HIERARCHICAL FRAMEWORKS FOR EFFICIENT PREHENSILE REARRANGEMENT WITH A ROBOTIC MANIPULATOR

BY

ATHANASIOS KRONTIRIS

A dissertation submitted to the

School of Graduate Studies—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Kostas E. Bekris

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2017

ABSTRACT OF THE DISSERTATION

Hierarchical Frameworks for Efficient Prehensile Rearrangement with a Robotic Manipulator

by Athanasios Krontiris

Dissertation Director: Kostas E. Bekris

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 $\mathcal{C}$-spaces involving multiple movable bodies and complex kinematic constraints. This work aims to identify ways of decomposing such problems into a hierarchy of challenges that can be addressed effectively individually, while their composition can provide a solution to the overall instance.

The first direction for such a hierarchical decomposition aims to take advantage of developments in the multi-robot community, where there are efficient solvers for the "pebble motion on a graph" problem. Unlabeled rearrangement problems with a robotic manipulator are decomposed into a sequence of subproblems, each one of which can be viewed as a "pebble motion on a graph" problem. The labeled case, however, is not easily decomposed to a "pebble motion on a graph" problem instances.

To deal with general object rearrangement, including both the labeled and the unlabeled case, this work builds on top of prior work that was able to compute solutions for labeled monotone instances through a backtracking search process. Monotone instances

are those where every object needs to be transferred at most once to achieve a desired arrangement. This thesis extends the backtracking process to a method that addresses many non-monotone challenges. In order to solve the non-monotone cases the method is using solutions to the Minimum Constraint Removal (MCR) path problem so as to transfer each object to its target. An MCR path minimizes the number of constraints that need to be removed from the path of an object. This work then utilizes the monotone or the non-monotone backtracking search process as local connection primitives in the context of a higher-level task planner, which operates similar to a Probabilistic Roadmap Method (PRM), that searches the space of object placements. It is shown that the integration of these primitives with the higher-level planner achieves probabilistic completeness guarantees for the general object rearrangement problems.

To improve the efficiency of the above hierarchical framework, this work introduces approximate but significantly faster primitives for monotone and non-monotone rearrangement instances. The methods avoid backtracking search by building a dependency graph between objects given solutions to the Minimum Constraint Removal (MCR) path planning problem to transfer each object to its target. From this graph, the approach discovers the order of moving objects by performing topological sorting. These new approximate but fast primitives that do not need backtracking search are incorporated in a higher-level incremental search algorithm for general rearrangement planning, which operates similar to a Bi-directional Rapidly-exploring Random Tree (Bi-RRT). Given a start and a goal object arrangement, tree structures of reachable new arrangements are generated by using the new and fast approximate primitives as an expansion procedure.

These methods have been evaluated in simulation using models of robotics manipulators, such as a Baxter or a Motoman robot arm, in order to study their capability in solving difficult instances of rearrangement problems. This work compares the different alternatives in terms of success ratio, running time, scalability and path quality.

Overall, this work aims to emphasize the benefit of using more powerful primitives, which are reasoning about the combinatorial and the underlying multi-object nature of the rearrangement problem, in the context of high-level task planning for robotic manipulation.

Acknowledgements

At this moment of accomplishment I am greatly indebted to my research adviser, Dr. Kostas E. Bekris, who accepted me as one of his first Ph.D. students and offered me his mentorship, support and enthusiasm. This work would not have been possible without his guidance, dedication, his motivation and encouragement on daily basis, even through late nights, weekends and vacations, from the first day of our collaboration till the last day of delivering this work. Under his guidance I successfully overcome many difficulties, learn a lot and I became a successful researcher and scientist. His zeal for perfection, passion and dedication has always inspired me to do more. For all of these, I sincerely thank you **Dr. Kostas E. Bekris** for stood next to me through this journey.

I would like to thank each of my committee members, Jingjin Yu, Mubbasir Kapadia, and Mehmet R. Dogar, for dedicating their time to the review and consideration of this work.

I would also like to thank each and every PRACSYS lab member I have worked with. Ilias Apostolopoulos, Andrew Dobson, Alexis Oyama, Yanbo Li, James Marble, Ryan Luna, Andrew Kimmel, Zakary Littlefield, Rahul Shome, Justin Cardoza, Colin Rennie, Shaojun Zhu, Zacharias Psarakis, Hristian Courtev, Nick Stiffler and the external member Shuai Han. Your presents, support and great skill were important for me to survive through the long deadlines and the long nights we spend together in the lab.

I would also like to thank my external collaborators, Professor Jingjin Yu where our research was always in parallel and eventually I had the honor to collaborate with him on a paper. Professor Mubbasir Kapadia for his guidance and support on our joint project.

I would like to thank the DHS CCICADA Center of Excellence at Rutgers University

for their generous financial and intellectual support. I would especially like to thank the CCICADA director Fred Roberts for his support and his efforts on our joint project, along with all member of CCICADA who helped with their ideas and especial thanks to Brian Ricks, Aditya Chukka and Han Meng for their dedication towards the completion of the joint project.

I also thank my loved ones and my friends (too many to list here but you know who you are!) for providing support and friendship that I needed through this long journey.

To my beloved sister in law, Evita, that she has given birth and she is about to give birth again to the two newest members of the Krontiris family. Our little Erina and her sister have brought a lot of happiness to our family.

To my brothers Giannis and Tasos that have never left my side and are very special. I could not imagined having better brothers. I am always happy to see them at the airport waiting to pick me up and take me back home.

Finally, I would like to thank my parents for all their love and moral support throughout my life. Thank you both for giving me strength to reach for the stars and chase my dreams. You are always there for me.

Dedication

This thesis is dedicated to my parents, **Dimitris and Katerina Krontiris**.

For their endless love, support, and encouragement.

Thank you for helping me to grow and become the person I am today.

Table of Contents

Chapter 1

Introduction

Rearrangement is a computationally hard problem 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. For instance, in manufacturing, where multiple products may need to be orderly arranged on factory floors or grocery stores. Many manipulation challenges are not always directly solvable but may require a robot to first change the environment.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.

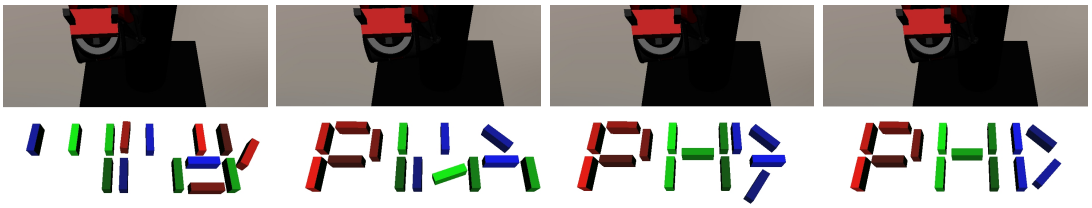

Figure 1.1: A rearrangement example where a Baxter robotic arm needs to arrange objects so that they form the letters PHD.

This work 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.

Certain instances, however, are easier to tackle. This was the realization behind a motivating work for manipulation among movable obstacles [110]. This previous work describes a backtracking search method for detecting the sequence of objects to be

moved so as to reach a desirable, previously unreachable object. The adaptation of this method to rearrangement problems, referred here as "monotone Rearrangement Solver" (`mRS`), addresses monotone instances, where each object needs to be moved at most once for a solution. Furthermore, `mRS` depends on exhaustive backtracking search to detect the right order with which objects need to be transferred. This search can be slow when many of the potential orderings result in failure as an exponential number of them need to be considered.

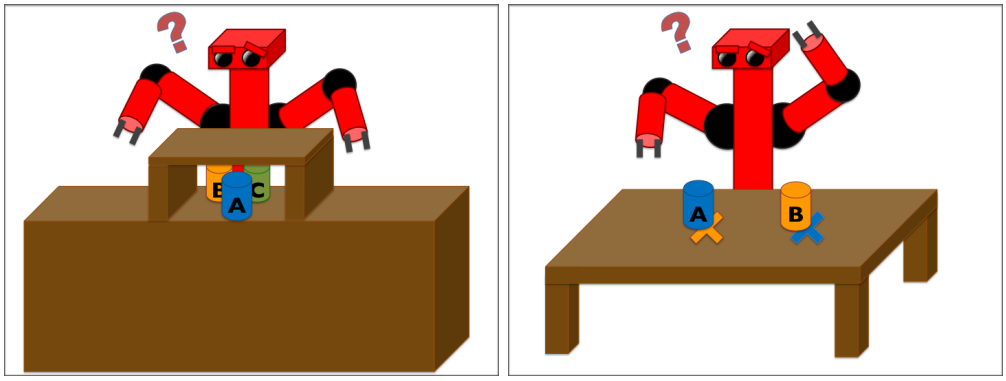

Figure 1.2: Cases of non-monotone problems that some objects have to be displaced in order to find a solution. (Left) Object A is occluded by object B and C, which makes object A unreachable from the robot. (Right) Overlapping between initial with final poses. As a result the task of placing the objects at their final poses it is not trivial.

Rearrangement problems are not always an easy problem, where the objects have to be moved only once. Occlusions of objects (Fig.1.2(left)) , or overlaps between initial and final poses (Fig.1.2(right)) are problems that can easily arose in such tasks. The harder instances require some objects to be placed in intermediate positions before moved to their target. These problems are recognized as non-monotone challenges. There are recent approaches for rearrangement that can deal with the non-monotone case under certain conditions [51, 106, 40, 72]. One method simplifies the problem by requiring that all objects are unlabeled, i.e., interchangeable and can occupy any pose in their final arrangement [72]. An alternative imposes a grid for placing the objects and then uses techniques, such as answer-set programming [51]. Others view the rearrangement problem as an instance of general integrated task and motion planning [106] and, in this context, they evaluate good heuristics for integrated planning [40].

1.1 Objective Statement

The objective of this work is to propose methods for efficiently solving **hard instances of general object rearrangement problems** through grasping using a single robotic arm. 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.

1.2 Dissertation Overview and Contributions

The outline and the main contributions of this work will be explained in greater detail in this section. The first part of the manuscript focuses on solving unlabeled rearrangement problems. As it is going to be explained later in section 3.1, these problems are slightly easier given that objects can occupy any pose in the final arrangement. However, this method could solve only problems with geometrically similar objects. The second part of this work that is comprised by chapters 5 and 6 is dealing with harder problems, where each object has a specific final pose in the final arrangement. Chapter 5 argues probabilistic completeness for solving rearrangement problems, while chapter 6 provides fast methods for hard rearrangement instances.

Chapter 2 introduces the basic concepts and foundational work in the areas of algorithmic motion planning, multi-body motion planning, robot task planning, manipulation planning and rearrangement planning. It begins by reviewing the most related work, planning among Movable Obstacles. Then it presents the minimum constraint removal path that is heavily used in this work. That chapter also introduces the basic

idea of multi-robot motion planning in the context of rearrangement planning. Moreover, it explains the basic concepts and terminology of task, motion, and manipulation planning and how these are related to rearrangement planning. Finally, the chapter concludes with a review about two important sampling-based motion planning algorithms the Probabilistic Roadmap Method (`PRM`) [61] and Rapidly-exploring Random Tree (`RRT`) [80]. Both algorithms have been used in order to search the arrangement space in order to find a solution. The `PRM` method is also used for searching in the configuration space for a valid collision-free path for the motion of the manipulator.

Chapter 3 presents the problem setup and explains the notation that this work uses through this manuscript. Most importantly, though, this chapter defines the different categories of a rearrangement problem. The first separation is based on the final arrangements. An arrangement can be either unlabeled, which means that the objects can occupy any pose in the final arrangement, or labeled, where each object has a specific final pose. A rearrangement problem can also be categorized based on how many times the robot needs to move an object before the object reaches its final pose. A problem will be called monotone if each object is transferred at most once, otherwise non-monotone. For non-monotone problems, at least one object will have to move to an intermediate pose before reaching its final pose. Finally, this chapter describes the motion planning primitives that the algorithms used in order to move the manipulator.

Chapter 4 proposes a method for efficiently computing manipulation paths to rearrange similar objects in a cluttered space, fig.1.3. Rearrangement is a challenging problem as it involves combinatorially large, continuous configuration spaces due to the presence of multiple bodies and kinematically complex manipulators. This work leverages ideas from multi-robot motion planning and manipulation planning to propose appropriate graphical representations for this challenge. These representations allow to quickly reason whether manipulation paths allow the transition between entire sets of object arrangements without having to explicitly store these arrangements. The proposed method also takes advantage of precomputation given a manipulation roadmap for transferring a single object in the space. The approach is evaluated in simulation for a realistic model of a `Baxter` robot and executed on the real system, showing that the

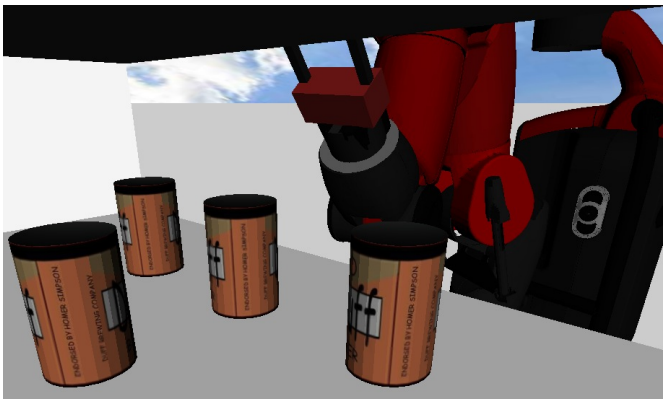

Figure 1.3: In order for `Baxter` to grasp the object at the back of the shelf, the other objects need to be rearranged.

method solves complex instances and is promising in terms of scalability and success ratio.

Contributions: This work can efficiently compute manipulation paths to rearrange similar objects in cluttered spaces, by integrating combinatorial multi-robot planning and manipulation planning. Continuous rearrangement challenges are abstracted as discrete pebble graph problems, i.e. Manipulation Pebble Graphs (`MPG`). In addition, the framework is able to deal with non-monotone problems, where multiple grasps of an object are needed in order to find a solution to the problem. Such problems are challenging for alternative methodologies.

Chapter 5 presents a probabilistically complete method for solving hard rearrangement problems. 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 including multiple movable bodies and complex kinematic constraints. This work initially revisits an existing backtracking search method for detecting the sequence of objects to be moved so as to reach a desirable, previously unreachable object. Following similar ideas with this motivating work, two powerful rearrangement primitives that could solve monotones and non-monotones rearrangement problems, are proposed. This chapter aims to emphasize the benefit of using more powerful rearrangement primitives in the context of task planning for object rearrangement than an individual pick-and-place. A traditional local planner

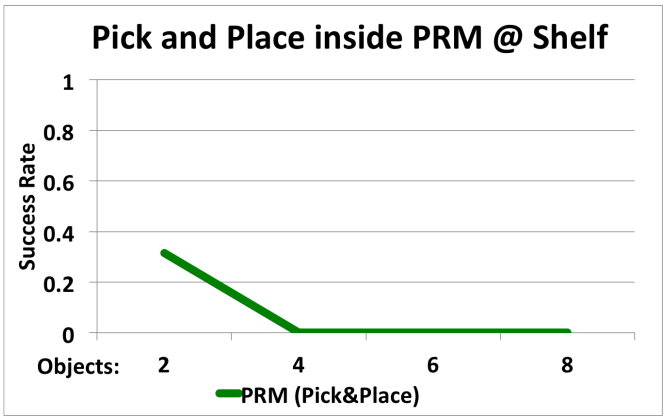

Figure 1.4: Success ratio given 30 minutes of computation when using an individual pick-and-place action as a local planner in a PRM-like task planner for rearranging objects in a shelf.

would be an individual pick-and-place, where a single object is displaced between two nodes. Fig.1.4 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 reliable, faster and with better path quality, i.e., fewer objects grasped during the rearrangement. 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.

Contributions: The first contribution is a method that addresses many non-monotone challenges (plRS), which is an extension of the monotone rearrangement primitive (mRS). The monotone rearrangement primitive was the result of adapting the backtracking search algorithm [110]. In order to solve the non-monotone cases, the method is using the minimum constraint removal paths (MCR) to transfer each object to its target. An MCR path minimizes the number of constraints that need to be removed from the path of an object. The second contribution is the use of either the monotone or of the new non-monotone method as a local connection primitives in the context of a higher-level task planner that searches the space of object placements and which provides probabilistic completeness guarantees. This chapter also provides the analysis

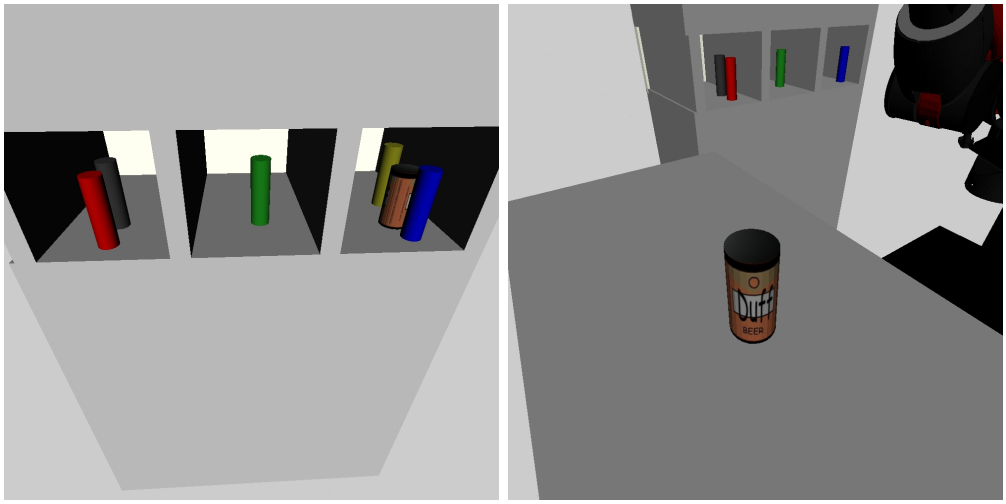

Figure 1.5: In this problem the manipulator has to remove a beer can from a shelf, where the can is obstructed by an object. The can needs to be placed on the table while all the other objects should be placed back in their original positions.

that argues the existence of probabilistic completeness for the integration of a task planner that searches the space of object arrangements.

Chapter 6 addresses the limitations of the previous framework for solving rearrangement problems, using backtracking search, and proposes an approximate but significantly faster alternative for general rearrangement instances while maintaining probabilistic completeness properties. The method defines a dependency graph between objects given minimum constraint removal paths (`MCR`) to transfer each object to its target. From this graph, the approach discovers the order of moving objects by performing topological sorting without backtracking search. The approximation arises from the limitation to consider only `MCR` paths, which minimize, however, the number of conflicts between objects. In order to efficiently solve non-monotone instances, these primitives are incorporated in a higher-level incremental search algorithm for general rearrangement planning, which operates similarly to `Bi-RRT`. Given a start and a goal object arrangement, tree structures of reachable new arrangements are generated by using any of the primitives as an expansion procedure. The integrated solution achieves probabilistic completeness for the general non-monotone case and based on simulated experiments it achieves very good success ratios, solution times and path quality relative to alternatives.

