Dealing with Difficult Instances of Object Rearrangement

Athanasios Krontiris and Kostas E. Bekris
Department of Computer Science, Rutgers-the State University of New Jersey, Piscataway, NJ, USA
Emails: tdk.krontir@rutgers.edu, kostas.bekris@cs.rutgers.edu

Abstract—Rearranging multiple objects is a critical skill for robots so that they can effectively deal with clutter in human spaces. This is a challenging problem as it involves combinatorially large, continuous $C$-spaces involving multiple movable bodies and complex kinematic constraints. This work initially revisits an existing search-based approach, which solves monotone challenges, i.e., when objects need to be grasped only once so as to be rearranged. The first contribution is the extension of this technique to a method that addresses many non-monotone challenges. The second contribution is the use of either the monotone or of the new non-monotone method as a local planner in the context of a higher-level task planner that searches the space of object placements and which provides stronger guarantees. The paper aims to emphasize the benefit of using more powerful motion primitives in the context of task planning for object rearrangement than an individual pick-and-place. Experiments in simulation using a model of a Baxter robot arm show the capability of solving difficult instances of rearrangement problems and evaluate the methods in terms of success ratio, running time, scalability and path quality.

I. INTRODUCTION

Robot manipulators must be able to rearrange objects to operate effectively in cluttered human environments. For instance, multiple products may need to be orderly arranged in factory floors or grocery stores. A robotic assistant may need to rearrange objects when tidying up a home or rearrange objects in a fridge when retrieving a refreshment from a fridge which is unreachable. This paper proposes methods for efficiently solving hard instances of such tasks through grasping using a single robotic arm.

Rearrangement is challenging because of the size of the corresponding configuration space ($C$-space) and the involved kinematic constraints. A complete method must operate in the Cartesian product of the robot’s and the objects’ $C$-spaces. The problem becomes harder when the objects are placed in tight spaces, such as shelves, where the arm has limited maneuverability. In these situations, a robot needs to carefully displace objects to reach previously unreachable items. This paper focuses on these combinatorial and geometric aspects of rearrangement. Several other issues arise in the real-world, such as accurate estimation of object locations and robust execution of grasps, which are not the focus of this effort.

Motivating work in manipulation planning [46] describes a backtracking search method for detecting the sequence of objects to be moved so as to reach a desirable, previously unreachable object. To reduce the size of the search space, the method focuses on monotone problems, where each object is allowed to be moved only once. When it is able to solve problems, it is able to do so relatively fast. After exhausting all possible orders for moving objects, it can report that it cannot find a solution. In this way, it is a useful primitive for identifying and solving simple instances.

Non-monotone challenges are recognized as hard rearrangement instances. There are recent approaches for rearrangement that can deal with the non-monotone case under certain conditions [25, 43, 19, 32]. One method simplifies the problem by requiring that all objects are unlabeled, i.e., interchangeable and can occupy any pose in their final arrangement [32]. An alternative imposes a grid for placing the objects and then uses techniques, such as answer-set programming [25]. Others view the rearrangement problem as an instance of general integrated task and motion planning [43] and, in this context, they evaluate good heuristics for integrated planning [19].

This paper is inspired by these efforts to propose simple but efficient solutions for hard instances of general object rearrangement. Specifically, hard problems correspond to:

1. non-monotone instances, where an object needs to be grasped multiple times to achieve the final arrangement;
2. unique, labeled objects, which need to occupy specific poses in the final arrangement;
3. tight, cluttered spaces, where it is not easy to displace objects to free space and bring them back to the desired arrangement. This is critical for static arms but also for minimizing the motion of mobile manipulators.

This paper first extends the backtracking search approach for monotone problems [46] to non-monotone instances. Every
time the method considers an object with a blocked path to its
goal, it tries to clear the path by finding appropriate interme-
diate poses for blocking objects. While completeness cannot
be guaranteed beyond tabletop challenges using overhand
grasps, experiments indicate that many challenging instances
in cluttered spaces are solved by this extension.

To address the general case, this paper proposes the use
of the monotone or the non-monotone backtracking search
approach as a local planner in the context of a higher-level
task planner for rearrangement. The task planner used in
the accompanying experiments searches the space of object
arrangements by building a roadmap, similar to PRM [28]. The
nodes correspond to object placements in the world, which are
connected with edges when the local planner for the robotic
arm can transfer objects between the two nodes.

A traditional local planner would be an individual pick-and-
place, where a single object is displaced between two nodes. Fig. 2 shows that the success ratio of solving rearrangement
problems using pick-and-place goes down quickly as the number
of objects increases.

The key insight is that the proposed powerful local planners,
especially the non-monotone primitive, can solve hard
instances more reliably, faster and with better path quality,
.i.e., fewer objects grasped during the rearrangement. Its
integration with the PRM-based task planner results in a
method that can solve effectively hard instances and achieves
probabilistic completeness.

