Lazy Rearrangement Planning in Confined Spaces

Rui Wang, Kai Gao, Jingjin Yu and Kostas Bekris

Rutgers University, The State University of New Jersey
{rw485, kg627, jy512, kb572}@cs.rutgers.edu

Abstract

Object rearrangement is important for many applications but remains challenging, especially in confined spaces, such as shelves, where objects cannot be accessed from above and they block reachability to each other. Such constraints require many motion planning and collision checking calls, which are computationally expensive. In addition, the arrangement space grows exponentially with the number of objects. To address these issues, this work introduces a lazy evaluation framework with a local monotone solver and a global planner. Monotone instances are those that can be solved by moving each object at most once. A key insight is that reachability constraints at the grasps for objects’ starts and goals can quickly reveal dependencies between objects without having to execute expensive motion planning queries. Given that, the local solver builds lazily a search tree that respects these reachability constraints without verifying that the arm paths are collision free. It only collision checks when a promising solution is found. If a monotone solution is not found, the non-monotone planner loads the lazy search tree and explores ways to move objects to intermediate locations from where monotone solutions to the goal can be found. Results show that the proposed framework can solve difficult instances in confined spaces with up to 16 objects, which state-of-the-art methods fail to solve. It also solves problems faster than alternatives, when the alternatives find a solution. It also achieves high-quality solutions, i.e., only 1.8 additional actions on average are needed for non-monotone instances.

Introduction

Rearrangement in confined spaces is a critical robot skill in setups such as aligning objects in shelves (Fig. 1), object retrieval and part assembly. These are harder setups than the less constrained tabletop case, where all objects can be directly accessed with top-down grasps and then lifted sufficiently high to avoid collisions. Tabletop rearrangement allows to ignore robot-object and object-object interactions, which leads to faster solutions and stronger guarantees. In the confined setup considered here, the robot arm has limited space to maneuver and cannot necessarily access all objects at any point, since top-down grasps are not available. Solving such tasks becomes harder even for a few objects, let alone maintaining high efficiency and good solution quality.

Many existing solutions for confined rearrangement using prehensile primitives follow a similar high-level strategy (Stilman et al. 2007; Wang et al. 2021; Wang, Miao, and Bekris 2022). They build a global tree \( T_G = (V, E) \) (Fig. 2(a)) where a node \( v \in V \) (circles in Fig. 2(a)) represents an arrangement state \( \alpha_v \) of the objects and an edge \( e(v, u) \in E \) (arrows in Fig. 2(a)) represents a collision-free arm path so as to pick-and-place an object to transition from arrangement \( \alpha_v \) to arrangement \( \alpha_u \). The pick-and-place path can be computed by calling a motion planner. For non-monotone problems, some objects need to be moved to an intermediate location different from both the object’s start and goal. For these instances, the global tree can be built hierarchically by concatenating subtrees \( T_{sub} \) returned by a local monotone solver as partial solutions until the final arrangement \( \alpha_F \) is connected to the initial arrangement \( \alpha_I \), as in Fig. 2(a). A basic monotone Rearrangement Solver (mRS) (Stilman et al. 2007) explores all possible object orderings that connect \( \alpha_I \) to \( \alpha_F \) via backtracking search (Fig. 2(c)). It faces scaling issues due to its \( O(n!) \) time complexity. Dynamic programming solutions, such as DFSap (Wang et al. 2021), observe that solving a subproblem from an arrangement \( \alpha \) (i.e., moving the remaining objects to their goals) does not depend on how \( \alpha \) is reached from \( \alpha_I \), i.e., the ordering with which the objects reach \( \alpha \) does not change the feasibility of the subsequent task \( \alpha \rightarrow \alpha_F \). In the toy example of Fig. 2(b), no solution is found from \( \alpha \) obtained from the branch \( o_1 \rightarrow o_2 \rightarrow o_3 \). This can also quickly invalidate the branch \( o_1 \rightarrow o_3 \rightarrow o_2 \) (Fig. 2(d)). Therefore, an arrangement state is sufficient to represent all orderings of objects, which re-

---

1i.e., using grasping, in contrast to non-prehensile primitives, such as pushing (King et al. 2015; King, Ranganeni, and Srinivasa 2017; Papallas and Dogar 2020; Vieira et al. 2022).
Figure 2: (a) A typical tree for non-monotone rearrangement problems, which grows incrementally local trees ($T_{sub}$, $T'_{sub}$), which are monotone, i.e., an object is moved at most once and only to its goal at the final arrangement $\alpha_F$. The edge connecting the two trees (dark blue arrow) corresponds to moving an object to an intermediate location. (b) A toy example with 5 objects (start positions are denoted as solid circles while goal positions as dashed circles). The goal position of $o_5$ after the placement of $o_2$. (c-e) The tree corresponding to mRS (c), DFS (d) and CIRS (e) on the toy example.

sult in the same placements. Given this observation, $\text{DFS}_{\text{op}}$ solves monotone problems with complexity $O(2^n)$, a significant improvement over $O(n!)$.

Nevertheless, neither mRS, nor $\text{DFS}_{\text{op}}$ take advantage of constraint reasoning to further prune the search space. As robot-object constraints arise often in confined and cluttered setups, such reasoning can increase efficiency. In the toy example of Fig. 2(b), the goal position of $o_2$ makes $o_5$ unreach-able, which can be identified just by considering robot configurations for grasping $o_5$. Therefore, moving $o_2$ should be treated as an invalid action for a monotone solution at any arrangement state where $o_5$ is still at its start. Such constraints can significantly prune the search space as in Fig. 2(e). Such constraints were first considered in prior related work (CIRS) (Wang, Miao, and Bekris 2022) but not comprehensively.

