Motion Planning," in *Algorithmic Foundations of Robotics XI*. Berlin, Germany: Springer, 2015, pp. 179–195.
[32] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2020, vol. 30, pp. 440–448.
[33] F. Lagriffoul and B. Andres, "Combining task and motion planning: A culprit detection problem," *Int. J. Robot. Res.*, vol. 35, no. 8, pp. 890–927, 2016.
[34] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson, "Constraint propagation on interval bounds for dealing with geometric backtracking," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 957–964.
[35] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *Int. J. Robot. Res.*, vol. 33, no. 14, pp. 1726–1747, 2014.
[36] A. K. Pandey, J.-P. Saut, D. Sidobre, and R. Alami, "Towards planning human-robot interactive manipulation tasks: Task dependent and human oriented autonomous selection of grasp and placement," in *Proc. 4th IEEE RAS EMBS Int. Conf. Biomed. Robot. Biomechatronics*, 2012, pp. 1371–1376.
[37] L. de Silva, A. K. Pandey, M. Gharbi, and R. Alami, "Towards combining HTN planning and geometric task planning," in *Proc. RSS Workshop Combined Robot Motion Plan. AI Plan. Practical Appl.*, 2013.
[38] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *Proc. Int. Joint Conf. Artif. Intell.*, 2015, pp. 1930–1936.
[39] M. Toussaint, K. Allen, K. Smith, and J. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Proc. Robot.: SCi. Syst.*, 2018.
[40] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Active model learning and diverse action sampling for task and motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 4107–4114.
[41] B. Kim, Z. Wang, L. P. Kaelbling, and T. Lozano-Pérez, "Learning to guide task and motion planning using score-space representation," *Int. J. Robot. Res.*, vol. 38, no. 7, pp. 793–812, 2019.
[42] R. Chitnis et al., "Guided search for task and motion plans using learned heuristics," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 447–454.
[43] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 1255–1262, Apr. 2019.
[44] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint, "Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 9563–9569.
[45] K. K. Hauser, "Minimum constraint displacement motion planning," in *Proc. Robot.: Sci. Syst.*, 2013, vol. 6, p. 2.
[46] A. Krontiris and K. Bekris, "Computational tradeoffs of search methods for minimum constraint removal paths," in *Proc. 8th Annu. Symp. Combinatorial Search*, 2015.
[47] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proc. ICRA. Millennium Conf. IEEE Int. Conf. Robot. Automat. Symposia Proc. (Cat. No 00CH37065)*, 2000, vol. 2, pp. 995–1001.
[48] D. McDermott et al., "PDDL-the planning domain definition language," Yale Center Comput. Vis. Contro, Tech. Rep. CVC TR-98-003/DCS TR-1165, 1998.
[49] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

Hao Tian received the Ph.D. degree from the School of Software Engineering, East China Normal University, Shanghai, China, in 2020.

He is currently a Post-doc with the Centre for Garment Production Limited, Hong Kong, and the Department of Computer Science, University of Hong Kong. His research interests are task and motion planning and robotic manipulation.

Chaoyang Song (Senior Member, IEEE) received the B.Eng. degree in mechanical engineering from Tongji University, Shanghai, China, in 2009, and the Ph.D. degree in mechanism and robotics from Nanyang Technological University, Singapore, in 2014.

From March 2013 to November 2015, he was a Postdoctoral Research Fellow with the Massachusetts Institute of Technology, Cambridge, MA, USA, and the Singapore University of Technology and Design, Singapore. Then, he was a Lecturer (Assistant Professor) with the Department of Mechanical and Aerospace Engineering, Monash University, Clayton VIC, Australia. He is currently an Assistant Professor with the Department of Mechanical and Energy Engineering and the Institute of Robotics, Southern University of Science and Technology, Shenzhen, China. His research interest includes the design science of bionic robotics and robot learning.

Changbo Wang received the Ph.D. degree in computer application technology from the State Key Laboratory of CAD and CG, Zhejiang University, Hangzhou, China, in 2006.

He was a Visiting Scholar with the State University of New York, Stony Brook from 2009 to 2010. He is currently a Professor with the School of Computer Science and Technology, East China Normal University, Shanghai, China. His main research interests include computer graphics, information visualization, and virtual reality.

Xinyu Zhang received the B.S. and M.S. degrees in material science from Zhejiang University, Hangzhou, China, in 1997, 2000, and the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 2004.

He is currently a Professor with the School of Software Engineering, East China Normal University, Shanghai, China. From 2012 to 2013, he was a Research Scientist of computer science and engineering with the University of North Carolina, Chapel Hill, NC, USA He was a Research Professor (2008–2012), a Fulltime Lecturer (2007–2008) and a Postdoctoral Research Fellow (2005–2007) with the Department of Computer Science, Ewha Womans University, South Korea. His research interests include robotics, virtual reality, computer graphics, and geometric modeling.

Jia Pan (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of North Carolina, Chapel Hill, NC, USA, in 2013.

He is currently an Associate Professor with the Department of Computer Science, University of Hong Kong, Hong Kong. He is also a member of the Centre for Garment Production Limited, Hong Kong. His research interests are robotics and artificial intelligence as applied to autonomous systems, particularly for navigation and manipulation in challenging tasks such as effective movement in dense human crowds and manipulating deformable objects for garment automation.