To significantly speed up online query resolution, it is pos-
sible to take advantage of appropriate preprocessing. Transfer
and transit roadmaps can be pre-built for the arm given the
static geometry and reused for the various objects and their
placement. The idea is related to the “conditional reachability
graph” proposed recently [19] and the “manipulation graph”
approach [40]. While not evaluated here, it is also possible
to define uni- or bi-directional tree versions of the high-level
planner [33] or follow a heuristic search procedure [19].

Simulated experiments using a model of a single Baxter arm
evaluate the methods in a variety of non-monotone, labeled
rearrangement problems, including setups in restricted spaces,
such as objects in shelves. The experiments show that the non-
monotone primitive can solve problems not easily addressable
by other alternatives. The experiments also reveal the required
computation time for finding a solution, the scalability as the
number of objects increases and resulting path quality of the
methods. Smoothing is used to further improve the quality of
the path followed by the arm for a rearrangement solution
given additional computation time.

II. RELATED LITERATURE

Rearrangement planning [4, 38] relates to various lines of
work in the robotics literature.

Planning among Movable Obstacles: A related chal-
lenge is the problem of navigation among movable obsta-
cles (NAMO), which was shown to be NP-hard [51], even
for simple instances with unit square obstacles [13]. Thus,
mmost efforts have dealt with efficiency [9, 37] and provide
completeness results only for problem subclasses [44, 45].
A probabilistically complete solution was proposed [48], but
works only for simple robots (2-3 DOFs). More recently, a
decision-theoretic framework was presented for NAMO, which
deals with the inherent uncertainty in real tasks [34]. A related
problem is the minimum constraint removal problem, which
seeks to minimize the number of constraints/obstacles along
a path [21, 31]. For this problem an asymptotically optimal
solution has been achieved [22] but the work does not capture
negative interactions between obstacles as in the current work.

Manipulation: Planning for high-DOF robotic arms can be
approached with a “manipulation graph” that contains “transit”
and “transfer” paths [1, 2, 40]. The graph can be constructed
with sampling-based planners [28, 33]. The current paper
employs a similar framework for precomputing manipulation
paths for the arm and employs asymptotically near-optimal
sampling-based planners for constructing the manipulation
graph [15, 36], i.e., roadmap spanners of PRM* [27].

Tree sampling-based planners have also been used suc-
cessfully for manipulation planning [5, 6]. A variety of ap-
proaches exist for manipulation planning beyond sampling-
based planners, which could also be employed in the context
of the proposed methods, such as heuristic search [10], or
optimization-based methods, e.g., CHOMP [54, 29].

Manipulation planning among multiple movable obstacles
has been considered for “monotone” problems where each
obstacle can be moved at most once [46]. The current work,
however, can reason about more complex challenges. As-
sembly planning similarly solves multi-body problems but it
focuses on separating a collection of parts and typically the
robot path is ignored [52, 20, 47].

Another paradigm for dealing with cluttered scenes involves
non-prehensile manipulation, such as pushing [11, 16], which
is not addressed in this work, but is a potential extension.

Task and Motion Planning: Rearrangement planning can
be seen as an instance of integrated task and motion planning
[8, 39, 43, 19]. Many approaches employ a high-level symbolic
planner [26, 18]. For instance, geometric constraints can
be incorporated into the high-level language [17], or it is
possible to plan in the cross product of the high-level symbolic
reasoning and the low-level configurations [8]. Multi-modal
roadmaps can deal with the combination of both discrete and
continuous parameters [7, 23, 24]. Recent methods generate
heuristics for symbolic manipulation planning with roadmaps
[19], or discover a symbolic language for manipulation on the
fly [30].
III. PROBLEM SETUP AND NOTATION

Consider a 3D workspace that contains obstacles and:
- A set of \( k \) movable rigid-body objects \( O \), where each object \( o_i \) in \( O \) can acquire a pose \( p_i \) in \( P_i \subseteq SE(3) \). An arrangement \( a \) in \( A \) specifies \( k \) poses \( \{p_1, \ldots, p_k\} \) for the objects in \( O \), where \( p_i \in P_i \). \( O' \subset O \) indicates the poses that the objects \( O' \subset O \) occupy according to \( a \).
- An arm able to move objects acquiring arm configurations \( q \) in \( Q \). Then, \( q(p_i) \) is an arm configuration grasping object \( o_i \) at pose \( p_i \) computed with inverse kinematics.

The \( C \)-space of the rearrangement problem \( Q = Q \times A \) is the Cartesian product of the arm’s \( C \)-space and arrangement space.

There are two different types of collision-free configurations:
- Stable: All objects rest on surfaces. ii) Grasping: An object is grasped and the others are resting. Then, valid configurations correspond to the union of stable and grasping configurations, while transition configurations are both stable and grasping, i.e., in transition configurations, one object is grasped and resting on a surface, while the rest are just resting.