The current work focuses on utilizing such constraints to reduce the computational overhead due to the most expensive primitive, i.e., the pick-and-place motion planning calls for each edge of the search tree, which involve collision checking. All the aforementioned methods call a motion planner for every edge of the search tree during its generation to check if the transition to the child arrangement is valid. Consequently, the number of motion planning queries is equivalent to the total number of edges in the search tree, which grows exponentially as the number of objects increase. The objective here is to avoid this combinatorial number of calls to motion planning, while still finding any solution that existing methods can discover and maintaining high-quality solutions, i.e., not requiring many intermediate locations for non-monotone problems.

If the approach detects early that there is no way to reach the final arrangement from a tree node, then significant time can be saved by not performing motion planning on edges out of that node. And it is possible to identify the infeasibility of arrangements quickly given only the objects’ locations and the robot’s grasps, i.e., if a set of objects in an arrangement blocks the grasps of another object at its start or its goal, then there is no arm path that can perform the pick-and-place at the corresponding arrangement (up to the resolution of grasps). These constraints are referred here as reachability constraints. They arise often and dominate the feasibility of rearrangements in confined spaces. Moreover, computing reachability constraints is relatively easy as grasps can be obtained by inverse kinematic (IK) solvers in a hundredth of a second; much faster than a motion planning routine for a pick-and-place sequence. In addition, given $k$ such grasps and total $n$ objects, those IK operations are performed $kn \times (O(n))$, which is much smaller than the worse case for the size of the search tree ($O(2^n)$).

Overall, this work proposes a lazy evaluation framework for rearrangement in confined spaces. It uses reachability constraints to quickly reveal the dependencies between objects without performing expensive motion planning queries during the search tree generation. Thus, this work:

1. **Proposes an efficient, lazy local monotone solver that utilizes reachability constraints.** Given the constraints that arise from grasp reachability, the method first builds a lazy search tree that respects these constraints without verifying edge validity via motion planning. It only performs such verification when a promising solution is found.

2. **Proposes a global planner for non-monotone problems with high success rate, efficiency and quality.** The global planner loads the local search tree from the monotone solver and explores how to move objects to intermediate spots, from where a local solver can be called to discover monotone solutions to the goal. Edges are checked lazily only if needed for moving an object to an intermediate position.

The overall framework is demonstrated to scale up to 16-object instances in confined spaces, which state-of-the-art alternatives fail to solve. The method significantly improves computational efficiency while maintaining high solution quality, i.e., only 1.8 additional actions are needed to fulfill non-monotone tasks on average. The solution of the proposed method is executed on a real robotic system.
Related Work

Object retrieval in confined setups requires relocating some objects to retrieve an object. The challenges lie in (1) partial observability (Xiao et al. 2019), (2) what and where to relocate obstacles (Ahn et al. 2021) and (3) which robot relocates which object (Ahn, Kim, and Nam 2021). The rearrangement problem considered here treats each object as a target to be relocated from its initial to final position and tends to be harder than object retrieval of a single target.

Rearrangement planning without violating constraints can be modeled as a constraint satisfaction problem (CSP), where a variable corresponds to a moving action at a time step and the value domain refers to a set of objects to move at that time step. The solution corresponds to an assignment of the variables that respects the constraints. (Haralick and Elliot 1980) discusses ways to increase tree efficiency to solve binary CSP while (Dechter, Cohen et al. 2003) focuses more on constraints processing. (Barták, Salido, and Rossi 2010) introduces definitions and techniques of constraint satisfaction in general AI planning and scheduling, many of which are adopted in this work for consistency. Constraint satisfaction techniques are increasingly used in robot planning problems. (Havur et al. 2014) uses Answer Set Programming (ASP) to decompose the cluttered workspace to place objects. Forward state-space search algorithms are developed to guide robotic spatial extrusion tasks (Garrett et al. 2020) under geometric and stiffness constraints. (Kavraki and Bohlin 2000; Sanchez and Latombe 2003), the properties of which have been examined (Hausser 2015). These approaches aim at reducing edge evaluation on computing a path (lazy motion planning), while the proposed framework aims at reducing the cost of path verification when computing a task plan (lazy rearrangement planning).

Problem Formulation

There are $n$ uniformly-sized cylindrical objects $\mathcal{O} = \{o_1, \ldots, o_n\}$ residing in a cuboid, bounded workspace $\mathcal{W} \subseteq \mathbb{R}^3$, each of which at a position $p_i \in \mathbb{R}^2, i \in \{1, \ldots, n\}$ (Fig. 3(a)). Such an arrangement of objects $\mathcal{O}$ to a set of object positions $\{p_1, \ldots, p_n\}$ defines an arrangement $\alpha \in \mathcal{A}$, where $\mathcal{A}$ is the arrangement space. $\alpha_1 = p_i$ indicates that object $o_i$ is at position $p_i$ given the arrangement $\alpha$.