Contributions: The first contribution of this work is the introduction of new methods for solving rearrangement instances by taking advantage of recent algorithmic insights [115, 47, 67]. The proposed methods, "fast monotone Rearrangement Solver" (`fmRS`) and "fast non-monotone Rearrangement Solver" (`fplRS`) avoid backtracking search given the following observation: if the paths for transferring the objects are fixed, then one can easily compute the sequence of moving objects without collisions - if one exists [115]. In particular, the sequence is the output of topological sorting on a "constraint graph" with objects as nodes and directed edges indicating the ordering dependence between objects. The fixed paths are computed using "minimum constraint removal" (`MCR`) paths [47] for minimizing the number of collisions with other objects. The second contribution is the adaptation of both the original monotone solver based on backtracking search (`mRS`) and the new fast monotone solver (`fmRS`) so that they return a partial solution when no complete solution is found. This involves moving as many objects as possible towards the target arrangement. Given the partial formulation of the rearrangement primitives, it is possible to use them in a high-level task planner as an expansion step from an initial arrangement instead of a connection between two arrangements. This allows for the efficient use of such methods in the context of higher-level search procedures similar to `Bi-RRT` [80, 79], where two tree data structures originating at the start and goal arrangements are built. The tree nodes correspond to object arrangements and are connected with edges, which are monotone rearrangement paths. The integrated solution achieves probabilistic completeness, efficiency and high success ratios.

Chapter 7 shares all the implementation details for a computationally efficient implementation. The object rearrangement is a challenging problem involving multiple movable bodies and complex kinematic constraints. However, it is possible to pre-build roadmaps for the manipulation motion given the static geometry. These roadmaps will help to have a fast manipulation planning framework. Building all these roadmaps thought it is not easy. This chapter discusses the implementation of primitives that will affect the performance of the high-level task planners. In order to further speed up the detection of the path a variation of bounded path length approach [67, 70] for

computing "minimum constraint removal " (MCR) path [47] is used as an alternative. This methodology bounds the length of paths that it searches.

Finally, **Chapter 8** restates the key contributions of the work, summarizes open questions and proposes interesting directions for future investigation. Furthermore, it cites the potential for this work to inform future investigation of these methods.

Chapter 2

Related Literature

Rearrangement planning [10, 92] relates to various lines of work in the robotics literature.

2.1 Planning among Movable Obstacles

A related and important challenge for mobile systems is the problem of navigation among movable obstacles (NAMO). Early on it was shown that NAMO is NP-hard [120] and later on it was confirmed that this is the case even for simpler instance that involves only unit square obstacles [30]. Due to problem complexity, most efforts have focused on efficiency [20, 91] and provide completeness results only for subclasses of the problem [108, 109]. A probabilistically complete solution for NAMO was eventually provided [116], but it can only be applied to lower dimensional robots (2-3 DOFS) and corresponds to uninformed brute force search. More recently, a decision-theoretic framework for NAMO has also been presented, which deals with the inherent uncertainty in both perception and control of real robots [82].

2.2 Minimum Constraint Removal Path

Related challenges to the problem considered in this work are disconnection proving, excuse-making and minimum constraint removal problems. The applications of disconnection proving correspond primarily to feasibility algorithms that try to detect if a solution exists given certain constraints in the environment [8, 15, 87, 124]. Nevertheless, this is a computationally expensive operation, where practical solutions are limited to low-dimensional or geometrically simple configuration spaces. A related challenge is to consider excuse-making in symbolic planning problems. In these approaches the

"excuse" is used to change the initial state to a state that will yield a feasible solution [43].

MCR formulations can be useful in the context of navigation among movable obstacles (NAMO), [108, 109, 120], as well as manipulation in cluttered environments [34], where it is necessary to evacuate a set of obstacles for an agent to reach its target. Such challenges were shown to be hard if the final locations of the obstacles are unspecified and PSPACE-hard when specified [120]. It is even NP-hard for simple instances with unit square obstacles [30]. Thus, most efforts have dealt with efficiency [20, 91] and provide completeness results only for problem subclasses [108, 109]. NAMO challenges relate to the Sokoban puzzle, for which search methods and proper abstractions have been developed [14, 93].

Similar work to the computation of MCR paths has addressed violating low priority tasks for multi-objective tasks specified in terms of LTL formulas [98]. An iteratively deepening task and motion planning method uses similar ideas in order to add and remove constraints on motion feasibility at the task level [26].

In the minimum constraint removal problem, the goal is to minimize the number of constraints that have to be displaced in order to yield a feasible path [48, 47, 67]. Different approaches have been proposed in the related literature. A computationally infeasible approach that searches all possible paths, as well as a faster greedy, but incomplete strategy were presented together with the formulation of the problem in the context of robotics challenges [48, 47]. The minimum constraint removal problem is proven to be NP-hard, even when the obstacles are restricted to being convex polygons [38].

2.3 Multi-robot Motion Planning

A motivation for the current work is to utilize progress in discrete solvers for multi-robot motion planning [71, 115, 84, 118, 122] in the context of rearrangement planning. Multi-robot motion planning is itself a hard problem, and coupled, complete approaches typically do not scale well with additional robots, even though there are methods that

decrease the number of effective DOFs [5]. On the other hand, decoupled methods, such as priority-based schemes [114] or velocity tuning [81], trade completeness for efficiency.

A related line of work in algorithmic theory deals with "pebble motion on graphs" problems, where pebbles need to move from an initial to a goal vertex assignment on a graph [66, 17]. Testing the feasibility of such problem instances can be answered in linear time [6, 44], inspiring a recent method for continuous multi-robot motion planning [104]. The first method for solving unlabeled cases employs sampling-based planners and reduces multi-robot planning into a sequence of discrete pebble problems, such that movement of the pebbles quickly translate to valid robot motions. The first part of the current work is motivated by this approach and defines "rearrangement pebble graphs" (MPGs) to solve manipulation challenges.

2.4 Manipulation Planning

The focus of this work is on building solutions for high-DOF robotic arms and solve manipulation challenges. Such problems can be approached with a multi-modal "manipulation graph" abstraction that contains "transit" and "transfer" paths [2, 1, 103]. The graph can be constructed through a sampling-based process [61, 80]. The current work follows a similar formalization and applies it appropriately to the case of multiple similar movable objects towards achieving an efficient solution. The current work is also utilizing asymptotically optimal near-optimal sampling-based planners for the computation of the manipulation graph [33, 86], i.e., roadmap spanners of PRM* [58].

Tree sampling-based planners have also been used successfully in the context of manipulation planning [12, 13]. A variety of approaches exist for manipulation planning beyond sampling-based planners, which could also be employed in the context of the proposed methods, such as heuristic search [23], or optimization-based methods, such as CHOMP [125, 63]. The focus of the current thesis is more on the combinatorial challenges that arise from reasoning about multiple objects and not the actual manipulation method. In this process, this work is making significant use of heuristic search primitives over sampling-based roadmaps. The output of the current algorithm could

also potentially be integrated with methods like CHOMP to improve the quality of the computed solution. There is also significant effort that focuses on identifying appropriate grasps in complex scenes [11, 22] but the current thesis does not focus on this aspect of the challenge.

The above manipulation efforts deal with moving and grasping individual objects. Manipulation planning among multiple movable obstacles has been considered before for "monotone" problems where each obstacle can be moved at most once [110, 88]. The solutions in the current work can reason about more complex challenges.

Manipulation planning among multiple movable obstacles has been considered for "monotone" problems where each obstacle can be moved at most once [110, 88]. Non-monotone instances are recognized as hard rearrangement challenges. There are recent approaches for rearrangement that can deal with non-monotone instances under certain conditions [51, 106, 40, 72, 92]. One method simplifies the problem by requiring that all objects are unlabeled, i.e., interchangeable and can occupy any pose in a final arrangement [72]. An alternative method imposes a grid for placing the objects and then uses techniques, such as answer-set programming [51]. The current work, however, can reason about more complex challenges. Assembly planning is also solving similar multi-body problems but the focus there is on separating a collection of parts and typically the robot path is ignored [121, 45, 111]. Another paradigm for dealing with cluttered scenes involves non-prehensile manipulation, such as pushing objects [25, 34]. The current solution could potentially be extended to include such actions but the focus in this work is on grasping primitives.

2.5 Task and Motion Planning

Solving complex problems require the integration of task and motion planning. There are cases where the task planner is calling the motion planner directly for just feasibility checks for an individual task action [35]. However, most of the times the communication between the task and motion planner is more intense, where more advanced methods are needed. Several methods by extending Hierarchical Task Networks (HTN) [39]

with geometric primitives, introduced a new hierarchical combined task and motion planner [29, 42], where the symbolic planner backtracks at the geometric level to detect different geometric solutions. Similar to Hierarchical Task Networks (HTN) idea, several approaches are using a Satisfiability Modulo Theories (SMT) [28] solver to generate task and motion plans using a static graph [89, 119]. Formulating the task and motion planning problem as a satisfiability modulo theories (SMT) problem will result in a probabilistically complete algorithm [26], where the algorithm will iteratively increase the plan depth and the motion planning timeouts until it finds a solution to the problem.

Rearrangement planning can be seen as an instance of integrated task and motion planning, which can be seen as an important step towards solving more complex, cognitive challenges in robotics and especially in manipulation [18, 95, 106, 40]. Many approaches employ a high-level symbolic planner [56, 37]. For instance, geometric constraints can be incorporated into the high-level language [36], or it is possible to plan in the cross product of the high-level symbolic reasoning and the low-level configurations [18]. Multi-modal roadmaps can deal with the combination of both discrete and continuous parameters [15, 49, 50], which is also useful for the current challenge. Recent methods generate heuristics for symbolic manipulation planning with lazily-expanded roadmaps [40], or discover a symbolic language for manipulation on the fly [65]. Some methods are splitting the rearrangement problem into two types of actions, in order to deal separately with the objects and the robot and be able to use non-prehensile interactions between the arm and the objects. The current method deals only with prehensile interactions.

2.6 Sampling-based Methods for Motion Planning

Informally, motion planning is the process by which a robot system uses an internal representation of its environment to compute a sequence of controls to actuate the system, so as to move the robot system from an initial position to a target position while remaining safe, i.e., avoiding collisions.

Algorithms dealing with the motion planning problem generally operate in the

robotic system's configuration space ($\mathcal{C}$), where a point in this space is called a configuration q.

Definition 1 (Configuration q)**.** *A configuration is the parameterization of a robotic system's degrees of freedom, where this set of parameters fully specifies the location of every rigid body geometry of the robotic system.*

The configuration space can be partitioned into two sets: the invalid or the obstacle set $\mathcal{C}^{\text{inv}}$ where the robotic system is violating constraints of the problem, i.e. collision with obstacles, and the collision-free space, $\mathcal{C}^{\text{free}}$, where the robotic system's configuration is valid and safe. These two partitions cover the whole configuration space, $\mathcal{C} = \mathcal{C}^{\text{free}} \cup \mathcal{C}^{\text{inv}}$.

A complete algorithm for solving the motion planning problem, known as the **Piano Mover's Problem**, was presented in the thesis of Canny [19]. Even this simple version of the problem is $\texttt{PSPACE} - \texttt{HARD}$ [96], where the difficulty of the problem scales exponentially with the number of degrees of freedom of the robotic system. This algorithm also introduced the concept of a roadmap, a graph data structure that maintains connectivity in the $\mathcal{C}^{\text{free}}$ space. The algorithm, however, is impractical, similar to other early methods that attempted to leverage $\mathcal{C}^{\text{free}}$ approximations [16, 83, 57].

Other approaches not explored in this work approach the problem from different directions. Some approaches apply a grid discretization over the configuration space and perform discrete search directly [107]. Another paradigm designed to solve these problems is based on artificial potential fields [62, 55, 41], though complete versions of this methodological approach are difficult to apply in general configuration spaces [64, 99]. A similar technique that solved difficult problems took a stochastic approach to avoid local minima [7, 75]. The difficulties faced by these methods and the advent of efficient collision checking primitives would motivate the development of sampling-based motion-planning approaches.

Sampling-based motion planning techniques provide efficient solutions in practice, even for high-dimensional, geometrically complex problems [76, 78, 21]. Two primary families of planners have emerged. Probabilistic Roadmap Methods (PRMs) preprocess

a robot configuration space ($\mathcal{C}$ -space) to create a multi-query structure [61, 60]. Tree-based planners, such as the Rapidly-exploring Random Tree (`RRT`) are suited to rapid single-query planning, especially for dynamic systems [77, 79]. There are also methods that lie somewhere between, in order to reap the advantages of both algorithmic approaches [94, 3]. In general, these methods rely on several primitives to construct a planning structure in the configuration space, which are explained next.

2.6.1 Sampling-based Planning Primitives

As implied by the name, these methods employ some sampling method to generate free configurations in the configuration space. There has been success in the field with several approaches to sampling, though most commonly these methods are leveraged using uniform random samplers.

In the current work, the sampling module will have to sample a configuration point in the arrangement space. This means that multiple poses, equal to the number of the objects in the specific instance, will be selected and placed as a single point. In order for the point, i.e. arrangement, to be valid the selected poses must be collision-free between them.

Traditionally, these approaches are applied to find shortest paths through the configuration space according to a distance function.

Definition 2 (Distance Function)**.** *The distance function $d(\cdot,\cdot)$ takes two configurations in $\mathcal{C}$ and returns a real value, i.e., $d(q_i, q_j) \rightarrow \mathbb{R}$, which expresses the distance of the two configurations in the absence of obstacles.*

Given that a sample in the current work represents an arrangement of objects, the distance function will measure the distance between each pose in order to define the distance between two configurations. For example for the arrangements α and α' the distance will be $\sum_{\forall o \in \alpha} d(\alpha[o], \alpha'[o])$.

In order to construct the planning structure in $\mathcal{C}^{\text{free}}$, sampling-based methods rely on being able to accept or reject samples based on whether a configuration is safe. That is, the approach leverages a method which determines whether configuration q

lies within $\mathcal{C}^{\text{free}}$ or $\mathcal{C}^{\text{inv}}$. Prototypically, these methods rely on the availability of a collision checker to determine the validity of samples, but the general term for such a module is a validity checker.

Definition 3 (Validity Checker)**.** *Given an individual configuration q, a validity checker returns whether q lies within $\mathcal{C}^{\text{free}}$ or $\mathcal{C}^{\text{inv}}$.*

The validity checker will have to check if the objects on the selected poses are in collision both with the static obstacles and the objects between them. Only if both of these checks are collision-free then the sample is a valid sample.

The following section will outline Probabilistic Roadmap Methods, which produce a planning structure that is a graph. Samples drawn within $\mathcal{C}^{\text{free}}$ will be added as nodes, but in order to build the planning structure, the sampled configurations must be connected by edges. This is typically accomplished with the aid of a local planner.

Definition 4 (Local Planner)**.** *Given configurations q_{begin}, q_{end}, a local planner $\mathcal{L}(\cdot,\cdot)$ returns the optimal path between the configurations* **in the absence of obstacles***, i.e., $\mathcal{L}(q_{begin}, q_{end}) \rightarrow \pi_{\mathcal{L}}$, where $\pi_{\mathcal{L}}(t) \rightarrow \mathcal{C}$ and $t \in [0,1]$, satisfying $\pi_{\mathcal{L}}(0) = q_{begin}$ and $\pi_{\mathcal{L}}(1) = q_{end}$.*

Typically, a straight line between q_{begin} and q_{end} in $\mathcal{C}$ is used for local planning. In order to add local paths as edges in the planning structure, it must be that for all $q \in \mathcal{L}(q_{\text{begin}}, q_{\text{end}})$, $q \in \mathcal{C}^{\text{free}}$. While there exist efficient and complete methods for checking if an entire path lies entirely within $\mathcal{C}^{\text{free}}$ [101], a sampling-based process is commonly used as an approximation.

In the current setup, the "straight" line between the two sample will be the rearrangement of the objects from the one arrangement, i.e. q_{begin}, to the final arrangement, i.e. q_{end}. For connecting these two samples rearrangement primitives will be used that will be explained in detail in sections 5.1 and 6.2. This work aims to emphasize the benefit of using more powerful connection primitives in the context of building a graph representation in order to solve object rearrangement problems, than an individual pick-and-place.

2.6.2 Probabilistic Roadmap Methods

Algorithm 1: PRM(n)

```
1 V ← ∅; E ← ∅;
2 for i = 1 ... n do
3     v ← SAMPLE();
4     V ← V ∪ v;
5     V_near ← NEAR(V, v);
6     for u ∈ V_near do
7         if π(v, u) ∈ C^free then
8             E ← E ∪ {π(v, u)};
9 return G = (V, E);
```

One of the first popular sampling-based motion planning methods, in order to construct a roadmap in $\mathcal{C}^{\text{free}}$, was the **Probabilistic Roadmap Method** (PRM) [61]. The high-level operations of PRM are outlined in Algorithm 1. For a set number of iterations n (Line 2), the algorithm samples a configuration in $\mathcal{C}^{\text{free}}$ (Line 3) and adds it as a node to the roadmap (Line 4). It then tries to connect it with a local path to a set of k-closest neighbors among the existing nodes (k-PRM) (Lines 5,6). If any local path lies entirely within $\mathcal{C}^{\text{free}}$ (Line 8), an edge is added to the roadmap (Line 9). The advantages of the method lie in its simplicity, and generality while scaling to higher-dimensional problem instances than other competing methods were capable of. While strict completeness and path non-existence cannot typically be proven for the PRM, probabilistic completeness can be provided instead.

Definition 5 (Probabilistic Completeness)**.** *Let* $(\mathcal{C}^{\text{free}}, q_{start}, q_{goal})$ *be a motion planning problem that admits a continuous trajectory* $\pi : [0, 1] \rightarrow \mathcal{C}^{\text{free}}$ *subject to* $\pi(0) = q_{start}$ *and* $\pi(1) = q_{goal}$*. Then, an algorithm ALG that runs for n iterations is probabilistically complete if the probability that ALG returns a solution to the motion planning problem converges to* 1 *as n tends to infinity.*

This was originally proven for d-dimensional manifolds [59, 52], for non-holonomic robots [112], and then later for a broad class of problems [73].

The PRM framework has been adapted so as to solve a variety of different challenges

beyond the basic motion planning problem, involving multiple robots [100], manipulation planning [90], assembly planning [111], planning for flexible objects [74] and bioinformatics applications [4].

In this work the PRM framework is used as high-level task planner in order to solve object rearrangement problems [68, 72]. A PRM-like roadmap is constructed in the arrangement space in order to find a solution path and connect the initial with the final arrangement. Chapter 5 will give in details how the PRM framework is being used and how is proven to be probabilistically complete for the object arrangement problem.

2.6.3 Random Tree Methods

An extremely popular alternative to roadmap-based methods explore the $\mathcal{C}$-space by incrementally propagating from existing nodes in the planning structure to grow a tree. One such common and popular method is the Rapidly-exploring Random Tree (RRT) approach [80]. This method is commonly employed due to the inherent "Voronoi Bias" it exhibits. That is, the tree is automatically biased toward quickly growing toward unexplored regions of the space. Around the same time, a similar tree-based approach was proposed that probabilistically biases the expansion of these trees toward unexplored regions of the space [53, 54].

These methods tend to be more efficient in quickly answering individual motion queries and they can be easily applied to problems involving robot dynamics because they do not depend on the existence of a steering method that exactly connects two configurations of the system. Such a steering function is typically required in the construction of a roadmap. Furthermore, by definition they already return sparse data structures, as a tree is minimally sparse; however, it does not ensure that the number of nodes in this tree remains low.