Furthermore, there are \( k + 1 \) different modes \( m \in M \):
- A single transit mode: The arm is not carrying an object and the problem’s configuration is stable. This corresponds to a planning primitive:
  
  \[
  \text{TRANSIT}(q_I, p_F, \alpha),
  \]
  which computes a collision-free path from arm configuration \( q_I \) to a grasping configuration \( q(p_F) \) for a pose \( p_F \) of an object given arrangement \( \alpha \).
- \( k \) different transfer modes: The problem is in a grasping configuration for object \( o_i \), which corresponds to the primitive:
  
  \[
  \text{TRANSFER}(o_i, p_I, p_F, \alpha),
  \]
  It computes a path to transfer an object \( o \) from pose \( p_I \) to pose \( p_F \) without collisions with the objects \( O \setminus o_i \) in poses specified by arrangement \( \alpha \).

A legal mode change is between the transit mode and one of the transfer modes or vice versa, only if the final configuration of the previous mode and the first configuration in the next mode correspond to the same transition configuration.

Then, the problem’s state space is defined as \( X : Q \times M \), where its valid subset \( X^v \) is defined for valid configurations and its stable subset \( X^s \) is defined for stable configurations.

A rearrangement path \( \pi \in \Pi : [0, 1] \rightarrow X^v \) is a continuous sequence of alternating stable/transit and grasping/transfer sets of states in \( X^v \) with legal mode changes.

Prehensile Rearrangement Problem: Given an initial state \( x_I = ((q_I, \alpha_I), t_I) \in X^s \) and a final state \( x_F = ((q_F, \alpha_F), t_F) \in X^s \), compute a rearrangement path \( \pi \in \Pi : [0, 1] \rightarrow X^v \), such that \( \pi(0) = x_I \) and \( \pi(1) = x_F \).

IV. FOUNDATIONS

Motivation: This section focuses on a previously proposed approach for manipulating objects [36] that deals with monotone problem instances. A monotone path moves each object at most once. Fig. 3 (left) provides an example of a monotone problem ignoring the arm. For such problems, the algorithm [36] performs a backtracking search in the space of possible orders of transferring objects directly to their final poses. Fig. 3 (right) shows possible orders considered given object \( B \) is moved first, which results in a solution.

Algorithm: The “monotone Rearrangement Search” algorithm (mRS) is given in Alg. 1 and is an adaptation of an existing manipulation approach for clearing a path to an unreachable object [45]. The original approach was searching backwards to clear a path to an unreachable object but in the case of rearrangement the two search directions are equivalent and forward search is easier to understand. The method receives:
- an object \( o \) to be transferred given that the manipulator’s last configuration was \( q \),
- the set of objects \( O_R \) (“remaining objects”) not yet moved to a final pose along the current branch of the search tree,
- the current and final arrangements: \( \alpha_C \) and \( \alpha_F \).

Algorithm 1: mRS \( (o, q, O_R, \alpha_C, \alpha_F) \)

1. \( \pi_N \leftarrow \text{TRANSIT}(q, \alpha_C[o], \alpha_C) \);
2. \( \pi_M \leftarrow \text{TRANSFER}(o, \alpha_C[o], \alpha_F[o], \alpha_C) \);
3. if \( \{\pi_M \mid \pi_N \} \) is collision free then
   4. \( \alpha_C[o] \leftarrow \alpha_F[o] \);
   5. if \( O_R = \emptyset \) then
      6. return \( \pi_M \);
   7. for each \( o_r \in O_R \) do
      8. \( \pi \leftarrow \text{mRS}(o_r, q(\alpha_F[o]), O_R \setminus o_r, \alpha_C, \alpha_F) \);
   9. if \( \pi \neq \emptyset \) then return \( \{\pi_M \mid \pi\} \);
10. return \( \emptyset \);

The method first computes paths for grasping the object \( o \) at its current pose \( \alpha_C[o] \) and for transferring it to its final pose \( \alpha_F[o] \) (lines 1-2). If the concatenation of the resulting paths is collision-free (line 3), the object is transferred (line 4). If this was the last remaining object (line 5), then the problem has been solved (line 6). Potentially at this point, the arm can return to its initial or safe configuration. Otherwise, the method recursively calls itself for all remaining objects that have not been transferred to their final pose (lines 7-8). If any of those calls is successful, a solution has been found and the path \( \pi_M \) for object \( o \) can be concatenated to the paths for the remaining objects in \( O_R \) (line 9). The method needs to be initially called for all possible objects, i.e., call \( \forall o : \text{mRS}(o, q_I, O \setminus o, \alpha_I, \alpha_F) \).
Properties: Assuming (probabilistic) completeness for the TRANSIT and TRANSFER primitives, the method is trivially (probabilistically) complete for monotone challenges since it exhaustively considers all possible orders for transferring the objects to their final arrangement with monotone paths. Similarly, the method fails if it is necessary to find an intermediate position for an object so as to solve the problem. In the worst case, the method needs to visit a complete search tree and it has to call the TRANSIT and TRANSFER primitives an exponential number of times. While this is bad asymptotic performance - the problem is hard after all - in practice, it can return early, i.e., as soon as a solution has been found. Even if it fails, the number of calls is typically orders of magnitude smaller than the worst case, as most branches will fail early.