A robot arm $\mathcal{M}$ can access the objects $\mathcal{O}$ from only one side of the workspace $\mathcal{W}$ and can move them one at a time. The arm $\mathcal{M}$ acquires a configuration $q \in \mathcal{Q}$ where $\mathcal{Q}$ is the space of all possible configurations that $\mathcal{M}$ can acquire. The swept volume $V(q)$ represents the space occupied by $\mathcal{M}$ at $q$. If $\mathcal{M}$ is grasping an object, the swept volume also includes the object’s volume given the grasp. $q(\alpha_1)$ represents a configuration where the arm can grasp $o_i$ at position $p_i = \alpha_1$. An arm path $\pi_i : [0, 1] \rightarrow \mathcal{Q}$ for an object $o_i$ corresponds to a sequence of configurations that move object $o_i$ from its current position $p_i = \alpha_1$ to another position $p_i'$, resulting in a new arrangement $\alpha'$ where $\alpha'_i = p_i'$ and $\forall j \in \{1, \ldots, n\}, j \neq i : \alpha'_j = \alpha_j$ (Fig. 3(b)). Such a path is valid if no collision arises between $\bigcup_{t=0}^{1} V(\pi_i(t))$ and other static geometries ($\mathcal{W}$ and static objects in $\mathcal{W}$).

The workspace is decomposed into a set of possible positions $\mathcal{P}_T$ where objects can be placed (Fig. 3(c)(d)). $\mathcal{P}_T$ are sampled with a resolution $\delta$, defined as the distance between adjacent candidates in the same row (or column). Goal positions of objects and intermediate positions called buffers are chosen from $\mathcal{P}_T$. The rearrangement problem is...
defined as follows: given any possible initial arrangement $\alpha_I$ (Fig. 4(c)) and a specified final arrangement $\alpha_F$ of $n$ objects $\Omega$ (Fig. 4(d)), find a sequence of valid arm paths $\Pi = (\pi_0, \pi_1, \ldots)$, which moves all objects from $\alpha_I$ to $\alpha_F$. A problem is monotone if the sequence $\Pi$ consists of at most one arm path for each object. Otherwise, the problem is non-monotone and at least one object needs to be moved to a buffer before being moved to its goal.

The above reachability constraints readily generalize to $k$ grasping configurations and $n$ objects. All the reachability constraints can be stored in a container $\mathcal{C}$ to indicate invalid actions of moving certain objects at a certain arrangement.

Grasping configurations can be acquired via an IK solver. An improved version of the Samuel Buss IK library was used (Buss and Kim 2005). In practice, different IK solutions place the last few links close to the gripper in similar locations, inducing similar reachability constraints. Given that, it usually suffices to consider only one IK per grasp.

### Monotone Instances

A lazy rearrangement solver (LRS) is proposed to solve local, monotone tasks $\alpha_S \rightarrow \alpha_F$ where $\alpha_S$ is the start arrangement. Alg. 1 summarizes the two steps of the solver: (1) obtain reachability constraints $\mathcal{C}$ from the task (Line 1); (2) use these constraints to grow lazily a search tree rooted at $\alpha_S$ (Line 2). Two definitions are introduced here to avoid potential confusion: In the context of a lazy tree, a node $\alpha_j$ is connected to another node $\alpha_i$ if the edge $\alpha_i \rightarrow \alpha_j$ does not violate reachability constraints. In contrast, $\alpha_j$ is accessible from $\alpha_i$ if $\alpha_i \rightarrow \alpha_j$ is verified by a motion planner to exist. Connectivity is a prerequisite for accessibility.

Step 2 of Alg. 1 is detailed in Alg. 2, which is a recursive routine to grow a local tree $T_{sub}$ from $\alpha_C$ to $\alpha_F$ that respects the reachability constraints $\mathcal{C}$. Here $\alpha_C$ refers to the arrangement where the search process is currently at. The output refers to a local, monotone tree $T_{sub}$ and a flag indicating if a solution is found (initially set to false, Line 1).

For each object $o \in \partial(\alpha_C)$ not yet at its goal given $\alpha_C$ (Line 2), a forward-checking routine is performed (Line 3) to evaluate if moving $o$ violates constraints $\mathcal{C}$ at $\alpha_C$. If it does, then this action is pruned at $\alpha_C$ (Line 4, also see Fig. 5(a)). Therefore, LRS identifies dead-ends and prunes the search space by reducing the action space. If $\mathcal{C}$ is satisfied, the resulting node $\alpha_{new}$ is generated (Line 5) where $\alpha_{new}[o] = \alpha_F[o], \alpha_{new}[\Omega \setminus \{o\}] = \alpha_C[\Omega \setminus \{o\}]$. If $\alpha_{new}$ is not in the tree, it will be added to the tree lazily (Line 6-7) without path verification to save computation. The tree keeps growing lazily in a recursive manner (Line 13) until $\alpha_F$ is connected to $\alpha_S$ via a branch (Line 8, also see Fig. 5(b)) where the path verification will be performed on the edges of the branch (Line 9), which is described in Alg. 3.

A global parameter mode is used to indicate the current search mode (backtracking or backjumping). It is in backtracking mode when Alg. 2 is first called in step 2 of Alg. 1. If the branch is valid, the solution is found and it quits Alg. 2 with backtracking (Line 11, 13-14). If not, the search

---

**Figure 4:** Three examples of the robot arm moving object $\alpha_3$ (gray solid circle). Start and goal positions of objects are denoted as solid circles and dashed circles, respectively. For simplicity, only relevant object positions are displayed.