Chapter 6 will use the RRT framework again as a high-level task planner in order to deal with object rearrangement problems. The incremental propagation idea maintains good connectivity of the graph which is important for rearrangement problems. A variation of the method is used, Bi-RRT, where a tree is growing for both the initial point and the final point. More details will be presented in section 6.3.

Chapter 3

Problem Setup and Notations

Consider a 3D workspace $\mathcal{W}$, which contains static obstacles. There is also a set of k **movable rigid-body objects** $\mathcal{O}$. Each object $o_i \in \mathcal{O}$ can acquire a **pose** $p_i \in \mathcal{P} \subseteq SE(3)$, i.e., a pose p_i specifies both the three dimensional position and orientation of a movable object o_i. This work uses the set of stable poses $\mathcal{P}^s \subset \mathcal{P}$ corresponding to the placement of the objects on a horizontal surface, i.e., $\mathcal{P}^s \equiv SE(2)$.

Beyond the objects, the workspace also contains a **robotic arm** $\mathcal{R}$, which is able to move the objects. The robot acquires **arm configurations** $q \in \mathcal{Q}$, which together with the static parameters of the arm fully specify the 3D volume occupied by the robot. Given the pose p_i of an object o_i, the notation $q(p_i, e)$ will denote a **grasping arm configuration**, i.e., a configuration that allows the robotic arm's end effector to grasp object o_i placed at pose p_i, where the pose of the end effector relative to the object's pose is denoted as $e \subseteq SE(3)$. One way to compute a grasping configuration is through the use of inverse kinematics. Given a redundant manipulator, there may be multiple arm configurations that manage to grasp an object at a specific pose p_i but they are uniquely differentiated by the relative end effector pose e.

The state space $\mathbb{X} = \mathcal{Q} \times \mathcal{A}$ of the rearrangement problem is the Cartesian product of the arm's configuration space $\mathcal{Q}$ and the space of object arrangements $\mathcal{A}$. The collision-free subset $\mathbb{X}_{\text{free}} \subset \mathbb{X}$ does not allow collisions between the arm and objects, among the objects and between the static obstacles and the arm or objects. The set $\mathbb{X}_{\text{free}}$ does allow intersections between the surface of the arm and the surface of the movable objects, so that grasps can take place.

There are $k + 1$ types of collision-free states of interest for robot rearrangement problems:

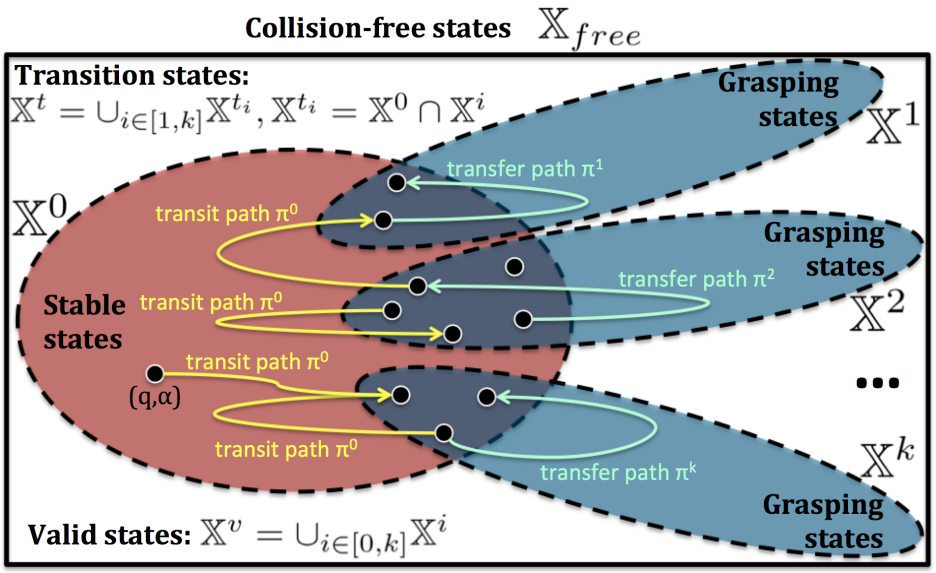

Figure 3.1: The relevant subsets of collision-free states for rearrangement problems and the motion planning primitives (i.e., transit and transfer paths) used by the algorithms in this thesis.

i) **Stable** states $\mathbb{X}^0$: These are collision-free states where all objects stably rest on surfaces without external forces applied to them beyond gravity.

ii) **Grasping** states $\mathbb{X}^i \in \mathbb{X}^g$ for each object o_i, where $i \in [1, k]$: These are collision-free states where the object $o_i \in \mathcal{O}$ is grasped by the arm's end effector and the other objects stably rest.

Then, **valid** states $\mathbb{X}^v$ correspond to the union of stable and grasping states, i.e., $\mathbb{X}^v = \cup_{i\in[0,k]}\mathbb{X}^i$. The subset of grasping states $\mathbb{X}^i$ for an object o_i, which are also stable, are defined as the transition states $\mathbb{X}^{t_i}$ for the object o_i, i.e., $\mathbb{X}^{t_i} = \mathbb{X}^0 \cap \mathbb{X}^i$. In a transition state for object o_i, the object o_i is both grasped and stably resting on a surface, while the remaining objects are stably resting. The union of all such states is the set of **transition** states $\mathbb{X}^t = \cup_{i\in[1,k]}\mathbb{X}^{t_i}$. The topology of $\mathbb{X}_{\text{free}}$ for the rearrangement problem is illustrated in Figure 3.1.

A **transit path** $\pi^0 \in \Pi^0 : [0, 1] \to (\mathcal{Q}, \alpha) \subset \mathbb{X}^0$ is a sequence of stable states, where the arrangement of objects α remains the same and the sequence of arm configurations along π^0 follows a continuous function. The following discussion will assume a motion planning primitive:

$$\texttt{TRANSIT}(q_I, q_F, \alpha) \to \Pi^0,$$

which computes a transit path so that the arm moves between arm configurations q_I and q_F in a collision-free manner given object arrangement α.

A **transfer path** for object o_i: $\pi^i \in \Pi^i : [0,1] \rightarrow \mathbb{X}^i$, where $i \in [1,k]$, is a sequence of grasping states for object o_i. The poses of all other objects $\{\mathcal{O} \setminus o_i\}$ remain the same but the pose p_i of object o_i changes in a continuous manner. The arm's configuration also changes in a continuous manner so that it always corresponds to a grasping configuration $q(p_i, e)$ for the same relative end effector pose e along π^i. The following discussion will assume a motion planning primitive:

$$\texttt{TRANSFER}(o_i, p_I, p_F, e, \alpha) \rightarrow \Pi^i,$$

which computes a valid path for the arm to transfer the object o_i from pose p_I to pose p_F, while grasping the object with a relative end effector pose e. This path is collision-free given the poses of the remaining objects specified by $\alpha[\mathcal{O} \setminus o_i]$. It has to be that $p_I = \alpha[o_i]$. The actual implementation of the transit and transfer primitives will be discussed in Chapter 7.

A **legal transition** between a transit path π^0 and a transfer path π^i can occur as long as $\pi^0(1) = \pi^i(0) \in \mathbb{X}^{t_i}$. Similarly, the transition between a transfer path π^i to a transit path π^0 is legal as long as $\pi^i(1) = \pi^0(0) \in \mathbb{X}^{t_i}$. This means that a legal transition arises if the last state of the first path and the first state of the following path correspond to the same transition state for the object moved by the transfer path.

A **rearrangement path** $\pi \in \Pi : [0,1] \rightarrow \mathbb{X}^v$ is a sequence of valid rearrangement states, which can be decomposed into an alternating sequence of transit and transfer sub-paths $\pi = \{\pi_1^0, \pi_1^{g_1}, \pi_2^0, \pi_2^{g_2}, \ldots\}$, where $g_j \in [1,k]$, so that all the transitions between subpaths are legal.

Definition 6 (**Prehensile Single-Arm Rearrangement Problem**). *Consider a workspace $\mathcal{W}$ that contains a robot arm $\mathcal{R}$ and a set of k movable objects $\mathcal{O}$. Given an initial state $x_0 = (q_0, \alpha_0) \in \mathbb{X}^0$ (Fig.1.1 first) and a final state $x_1 = (q_1, \alpha_1) \in \mathbb{X}^0$ (Fig.1.1 last), compute a solution rearrangement path $\pi \in \Pi : [0,1] \rightarrow \mathbb{X}^v$ such that $\pi(0) = (q_I, \alpha_I)$ and $\pi(1) = (q_F, \alpha_F)$. The path is an alternating sequence of transit and transfer states. The types of states along path π change only when $\pi(s) \in \mathbb{X}^t$.*

3.1 Unlabeled vs Labeled Problem

The rearrangement problems can be split into two categories based on the initial and target arrangements, the unlabeled and labeled problems.

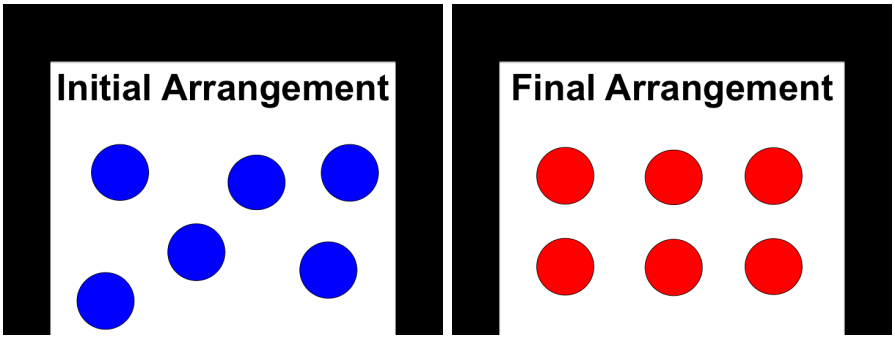

Figure 3.2: Initial and final arrangement of 6 objects. Each object can occupy any of the final poses.

An **unlabeled arrangement** $\alpha \in \mathcal{A}$ of k objects specifies a k-combination of poses $\{p_1, \ldots, p_k\}$, where $p_i \in \mathcal{P}$ and results in a *collision-free* placement of the objects in $\mathcal{W}$. Permutations of α are indistinguishable, since the objects are unlabeled. In an unlabeled problem the final arrangement does not specify the final pose for each of the k objects. A problem is considered solved when all the final poses are occupied by an object.

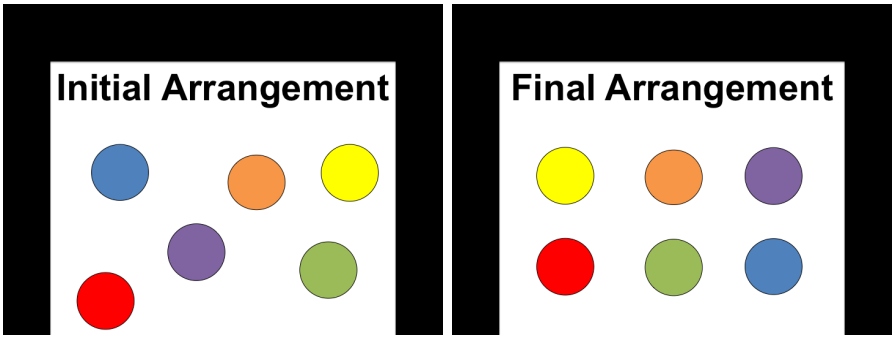

Figure 3.3: Initial and final arrangement of 6 objects. Each object has a specific final pose, that need to occupy in order to consider the problem solved.

A **labeled arrangement** $\alpha \in \mathcal{A}$ specifies the k poses $\{p_1, \ldots, p_k\}$ for all the objects in $\mathcal{O}$. Then, the notation $\alpha[\mathcal{O}' \subseteq \mathcal{O}]$ will indicate the poses that the objects $\mathcal{O}' \subseteq \mathcal{O}$ occupy according to the arrangement α. The notation $\alpha_F[\mathcal{O}]$ (Fig.3.3 right) specify the

final pose for each individual object. Each object can be place on any of these poses, temporarily. However, the problem is solved only when each object occupies the final pose that is assigned by the final arrangement to this object.

3.2 Monotone vs Non-Monotone Problem

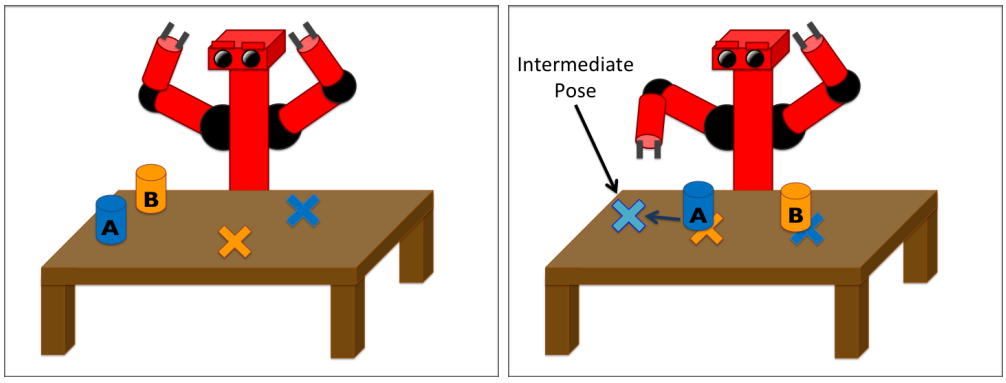

Figure 3.4: Left: A monotone problem where the robot could easily solve by first moving object B to its final pose following by object A. Right: A non-monotone problem, where the robot will have to move object A to an intermediate pose before move it to its final pose.

The rearrangement problems can also be categorized based on how many times an object has to be transferred before a solution to the problem is found. These two categories depict in figure 3.4. The monotone problems are in general easier problems given that the robot needs to find the correct order to move the object to their final location.

Definition 7 (Monotone Rearrangement Problem)**.** *A rearrangement problem is called monotone when the solution path, from an initial α_I arrangement to a final α_F arrangement, will have at most k transfer subpaths, where each object is transferred at most once (Fig. 3.4 (left))*

Frequently, there will be overlaps between initial and final poses. This does not mean, necessarily that the problem is not a monotone problem. However, there are many cases (Fig.3.4(right)) where the robot will have to move an object to an intermediate pose before bringing it to its final pose. These problems will be called

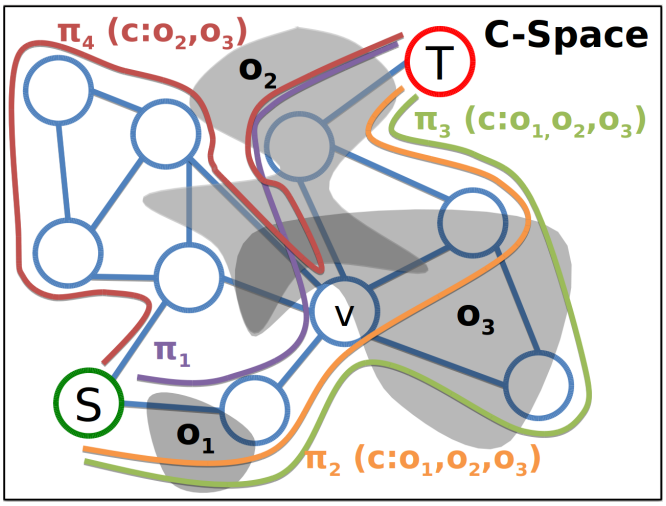

Figure 3.5: An example of a graph embedded in a space with constraints. The dark areas correspond to regions with constraints, which if removed they allow for a feasible path along the graph.

non-monotone and in general are harder problems, because an intermediate pose has to be detected.

Definition 8 (Non-monotone Rearrangement Problem)**.** *A rearrangement problem is called non-monotone when in order to find a solution at least one object will have to move to any intermediate pose before reaching its final pose (Fig.3.4 (right)).*

3.3 Minimum Constraint Removal Paths for Manipulation Planning

Frequently, motion planning challenges do not have a solution, given the presence of constraints in the environment. In scenarios where the path is blocked by movable objects, the minimum constraint removal problem asks for the minimum set of constraints, which if they removed from the scene, then a feasible solution exists.

Consider a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ that represents the connectivity of a state space. Each node corresponds to a state of the manipulator, while an edge expresses a local trajectory between two states. The weight of an edge is set equal to the distance between the two nodes defining the edge in the manipulator's state space. Moreover, it is possible to define for each edge $e \in \mathcal{E}$ a set of constraints c_e. The objective is to compute a

path on $\mathcal{G}$ that minimizes the number of constraints that the path traverses. Figure 3.5 describes a relevant setup.

Definition 9 (Constraints)**.** *All the objects that are blocking the path π of the object o_i to its final pose $\alpha_F[o_i]$ will be called "constraints" of the path π, $\pi_{\mathcal{B}}(\pi)$.*

The constraints along a path $\pi(v,u) = \{e_1, \ldots, e_n\}$ will be denoted as $\pi_{\mathcal{B}}(\pi(v,u))$ and correspond to the union of the constraints along the edges of the path: $\pi_{\mathcal{B}}(\pi) = \cup_i c_{e_i}$.

Definition 10 (`MCR`: Minimum Constraint Removal Path [47])**.** *Consider the state space $\mathbb{X} = \mathcal{Q} \times \mathcal{A}$, k obstacle regions $O_1, \cdots, O_k \subset \mathbb{X}$ and endpoints q_s,$q_f \in \mathbb{X}$. Also, $c(q)$ determines the set of obstacles violated at configuration q : $c(q) = \{i | q \in \mathcal{Q}\}$. A configuration q_f is reachable from q_s if there exists a continuous path $y \in \mathbb{X}$ starting at q_s and ending at q_f. A minimum constraint path (`MCR`) is the path between q_s and q_f that minimizes the number of violated obstacle regions.*

3.3.1 The case of manipulation

Consider the following setup in a 3D workspace:

- A robotic manipulator, that is able to acquire configurations $q \in \mathcal{Q}$, where $\mathcal{Q}$ is the manipulator's configuration space.
- A set of static obstacles $\mathcal{S}$. Given the presence of $\mathcal{S}$ it is possible to define the subset of $\mathcal{Q}$ that does not result in collisions with the static obstacles: $\mathcal{Q}_{free}$.
- A set of movable rigid-body objects $\mathcal{O}$, where each object $o_i \in \mathcal{O}$ can acquire a pose $p_i \in SE(3)$.
- A target object o, which is located in a starting pose $p^s \in SE(3)$ and needs to be transferred to a target pose $p^t \in SE(3)$. The target object o does not belong in the set $\mathcal{O}$.

Given the pose p of object o, it is possible to define a grasping configuration $q(p) \in \mathcal{Q}$ for the manipulator. For instance, this can be achieved through the use of inverse

kinematics. The underlying manipulation challenge considered here is to find a path for the robot manipulator, which starts from a given grasping configuration $q(p^s) \in \mathcal{Q}_{free}$ for the start pose p^s of object o and transfers the object to a target pose p^t given a grasping configuration $q(p^t) \in \mathcal{Q}_{free}$. The relative pose between the robot's end-effector and the object - i.e., the grasp - is the same given the arm configuration/object pose pairs $(q(p^s), p^s)$ and $(q(p^t), p^t)$, so no in-hand manipulation is necessary to transfer the object.