This method already solves tabletop problems where the arm can use overhand grasps at both the initial and final poses as long as there is no overlap between these poses. But this requirement is not satisfied by all tabletop challenges, since frequently there will be overlap between the initial and final poses of objects in a tabletop setup. Moreover, for objects in shelves, the arm’s maneuverability is limited, overhand grasps are frequently impossible and non-monotone challenges arise easily. Consider Fig. 4 (left), referred here as “simplified Towers of Hanoi”, where A and B need to be placed in intermediate poses so that A reaches its final pose. A setup like this one can arise when a manipulator has to reach occluded objects in shelves or move them between bins while keeping them in the same order. Such non-monotone challenges motivate the development of the following methods.

V. AN EXTENSION FOR NON-MONOTONE INSTANCES

Motivation: This section extends the previous approach to the case that an object’s path to its final pose is blocked. In many cases it is possible to easily evacuate the blocking objects \( O_b \). Specifically, the method employs a subroutine that searches for a monotone path of the blocking objects \( O_b \) so as to evacuate the path of the blocked object. While the subroutine does not guarantee that it will always find a solution when one exists, experiments suggest that many non-monotone rearrangement problems are addressable in this manner.

For instance, consider the “simplified Towers of Hanoi” example of Fig. 4 and the case object A is at the top of the search tree. The path to A’s final pose is obstructed by B and C. Then a subproblem arises (the top triangle in Fig. 4 (right)): “Move B and C to intermediate poses so that A can move to its final pose”. The intermediate poses for B and C should correspond to the shortest manipulation path among the poses that allow the transfer of object A. In this case, the method will work regardless of the order with which objects B and C are considered. For instance, if B is selected, it can move to the front/open part of the second bin, given A’s path as a constraint. But then it is blocked by C. Then C needs to evacuate both A’s and B’s paths and moves further up the second bin. This results in the configuration at the top right corner of Fig. 4 (right).

At that point objects B and C still have not reached their goal. Then, there are two possible orders to consider. Either B moves first (left subtree) or C moves first (right subtree). If B moves first, the remaining problem is monotone and this branch leads to success. If C moves first, then its path is blocked by B and another subproblem is defined (the lower right triangle in Fig. 4 (right)): “Move B to an intermediate position so that C can then move to its final pose”. If B moves back to the first bin, as is shown in the figure, the subproblem for C is solved but this branch fails, since C is at its goal and blocks the path of B.

If C is selected to evacuate first A’s path, it can be moved in an unobstructed manner to the opening of the second bin. Then, object B still needs to evacuate A’s path. It will find the same position as C to clear A’s path. Its path is blocked by C, which can be pushed further up the second bin so as to allow B to follow its path and the problem will be solved.

Algorithm: The “piecewise linear non-monotone Rearrange- ment Search” algorithm (p1RS) is given in Alg. 2 and directly extends its monotone counterpart.

**Algorithm 2: p1RS**

1. \( \pi_N \leftarrow \text{TRANSIT}(q, o, \alpha_C, O_b \setminus \{C\}); \)
2. \( \pi_M \leftarrow \text{TRANSFER}(o, \alpha_C, O_b \setminus \{C\}); \)
3. if \( \pi_U \leftarrow \{\pi_N \mid \pi_M\} \text{ is collision free then} \)
4. \( \alpha_C \leftarrow \alpha_F \)
5. if \( O_b \equiv \emptyset \) then \( \pi_U \)
6. for each \( o_r \in O_r \) do \( \pi \leftarrow \text{p1RS}(q, o, \alpha_F, O_R \setminus o_r, \alpha_C, O_b); \)
7. if \( \pi \neq \emptyset \) then return \( \{\pi_U \mid \pi\}; \)
8. else \( o_b \leftarrow \text{a blocking object along } \pi_U; \)
9. if \( o_b \in O_R \) then \( \{p, \alpha_C, \pi\} \leftarrow \text{CLEAR}(o_b, q, O_R \setminus o_b, \alpha_C, \pi_U); \)
10. if \( \pi' \neq \emptyset \) then \( \pi \leftarrow \text{p1RS}(q, p, O_R, \alpha_C, \alpha_F); \)
11. if \( \pi' \neq \emptyset \) return \( \{\pi' \mid \pi\}; \)
12. return \( \emptyset; \)
The pRS method has two differences from mRS:

- The transit and transfer paths it computes for object \( o \) must be collision-free only with objects that have been already moved to their target, i.e., in the set \( O \setminus O_R \). For the remaining objects \( O_R \), minimum constraint removal paths are computed [22].
- It does not directly return failure if the path of object \( o \) to its final pose \( \alpha_F[o] \) is in collision with one of the remaining objects \( O_R \) (lines 10-16).

In the case the path is not collision-free, the algorithm considers the first blocking object \( o_b \) (line 11). If \( o_b \) has not been moved to its final pose, i.e., it is in the list of “remaining objects” \( O_R \) (line 12), then a subroutine CLEAR is called to clear \( o_b \)’s path from object \( o_b \) (line 13). The CLEAR function receives the path \( \pi_U \) of object \( o \) as a constraint. If the subproblem of CLEAR can be solved and \( o_b \) can evacuate \( o_b \)’s path (line 14), then the pRS algorithm is called again for object \( o \) (line 15), so as to try again to move it along its path. If this eventually succeeds - by potentially making additional calls to CLEAR- the path is returned (line 16).