---

**Algorithm 1:** Lazy Rearrangement Solver (LRS)

**Input:** $\alpha_S, \alpha_F, \mathcal{C}$

**Output:** $T_{sub}$

1. $\mathcal{C} = \text{obtain constraints}(\alpha_S, \alpha_F, \mathcal{C})$
2. return $T_{sub}, \text{flag} = \text{grow local tree}(\emptyset, \alpha_S, \alpha_F, \mathcal{C})$

where $G_2 \cap G_4$ can be handled similar to the constraint (1) of Fig. 4(a), while $G_2 \cap G_4$ can be handled similar to constraint (2) of Fig. 4(b).

### Methodology

This section introduces lazy rearrangement planning and describes: (1) how reachability constraints are generated and used to prune the search space, (2) how the local, monotone solver builds a lazy search tree, and (3) how the global planner operates over these trees for non-monotone instances.

### Reachability Constraints

Consider Fig. 4(a) where there are two arm configurations for grasping $\alpha_3$: $\alpha_3$ is unreachable from one configuration, if $\alpha_2$ is at its goal; while $\alpha_3$ is unreachable from the other, if $\alpha_1$ is at its start. Define $S_i$ as the event that $\alpha_i$ is at its start while $G_i$ as the event that $\alpha_i$ is at its goal. Then in Fig. 4(a), $\alpha_3$ is unreachable if the event $G_2 \cap S_1$ occurs, which generates the reachability constraint:

**Don’t move $\alpha_3$, if $\alpha_2$ is at its goal and $\alpha_1$ is at its start.** (1)

In Fig. 4(b), $\alpha_3$ is unreachable if the event $G_2 \cap G_1$ occurs:

**Don’t move $\alpha_3$, if $\alpha_2$ is at its goal and $\alpha_1$ is at its goal.** (2)

Note that when all reachability constraints arise from goal positions, additional constraints can be elicited for the constraining objects, i.e., $\alpha_2$ and $\alpha_1$ in this example. In particular, the additional constraints are:

**Don’t move $\alpha_2$, if $\alpha_1$ is at its goal and $\alpha_3$ is at its start.** (3)

**Don’t move $\alpha_1$, if $\alpha_2$ is at its goal and $\alpha_3$ is at its start.** (4)

For generalization, one more example (Fig. 4(c)) is given where a grasping configuration intersects with more than one object position (start position of $\alpha_1$ and goal position of $\alpha_2$). In this case, $\alpha_3$ is unreachable if the event $G_2 \cap (S_1 \cup G_4)$ occurs. The event can be decoupled as:

$$G_2 \cap (S_1 \cup G_4) = (G_2 \cap S_1) \cup (G_2 \cap G_4)$$ (5)
switches to the backjumping mode (Line 10), in which the search tree jumps back to the last accessible node \( \alpha_{\text{last}} \) to continue (Line 15-17). The tree is returned when all alternatives have been exhausted (Line 18).

Alg. 3. verifies the branch that connects \( \alpha_F \) from \( \alpha_S \). It first traces back to the nearest node \( \alpha' \) that is accessible from the root (Line 1) to avoid duplicate edge verification. From there, it checks the validity of each edge (Line 3-4). If an edge \( \alpha' \rightarrow \alpha_{\text{child}} \) is valid (Line 5), it marks \( \alpha_{\text{child}} \) as accessible from \( \alpha' \) (Line 6) and moves on to the next edge (Line 7). If not (Line 8), the branch verification stops at the edge \( \alpha' \rightarrow \alpha_{\text{child}} \) (see Fig. 5(d)) and deletes any subtree rooted at \( \alpha_{\text{child}} \) (Line 9) as \( \alpha_{\text{child}} \) is not accessible from \( \alpha' \). It returns the trimmed lazy tree \( T_{\text{sub}} \) and records the last accessible node \( \alpha_{\text{last}} = \alpha' \) (Line 10, also see Fig. 5(e)). If all the edges are valid, the task is solved and the branch on \( T_{\text{sub}} \) is the solution (Line 11, also see Fig. 5(c)).

**Theorem 1** (Completeness of LRS). LRS achieves the same completeness properties as the underlying motion planner.

**Proof.** Assume there is a monotone solution \( \Pi' \) but LRS fails to find it. Since \( \Pi' \) is monotone, each object moves directly from its start to its goal. Following the order, there is a corresponding path \( P' \) in the state space of LRS, which is a sequence of states with the first state being \( \alpha_S \) and the others being the resulting states of actions in \( \Pi' \) in the order. Given the failure assumption, let the first edge in \( P' \) that is not in \( T_{\text{sub}} \) be \( \alpha_1 \rightarrow \alpha_2 \). Since \( \alpha_1 \rightarrow \alpha_2 \) is a valid edge that respects constraints \( C \), it should be eventually discovered by a motion planner (e.g., assuming probabilistic completeness). Therefore, the only reason that \( \alpha_2 \) is not linked to \( \alpha_1 \) when LRS explores \( \alpha_1 \) is that \( \alpha_2 \) corresponds to a node that has been considered already (with a different parent than \( \alpha_1 \)) and deemed as a “dead end”, i.e., can’t be connected to \( \alpha_F \).

If \( \alpha_2 = \alpha_F \), then it means LRS finds a path from \( \alpha_1 \) to \( \alpha_F \) and the path succeeds in the branch verification process, which contradicts the failure assumption. Otherwise, since the search tree is developed in a depth first manner, it suggests that LRS cannot find a path in the state space from \( \alpha_2 \) to \( \alpha_F \). But there is a sub-path \( P'' \) in \( P' \) from \( \alpha_2 \) to \( \alpha_F \). Recursively, all the edges in \( P'' \) are in \( T_{\text{sub}} \) or there is another path from \( \alpha_F \) to \( \alpha_F \) in \( T_{\text{sub}} \).