Beyond the static geometry, the problem is further complicated by the presence of the movable objects $\mathcal{O}$. Each object $o_i \in \mathcal{O}$ at pose p_i defines a subset of manipulator configurations $\mathcal{Q}^{o_i} \subset \mathcal{Q}_{free}$ that result in a collision between object o_i and the manipulator or the transferred object o given the grasp. The focus is on situations where there is no solution path that takes the manipulator carrying the object o from $q(p^s)$ to $q(p^t)$ without intersecting any of the sets $\mathcal{Q}^{o_i}$, where $o_i \in \mathcal{O}$. In this context, the objective is then adapted so as to identify the minimum set of movable objects o_i that need to be removed from the scene so as to be able to solve the original manipulation objective.

To deal with this objective and along the lines of the abstract `MCR` problem definition, this work assumes that a graph structure in the collision-free configuration space $\mathcal{Q}_{free}$ of the manipulator, i.e., a roadmap, is computed first in order to compute paths for the manipulator. This can be done either by using sampling-based planners that sample nodes and edges of a roadmap in the configuration space, such as with a `PRM` approach or other sampling-based planners [61, 80, 13, 2], or search-based methods that implicitly consider a discretization of the configuration space and directly search over it [24].

Such a roadmap for the manipulator can be seen as the equivalent to the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ considered in the abstract `MCR` definition. The nodes $\mathcal{V}$ of the roadmap correspond to configurations of the manipulator in $\mathcal{Q}_{free}$ and the edges $\mathcal{E}$ to straight-line paths in $\mathcal{Q}_{free}$ that connect pairs of nodes. The roadmap can be used to search for transfer paths for the object o by placing the object at the end-effector given the grasp defined according to $q(p^s)$ to $q(p^t)$ during query resolution. Then, traversing an edge e of the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, corresponds to a sequence of motions for the manipulator carrying the object o, which are collision-free with the static geometry $\mathcal{S}$ but may intersect a

subset of the $\mathcal{Q}^{o_i}$ configuration sets.

Intersections with one of the $\mathcal{Q}^{o_i}$ along an edge $e \in \mathcal{E}$ are equivalent to the edge e having a constraint that needs to be avoided in the context of `MCR`. The set of constraints of an edge e of the roadmap will be denoted as c_e, and correspond to the set of objects in $\mathcal{O}$ that cause a collision with the manipulator or the carried object o along the edge e. An example is shown in Figure 3.5, where a graph is embedded in the configuration space $\mathcal{Q}_{free}$ and the gray regions correspond to constraints arising from different movable obstacles.

Then, similar to the abstract problem, the objective is to find the path $\pi = \{e_1, \ldots, e_n\}$ along the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ storing manipulator configurations, which has the minimum number of constraints. This path will correspond to the minimum number of objects $\mathcal{O}$ that need to be removed in order for the manipulator to be able to move the object o_i from pose p^s to pose p_i^t.

Note that a solution to this `MCR` problem can be easily applied both to the computation of transit and transfer paths. In transit problems, the manipulator is not holding an object and needs to move between two configurations in $\mathcal{Q}_{free}$. The experimental evaluation of this work includes both transit and transfer challenges.

3.4 Manipulation Primitives

As mentioned before the low-level motion primitives that will be used in this work are `TRANSIT` and `TRANSFER`. However, most of the methods require from the manipulator to return to its initial configuration after a successful transfer. For convenience this work will use the manipulation primitive **pick and place path**

Definition 11 (pick and place path)**.** $\pi \in \Pi : [0, 1] \rightarrow \mathbb{X}^v$ *is a sequence of a transit, transfer and again a transit path, where the transitions between the subpaths are legal. The following discussion will assume the manipulation primitive*

$$\texttt{PICK\_AND\_PLACE}(o, q, p_0, p_1, c) \rightarrow \mathbb{X}^v$$

which computes a collision-free path for the manipulator to pick an object o from the

position p_0, transfer it to the position p_1 and finally, return the arm back to its initial state q, while respect the constraints c.

Given that the manipulator is moving among movable objects that can be considered constraints, a collision-free path might not exist. However, a valid path with constraints might be feasible. For that reason most of the algorithm will use `MCR` paths in order to find a solution, i.e. the manipulation primitive **minimum constraint removal pick and place path** will be used when the algorithm is able to deal with constraints.

Definition 12 (minimum constraint removal pick and place path)**.** *The minimum constraint removal pick and place path*

$$\texttt{MCR\_PICK\_AND\_PLACE}(o, q, p_0, p_1, c) \rightarrow \mathbb{X}^v$$

will return a similar path to move the object o from position p_0 to p_1 while respects the constraints c. The difference between the two manipulation primitives is that the second one will return a path with the minimum number of constraints if a collision-free path does not exist.

Similarly, the motion primitives `TRANSIT` and `TRANSFER` will also use the `MCR` method, in order to compute a solution path. This way both `MCR_TRANSIT` and `MCR_TRANSFER` will return a path with the minimum number of objects that have to be relocated in order to get a feasible path.

Chapter 4

Solving Unlabeled Rearrangement Problems Using Pebble Graphs

The focus of this chapter is on the case when the target objects are geometrically similar and interchangeable. A key challenge in developing practical algorithms for such problems is the size of the search space. A complete method must operate in the Cartesian product of the configuration spaces of all the objects and the robot. This work deals primarily with these combinatorial and geometric aspects and proposes motion planning methods that return collision-free paths for manipulating rigid bodies.

The approach reduces the continuous, high-dimensional rearrangement problem into several, discrete rearrangement challenges on "manipulation pebble graphs" (`MPG`s). The inspiration comes from work in algorithmic theory on "pebble graphs" [6] and related recent contributions in multi-robot motion planning [104]. The transfer of the idea of pebble graphs to the problem of rearrangement is not trivial. The presence of a manipulator in rearrangement planning induces additional constraints relative to the previous work on multi-robot motion planning [104], which required the development of different solutions for the connection of `MPG`s. The current work builds on top of work in manipulation planning [103] so as to allow for the efficient use of "pebble graphs" in the context of a manipulator rearranging similar objects.

The nodes of an `MPG` correspond to potential stable poses for the objects. Edges indicate that there is a path for the manipulator to transfer an object between the poses, even if all other nodes are occupied. For most interesting queries, it is difficult to find a single pebble graph that contains both the start and the goal arrangement and solves the problem. It is helpful to generate multiple such graphs and identify transitions between them to find a solution. `MPG`s have the benefit that they can implicitly represent an entire

set of object arrangements over the set of stable poses that they correspond to. Only once a sequence of pebble graph transitions that solve the problem has been found, does the algorithm need to extract the sequence of object placements and the corresponding arm paths. The encapsulation of multiple object arrangements within an individual graph helps with the combinatorial aspects of rearrangement. Furthermore, the method can effectively utilize precomputation. For a known workspace and geometry for the objects, it is possible to precompute a manipulation graph and perform many expensive collision checking operations offline.

The approach can efficiently address many rearrangement challenges, albeit with certain concessions. For instance, it must be possible to retract the arm to a safe configuration from every stable grasped pose in a solution sequence. Furthermore, this chapter deals only with similar geometry, interchangeable objects. Nevertheless, given the motivating work in multi-robot motion planning [104], it is possible to extend the proposed framework to dissimilar objects.

4.1 Unlabeled Rearrangement Planning

A naive approach to solve unlabeled rearrangements would be to build a manipulation graph [1] in the entire $\mathbb{X}$. This is intractable, however, as the dimensionality increases with the number of objects. The idea here is to abstract out the motion of the manipulator and then reason directly about the movement of objects between different stable poses in $\mathcal{P}^s$. Reasoning about the movement of multiple objects can take place over discrete graphical representations so as to take advantage of linear-time path planning tools for rearranging unlabeled "pebbles" on a graph from an initial to a target arrangement [6].

A sampling approach can be used to define graphs where nodes correspond to stable poses in $\mathcal{P}^s$ and edges correspond to collision-free motions of the arm that transfer an object between stable poses. If such a graph is connected and contains all the poses from the initial and target arrangements, then a discrete solver can be used to define a solution in the continuous space as long as placing objects in different poses does not

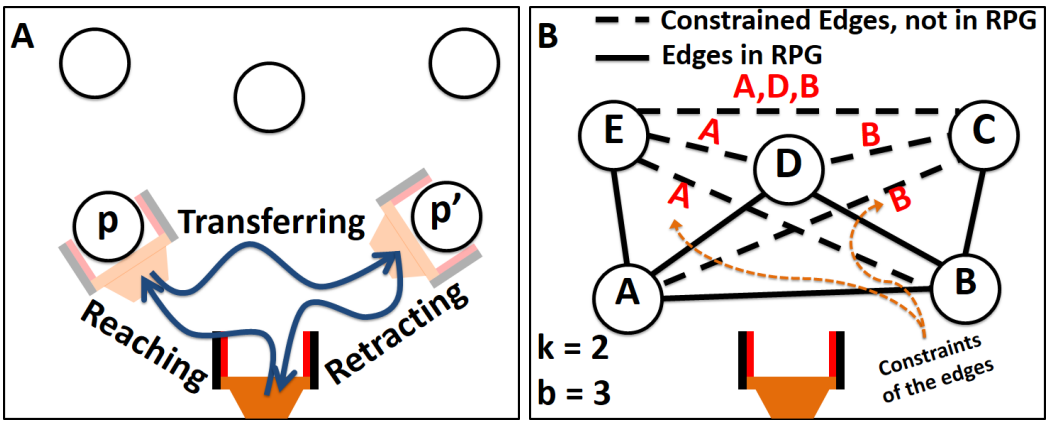

Figure 4.1: Each edge on an MPG is the combination of a reaching, transferring and retracting path. Edges can be constrained by other nodes of the MPG.

cause collisions [6].

It may be difficult or even impossible, however, to construct a single such graph that directly solves the problem. For example, the poses in the initial and target arrangements could be already overlapping, or it may not be possible to ensure connectivity with collision-free motions of the arm. Motivated by work in the multi-robot motion planning literature [104], the current work considers multiple such graphs, referred to as "manipulation pebble graphs" (MPGs), as shown in Fig.4.1B.

4.1.1 Manipulation Pebble Graphs

MPGs are built so that the objects are placed in collision-free, stable poses. They have at least k poses and b additional poses as nodes (called "blanks"). The extra nodes allow the rearrangement of k objects on the MPGs. The poses used in an MPG are defined as a pumped arrangement.

Definition 13 (Pumped Arrangement)**.** *A pumped arrangement $\alpha^{\mathcal{P}}$ is a set of $n = k+b$ poses where: (a) These poses are collision-free, stable poses (i.e., subset of $\mathcal{P}^s$); (b) No two objects will collide if placed on any two poses of the pumped arrangement.*

Definition 14 (Manipulation Pebble Graph (MPG))**.** *An MPG is a graph $\mathcal{G}_P(\alpha^{\mathcal{P}}, \mathcal{E}^P)$, where the set of nodes is a pumped arrangement $\alpha^{\mathcal{P}}$. The set of edges $\mathcal{E}^P$ corresponds to pairs $(p, p') \in \alpha^{\mathcal{P}}$, for which the manipulator can transfer objects between poses p, p'*

without collisions, given that potentially every other pose of $\alpha^{\mathcal{P}}$ is occupied.

Algorithm 2 describes the construction of an MPG. The algorithm receives as input a "safe" configuration of the manipulator $q_s^{\mathcal{R}}$, which is one that does not interfere with the objects placed on any stable poses, and typically is a retracted arm configuration. The algorithm also needs the size of the MPG, i.e., the parameters k and b. The algorithm will return an MPG and a set of additional constrained edges E_c together with the poses that block them. The constrained edges are edges that are not added in the MPG.

The algorithm starts by selecting a non-intersecting set of $k + b$ poses that create a pumped arrangement (Line 1). For each pair of poses p and p', a path has to be computed that allows the manipulator to move an object from p to p'. This path consists of three segments as shown in Fig. 4.1:

i) A path for the arm from its safe configuration $q_s^{\mathcal{R}}$ to a transition state $x_p^t \in \mathbb{X}^t$ for pose p.

ii) A path that transfers the object from $x_p^t \in \mathbb{X}^t$ to a state $x_{p'}^t \in \mathbb{X}^t$ for p'.

iii) A retraction path from $x_{p'}^t$ back to $q_s^{\mathcal{R}}$.

Algorithm 2: CREATE_MPG$(q_s^{\mathcal{R}}, k, b)$

```
1  V_P ← SAMPLE_VALID_PUMPED_ARRANGEMENT(k + b);
2  E_P ← ∅, E_c ← ∅;
3  for p, p' ∈ V_P and p ≠ p' do
4      π ← MCR_PICK_AND_PLACE(o, q_s^R, p, p', V_P);
5      if is_valid(π) then
6          E_P ← E_P ∪ ((p, p'), π);
7      else
8          π_B ← COMPUTE_CONSTRAINTS(π);
9          E_c ← E_c ∪ ((p, p'), π, π_B);
10 return {G_P(V_P, E_P), E_c}
```

For each edge, the algorithm aims to compute a "minimum constraint removal path" given that objects may be placed in all poses but p'. The "minimum constraint removal path" is computed heuristically by searching for a path that is within a bound of the shortest path [67] ignoring the other poses and which minimizes conflicts with the

other poses. If the computed path is collision-free, then an edge (p,p') is added to the MPG (Line 6), which stores the corresponding path. In case the "minimum constraint removal path" π collides with objects on other poses of the MPG, the edge (p,p') is not added. The algorithm identifies these potential collisions and stores them in a set of constraints $\pi_{\mathcal{B}}$ (Line 8). Such edges with constraints are stored in the data structure E_c (Line 9). The algorithm returns the MPG $\mathcal{G}_P(V_P, E_P)$ and the set E_c of constrained edges. The use of E_c is described later on.

Signatures in a Pebble Graph

Objects within each connected component of an MPG can be safely rearranged, since MPG's edges correspond to valid motions of the manipulator regardless of object placement on the graph's node. A discrete solver [6] can be used to achieve all feasible arrangements, given an MPG's connectivity. There is no need to explicitly store all the possible arrangements over an MPG, since the "signature" of an arrangement α is sufficient to describe all reachable rearrangements.

Definition 15 (Signature)**.** *A signature, $\sigma_{\mathcal{G}_P}(\alpha)$ is the number of objects contained in each connected component of* MPG $\mathcal{G}_P$ *according to an arrangement α.*

4.1.2 Building a Super-Graph

It is also possible to move objects between poses belonging to different MPGs if the MPGs share at least k poses that can be simultaneously occupied by objects, given the corresponding signatures.

These observations give rise to a super-graph structure, where each super-node corresponds to an MPG and a signature. Super-edges correspond to transitions between such super-nodes. The initial and target arrangements define two such super-nodes. Then the approach generates and connects super-nodes (i.e., MPGs and signatures) until the initial and target arrangement are connected on the super-graph. At that point, the rearrangement problem is solved and the necessary motions of the manipulator can be extracted along the path connecting the initial and target nodes on the super-graph.

Constructing Super-Nodes

All arrangements in an MPG with the same signature are reachable from each other, i.e., if $\sigma_{\mathcal{G}_P}(\alpha) = \sigma_{\mathcal{G}_P}(\alpha')$, then there is some sequence of transitions and a corresponding path for the manipulator, π, which brings α to α', and vice versa.

Definition 16 (super-node)**.** *A super-node is an* MPG *and a signature σ shared by a set of reachable arrangements of objects on the* MPG*.*

Definition 17 (Sibling nodes)**.** *Sibling nodes correspond to super-nodes with the same* MPG *but different signature.*

Algorithm 3: CREATE_SUPERNODES$(\mathcal{H}(V_H, E_H), q_s^{\mathcal{R}}, k, b)$

```
1 {G_P, E_c} ← CREATE_MPG(q_s^R, k, b);
2 Σ ← GENERATE_SIGNATURES(G_P);
3 V_n ← ∅;
4 for σ ∈ Σ do
5 |   v_H ← (G_P, σ), V_n ← V_n ∪ v_H;
6 |   V_H ← V_H ∪ v_H;
7 |   CONNECT_NODE(H, v_H);
8 CONNECT_SIBLINGS(H, V_n, E_c);
```

Algorithm 3 takes as an argument the existing super-graph $\mathcal{H}(V_H, E_H)$ and adds new super-nodes and edges. It needs to be aware of the safe configuration $q_s^{\mathcal{R}}$, and the size of the MPGs it will construct: $n = k + b$. When this function completes, the new "sibling" nodes will be in the super-graph and appropriately connected with edges.

The algorithm begins by constructing a random, valid MPG (Line 1). Then, it computes all possible signatures that the generated MPG can attain (Line 2), i.e., for G_{P2} in Fig.4.2, the possible signatures are {3,0},{2,1},{1,2}. A new super-node v_H is created using the created MPG and one of the signatures (Line 5). When v_H is added in the super-graph $\mathcal{H}$, the functions CONNECT_NODE and CONNECT_SIBLINGS try to connect v_H with other super-nodes (Lines 8, 9).

Connecting Super-Nodes

The super-nodes in $\mathcal{H}$ have two different ways to connect to each other.

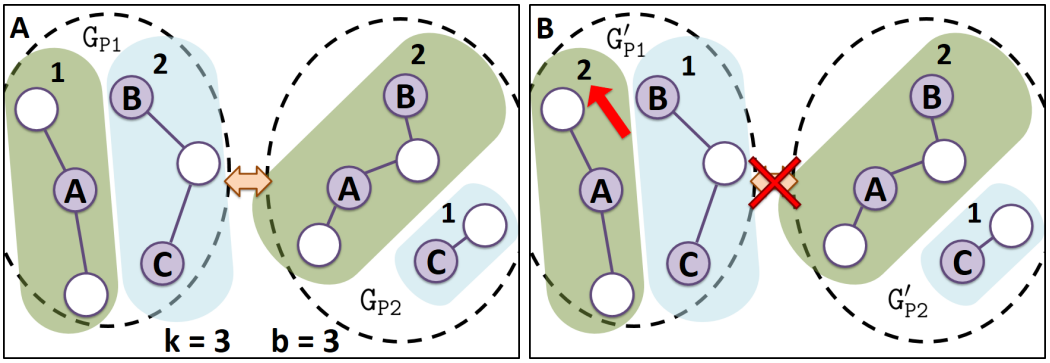

Figure 4.2: CONNECT_NODE: both MPGs share the shaded positions (A,B,C). (left) The nodes are connected since both signatures allow the three common positions to contain an object. (right) The nodes cannot be connected, since on $\mathcal{G}'_{P1}$ it is not possible to place objects both on B and C.

(A) CONNECT_NODE connects super-nodes created by a new MPG with super-nodes created using previous, different MPGs. The function first identifies whether super-node v_H shares at least k common poses with an existing super-node. If true, the method checks whether both super-nodes can achieve placing k objects on the k common poses given the super-nodes' signatures (Fig. 4.2). If they both can, an edge is added between the two super-nodes, which represents a context switch between two arrangement sub-problems. This switch is feasible because only the b unoccupied poses of the first MPG will be replaced with the b unoccupied poses of the other MPG, while the k objects will have to stay in their current positions.