Alg. 3 provides the CLEAR function, which first finds an intermediate pose \( p \) for the calling object \( o \) and the corresponding path \( \pi_U \) (line 1). The intermediate pose must be (i) collision-free with the current arrangement \( \alpha_C[O \setminus O_R] \) of objects that have reached their target; and (ii) it does not collide with any configuration along the input constraint paths \( \pi_B \). Among the possible poses, the approach selects the one for which it can compute a minimum constraint removal path from the current arm configuration \( q \) given all objects in \( O_R \). If there are multiple poses with paths that have the same number of minimum constraints, the method returns the one corresponding to the shortest path. Section VI will describe how these reachability computations can be accelerated using precomputation.

**Algorithm 3: CLEAR\( (o, q, O_R, \alpha_C, \pi_B) \)**

```plaintext
1. \{p, \pi_U\} ← INTERMEDIATE_POSE\( (\alpha_C[O \setminus O_R], \pi_B, q) \);
2. if \( \pi_U \) is collision free then
   3. \( \alpha_C[o] ← \alpha_F[o]; \)
   4. return \( \{p, \alpha_C, \pi_U\} \);
else
   6. \( o_b ← \) a blocking object along \( \pi_U \);
   7. if \( o_b ∈ O_R \) then
      8. \{p′, \alpha_C, \pi′\} ← CLEAR\( (o_b, q, O_R \setminus o_b, \alpha_C, \pi_U \cup \pi_B) \);
      9. if \( \pi′ \neq \emptyset \) then
         10. \{p, \alpha_C, \pi\} ← CLEAR\( (o, q(p′), O_R, \alpha_C, \pi_B) \);
      11. if \( \pi \neq \emptyset \) return \( \{p, \alpha_C, \pi′|\pi\} \);
12. return \( \emptyset \);
```

If a collision-free path \( \pi_U \) to an intermediate pose \( p \) is found (line 2), the object is moved there, and its pose, the updated assignment and the corresponding path are returned (lines 3-4). Otherwise, the path is blocked by another object and the function is called recursively for the blocking object, similar to what happened in pRS (lines 5-11).

**Properties:** The mRS approach from the previous section cannot solve tabletop challenges where there is an overlap between the initial and final poses. Nevertheless, it is easy to argue that pRS can solve such challenges where all objects are accessible with overhand grasps under the following sparsity requirement:

- For each pair of poses for object \( o_i \), there is enough space to place the remaining objects in poses that are collision-free among themselves and with \( o_i \)’s poses regardless of the order with which the blocking objects are considered.

Every time the algorithm discovers a path that is blocked under the above assumption, it can always evacuate the blocking agents from this path to a set of collision-free poses. In such setups, every time CLEAR is called from pRS, there will be no recursion needed. For every blocking object there is going to be an available collision-free pose regardless of the order the objects are considered. This property, however, still leaves a large set of non-monotone challenges for which it is not possible to argue that the algorithm will find a solution. The recursive nature of CLEAR, however, is helping in practically addressing a wider set of non-monotone instances as the accompanying experimental section shows.

![Fig. 5.](image)

(left) An example problem where the non-monotone extension of the backtracking search approach fails. (right) The corresponding search tree given object \( B \) as the first object. The subproblem succeeds but then object \( A \) cannot be returned to its target pose.

For the non-monotone problem of Fig. 5 (left), pRS will fail. When object \( A \) is considered first, object \( B \) needs to be cleared. But object \( B \) is not reachable by the arm. When object \( B \) is considered to be moved first, the search tree in Fig. 5 (right) shows that \( A \) can be cleared from object \( B \)’s path but then \( A \) cannot reach its own target. Solving such challenges relates to complete multi robot planning [49][35][50][53][41], which is itself hard. Nevertheless, complete methods will end up coupling objects and try to plan in a composite space, which increases computational cost.

**VI. USE OF THE PRIMITIVES AS LOCAL PLANNERS**

**Motivation:** This section considers the above primitives as local planners within a higher-level task planner to search the space of possible object arrangements and solve problems like the one in Fig. 5. The benefit of these primitives is they can connect an individual arrangement to a relatively large number of different ones. This is an advantage over pick and place, which can only connect arrangements that differ only by a single object pose [19][43]. The non-monotone extension is more powerful but more expensive than the monotone primitive, which is experimentally shown to be advantageous. The high-level task planning process can be performed in many different ways, e.g., through heuristic search [19], or in RRT-like fashion. Here a roadmap, similar to a PRM is used, referred to as ReArrange_PRM and summarized in Alg. 4.
Algorithm: The high-level planner builds a roadmap, where nodes are object arrangements. Initially, the roadmap starts with the initial and final arrangements \( \alpha_I \) and \( \alpha_F \) (line 1). Edges correspond to local rearrangement paths between two nodes. Such paths can be computed either by an individual pick-and-place, or by the monotone primitive \( \text{mpRS} \) or by the non-monotone variant \( \text{pRS} \). While the problem is not solved (line 2), the method samples new random arrangements \( \alpha_{\text{rand}} \) (line 3). Then, based on a distance estimate in the space of arrangements, a set of neighboring arrangements \( A_{\text{near}} \) in the roadmap is returned (line 4). For each neighbor \( \alpha_{\text{near}} \), a connection is attempted between the neighbor and the random arrangement (lines 5-10) with the preferred primitive. In the algorithmic, the non-monotone variant is used (line 7).