Thus, given a monotone rearrangement instance, whether LRS is able to find a solution depends on whether the underlying motion planner can find one. In other words, the proposed LRS does not degrade the properties of the underlying motion planner (e.g., probabilistic/resolution completeness). For instance, if a solution has not been found but exists, a straightforward iterative strategy can provide to the motion planner additional resources (more sampled configurations, object position candidates, grasps and IK solutions) and then the task planner is called again.

The motion planner used is the asympt. optimal Probabilistic Roadmap (PRM*) (Karaman and Frazzoli[2011]).

---

**Algorithm 2: GrowLocalTree**

Input: \( T_{\text{sub}}, \alpha_C, \alpha_F, C \)

Output: \( T_{\text{sub}}, \text{flag} \)

1. \( \text{flag} = \text{False} \)
2. for \( o \in \mathcal{O}(\alpha_C) \) do
   3. \( \text{valid} = \text{ForwardChecking}(o, \alpha_C, C) \)
   4. if \( \text{not valid} \) then continue
   5. \( \alpha_{\text{new}} = \text{GetNewNode}(o, \alpha_C, \alpha_F) \)
   6. if \( \alpha_{\text{new}} \notin T \) then continue
   7. \( T_{\text{sub}} = \text{AddNode}(\alpha_{\text{new}}, \alpha_C, T_{\text{sub}}) \)
   8. if \( \alpha_{\text{new}} = \alpha_F \) then
      9. \( T_{\text{sub}}, \text{success}, \alpha_{\text{last}} = \text{VerifyBranch}(\alpha_{\text{new}}, T_{\text{sub}}) \)
   10. if \( \text{not success} \) then mode = backjumping
   11. return \( T_{\text{sub}}, \text{success} \)
   12. else
      13. \( T_{\text{sub}}, \text{flag} = \text{GrowLocalTree}(T_{\text{sub}}, \alpha_{\text{new}}, \alpha_F, C) \)
      14. if \( \text{flag} \) then return \( T_{\text{sub}}, \text{flag} \)
      15. if mode = backjumping then
        16. if \( \alpha_{\text{new}} = \alpha_{\text{last}} \) then mode = backtracking
        17. else return \( T_{\text{sub}}, \text{flag} \)
18. return \( T_{\text{sub}}, \text{flag} \)

---

**Algorithm 3: VerifyBranch**

Input: \( \alpha_C, T_{\text{sub}} \)

Output: \( T_{\text{sub}}, \text{success}, \alpha_{\text{last}} \)

1. \( \alpha' = \text{NearestNode}(\alpha_C, T_{\text{sub}}) \)
2. while \( \alpha' \neq \alpha_C \) do
   3. \( \alpha_{\text{child}} = \alpha'_F \)
   4. \( \text{success} = \text{PathVerification}(\alpha', \alpha_{\text{child}}) \)
   5. if \( \text{success} \) then
      6. \( T_{\text{sub}} = \text{AddEdge}(\alpha', \alpha_{\text{child}}, T_{\text{sub}}) \)
      7. \( \alpha' = \alpha_{\text{child}} \)
   8. else
      9. \( \text{DeleteTree}(\alpha_{\text{child}}, T_{\text{sub}}) \)
   10. return \( T_{\text{sub}}, \text{success}, \alpha' \)
11. return \( T_{\text{sub}}, \text{success}, \alpha' \)

---

Figure 5: Lazy tree in LRS. Black dashed circles and arrows represents nodes and edges without path verification while green solid ones represents those with path verification. Here \( \alpha_C \) indicates the current stage of the search. (a) Forward-checking phase ( Alg. 2, Line 3-4). A tree cannot expand upon \( \alpha_C \) due to constraint violation. (b) The tree connects \( \alpha_C \) to \( \alpha_F \) lazily (Alg. 2, Line 8). (c) The branch pass path verification (Alg. 3, Line 11). (d) Path verification fails at the edge \( \alpha' \rightarrow \alpha_{\text{child}} \) and is terminated. (e) The subtree rooted at \( \alpha_{\text{child}} \) is trimmed (Alg. 3, Line 9-10) and the search jumps back to the last accessible node \( \alpha_{\text{last}} \) and backtracks to grow the tree lazily (Alg. 2, Line 15-17).
Non-Monotone Instances

For non-monotone problems where some objects need to be moved to buffers first to admit a feasible solution, a global planner is needed, as the local solver only examines the monotonicity of a local task. The global planner assigns a local task to the local solver and concatenates the local tree $T_{sub}$ returned by the local solver to build a global tree $T_g$. If the resulting $T_g$ does not lead to a solution, an action is needed to move an object to a buffer at an arrangement. Such actions are called perturbations in the context of the global planner. A new tree node $\alpha_{pert}$ (the blue node in Fig. 2(a)) will be generated upon perturbation, from which a new local task $\alpha_{pert} \rightarrow \alpha_F$ is assigned to the local planner.