Algorithm 4: CONNECT_NODE$(\mathcal{H}(V_H, E_H), v_H)$

```
1 for v'_H ∈ V_H do
2     p_c ← COMMON_POSITIONS(v_H, v'_H);
3     if |p_c| ≥ k then
4         if CAN_ASSIGN(poses, v_H, v'_H) then
5             E_H ← E_H ∪ (v_H, v'_H);
```

(B) CONNECT_SIBLINGS connects "sibling" super-nodes by using the constraint edges E_c to switch between different signatures on the same MPG. The algorithm connects super-nodes only if there is a motion that allows transition between signatures of the "siblings" (Fig. 4.3). The arm is able to follow a constrained edge in E_c if the constraining poses can be emptied given the super-node's signature. For all constrained edges,

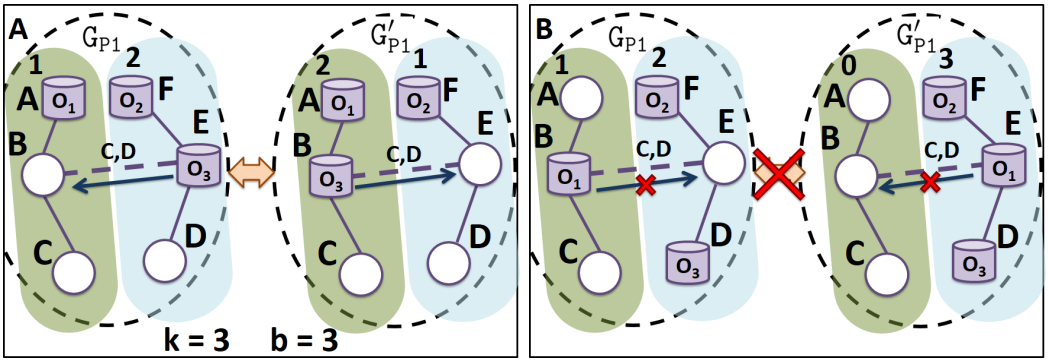

Figure 4.3: `CONNECT_SIBLINGS`: Nodes with same `MPG` but different signatures. The constrained (dashed) edge is not in the `MPG`. (left) An edge is added since an object can be moved along the edge given constraint (C,D). (right) The edge is not added since pose D cannot be emptied.

the algorithm finds the connected components of the `MPG` that this edge is connecting. If the connected components are different and the edge is feasible given the signature, then this edge can be used for potential connections between two "siblings". An edge is feasible if the poses that constrain the edge and the target pose p' of the constrained edge can be emptied given the signature of v_H. Furthermore, it must be possible to bring an object to the source pose p of the edge. Moving along such edges results in a change of signature relative to that of v_H and the new signature σ_n can be computed. Then an edge is created that connects the two "siblings" in the super-graph.

Algorithm 5: `CONNECT_SIBLINGS`$(\mathcal{H}(V_H, E_H), V_n, E_c)$

```
1 for v_H ∈ V_n do
2     for e ∈ E_c do
3         {From_cc, To_cc} ← FIND_COMPONENTS(v_H, e);
4         if From_cc ≠ To_cc and EDGE_IS_FEASIBLE(v_H, e) then
5             σ_n ← COMPUTE_NEW_SIGNATURE(v_H, e);
6             v'_H ← FIND_SIBLING(v_H, σ_n);
7             if v'_H exists then
8                 E_H ← E_H ∪ (v_H, v'_H);
```

Building the Graph Using Incremental Approach

Generating new super-nodes can take place in a fashion similar to a bidirectional sampling-based tree planner. The proposed approach first generates two super-nodes of

size k for the initial and target arrangements. The b "blank" poses are unnecessary for these nodes. Then, k poses are selected from an already existing super-node to be used as poses for the new `MPG`. These poses can be occupied by objects given the signature of the selected super-node. An additional k poses are sampled for the new `MPG`. Alg.3 will generate all the "siblings" super-node for the new `MPG` that will be added in the super-graph and attempt connections with existing super-nodes. The process repeats until the initial and target super-nodes are in the same connected component.

Answering Queries on the Super-Graph

For each super-node, there exists a way to move between all possible arrangements on the corresponding `MPG` that have the same signature. The actual paths that accomplish these rearrangements are not explicitly stored. Only during the query phase, when a graph search returns a solution sequence of super-nodes that connects the start and target arrangements, the algorithm computes solutions to the pebble motion problem within each `MPG`.

The super-graph path is transformed into a manipulation path by solving discrete graph problems [6] on individual `MPG`s given start and final arrangements on them according to information stored on the hyper-edges. Since the `MPG` contains edges which have safe motions for the manipulator, any arrangement produced by the approach will be feasible for the robotic arm if it follows the sequence of actions that were used to validate the edge. The start arrangement on the `MPG` is simply the latest arrangement the objects have reached in the previous `MPG`. The end arrangement is stored on the edges of the super-graph. For connections between nodes generated from the same `MPG`, the edges involve a motion and have the constraints associated with this motion encoded. The end configuration is such that those constraints are satisfied.

4.1.3 Smoothing Solution

Figure 4.4 shows examples of smoothing the solution path. The colored disks represent occupied poses. The uncolored discs represent the safe configuration $q_s^{\mathcal{R}}$. The edges connecting the nodes represent the trajectory taken by the manipulator to move the

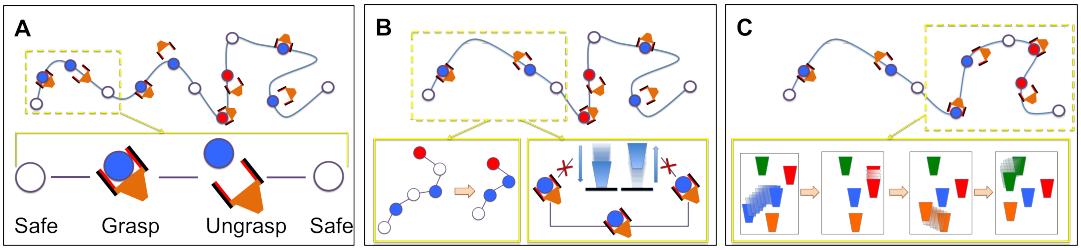

Figure 4.4: A: A path decomposed into reaching, transferring and retracting motions. B: If the arm grasps the same object twice, then the intermediate stable state is removed. C: Redundant movements of objects can be removed.

objects. A trajectory can be separated into distinct sequences of transit to grasp from $q_s^{\mathcal{R}}$, transfer an object between poses (p,p'), and retract to $q_s^{\mathcal{R}}$, as shown in Figure 4.4A.

Phase 1 - Consecutive, Identical Grasps: The smoothing process first looks for consecutive grasps of the same object in a solution sequence. If such pairs of grasps exist, the intermediate stable pose is removed, along with any redundant states, as shown in Figure 4.4B. This has the effect of removing motions, such as raising and lowering the object, which are unnecessary when consecutively grasping the same object.

Phase 2 - Standard Trajectory Smoothing: Each reaching, transferring and retracting sequence is smoothed by checking for shortcuts between pairs of states. If these shortcuts exist, they replace their corresponding intermediate trajectory.

Phase 3 - Maximizing Consecutive Grasps: The solution returned by the algorithm is conservative. This can potentially lead to redundant movements of objects. The trajectory that moves an object o_x from position p^i to p^{i+1} can be represented as $p_x^i \rightarrow p_x^{i+1}$. The trajectory sequence:

$$\{p_a^i \rightarrow p_a^{i+1},\ \ p_b^j \rightarrow p_b^{j+1},\ \ p_a^{i+1} \rightarrow p_a^{i+2}\},$$

contains an intermediate placement of o_a at pose p_a^{i+1}, before eventually moving it to p_a^{i+1} (Figure 4.4C). If, i) o_a can be moved from pose p_a^i to pose p_a^{i+2} with a collision free path, while o_b is at pose j and ii) the trajectory $p_b^j \rightarrow p_b^{j+1}$ is collision free given that o_a is at pose p_a^{i+2}, then the original trajectory can be replaced with $\{p_a^i \rightarrow p_a^{i+2},\ \ p_b^j \rightarrow p_b^{j+1}\}$.

These phases can be repeated, starting from phase 1, to continually improve the path quality. Once the smoothing phases are complete, phase 2 can be applied over the entire trajectory, resulting in the final smoothed solution. In practice, a single

application of each phase and a final application of phase 2 are sufficient.

4.2 Evaluation

The proposed algorithm was evaluated in a simulation environment to determine its scalability and showcase the class of difficult non-monotone problems it can solve. These are challenges which require an object to be grasped multiple times in order to be solved. Such problems are challenging for state-of-the-art approaches. A 7-DOF arm of a Baxter robot is used in the simulation. The solution paths were executed on a Baxter robot in open loop trials. The identical objects being rearranged are cylinders with height $14cm$ and radius $4cm$. A brute-force approach has not been evaluated in comparison since a search on the combined configuration space of the manipulator and the objects quickly becomes intractable.

Different types of rearrangement problems are considered. These problems include a) environments with random initial configurations to a final grid rearrangement of the objects in a shelf, and b) two non-monotone problems with $k = 3, 4$ (Figure 4.5). The execution time and the length of the solution trajectory have been measured. These metrics are reported for different values of b, i.e., the number of "blank" poses in the `MPGs`. An informed pre-computed roadmap $\mathcal{G}_{\mathcal{R}}$, as described in section 7.1, has been used in the implementation. A trial is deemed a failure if the method cannot find a solution within the time limit of 600 seconds. The simulations were performed on a machine running on an Intel(R) Xeon(R) CPU E5-4650 0 @ 2.70GHz.

4.2.1 Randomized Grid Evaluation

For the first set of experiments, objects are placed randomly on a shelf-like environment, reachable by the manipulator. The mutually exclusive initial pose of the objects is selected as a random set of non-intersecting poses for every trial. The dimensions of the shelf prevent the grasping of the objects from the top. This causes objects to be occluded by other objects in front of them. These properties of the environment make the solution to a rearrangement challenge non-trivial. For a given number of

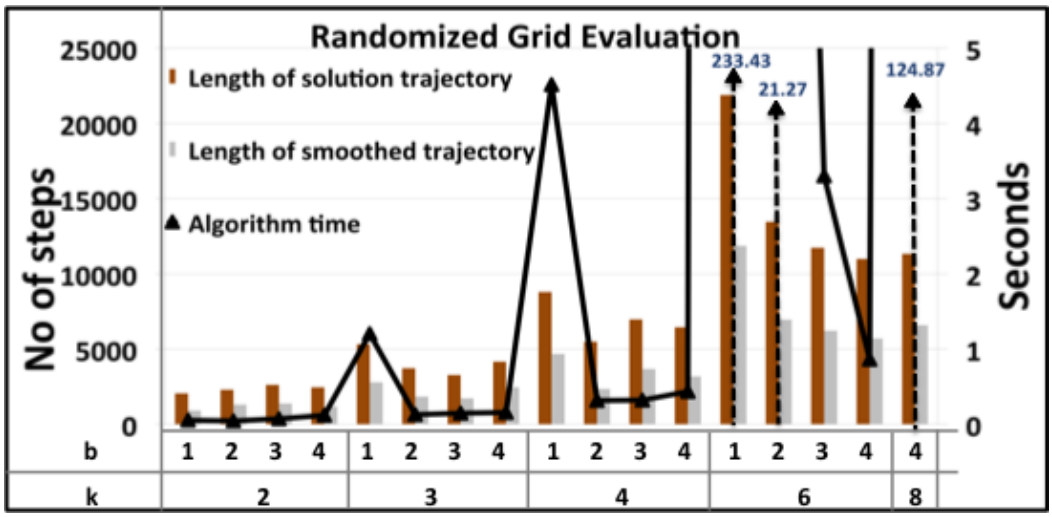

Figure 4.5: Average length of the trajectory before and after smoothing (bars) and average solution time (line) for the randomized grid for different values of k and b.

objects, the goal arrangement is a uniform grid formation on the shelf and the same goal arrangement is used for a value of k. For every pair of k and b, 10 trials are executed. Experiments for $2, 3, 4, 6$, or 8 objects were performed. For a specific number of objects, k, values of b, which are tested, are $1, 2, 3$, and 4. The manipulation roadmap used for these experiments employed 20 different poses and computed 517 vertices and 6032 edges.

Execution time: Fig. 4.5 shows that the algorithm achieves a high success ratio in the randomized grid setup for $k = 2, 3, 4, 6$. Trials with 8 objects could not finish within the stipulated limit of $600s$. $k = 8$ causes an increase in the problem complexity. The case with 8 objects is constrained by the size of the shelf in terms of the availability of free poses on the shelf and free volume required to move. This affects the motion planning complexity for pose connections. The relatively sparse roadmap used in the experiment helps the running time of the individual motion planning problems, but in a constrained space, the solution to the rearrangement problem is not always achieved within the time limit.

The value of b seems to have a significant role in the execution time for the same k, as illustrated in Fig.4.5. $b = 1$ seems to deteriorate the algorithm's running time for higher values of k. This is a result of a very slow rate of exploration of the poses

available on the shelf for rearrangement, due to only one empty pose available in the MPG. However, the connectivity in an MPG is maximized. High values of b introduce a high combinatorial component of k objects in $k+b$ poses in every MPG and the connectivity of the MPG decreases. For every value of k, the value of b with the best performance in terms of execution time indicates this trade-off.

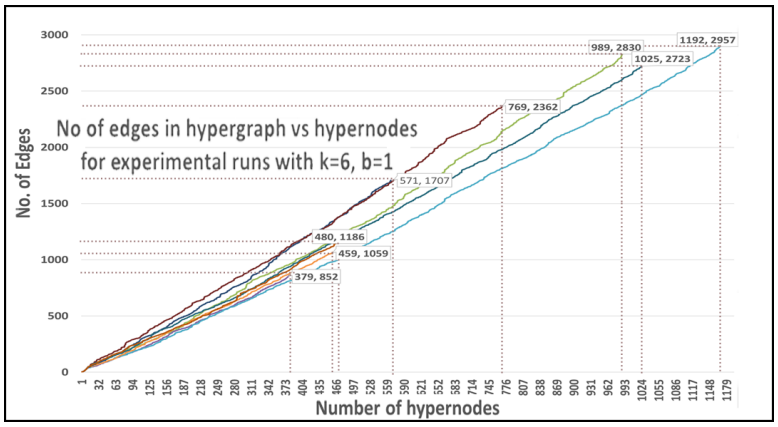

Figure 4.6: The number of edges in the super-graph over the number of super-nodes for $k = 6$ and $b = 1$.

Connectivity: The key feature of the algorithm is the way it constructs the super-graph from the super-nodes. The connectivity of this super-graph is crucial in finding the solution to the rearrangement problem. Fig.4.6 shows the rate at which new edges are created, with the introduction of every new super-node, for the different trials for the randomized grid experiment with $k = 6$,$b = 1$. The random trials corresponding to the best values of b for every $k = 2, 3, 4, 6$ from Fig.4.5 are analyzed for the number of super-nodes that were required for finding the solution and the time taken, Fig.4.7. Complex solutions are efficiently discovered with short execution times. For $k = 6$,$b = 4$, the solution grasps every object 2.67 on average times over all the trials.

Solution quality: The quality of the solution in the randomized grid experiment, is shown in Fig. 4.5. The value of $b = 1$ introduces a low exploration rate, which increases the length of the solution. The conservative solution trajectories are shortened by an average of 48%.

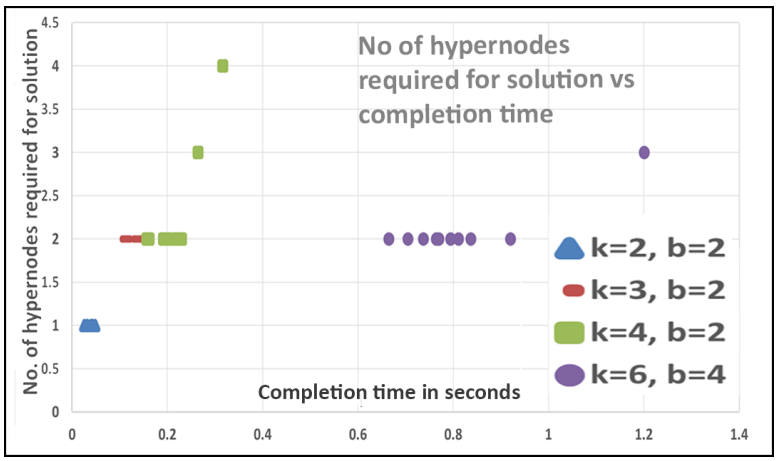

Figure 4.7: The amount of time taken and size of the super-graph required to achieve the solution, for different runs of the randomized grid problem and different values of k and b .

4.2.2 Non-monotone Benchmark Evaluation

The second set of experiments consists of two non-monotone benchmark problems. For the first problem, 4 objects are placed on a platform which resembles the shelf with the sides removed, figure 4.8. The goal arrangement requires the two front objects to be moved at least twice. The trials are performed for different values of $b = 1, 2, 3, 4$. The same roadmap as before is used.

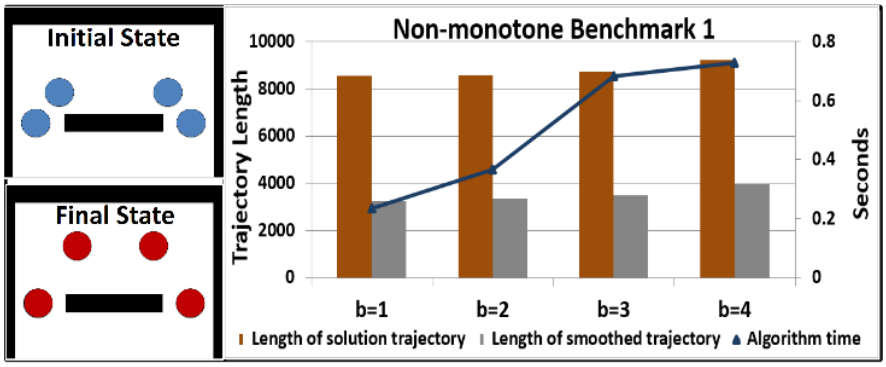

Figure 4.8: The figure on the left-top shows the initial arrangement of the non-monotone benchmark with 4 objects. The left bottom image indicates the intended goal positions. The problem involves 4 objects on a platform which resembles the shelf with the sides removed. The wall in the center at the front makes both the initial and goal positions of two of the objects inaccessible. The goal positions require the two objects in the front to necessarily be moved at least twice. The figure on the right shows the performance of the algorithm for the benchmark. Average length of the trajectory before and after smoothing (bars) and average solution time (line).

For the second benchmark, 3 objects are placed on a shelf with two static obstacles that form a narrow cavity between them (Fig.4.9(top left)). Because of the smaller k only values of $b = 1, 2, 3$ are tested. The second benchmark is solved on a roadmap with 684 vertices, 9628 edges and 30 poses. The bigger roadmap, used in this benchmark, is necessary to deal with the constraining nature of the problem. Increasing the size of the roadmap and the number of poses greatly helps in making more constrained problems solvable. The results of this benchmark are depicted in figure 4.9.