Algorithm 4: \textbf{Rearrange\textunderscore PRM}(q, \mathcal{O}, \alpha_I, \alpha_F)

1. \( G \leftarrow \{ \mathcal{V}, \{\alpha_I, \alpha_F\}, \mathcal{E} \leftarrow \{\emptyset\} \} \);
2. while \((\mathcal{W} \leftarrow \text{FIND\_PATH}(G, \alpha_I, \alpha_F)) == \emptyset \) do
3. \( \alpha_{\text{rand}} \leftarrow \text{SAMPLE\_ARRANGEMENT}() \);
4. \( A_{\text{near}} \leftarrow \text{CLOSEST}(G, \alpha_{\text{rand}}) \);
5. for \( \alpha_{\text{near}} \in A_{\text{near}} \) do
6. for each \( q \in \mathcal{O} \) do
7. \( \pi \leftarrow \text{pRS}(q, \mathcal{O} \setminus q, \alpha_{\text{rand}}, \alpha_{\text{near}}) \);
8. \( \mathcal{E} \leftarrow \mathcal{E} \cup \{(\alpha_{\text{rand}}, \alpha), \pi\} \);
9. if \( \pi \neq \emptyset \) then
10. \( \text{BREAK} \);
11. return \( \mathcal{W} \);

Similar to PRM, this approach can be parameterized based on the selection of:

a) Sampling arrangements: On top of uniform sampling, a heuristic choice is to include with certain probability (5%) the selection of object poses either from the initial or the goal arrangement.

b) Distance metric: An estimation of the number of necessary transfers between two input arrangements can be used (e.g., through problem relaxations). The implementation uses the sum of the distances between poses.

c) Number of neighbors: The value from PRM* [27] and a linear search for closest neighbors were used. If path quality is not a priority, then if two arrangements are already in the same connected component, they do not need to be connected. Alternatively, a roadmap spanner can be applied where only useful edges in terms of path quality are considered [36].

Properties: To keep the arguments simple, consider a version of \textbf{Rearrange} – PRM, where uniform sampling is used and all roadmap nodes are returned as neighbors. Then, consider a solution path \( \pi \) for a prehensile rearrangement problem \( (X, x_1, x_F) \). The solution path can be decomposed into a sequence of segments \( \{\pi_1, \pi_2, \ldots, \pi_m\} \), where each \( \pi_i \) corresponds to a \text{TRANSIT} and a \text{TRANSFER} operation for an object. Denote the object moved during path segment \( \pi_i \) as \( \alpha_i \). The start state of \( \pi_i \) is \( x_{i-1} = ((q_{i-1}, \alpha_{i-1}), \text{TRANSIT}) \) and the final state is \( x_i = ((q_i, \alpha_i), \text{TRANSFER}) \), where \( q_i = q(\alpha_i[x_0]) \) (and \( q_0 = q_I \)).

Given the algorithm samples the start arrangement \( \alpha_{i-1} \) of \( x_{i-1} \) and the final arrangement \( \alpha_i \) of \( x_i \) for segment \( \pi_i \), it will manage to connect them using any of the considered primitives (pick-and-place, \text{mpRS} or \text{pRS}). This is because connecting the two states \( x_i \) and \( x_{i-1} \) can be achieved with an individual pick-and-place, since all objects are in the same pose, with the exception of \( o_i \). Since all arrangements can be eventually sampled and they can be pair-wise connected with any of the primitives, the approach will eventually generate the solution path \( \pi \) in every case.

This raises the question of why should one use the rearrangement search primitives since the method is probabilistically complete even for a pick-and-place local planner. For a pick and place to succeed, however, it has to be that the two arrangements are different only by a single pose. This relates to the probability of sampling the right sequence of arrangements in PRM. Sampling in continuous space for two arrangements of \( k \) objects that have the same \( k - 1 \) poses has probability 0. Nevertheless, the search primitives can work successfully in connecting pairs of arrangements that do not share any pose. For instance, the \text{mpRS} primitive will succeed, if there is a \( \delta \)-ball around each arrangement \( \alpha_i \) along a segment \( \pi_i \) so that for all \( \alpha_i' \) in the \( \delta \)-ball: i) the path segment \( \pi_i' \) connecting arrangements \( \alpha_{i-1}' \) to \( \alpha_i' \) is also part of a solution path \( \pi' \), and ii) the path segment \( \pi_i' \) remains monotone. The probability a local segment \( \pi_i' \) to be monotone has positive probability.