Consider a lazy local tree $T_{sub}$ in Fig. 6(a). There are different ways of concatenating $T_{sub}$ to the global tree $T_g$. A greedy global planner concatenates only the part of the tree which has been path verified (Fig. 6b), while a conservative global planner first verifies all unverified edges in the lazy tree and then concatenates the verified tree (Fig. 6c). In the greedy case, the planner operates lazily by only accepting the verified part of the tree to avoid performing any expensive path verification. As a result, it quickly concatenates the tree but giving up unverified nodes which could lead to a solution given a proper perturbation. In the extreme cases where no nodes except the root of the local tree is accepted, the only way the tree can grow in the greedy case is through perturbation at the root. But a perturbation is both slow (just extends one edge) and unpredictable (the object is moved to a random buffer). In the conservative case, the planner performs additional work to verify the unverified part of the tree, thus maintaining those nodes in the global tree. This, however, undermines the utility of lazy evaluation as the global planner takes time to make up the laziness of the local planner. The only advantage lies in the last iteration if a solution is found (in that case, the global planner only needs to concatenate the verified branch).

To leverage the benefits of both the greedy and the conservative planner, this work introduces a global planner LRS$_{hybrid}$, which concatenates the lazy local tree as it is and only performs path verification if the node being selected for perturbation is unverified. In this manner, the global tree maintains both verified and unverified edges (hybrid) throughout the search. LRS$_{hybrid}$ is described in Alg. 4. It solves a global task $\alpha_I \rightarrow \alpha_F$ with $n$ objects and all possible positions $P_T$ and outputs a path sequence $\Pi$. The global task is first assigned to the local solver, from where the solver tests the monotonicity of the problem (Line 1-3). If the problem is non-monotone (Line 4), a node $\alpha_C$ is selected to perform a perturbation (Line 5, also see Fig. 6a). Before perturbation, $\alpha_C$ is checked to see if it is accessible from the root $\alpha_I$ (Line 6). If not, the perturbation terminates and restarts (Fig. 7c(d)). If $\alpha_C$ is accessible from $\alpha_I$ (Line 7), the perturbation is performed on $\alpha_C$ by randomly selecting an object to be placed in a buffer, which is randomly selected from $P_T$ (Line 8). If the perturbation is not successful, it terminates and restarts (Line 4). Otherwise (Line 9), a perturbation node $\alpha_{pert}$ is added to the tree (Line 10, also see Fig. 7b), from where the local solver is called to solve a local task $\alpha_{pert} \rightarrow \alpha_F$. The local solver call and the perturbation alternate until a solution is found (Line 12-13) or the time exceeds a threshold (Line 4).

Despite different ways (greedy, conservative or hybrid) of tree concatenation, the planner generally follows the structure of alternating between random perturbations and subtree concatenation. The high-level structure is referred to as Perturbation Search (PERTS). The following discussion provides a proof that PERTS guarantees probabilistic completeness, regardless of how local trees are concatenated or the type of monotone solver used. The only requirement is that the clearance of paths $\delta$ in the rearrangement plan is lower bounded by a constant. Here $\delta$ is measured as the minimum distance between the robot arm and the obstacles in the plan.

![Figure 6](image6.png)  
(a) A lazy search tree returned by the local solver LRS with both verified (green solid arrows) and unverified (black dashed arrows) edges. (b) The tree concatenated by a greedy global planner. (c) The tree concatenated by a conservative global planner.

![Figure 7](image7.png)  
(a) A node $\alpha_C$ is selected to perform perturbation on a tree in Fig. 6a. (b) A perturbation succeeds only if (1) $\alpha_C$ is accessible from the root (green arrows) and (2) the perturbation node $\alpha_{pert}$ is accessible from $\alpha_C$ (blue arrow). (c) An unsuccessful perturbation given that $\alpha_C$ is not accessible. (d) The tree after trimming the subtree rooted at $\alpha_C$.
A lemma for the completeness theorem is proved as follows.

**Lemma 1.** Given the set of position candidates $\mathcal{P}_T$ with resolution $\delta_r$ and a complete motion planner, if there is a feasible rearrangement plan $\Pi$ from $\alpha_I$ to $\alpha_F$ with clearance $\delta_I \geq \sqrt{2}\delta_r$, then there is a feasible path $P$ from $\alpha_I$ to $\alpha_F$ in the state space of the proposed global planner.

**Proof.** The lemma can be proven by moving all the placing positions in $\Pi$ (goal and buffer positions) to their nearest neighbors in $\mathcal{P}_T$. Given the resolution $\delta_r$, each point in the rectangular region is at most $\sqrt{2}/2\delta_r$ away from its nearest neighbor in $\mathcal{P}_T$. With a complete motion planner, there is a feasible path $P$ in the state space $S$ of the proposed global planner given clearance: $\delta_I \geq \delta - 2*\sqrt{2}/2\delta_r > 0$. \hfill \Box

**Theorem 2** (Probabilistic Completeness of PERTS). Given a complete motion planner and a constant $\delta^*$, if there is a rearrangement plan with clearance $\delta \geq \delta^*$, then the probability of the global planner to find a feasible rearrangement plan approaches 1 as the number of perturbations increase, regardless of which monotone solver is used.

**Proof.** Given the tolerating clearance $\delta^*$, generate $\mathcal{P}_T$ with resolution $\sqrt{2}/3\delta^*$. Assume that there is a feasible rearrangement plan $\Pi$ for a rearrangement instance. According to the Lemma above, there is a feasible path $P$ from $\alpha_I$ to $\alpha_F$ in the state space $S$ of PERTS.