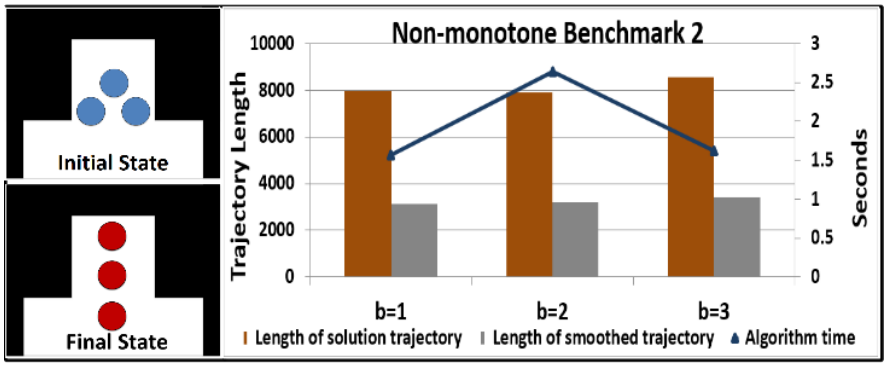

Figure 4.9: Average length of the trajectory before and after smoothing (bars) and average solution time (line) for the second non-monotone benchmarks. In this problem the manipulator has to change the arrangement of the objects from a triangle to a line arrangement.

4.3 Discussion

The method proposed in this chapter solves rearrangement problems for similar objects, using a high DOF robot arm, including non-monotone instances that are hard to address with existing methods [110]. Precomputation using a manipulation graph can be appropriately utilized to achieve fast solution times.

The current algorithm does not address objects with different labels or geometries. The motivating work on pebble graphs [104], however, provides a framework to extend the current approach to the case of different objects. Furthermore, the current implementation takes advantage of grasp symmetries about the Z axis for cylindrical objects placed upright. Dealing with general grasps and general resting poses for the objects needs further investigation. There are also interesting variations that can be explored

in relation to different ways for connecting super-nodes that may not require the two nodes to share k poses.

The paths computed in the simulation have been tested in open-loop (for known initial and target object arrangement) on a real Baxter system arranging cylindrical objects in a shelf. Long paths that involve multiple grasps were unsuccessful as the robot's path deviated due to errors from the computed one. Future efforts will focus on the computation of robust rearrangement trajectories and their robust execution given appropriate sensing input [82].

Chapter 5

A Probabilistically Complete Hierarchical Framework for General Object Rearrangement

The method presented in chapter 4 can solve both monotone and non-monotone problems under the assumption that all objects are unlabeled. This means that they are interchangeable and can occupy any pose in their final arrangement. While unlabeled problems occur frequently in applications, it is similarly important to be able to solve cases where the objects are labeled and could be dissimilar, i.e., each object has to acquire a unique final pose in the target, final arrangement.

This chapter presents a framework for solving hard instances of general object rearrangement. It initially revisits an existing backtracking search approach (`mRS`) for manipulation among movable obstacles [110]. This method can be used to solve monotone labeled challenges. Then, this chapter presents the extension of the backtracking search approach [110] to a similar backtracking-search strategy that addresses many labeled non-monotone challenges (`plRS`) [68]. In order to solve the non-monotone cases, the method is using a solution to the minimum constraint removal path planning problem (`MCR`) [47, 70]. Solutions to `MCR` are used to compute the intermediate placement of objects in the context of non-monotone solutions. An `MCR` path minimizes the number of constraints that need to be removed from the path of an object. Every time the method considers an object with a blocked path to its goal, it tries to clear the path by finding appropriate intermediate poses for blocking objects.

The objective of this chapter is to achieve probabilistic completeness guarantees by employing either the monotone (`mRS`) or of the new non-monotone method (`plRS`) as a local connection primitive in the context a higher-level task planner. The higher-level planner searches the space of object placements. This integration demonstrates the

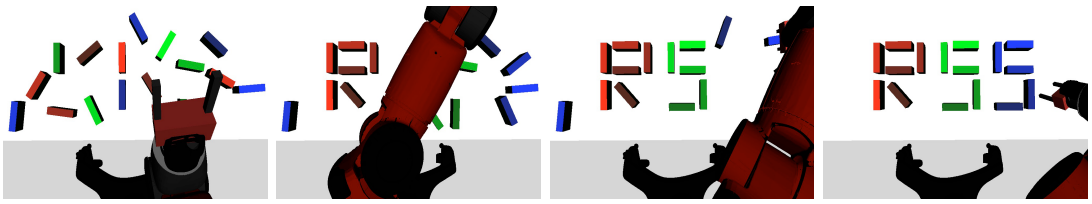

Figure 5.1: An example of a challenge considered in the accompanying simulated experiments: 16 objects are manipulated by a Baxter arm from an initial arrangement to a final one, where the letters `RSS` are spelled.

benefit of using more powerful connection primitives in the context of task planning for object rearrangement than an individual pick-and-place.

Simulated experiments using a model of a Baxter arm evaluate the methods in a variety of non-monotone, labeled rearrangement problems, including setups in restricted, tight 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. The methods are evaluated in terms of success ratio, running time, scalability and path quality.

5.1 Rearrangement Primitives

This section presents two rearrangement primitives that under certain conditions can provide a solution for connecting two object arrangements. The advantage over an individual pick and placeof using these primitives as local planners in a higher-level task planner is that they can connect arrangements that have more than one object in a different pose. Pick and place can only connect arrangements that differ only by a single object pose [40, 106]. The non-monotone extension is more powerful but more computationally expensive than the monotone primitive, which is experimentally shown to be advantageous.

5.1.1 A Primitive for Monotone Rearrangements

This section focuses on a previously proposed approach for manipulating objects [110], which deals with monotone problem instances.

For such problems, the algorithm [110] performs a backtracking search in the space of possible orders of transferring objects directly to their final poses. Fig. 5.2 (right) shows possible orders considered given object B is moved first for the scene of Fig. 5.2(left). One of the branches in the search tree results in a solution. In the context of the algorithm for manipulation purposes, between each pair of transfer paths, there is a transit path that allows the arm to switch between objects. There is also an initial and final transit path that allows the manipulator to reach the first object and retract after completing the transfer of the last object.

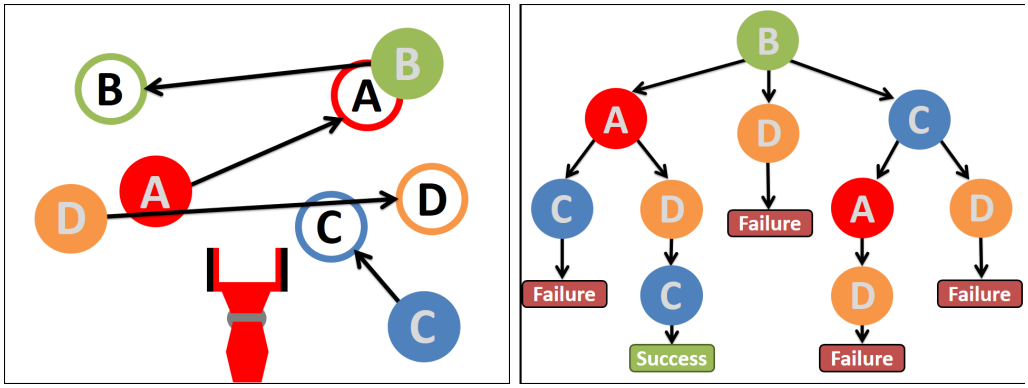

Figure 5.2: (left) An example of two arrangements for four objects (initial: light colored, final: darker colored) and a possible order that allows their monotone rearrangement for linear paths. (right) Corresponding backtracking search.

The "monotone Rearrangement Search" algorithm (`mRS`) is described in Alg. 6 and is an adaptation of the existing manipulation approach for clearing a path to an unreachable object [110]. The original approach was searching backward 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:

- the workspace $\mathcal{W}$ and the robot $\mathcal{R}$,
- the set of objects $\mathcal{O}_R$ ("remaining objects") not yet moved to their final pose along the current branch of the search tree,

- an object o to be transferred,
- the manipulator's last configuration q,
- the current and final arrangements: α_C and α_F.

Algorithm 6: $\texttt{mRS}(\mathcal{W}, \mathcal{R}, \mathcal{O}_R, o, q, \alpha_C, \alpha_F)$

```
1  π_N ← TRANSIT(q, q(α_C[o], e), α_C);
2  π_M ← TRANSFER(o, α_C[o], α_F[o], e, α_C);
3  if (π_U ← {π_N | π_M}) is collision free then
4      α_C[o] ← α_F[o];
5      if O_R == ∅ then
6          return π_U;
7      for each o_r ∈ O_R do
8          π ← mRS(W, R, O_R ∖ o_r, o_r, q(α_F[o], e), α_C, α_F);
9          if π ≠ ∅ then return {π_U | π};
10 return ∅;
```

Given a start configuration q for the arm and the first object o to be transferred, the algorithm first aims to find a transit path $\pi_{\mathcal{N}}$ to grasp the object at its current (initial) location $\alpha_C[o]$ (line 1 - path $\pi_{\mathcal{N}}$) and then transfer it to $\alpha_F[o]$ (line 2 - path $\pi_{\mathcal{M}}$). These paths must be collision-free (line 3), given the poses of the other objects in the current arrangement α_C. If such paths $\pi_{\mathcal{N}}$ and $\pi_{\mathcal{M}}$ are found for object o, then the object is moved (line 4). If all of the objects have been moved to the pose according to the final arrangement α_F (line 5), then the method terminates as a solution path $\pi_{\mathcal{U}}$ has been found, which concatenates $\pi_{\mathcal{N}}$ and $\pi_{\mathcal{M}}$ (line 6). If there are objects that still have not reached their final pose, the method is called recursively for each one of them as the first to be transferred next (lines 7-8). If any of these calls is successful, a solution has been found and the path $\pi_{\mathcal{U}}$ for object o can be appended to the existing solution path (line 9). If no paths were found for object o, or if all the recursive calls to move the remaining objects fail, then the process returns failure (line 10). This means that the current branch of the backtracking search failed to give a solution. The method needs to be initially called for all possible objects, i.e., call $\forall\ o : \texttt{mRS}(o, q_I, \mathcal{O} \setminus o, \alpha_I, \alpha_F)$.

The overall solution path returned by `mRS` can be decomposed into `TRANSIT` and

TRANSFER sub-paths. Given this decomposition, it is straightforward to argue the algorithm's (probabilistic) completeness for monotone rearrangement problems.

Lemma 1. *Assuming completeness for the* TRANSIT *and* TRANSFER *primitives, the* mRS *algorithm is complete for monotone challenges.*

Proof. For a monotone problem, there is a solution path that contains a single transfer for each object where it is moved from its initial to its final pose. Between these transfer paths, the robot executed transit paths. This means that the path can be decomposed into a sequence of TRANSFER and TRANSIT sub-paths for each object. The mRS method will exhaustively consider all possible orders for transferring the objects to their final arrangement. Given the assumption that the TRANSIT and TRANSFER sub-paths are complete, then the method is also complete. □

The above lemma can also be extended to the case of probabilistic completeness, given a small variation to the algorithmic process. For instance, the TRANSIT and TRANSFER paths can be computed given access to an underlying probabilistic roadmap for the robotic arm. Then a higher-level loop can call the mRS algorithm and when it fails it can increase the size of the corresponding roadmap through a sampling process and then call mRS again until a solution is found.

Corollary 1. *Assuming that the* TRANSIT *and* TRANSFER *primitives are computed with the aid of a probabilistically complete sampling-based roadmap planner for the robotic arm (e.g.,* PRM*), then consecutive calls to the* mRS *algorithm and extending the sampling-based roadmap corresponds to a probabilistically complete solution for monotone challenges.*

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 a factorial number of times, i.e., the number of all possible permutations of the movable objects in the scene. While this is a bad asymptotic performance - the problem is hard after all - in practice, the backtracking search can return early, i.e., as soon as a solution has been

found. Even if it fails, the number of calls is quite smaller than the worst case, as most branches will fail early.

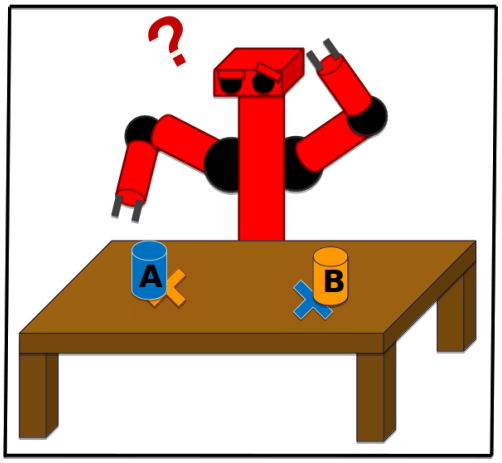

Figure 5.3: An easy table top example where the monotone algorithm will fail.

This method already solves tabletop tasks 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 (Fig.5.3).

Moreover, for objects placed in a tight environment, such as shelves, the arm's maneuverability is limited and overhand grasps are frequently impossible. In these setups, non-monotone challenges arise easily. Consider Fig. 5.4 (left), referred here as "simplified Towers of Hanoi", where B and C 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 method.

5.1.2 Extension to Non-Monotone Challenges using Minimum Constraint Removal Paths

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

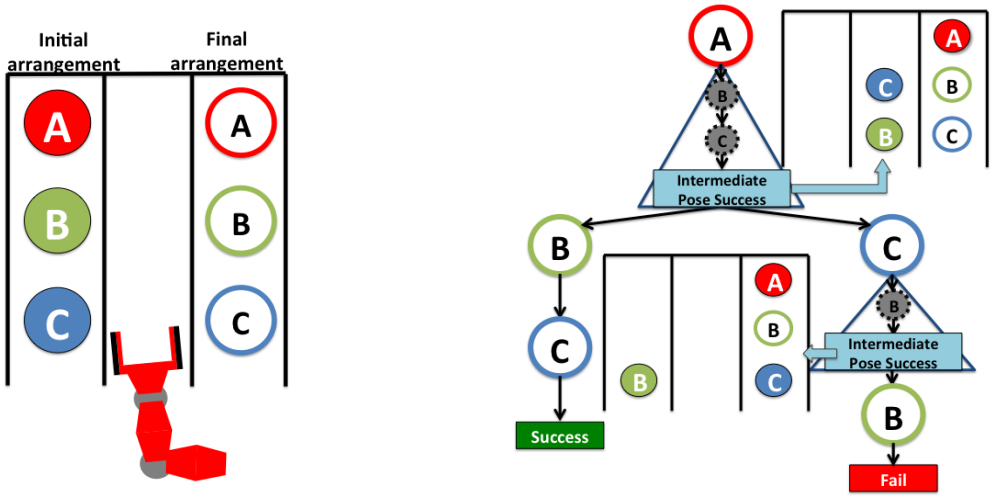

Figure 5.4: (left) A non-monotone challenge: 3 objects from the left shelf must be transferred to the right shelf in the same order. (right) The corresponding search tree starting with object A for `plRS`. Every time an object cannot be moved to its final pose, the remaining objects need to clear its path.

the blocking objects O_b so as evacuate the path of the blocked object. By clearing the path of the blocked object so that it reaches its final pose, there is going to be progress towards the solution of the problem. If one by one the objects move to their final pose without disturbing the objects that are already in their final pose, then the problem will be solved [85]. 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. 5.4 and the case object A is at the top of the search tree. The path for A's to reach its final pose is obstructed by B and C. Then, a subproblem arises (the top triangle in Fig. 5.4 (right)): *"Move B and C to intermediate poses that allow the transfer of object A to its final pose"*. In this example, the problem can be resolved 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 arrangement at the top right corner of Fig. 5.4 (right). On the other hand, if C is selected to evacuate first, it can move to the front/open part

of the second bin. Nevertheless, A is still blocked by B, which must also be moved to an intermediate pose. Given the same reasoning, B should select the same front/open part of the second bin, where C has been placed. As a result, C will be blocking B. Since C is not at its final pose, it can be relocated to a new intermediate pose, such as further up the second bin. This will open the spot for B and will result in the same arrangement as before, i.e., that shown in Fig. 5.4 top right corner.

Once object A is moved to its final pose, 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 resulting problem is monotone and this branch leads to success based on the standard backtracking search approach. If C moves first, then its path is blocked by B and another subproblem is defined (the lower right triangle in Fig. 5.4 (right)): *"Move B to an intermediate position so that C can then move to its final pose".* Once B moves back to the first bin so as to clear C's path, 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.

Given that objects already on their final pose, $\mathcal{O}_P$, should not move again, the proposed approach considers these objects as obstacles. All the objects $\mathcal{O}_R$ that are not on their final pose, i.e., $\alpha_C[\mathcal{O}_R] \neq \alpha_F[\mathcal{O}_R]$ and are blocking the path π of the object o_i to its final pose $\alpha_F[o_i]$ are viewed as potentially avoidable constraints, which need to be minimized. For this reason the idea of a minimum constraint removal path [47, 46, 70] will be used in order to minimize the number of objects that overlap with the path that the manipulator will follow to transfer the target object to its final pose. Figure 5.5 provides a comparison between the shortest path ignoring the other movable objects and the minimum constraint removal path. It shows that the minimum constraint path (`MCR`) will collide only with two objects, which means that a rearrangement algorithm will have to detect intermediate poses only for these two blocking objects. On the other hand, the shortest path will go through almost all of the objects. As a result, if the shortest path is considered, then an intermediate pose for all the objects in the scene has to be detected in order to move the target object.

All the transit and transfer paths that are computed for this algorithm will be

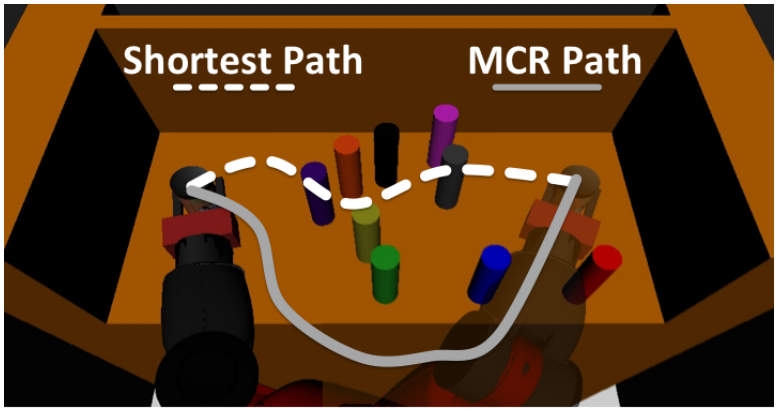

Figure 5.5: A minimum constraint removal challenge. (Dashed line) The shortest path that ignores obstacles can be computed quickly but will return many obstacles as constraints. (Solid line) It is more expensive to compute the path that returns the minimum number of obstacles as constraints.

`MCR` paths, i.e., `MCR_TRANSIT` and `MCR_TRANSFER`, where a fast approximation for the computation of `MCR` paths is used [70] and which is explained later on in this thesis. In this way, the algorithm aims to minimize the number of intermediate poses needed in order to evacuate potential object "constraints". The "piecewise linear non-monotone Rearrangement Search" algorithm (`plRS`) is given in Alg. 7 and directly extends its monotone counterpart.

The `plRS` method has two differences from `mRS`:

- The `MCR_TRANSIT` and `MCR_TRANSFER` paths for object o must be collision-free only with objects that have been already moved to their final pose, i.e., in the set $\mathcal{O} \setminus \mathcal{O}_R$. The remaining objects $\mathcal{O}_R$ constitute constraints in the context of minimum constraint removal paths [47].