The issue with pick-and-place can be addressed by restricting the problem to a discrete set of poses and give up on prob. completeness, which is actually a practical way of taking advantage of preprocessing. Alternatively, one can consider a tree-based approach instead of the PRM, where given an input arrangement a new arrangement is generated with the pick and place primitive. The problem is, however, that the number of possible new arrangements that can be generated at each step of the algorithm is small, i.e., equal to the number of objects \( k \). On the other hand, the rearrangement search primitives can connect a continuum of arrangement pairs.

The resulting tradeoff is computational in nature. Consider a different decomposition of the same solution \( \pi \) into segments \( \{\pi_1, \pi_2, \ldots, \pi_n\} \), where each segment \( \pi_i \) corresponds to a monotone subproblem. Such a decomposition exists, because the initial pick-and-place decomposition exists and is also monotone. The length of the pick-and-place decomposition \( m \) is typically significantly greater than the length \( n \) of the monotone one, which in turn is greater than a decomposition to subproblems of \text{pRS}. This means that a significantly larger number of arrangements needs to be sampled and connected until the pick-and-place sequence is found. On the other hand, there is an increased cost of generating a monotone segment \( \pi_i' \) versus generating a pick-and-place segment \( \pi_i \), since it involves evaluating a larger number of transit/transfer paths. This cost has to be also paid for failed connection attempts. As the experiments accompanying this paper show, and as the authors’ experience suggests, this tradeoff turns out to be in favor of the more expensive connection primitive.
In order to speed up online query resolution, it is possible to preprocess a scene given the arm’s placement, the static obstacles and the geometry of the movable obstacles.

Preprocessing: The first step is the sampling of a discrete set of useful stable poses \( \mathcal{P} \) reachable by the arm for the movable obstacles. These are poses that can provide transition states. For each pose, multiple grasps for the end-effector are sampled. Then through inverse kinematics it is possible to get the corresponding arm configurations.

These configurations are used as seeds during the generation of sampling-based roadmaps \([27]\). One roadmap is built for each mode of the rearrangement problem. One roadmap is constructed for the transit mode, which corresponds just to the arm moving in the environment and avoiding collisions with the obstacles. Then, one roadmap is built for each transfer mode and corresponds to the arm grasping one of the objects. If all of the \( k \) movable objects have different geometries, then \( k + 1 \) pairs of transit and transfer roadmaps are generated. If multiple objects share the same geometry, they can use the same roadmap. Thus, for geometrically similar objects, a single pair of roadmaps is sufficient.

Then, it is possible to precompute collision information and minimize the cost of collision checking during the online phase. Objects are placed in the poses \( \mathcal{P} \) and then for each edge of the roadmap, the set of object poses that lead into collisions are discovered and stored. This type of precomputation is similar to the “conditional reachability graph” data structure \([19]\).

Query Resolution: During the online operation of the methods, every time that a TRANSIT and a TRANSFER primitive are executed, a multi-goal \( A^\ast \) algorithm is executed on the corresponding precomputed roadmaps \([12] [14]\). The multiple goals for the \( A^\ast \) correspond to multiple potential grasps for the pose of the object. During this process, when an edge of the roadmap is explored, instead of performing collision checks, the set of colliding poses for the edge discovered during the precomputation are tested against the poses in the current arrangement \( \alpha_C \). If there is any overlap, then the edge is in collision. In the case of pLRS, the same information can also facilitate the computation of minimum constraint removal paths.

If the poses corresponding to the initial and final arrangement of a prehensile manipulation problem are known during the offline phase, then no actual collision checks need to be performed during online query resolution. Otherwise, collision checking needs to be performed only for these poses. Any pose that is used as an intermediate free pose in INTERMEDIATE_POSE or for SAMPLE_ARRANGEMENT can come from the list of precomputed poses.

Considering only the precomputed poses affects the method’s probabilistic completeness as it will have to operate over only a discrete set of poses. To provide with prob. completeness in this setup, if the algorithm fails to find a solution given the precomputation, then the set of considered poses for the objects needs to be augmented online.

Experimental Setup: The methods have been tested in 3 workspaces with a model of a Baxter arm: “grid@tabletop” (Fig. 6 (top)), “grid@shelf” (Fig. 6 (bottom)) and “RSS challenge” (Fig. 1). The “RSS challenge” involves 6, 11 or 16 boxes placed on a tabletop. Initially the arrangement is random and the objective is to form the characters: “R”, “RS” and “RSS”. The “grid@tabletop” benchmark places 2 to 14 objects on a tabletop. The “grid@shelf” challenge has 2 to 8 cylinders placed in a shelf that limits the arm’s reachability and does not allow overhand grasps. In both cases, the objective is to rearrange the objects from a random to a grid arrangement. Four different methods are tested: (a) mRS, (b) pLRS, (c) the Rearrange_PRM approach with mRS and (d) the Rearrange_PRM with pLRS. The time limit provided to the methods was 30 mins. 20 experiments were performed for each combination of method and environment.