To prove the probabilistic completeness, regardless of which monotone solver is used, it suffices to prove that the probability of finding $\alpha_F$ via perturbations approaches 1 as the perturbation number $m$ goes to infinity.

In each perturbation, PERTS moves a randomly selected object from a random state to a randomly selected position in $\mathcal{P}_T$. Since the size of the state space $S$ is upper bounded by $|\mathcal{P}_T|^n$, for all $2 \leq i \leq |P|$, when $P[i-1]$ is in the search tree, the probability that $P[i]$ can be added into the search tree in the next perturbation is lower bounded by $p > 1/(|\mathcal{P}_T|+n)$. Given the probability of finding $\alpha_F$ via perturbations approaches 1 as the number of perturbations increase, regardless of which monotone solver is used.

In each perturbation, PERTS moves a randomly selected object from a random state to a randomly selected position in $\mathcal{P}_T$. Since the size of the state space $S$ is upper bounded by $|\mathcal{P}_T|^n$, for all $2 \leq i \leq |P|$, when $P[i-1]$ is in the search tree, the probability that $P[i]$ can be added into the search tree in the next perturbation is lower bounded by $p > 1/(|\mathcal{P}_T|+n)$. Given the probability of finding $\alpha_F$ via perturbations approaches 1 as the number of perturbations increase, regardless of which monotone solver is used.

**Evaluation on Monotone Problems:** The proposed LRS is compared to the aforementioned alternatives (mRS, DFS-DP, CIRS). A limitation of 100 seconds is given to solve a monotone problem. Fig. 9(left column) demonstrates that the success rate for LRS stays 100% even in harder problems (12, 14 objects) while that for mRS and DFS-DP drops significantly (16.7%, 4.2% for mRS and 46.7%, 8.3% for DFS-DP). CIRS shows relatively high success rate (83.3% for 14 objects)
but LRS outperforms CIRS (Fig. 9 (right column)) with faster computation time (10.2 and 16.8 seconds for 12 and 14 objects, compared to 28.3 and 47.3 seconds for CIRS). The lazy evaluation structure of LRS significantly reduces the path verification time. For 14 objects, LRS spends 16.2 seconds (96.4% of the total time) on path verification, while the time is 46.9 (99.2%), 93.9 (99.3%) and 97.3 (99.3%) seconds for CIRS, DFSdp and mRS, respectively (Fig. 9 (right column), colored part of the bars). It also confirms the observation that path verification is the computational overhead.

**Evaluation on Non-monotone Problems:** The proposed global planner LRS hybrid is compared to global planners which use the alternatives (mRS, DFSdp, CIRS) as the integrated local solver. For simplicity, the names mRS, DFSdp, CIRS are still used to represent the corresponding global planner. A limitation of 240 seconds is given to solve a non-monotone problem. Fig. 10 (left) indicates that mRS and DFSdp fail in every instance for 13-16 objects and CIRS shows a stable decrease of success rate (44.1%, 37.9% and 25.8% for 14, 15 and 16 objects, respectively). In contrast, LRS hybrid has 100% success rate for 12-14 objects and stays high for harder problems (85.7% for 15 objects and 82.4% for 16 objects). LRS hybrid solves non-monotone problems 61% faster than alternatives, as it lazily loads the local tree and only performs path verification if the node selected for perturbation is unverified. Moreover, the time used other than path verification (Fig. 10 (right), white part of the bars) for LRS hybrid is higher than other methods (e.g., for 16 objects, that time is 23.2 seconds for LRS hybrid, compared to 1.9 seconds for other methods), as LRS hybrid requires steps such as obtaining constraints (Alg. 1, Line 1) and trimming inaccessible subtrees (Alg. 3, Line 9). However, the path verification time for LRS hybrid is 98.8 seconds, much smaller than that of other methods (231.6 seconds on average). In that regard, LRS hybrid trades heavy path verification time for some moderate increase in task planning time.

**Ablation Study:** An ablation study is also performed to reveal the impact of different versions of the global planner (greedy, conservative, hybrid) on non-monotone problems. Fig. 11 (left column) shows that the success rate for LRS hybrid is higher than LRS greedy and LRS conservative. The two alternatives suffer from relatively low success rates for different reasons. LRS greedy only accepts the verified part of tree to avoid path verification, giving up potential nodes. As a result, it involves more perturbation steps to grow the tree, which can be slow. LRS conservative further verifies the unverified part of the local lazy tree before concatenating it to the global tree. As a result, it has more path verification steps, which is also slow. Fig. 11 (right column) also confirms these reasons. LRS greedy has a smaller path verification time (colored part of the bars) but a higher other time (white part of the bars) and LRS conservative has both time in the middle compared to the other two, achieving the fastest computation time.

The number of buffers needed to solve non-monotone problems is provided in Table 1. With the PERTS structure, the problems can be mostly solved with one or two buffers.

**Conclusion and Future Work**

This work introduces a lazy evaluation framework, which involves a local monotone solver and a global planner for solving rearrangement in confined spaces. The proposed framework is capable of solving hard instances up to 16 objects with high-quality by improving computational efficiency. This paper argues the completeness of the local solver and the probabilistic completeness of the global planner. The solutions are also demonstrated on a real robotic system. The confined setup does not only pose reachability challenges, but also visibility challenges. As the objects cannot be accessed from above by the robot arm, they cannot be always detected from above by a camera. This results in partial observability where the rearrangement objective changes, i.e., rearranging objects which improve the visibility of others. Not all objects may be known initially and replanning is needed as the scene is updated. The proposed techniques can be modified to be deployed in these setups. For instance, building a stochastic tree where the edge incorporates the probability the object is movable given uncertainty.