- 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 $\mathcal{O}_R$ (lines 10-16).

The first part of the `plRS` algorithm is similar to the monotone counterpart mRS (lines 1-9). In the case the path $\pi_{\mathcal{U}}$ for the target object o is not collision-free (line 10), the algorithm considers a blocking object o_b along $\pi_{\mathcal{U}}$ (line 11). If o_b has not been moved to its final pose, i.e., it is in the list of "remaining objects" $\mathcal{O}_R$ (line 12), then a subroutine `CLEAR` is called to clear o's path from object o_b (line 13). The `CLEAR` function receives

Algorithm 7: `plRS`$(\mathcal{W}, \mathcal{R}, \mathcal{O}_R, o, q, \alpha_C, \alpha_F)$

```
1  π_N ← MCR_TRANSIT(q, q(α_C[o], e), α_C[O \ O_R], α_C[O_R]);
2  π_M ← MCR_TRANSFER(o, α_C[o], α_F[o], e, α_C[O \ O_R], α_C[O_R]);
3  if (π_U ← {π_N | π_M}) is collision free then
4      α_C[o] ← α_F[o];
5      if O_R == ∅ then
6          return π_U;
7      for each o_r ∈ O_R do
8          π ← plRS(W, R, O_R \ o_r, o_r, q(α_F[o], e), α_C, α_F);
9          if π ≠ ∅ then return {π_U | π};
10 else
11     o_b ← a blocking object along π_U;
12     if o_b ∈ O_R then
13         {p, e, α_C, π'} ← CLEAR(O_R \ o_b, o_b, q, α_C, π_U);
14         if π' ≠ ∅ then
15             π ← plRS(W, R, O_R, o, q(p, e), α_C, α_F);
16             if π ≠ ∅ return {π' | π};
17 return ∅;
```

the path $\pi_{\mathcal{U}}$ of object o as a constraint to be avoided. If the subproblem of `CLEAR` can be solved and o_b can evacuate o's path (line 14), then the `plRS` algorithm is called again for the 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. 8 provides the `CLEAR` function, which first finds an intermediate pose p for the calling object o and the corresponding path $\pi_{\mathcal{U}}$ to bring o to the intermediate pose p (line 1). The intermediate pose must:

- Be collision-free with the current arrangement $\alpha_C[\mathcal{O} \setminus \mathcal{O}_R]$ of objects that have reached their final pose,
- Not collide with the swept volume of the input constraint paths $\pi_{\mathcal{B}}$; these are the paths of objects that need to be cleared by object o.

Among all the possible poses that satisfy the above requirements, 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 $\mathcal{O}_R$. 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 among them. Section 7.3 will describe how these reachability computations can be accelerated using precomputation.

Algorithm 8: CLEAR$(\mathcal{O}_R, o, q, \alpha_C, \pi_\mathcal{B})$

```
1  {p, e, π_U} ← INTERMEDIATE_POSE(α_C[O \ O_R], π_B, q);
2  if π_U is collision free then
3      α_C[o] ← α_F[o];
4      return {p, e, α_C, π_U};
5  else
6      o_b ← a blocking object along π_U;
7      if o_b ∈ O_R then
8          {p', e', α_C, π'} ← CLEAR(O_R \ o_b, o_b, q, α_C, π_U ∪ π_B);
9          if π' ≠ ∅ then
10             {p, e, α_C, π} ← CLEAR(O_R, o, q(p', e'), α_C, π_B);
11             if π ≠ ∅ return {p, e, α_C, π'|π};
12 return ∅;
```

If a collision-free path $\pi_\mathcal{U}$ to an intermediate pose p is found (line 2), the object is moved there (line 3), and its pose p, the updated assignment α_C and the corresponding path $\pi_\mathcal{U}$ 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 the operation of the `plRS` algorithm itself (lines 5-11).

Below is a discussion of the `plRS` approach's properties. The first point to note is that `plRS` will at least solve all monotone problem instances.

Lemma 2. *Algorithm* `plRS` *will succeed for all problems that* `mRS` *succeeds, as long as both methods are called for all possible objects as the first object to be moved .*

Proof. When the permutation of objects that results in a solution for `mRS` is considered, the `MCR_TRANSIT` and `MCR_TRANSFER` subroutines of `plRS` will always return the same collision-free paths as the `TRANSIT` and `TRANSFER` subroutines used by `mRS`. Then, the first part of the `plRS` (lines 3-9) will result in the same solution as `mRS`. □

`plRS` may return a different, potentially lower quality solution than `mRS`, since `plRS` may try to resolve a branch with a non-monotone solution although other branches may

provide a monotone one.

The mRS approach from the previous section cannot solve tabletop challenges where there is an overlap between the initial and final poses (Fig. 5.3). In order to find a solution for this kind of problems, it is necessary to detect at least one intermediate pose for the involved objects. It is possible to argue that plRS can solve a large set of problems where objects are placed on a table-top and reachable with overhand grasps, i.e., in situations where when the arm grasps objects, there is no collision between the arm and the remaining objects.

Proposition 1. plRS *can solve all rearrangement challenges* $(\mathcal{W}, \mathcal{R}, \mathcal{O}_R, o, q, \alpha_C, \alpha_F)$ *where all* k *objects are accessible with overhand grasps under the following sparsity requirement for the setup: There are at least* $k-1$ *discrete intermediate poses, which allow each of any* $k-1$ *objects to be placed in the corresponding intermediate poses so that no collisions arise among the objects in the intermediate poses or with objects in the initial and final poses.*

Proof. Given these assumptions, every time that the algorithm considers the k-th object, even if its path to its final pose is blocked, the algorithm will always be able to evacuate the $k-1$ blocking objects from this path to the set of intermediate collision-free poses. Given the overhand grasp setup, an object o_b can block another object's o path only when o_b is (partially) occluding object's o final pose. In this situation, CLEAR will select the intermediate pose for o_b and will not be called recursively.

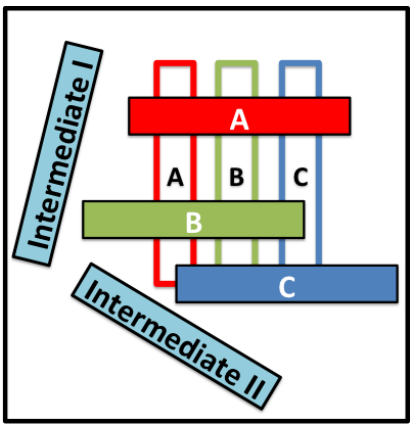

Figure 5.6: In order to solve this problem two out of the three objects have to move to an intermediate pose.

Figure 5.6 depicts an example of a worst case setup where $k-1$ objects have to move to intermediate poses in order to solve the problem. Given that the intermediate poses are collision-free among themselves and with all the final poses CLEAR will be able

to find an intermediate pose for all the blocking objects. The monotone primitive `mRS` will not be able to solve this problem. □

This property still leaves a large set of non-monotone challenges for which it is not possible to argue that the algorithm will find a solution. In particular, scenarios that the objects are not accessible with overhand grasps will generate more constraints. The interactions between the objects are not only constraints arising from the final poses. For example, in a shelf, the volume of the hand will generate additional constraints between the objects. For the non-monotone problem depicted in Fig. 5.7 (left) `plRS` 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. 5.7 (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 [115, 84, 118, 122, 104], which is itself hard.

Complete methods will end up coupling objects and try to plan in a composite space, which increases computational cost. For many of these challenges, the recursive nature of `CLEAR` helps in resolving the problem without too much search effort. The recursion results in a quadratic number of calls to transfer/transit primitives in the worst case for each edge of the backtracking search. This is still significantly less than considering all possible permutations if one considered all possible orders for moving the blocking objects. The question is whether this overhead is justified in practice given the increased capability of the approach to resolving problems. The recursive nature of `CLEAR`is practically helping in addressing a wide set of non-monotone instances as the accompanying experimental section shows (section 5.3).

It is understood, however, that is difficult to address the general case using a complete rearrangement primitive. The following section introduces the idea of a hierarchical approach, where the above primitives are considered as local planners within a higher-level task planner that searches the space of possible object arrangements. The benefit of these primitives is they provide good connectivity in the space of object arrangements.

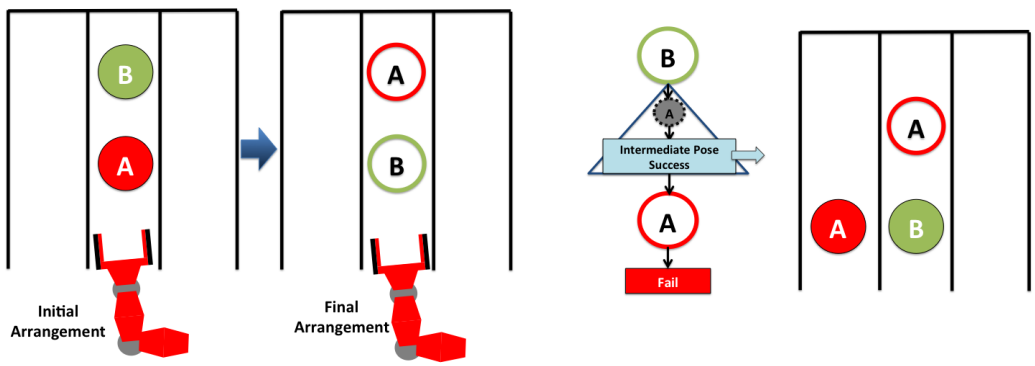

Figure 5.7: (left) An example problem where the non-monotone extension of the backtracking search approach fails. (right) The corresponding search tree given object B as the first object. The subproblem of finding an intermediate pose for A succeeds but then object A cannot reach its final pose.

5.2 A Hierarchical, Graph-theoretic Approach

The main idea here is to limit the search for a solution in the valid subset of the state space $\mathbb{X}^v$ and focus the search process on identifying transfer paths that allow the arm to rearrange objects. This is achieved in a hierarchical manner. The top-level algorithm searches globally the space of object arrangements by constructing a graphical representation that expresses which pairs of arrangements are reachable to each other using rearrangement primitives. This top-level method does not explicitly identify the motions of the arm that achieve the corresponding rearrangement. This is the responsibility of a connection rearrangement primitive between pairs of similar, "neighboring" arrangements. The similarity is defined based on a distance estimate in the space of object arrangements. If the connection primitive can connect all possible pairs of object arrangements in an efficient manner, then there would be no need for the top level search process. The proposed hierarchical approach integrates the available incomplete connection primitives in a higher-level search process that achieves probabilistic completeness. This section describes the top-level algorithm for searching in the space of object arrangements and its properties.

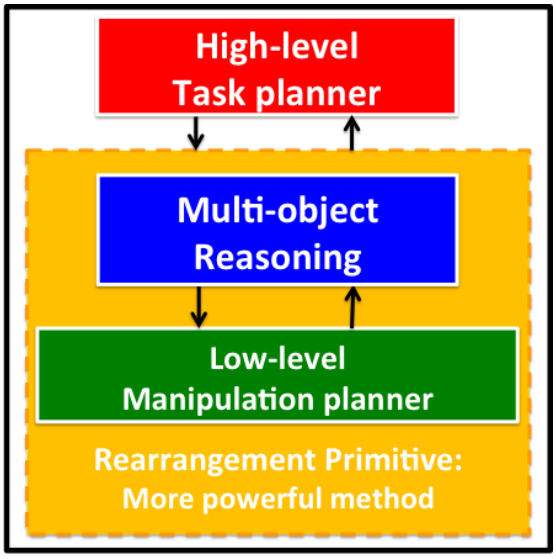

Figure 5.8: The hierarchical approach for solving rearrangement problems. A rearrangement primitive is used as a local planner for a high-level task planner.

5.2.1 Top-level Search over Transition Rearrangement States

The top-level search process can be performed in many different ways. For instance, it is possible to use heuristic search [40], or follow an incremental sampling-based algorithm, such as `RRT` [80]. Here a roadmap, similar to sampling-based roadmap methods, such as `PRM` [61] is built, which simplifies the reasoning about the coverage of the underlying state space and opens the door for potential path quality arguments. The method is referred to as `REARRANGE_PRM` and is summarized in Alg. 9.

The algorithm constructs a graph $G(V, E)$ where nodes $v \in V$ correspond to transition rearrangement states of the form $x = (q(\alpha[o_i]), \alpha) \in \mathbb{X}^{t_i}$ for some $i \in [1, k]$. In these states, all the objects are stably resting and the arm is grasping one of them. Edges $e \in E$ correspond to local rearrangement paths between such transition states. The algorithm is provided the specification of a "prehensile single-arm rearrangement problem", i.e., the workspace $\mathcal{W}$, the robot specification $\mathcal{R}$, the set of movable objects $\mathcal{O}$ and the start and goal states (q_I, α_I) and (q_F, α_F).

After the initialization of the graph (line 1), the method constructs the set of nodes that correspond to possible start and goal object arrangements, where the arm may be grasping one of the objects (lines 2-8). In particular, for every object $o \in \mathcal{O}$, the

Algorithm 9: STATE_BASED_REARRANGE_PRM($\mathcal{W}, \mathcal{R}, \mathcal{O}, q_I, \alpha_I, q_F, \alpha_F$)

```
1  G ← {V ← {∅}, E ← {∅}};
2  for each o ∈ O do
3      q_start = q(α_I[o]);
4      if TRANSIT( q_I, q_start, α_I) then
5          V ← V ∪ { (q_start, α_I) };
6      q_end = q(α_F[o]);
7      if TRANSIT( q_end, q_F, α_F) then
8          V ← V ∪ { (q_end, α_F) };
9  while ((Π, v_start, v_end) ← FIND_PATH(G, α_I, α_F)) == ∅ do
10     α_rand ← SAMPLE_STABLE_ARRANGEMENT(W, O, G);
11     for each o ∈ O do
12         q = q(α_rand[o]);
13         if COLLISION_FREE(W, R, O, q, α_rand) then
14             v_new ← (q, α_rand);
15             V ← V ∪ {v_new};
16             V_near ← NEAR_STATE(G, v_new);
17             for v_near ∈ V_near do
18                 π ← CONNECT_STATES(W, R, O, q, v_new, v_near);
19                 if π ≠ ∅ then
20                     E ← E ∪ { ((v_new, v_near), π) };
21 q_start = v_start.q; q_end = v_end.q;
22 Π ← TRANSIT( q_I, q_start, α_I) | Π | TRANSIT( q_end, q_F, α_I);
23 return Π;
```

method identifies whether it is feasible to move the arm from its initial configuration q_I to a grasping configuration $q(\alpha_I[o_i])$ given the initial arrangement α_I (lines 3-5). If this is possible, then the corresponding transition state, where the arm is grasping that object in the initial arrangement is added as a node. The same process is repeated for the final arrangement (lines 6-8), as long as it is possible to retract the arm from grasping an object $o \in \mathcal{O}$ in the final arrangement to the final arm configuration q_F.

The main loop of the algorithm (lines 9-20) searches for intermediate arrangement states that will allow connecting any of the nodes corresponding to the initial and final arrangements on the roadmap. In particular, the function FIND_PATH corresponds to a multi-start, multi-goal A^* search on the graph G between all the initial and final states (line 9). For as long as no path has been found, the algorithm samples a stable arrangement for the objects, i.e., one where all of them rest stably (line 10). For

each object, the method considers the corresponding grasping arm configuration (lines 11-12) and evaluates whether this is reachable by the robot and collision-free given the workspace and the sampled arrangement α_{rand} (lines 13). If it is reachable and collision-free, then it is possible to define the transition state where the objects are in the arrangement α_{rand} and the arm configuration allows the robot to grasp object o. This state is stored as a node v_{new} in the roadmap (line 14-15) and a set of neighboring nodes/states V_{near} are identified based on a distance estimate in the space of object arrangements (line 16). For each one of these nodes $v_{near} \in V_{near}$, the method attempts to connect v_{new} and v_{near} directly using the primitive `CONNECT_STATES` (lines 17-18). This is the lower-level primitive that is responsible to compute the actual manipulation path that allows moving between two neighboring object arrangements. If a local path π is found, then an edge between v_{new} and v_{near} is added to the roadmap (lines 19-20). This edge also stores the local rearrangement path π on the graph.

Once the function `FIND_PATH` finds a manipulation path Π between the initial and final poses, it also returns that actual start v_{start} and end state v_{state} along the corresponding path. These are used in order to compute the transit paths that allow the arm to move from q_I to the initial grasping state and from the final grasping state to q_F, which is then properly appended to Π so as to return the final solution (lines 21-23).

5.2.2 Conditions for Probabilistic Completeness for Single Pick and Place

The following discussion will first consider the integration of the above search process with an individual pick and place primitive and argue that under certain conditions it is possible to argue probabilistic completeness guarantees for this integration. A single pick and place action corresponds to a `TRANSIT` and a `TRANSFER` path for a single object o. The focus is on describing the conditions for Algorithm 9 to achieve probabilistic completeness for rearrangement problems using pick and place actions.

Consider a rearrangement problem $(\mathcal{W}, \mathcal{R}, \mathcal{O}, q_I, \alpha_I, q_F, \alpha_F)$, which has a solution path π with clearance properties as defined below in Assumption 1. Note that by definition the solution path can be decomposed into an alternating sequence of transit

and transfer sub-paths $\pi = \{\pi_1^0, \pi_1^{g_1}, \pi_2^0, \pi_2^{g_2}, \ldots\}$, where $g_j \in [1, m]$, so that all the transitions between subpaths are legal. In the context of rearrangement problems, the assumed clearance involves both object arrangements and the robot's motion.

Assumption 1. *For the rearrangement problem $(\mathcal{W}, \mathcal{R}, \mathcal{O}, q_I, \alpha_I, q_F, \alpha_F)$ there is a solution path π, for which if the poses of objects in the initial, final and any intermediate transition states along path π are disturbed by at most a certain distance $\delta_{\mathcal{O}}$, there is still a solution π' for that rearrangement problem. Moreover, all transit subpaths π_i^0 and transfer subpaths $\pi_i^{g_i}, g_i \in [1, m]$ for the arm have a clearance $\delta_{\mathcal{R}}$ from collision states. This implies that the solution path π has $\delta = f(\delta_{\mathcal{O}}, \delta_{\mathcal{R}})$ clearance in the state space $\mathbb{X}$ of the problem.*

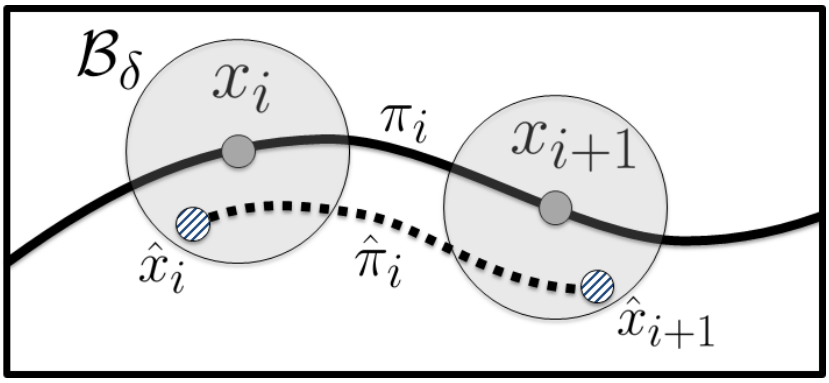

Figure 5.9: If there is a path to connect the states x_i and x_{i+1}, then there will also be a path to connect $\hat{x}_i$ and $\hat{x}_{i+1}$.