![Fig. 6. (top) The “grid@tabletop” problem. An initial and the final setup. (bottom) The “grid@shelf” problem. An initial and the final setup.](image)

Results: Fig. 7 provides the results for the “RSS challenge” and the two primitives. While for 6 objects, both methods were able to find a solution because the challenge is monotone, the success rate of the monotone solver goes down quickly with additional objects. The non-monotone solver, however, manages to solve all problems. In terms of running time, there is an increase for pLRS as the number of objects increases but overall it performs good across the board. The Rearrange_PRMs mimic the running times of their primitives.

![Fig. 7. (left) Success ratio in the “RSS challenge”: pLRS always succeeds. (right) Execution time for successful runs for 6, 11 and 16 objects.](image)
difference is more significant in the larger-scale examples but the monotone solver fails in most cases there. The running time of the Rearrange_PRMs tracks those of the primitives as they typically succeed because the first connection works.

Fig. 8. (top left) Success ratio for the different methods in the “grid@tabletop” benchmark. Rearrange_PRM(p1RS) has the best success ratio. (remaining) Execution time for successful runs for 4, 8 and 14 objects.

Fig. 9 provides the results for “grid@shelf”. Here the arm has reduced reachability and is not able to use overhand grasps. As the number of objects increases, the monotone solver’s success ratio goes down quickly. The non-monotone primitive has good success ratio, despite the difficulty of the challenge and the limited reachability. Using the non-monotone primitive within Rearrange_PRM results in even better success rate. Only two runs were not completed for 8 objects. As Fig. 2 shows, using pick-and-place as the local planner in this case results in significantly lower success ratio.

There is an expected increase in running time when transitioning from the monotone to the non-monotone solver. The Rearrange_PRM(mRS) method requires many nodes to solve 6 object challenges resulting in a high average time in this case. The Rearrange_PRM(p1RS) method has an increased cost for 8 objects but these are the hardest problems. The graphs show the average time only for successful runs. The Rearrange_PRM(p1RS) manages to solve problem instances that the other methods did not manage to address within the time limit. These harder instances lead to increased averages.

Table [10] provides the average path length found for the “grid@tabletop” problem. The outcome is similar across the methods with the exception of using pick-and-place actions. For larger-scale problems, the Rearrange_PRM(p1RS) performs effectively in terms of path quality.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>mRS</td>
<td>18.29</td>
<td>35.55</td>
<td>44.12</td>
<td>N/A</td>
</tr>
<tr>
<td>p1RS</td>
<td>15.76</td>
<td>31.36</td>
<td>57.15</td>
<td>78.06</td>
</tr>
<tr>
<td>PRM(mRS)</td>
<td>18.16</td>
<td>43.87</td>
<td>84.34</td>
<td>N/A</td>
</tr>
<tr>
<td>PRM(p1RS)</td>
<td>15.85</td>
<td>31.36</td>
<td>58.79</td>
<td>59.81</td>
</tr>
<tr>
<td>PRM(Pick&amp;Place)</td>
<td>28.18</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Fig. 10. Average path duration in seconds for the “grid@shelves” problem.

IX. DISCUSSION

This work proposes a primitive for rearrangement, which provides improved connectivity among object arrangements relatively to pick-and-place actions. The method extends an existing technique [46] to non-monotone problems. It is integrated with a higher-level planner, which uses the proposed primitive as a local planner to connect object arrangements, and achieves probabilistic completeness. Experiments show that the proposed primitive solves many non-monotone challenges by itself. The integration with the higher-level task planner results in the most efficient solution in terms of success ratio, path quality and scalability, especially in setups with limited reachability, such as shelves. Typically, few arrangements are sampled by the high-level planner to solve relatively hard instances when the primitive is failing by itself.

The method should be tested with different geometry objects, general grasps and object poses, which in principle it can already address. A more formal study of p1RS’s properties is also desirable, as well as task planning alternatives to PRM. Preliminary indications with a bidirectional RRT show reduced computation time and similar performance between the primitives. For RRT, it makes sense to generate versions of the primitives that do not connect two arrangements but partially extend an initial towards a target one.

Using two arms can simplify a problem, as one of them can grasp an object to clear the scene and the other can perform transfers. The current setup can also be integrated with non-prehensile [11, 16] and mobile manipulation [19]. Pushing can easily replace some grasps. Mobility does not significantly alter the combinatorial aspects of the problem. Cloud computing can also be considered to improve performance through parallelization [3]. An alternative but important direction is to adapt complete but efficient multi-robot planning algorithms in the context of manipulation [50, 42]. Future efforts should also focus on the computation of robust rearrangement trajectories given actuation and observation noise.

ACKNOWLEDGMENTS

This work is supported by NSF awards IIS-1451737 and CCF-1330789. Any opinions or findings expressed in this paper do not necessarily reflect the views of the sponsors. The authors would like to thank the anonymous RSS reviewers for their comments.