**Acknowledgements**

The work is partially supported by an NSF HDR TRIPODS award 1934924.
References

Ahn, J.; Kim, C.; and Nam, C. 2021. Coordination of two robotic manipulators for object retrieval in clutter. arXiv preprint arXiv:2109.15320.

Ahn, J.; Lee, J.; Cheong, S. H.; Kim, C.; and Nam, C. 2021. An integrated approach for determining objects to be relocated and their goal positions inside clutter for object retrieval. In 2021 IEEE International Conference on Robotics and Automation (ICRA), 6408–6414. IEEE.

Bartáková, R.; Salido, M. A.; and Rossi, F. 2010. Constraint satisfaction techniques in planning and scheduling. Journal of Intelligent Manufacturing, 21(1): 5–15.

Buss, S. R.; and Kim, J.-S. 2005. Selectively damped least squares for inverse kinematics. Journal of Graphics tools, 10(3): 37–49.

Cohen, B. J.; Phillips, M.; and Likhachev, M. 2015. Planning Single-Arm Manipulations with N-Arm Robots. In SOCS.

Dechter, R.; Cohen, D.; et al. 2003. Constraint processing. Morgan Kaufmann.

Garrett, C. R.; Huang, Y.; Lozano-Pérez, T.; and Mueller, C. T. 2020. Scalable and Probabilistically Complete Planning for Robotic Spatial Extrusion. In Robotics: Science and Systems.

Haralick, R. M.; and Elliott, G. L. 1980. Increasing tree search efficiency for constraint satisfaction problems. Artificial intelligence, 14(3): 263–894.

Hauser, K. K. 2015. Lazy collision checking in asymptotically-optimal motion planning. 2015 IEEE International Conference on Robotics and Automation (ICRA), 2951–2957.

Havur, G.; Ozbilgin, G.; Erdem, E.; and Patoglu, V. 2014. Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach. In 2014 IEEE International Conference on Robotics and Automation (ICRA), 445–452. IEEE.

Karaman, S.; and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. The international journal of robotics research, 30(7): 846–894.

Kavraki, L.; and Bohlin, R. 2000. Path planning using lazy PRM. In Proc. IEEE Int. Conference on Robotics and Automation. San Francisco.

King, J. E.; Haustein, J. A.; Srinivasa, S. S.; and Asfour, T. 2015. Nonprehensile whole arm rearrangement planning on physics manifolds. In 2015 IEEE International Conference on Robotics and Automation (ICRA), 2508–2515. IEEE.

King, J. E.; Ranganeni, V.; and Srinivasa, S. S. 2017. Unobservable monte carlo planning for nonprehensile rearrangement tasks. In 2017 IEEE International Conference on Robotics and Automation (ICRA), 4681–4688. IEEE.

Kleinbort, M.; Solovey, K.; Littlefield, Z.; Bekris, K. E.; and Halperin, D. 2018. Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation. IEEE Robotics and Automation Letters, 4(2): x–xvi.

Kontiris, A.; and Bekris, K. E. 2016. Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner. In 2016 IEEE International Conference on Robotics and Automation (ICRA), 3924–3931. IEEE.

Lim, J.; Srinivasa, S.; and Tsiotras, P. 2021. Lazy Lifelong Planning for Efficient Replanning in Graphs with Expensive Edge Evaluation. arXiv preprint arXiv:2105.12076.

Mandalika, A.; Choudhury, S.; Salzman, O.; and Srinivasa, S. 2019. Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 29, 745–753.

Mandalika, A.; Salzman, O.; and Srinivasa, S. 2018. Lazy receding horizon A* for efficient path planning in graphs with expensive-to-evaluate edges. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 28.

Papallas, R.; and Dogar, M. R. 2020. Non-prehensile manipulation in clutter with human-in-the-loop. In 2020 IEEE International Conference on Robotics and Automation (ICRA), 6723–6729. IEEE.

Sánchez, G.; and Latombe, J.-C. 2003. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In Robotics research, 403–417. Springer.

Stilman, M.; Schamburek, J.-U.; Kuffner, J.; and Asfour, T. 2007. Manipulation planning among movable obstacles. In Proceedings 2007 IEEE international conference on robotics and automation, 3327–3332. IEEE.

Vieira, E. R.; Nakhimovich, D.; Gao, K.; Wang, R.; Yu, J.; and Bekris, K. E. 2022. Persistent Homology for Effective Non-Prehensile Manipulation. In 2022 IEEE International Conference on Robotics and Automation (ICRA). IEEE.

Wang, R.; Gao, K.; Nakhimovich, D.; Yu, J.; and Bekris, K. E. 2021. Uniform Object Rearrangement: From Complete Monotone Primitives to Efficient Non-Monotone Informed Search. In IEEE International Conference on Robotics and Automation.

Wang, R.; Miao, Y.; and Bekris, K. E. 2022. Efficient and high-quality prehensile rearrangement in cluttered and confined spaces. In 2022 International Conference on Robotics and Automation (ICRA), 1968–1975. IEEE.

Xiao, Y.; Katt, S.; ten Pas, A.; Chen, S.; and Amato, C. 2019. Online planning for target object search in clutter under partial observability. In 2019 International Conference on Robotics and Automation (ICRA), 8241–8247. IEEE.