For any solution path π that satisfies the above assumption, denote all the corresponding intermediate transition states after the application of each transit π_i^0 and transfer subpath $\pi_i^{g_i}$ as x_i. The solution path based on states will be

$$\pi = \{x_0, \ldots, x_i, x_{i+1}, \ldots, x_m\}$$

, where $x_F = x_m$. The path between states x_i and x_{i+1} corresponds to a `TRANSIT` and a `TRANSFER` path for an object o_i.

The corresponding arrangements α_i and α_{i+1} for the states x_i and x_{i+1} will differ only by a single pose. With a random sampler it is possible to sample $\hat{x}_i \in \mathcal{B}_\delta(x_i)$ and $\hat{x}_{i+1} \in \mathcal{B}_\delta(x_{i+1})$. This does not mean, however, that $\hat{x}_i$ and $\hat{x}_{i+1}$ will differ only by one pose. They could differ up to k poses, since all the objects may be moved.

Algorithm 10: SAMPLE_NEAR_ARRANGEMENT$(\mathcal{W}, \mathcal{O}, \mathcal{G}(\mathcal{V}, \mathcal{E}))$

1 $(q_{rand},\ \alpha_{rand}) \leftarrow (q,\ \alpha) \in \mathcal{V}$;
2 $o_q \leftarrow \{o | q_{rand}(\alpha_{rand}[o])\}$;
3 $\alpha_{new} \leftarrow \alpha_{rand}$;
4 $\alpha_{new}[o_q] \leftarrow \{p | p \notin \alpha_{rand}\}$;
5 **return** α_{new};

Instead of the abstract sampler SAMPLE_STABLE_ARRANGEMENT used in Algorithm 9, a special sampler is considered here: SAMPLE_NEAR_ARRANGEMENT$(\mathcal{W}, \mathcal{O}, \mathcal{G})$, in order to restrict the sampling process and achieve desirable properties using a pick and place primitive. The special sampler (Alg.10) will select randomly a node (line 1) that contains an arrangement α_{rand} and a grasping configuration for the object o_q at the pose $\alpha_{rand}[o_q]$ (line 2). Then it will generate a new object arrangement that differs only by one pose from the randomly selected object arrangement α_{rand} (lines 3-4). When a state $\hat{x}_i \in \mathcal{B}_\delta(x_i)$ is selected as input for SAMPLE_NEAR_ARRANGEMENT there is a positive probability that it will sample the object arrangement $\hat{\alpha}_{i+1}$ that differs only by one pose from $\hat{\alpha}$ and also is in the vicinity of α_{i+1} that corresponds to x_{i+1}.

Lemma 3. *Given Assumption 1 and* SAMPLE_NEAR_ARRANGEMENT*, Algorithm 9 has a probability lower bounded by a positive value $\rho_\delta > 0$ to sample states $\hat{x}_i \in \mathcal{B}_\delta(x_i)$ for all the intermediate transition states x_i, where $i \in [0, m]$, along a solution path π with clearance δ.*

Proof. The action between states x_i and x_{i+1} is a single pick and place. This means that the object arrangements of these states will differ by a single pose. Without loss of generality assume:

$$\alpha_i[o_i] \neq \alpha_{i+1}[o_i] \text{ and } \alpha_i[\mathcal{O} \setminus o_i] = \alpha_{i+1}[\mathcal{O} \setminus o_i]. \tag{5.1}$$

The state $\hat{x}_0 = x_0$ is already in the set of graph nodes. As an inductive step, assume state $\hat{x}_i \in \mathcal{B}_\delta(x_i)$ is already sampled. Given the sampler SAMPLE_NEAR_ARRANGEMENT it is possible to generate $\hat{x}_{i+1}$, where

$$\hat{\alpha}_i[o_i] \neq \hat{\alpha}_{i+1}[o_i] \text{ and } \hat{\alpha}_i[\mathcal{O} \setminus o_i] = \hat{\alpha}_{i+1}[\mathcal{O} \setminus o_i]. \tag{5.2}$$

Since all the poses for the arrangement $\hat{\alpha}_i \in \hat{x}_i$ will be within the vicinity of the arrangement $\alpha_i \in x_i$, then it is also true that $\hat{\alpha}_i[\mathcal{O} \setminus o_i] \in \mathcal{B}_{\delta_\mathcal{O}}(\alpha_i[\mathcal{O} \setminus o_i])$.

Given Eqs. 5.1 and 5.2, the objects $\mathcal{O} \setminus o_i$ will be in the vicinity of the arrangement $\alpha_{i+1} \in x_{i+1}$, i.e., $\hat{\alpha}_{i+1}[\mathcal{O} \setminus o_i] \in \mathcal{B}_{\delta_\mathcal{O}}(\alpha_{i+1}[\mathcal{O} \setminus o_i])$. Because the objects $\mathcal{O} \setminus o_i$ will be in the same poses for states x_i and x_{i+1} as well as states $\hat{x}_i$ and $\hat{x}_{i+1}$. Furthermore, $\hat{x}_i \in \mathcal{B}_\delta(x_i)$. Eventually, a state $\hat{x}_{i+1}$ expanded from $\hat{x}_i$ using `SAMPLE_NEAR_ARRANGEMENT` will be sampled, where $\hat{\alpha}_{i+1}[o_i]$ will be in the vicinity $\mathcal{B}_{\delta_\mathcal{O}}(\alpha_{i+1}[o_i])$, which means that $\hat{x}_{i+1} \in \mathcal{B}_\delta(x_{i+1})$. This shows that the method will eventually sample all the intermediate transition states x_i of the solution path π, given the sampler `SAMPLE_NEAR_ARRANGEMENT`.

□

As mentioned before tow states x_i and x_{i+1} of the solution path π differ only in the pose of a single object. This means that a pick and place action will work between these two states and the object o_i will be moved from its initial pose $\alpha_i[o_i]$ to the pose $\alpha_{i+1}[o_i]$.

Lemma 4. *Since a "pick and place" action for object o_i allows to transition between two consecutive states x_i and x_{i+1} of the solution path π, it will also work for the states $\hat{x}_i$ and $\hat{x}_{i+1}$ if:*

- $\hat{x}_i \in \mathcal{B}_\delta(x_i)$,
- $\hat{x}_{i+1} \in \mathcal{B}_\delta(x_{i+1})$ *and* $\hat{a}_{i+1}[o] = \begin{cases} \mathcal{B}_{\delta_\mathcal{O}}(\alpha_{i+1}[o]), & o = o_i; \\ \hat{\alpha}_i[o], & o \in \mathcal{O} \setminus o_i. \end{cases}$

Proof. Given lemma 3 the method will be able to sample all the intermediate states $\hat{x}_i$ along the solution path $\hat{\pi}$. The sampled states will be in the vicinity $\mathcal{B}_{\delta_\mathcal{O}}$ of the original states. Given assumption 1 the pick and place action will be able to work between the states $\hat{x}_i$ and $\hat{x}_{i+1}$ because it was able to work between the states x_i and x_{i+1} and all the objects are within distance $\delta_\mathcal{O}$. □

Theorem 1. *The high-level task planner* `STATE_BASED_REARRANGE_PRM` *(Alg.9) with the following components:*

- `CONNECT_STATES`*: a simple pick and place consisting by a* `TRANSFER` *and a* `TRANSIT` *path for the manipulator to grasp an object and move it to a new pose;*
- `SAMPLE_STABLE_ARRANGEMENT`*: the incremental sampler* `SAMPLE_NEAR_ARRANGEMENT` *and*
- `NEAR_STATE`*: searches all the nodes and returns as neighbors all the nodes that differ by a single pose.*

will find a solution to the rearrangement problem $(\mathcal{W}, \mathcal{R}, \mathcal{O}, q_I, \alpha_I, q_F, \alpha_F)$ *with probability reaching 1 as the number of samples increases, if a solution exists.*

Proof. Given a prehensile rearrangement problem $(\mathcal{W}, \mathcal{R}, \mathcal{O}, q_I, \alpha_I, q_F, \alpha_F)$ consider a solution path π. The solution path can be decomposed into a sequence of segments $\{\pi_1, \pi_2, \ldots, \pi_m\}$, where each π_i corresponds to a `TRANSIT` and a `TRANSFER` operation. Denote the object moved during path segment π_i as o_i. The start state of π_i is $x_i = (q_i, \alpha_i)$ and the final state is $x_{i+1} = (q_{i+1}, \alpha_{i+1})$, where $q_i = q(\alpha_i[o_i])$ (and $q_0 = q_I$).

The algorithm will eventually sample states in the vicinity $\mathcal{B}_{\delta_\mathcal{O}}$ of all the states along the solution path (lemma 3), and can connect these states using the simple pick and place action (lemma 4). Thus, it will generate a solution path $\hat{\pi}$ in every case. □

5.2.3 Conditions for Probabilistic Completeness using More Expressive Primitives

The previous subsection gave the proof that a high-level task planner using single pick and place is complete for rearranging problems. However, this method needs a special sampler for this argument to work. Furthermore, it requires a very large number of samples before it can achieve a solution as it has to densely sample in the vicinity of a solution path. Section 5.1 presented primitives where two states x_i and x_j can be connected even if they do not differ by a single position of an object o.

Given the previously used decomposition, the solution path π is split into a sequence of `TRANSIT` and `TRANSFER` sub-paths. Along the solution path π there could be two states x_i and x_j, which differ up to k poses. If each object is moved at most once

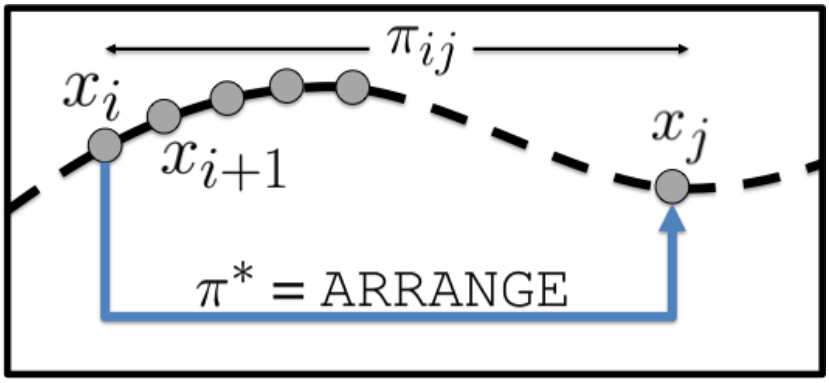

Figure 5.10: Along the solution path π there are states that can be connected using a local rearrangement primitive.

between the two states, then the states x_i and x_j can be connected with a path π_{ij}, which corresponds to a solution achieved by the monotone primitive. The path π_{ij} provides a sequence of `TRANSIT` and `TRANSFER` sub-paths for rearranging the objects from the initial state x_i to the final state x_j. The sequence can be detected by the monotone primitive and can connect directly the states x_i and x_j.

Given a more powerful primitive, it is possible to define a new decomposition of the solution path, into a sequence of local rearrangement sub-paths, fig.5.10. Each `ARRANGE` sub-path will be the solution of a local rearrangement problem between two neighboring states.

Lemma 5. *If two states x_i and x_j along the solution path π for the rearrangement problem can be connected with a primitive that is complete for monotone problems, then the states $\hat{x}_i \in \mathcal{B}_\delta(x_i)$ and $\hat{x}_j \in \mathcal{B}_\delta(x_j)$ can also be connected using the same primitive.*

Proof. If the two states x_i and x_j can be connected with a monotone path π_{ij}, then the path π_{ij} can be decomposed into a sequence of pick and place sub-paths $\{x_l, x_{l+1}\}$. The object arrangements of the states x_l and x_{l+1} will differ only by a single pose for an object o_l. The pose of object o_l in the state x_l will have to be the same as in the x_i state, while the one in x_{l+1} will have to be the same as in the x_j state since the π_{ij} path is monotone. In other words, the object arrangement of a state x_l will have some poses of objects borrowed from state x_i and the remaining one will come from x_j.

Then, for states $\hat{x}_i \in \mathcal{B}_\delta(x_i)$ and $\hat{x}_j \in \mathcal{B}_\delta(x_j)$, it is possible to define the monotone path $\hat{\pi}_{ij}$, which involves the same sequence of pick and place actions as the path π_{ij} and can be decomposed in a similar sequence of pick and place sub-paths $\{\hat{x}_l, \hat{x}_{l+1}\}$. Each one of the states $\hat{x}_l$ is guaranteed to be in the vicinity of $\mathcal{B}_\delta(x_l)$. Given that states x_i and x_j satisfy the desirable clearance properties, the state x_l is also satisfying them, implying that state $\hat{x}_l$ is a valid state. Given that the primitive is assumed complete for monotone problems and all the intermediate states are valid, then the states $\hat{x}_i \in \mathcal{B}_\delta(x_i)$ and $\hat{x}_j \in \mathcal{B}_\delta(x_j)$ can be connected with the corresponding primitive.

□

Theorem 2. *The high-level task planner* `STATE_BASED_REARRANGE_PRM` *(Alg.9), using as:*

- `CONNECT_STATES`*: any primitive that is complete for monotone problems,*
- `SAMPLE_STABLE_ARRANGEMENT`*: a random sampler and*
- `NEAR_STATE`*: all roadmap nodes are returned as neighbors,*

will find a solution to the rearrangement problem $(\mathcal{W}, \mathcal{R}, \mathcal{O}, q_I, \alpha_I, q_F, \alpha_F)$ *with probability reaching 1 as the number of samples increases, if a solution exists.*

Proof. Given a prehensile rearrangement problem $(\mathcal{W}, \mathcal{R}, \mathcal{O}, q_I, \alpha_I, q_F, \alpha_F)$ consider a solution path π. The solution path can be decomposed into a sequence of segments $\{\pi_1, \pi_2, \ldots, \pi_m\}$, where each π_i corresponds to a monotone path, i.e., a path that can be found by the primitives of Section 5.1.

The random sampling method will sample with positive probability states in the vicinity $\mathcal{B}_\delta$ of the states on the solution path. Given lemma 5 the primitive solver that is being used in this high-level task planner will be able to connect these states. Moreover, given that the algorithm will try to connect a new sampled state with all the existing states will generate the solution path π in every case.

□

Similar proof holds for the non-monotone primitives given the lemma 2, which says that a non-monotone primitive will work the same way as the monotone primitive in all the monotone cases. As a result, the `STATE_BASED_REARRANGE_PRM` is complete when using any monotone or non-monotone primitive as connection function.

Note that Theorem 2 does not require the previous special sampler, which was used in the case of the pick and place connection primitive. Instead, a naive random sampling process in the arrangement space is sufficient. This means that the algorithm is going to need fewer samples in order to sample all the states along a solution path π. This way the algorithm `STATE_BASED_REARRANGE_PRM` does not have to sample all the different pick and place actions in order to connect two states. Instead, states that differ up to k poses can be connected. Nevertheless, the near function of Theorem 2 seems to be more expensive, since the algorithm needs to try connections with all the existed samples. Experimentally, it is shown that it is beneficial to try connections with few neighboring samples instead of trying to sample all the states along a solution path.

A reasonable concern is why should one use the rearrangement search primitives since it is possible to show probabilistic completeness 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. On the other hand, the proposed primitives can work successfully in connecting pairs of arrangements that do not share any pose. For instance, the `mRS` primitive will succeed, if there is a δ-ball around each arrangement α_i along a segment π'_i so that for all α'_i in the δ-ball: i) the path segment π'_i connecting arrangements α'_{i-1} to α'_i is also part of a solution path π', and ii) the path segment π'_i remains monotone. The probability a local segment π'_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 probabilistic 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 trade-off is computational in nature. Consider a different decomposition of the same solution π into segments $\{\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_n\}$, where each segment $\hat{pi}_i$ 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 `plRS`. 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 π'_i versus generating a pick and place segment π_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 work show this tradeoff turns out to be in favor of the more expressive and computationally expensive connection primitives.

5.2.4 Faster Search over Arrangement Space can be also Complete

Algorithm 9 will generate a lot of nodes that are not needed for the solution of the problem. This is because each node corresponds to an object arrangement and a grasp for an individual object, e.g., $x = (q(\alpha[o]), \alpha)$. For each sampled arrangement of k objects, the method generates up to k new nodes for the graph, where the arm grasps a different object in each case. A more efficient search process is possible, which samples only objects arrangements and not problem states. For a pair of object arrangements, it is possible to evaluate whether they can be connected by assuming a safe arm configuration $q_s^{\mathcal{R}}$ and trying to use one of the primitives to move the arm from $q_s^{\mathcal{R}}$ back to the same safe configuration $q_s^{\mathcal{R}}$. This simplification works under the following assumption.

Assumption 2. *There is a solution path with grasps $q(\alpha[o])$ that are reachable from the configuration $q_s^{\mathcal{R}}$ of the robot. The configuration $q_s^{\mathcal{R}}$ will be called a "safe configuration".*

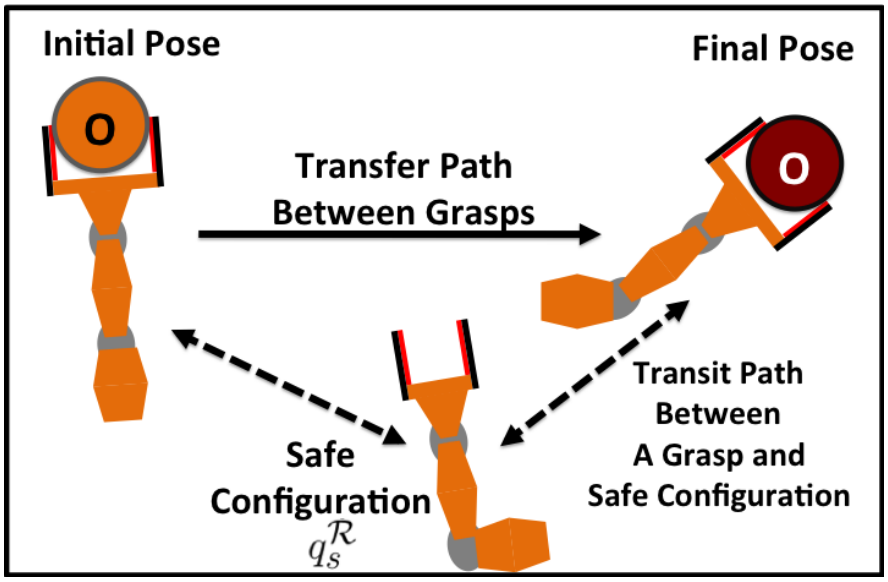

Figure 5.11: All the grasp states are reachable from the safe configuration $q_s^{\mathcal{R}}$ of the robot. As a result, the robot can start from the safe configuration grasp an object from an initial pose, transfer it to a final pose and then return back to the safe configuration.

The above assumption also means that all the generated grasps from the algorithm will be reachable from $q_s^{\mathcal{R}}$. See Fig.5.11 for a related illustration.

Algorithm 11 depicts how `REARRANGE_PRM` will work when sampling takes place only in the arrangement space. This new algorithm will construct a graph with fewer nodes, which searches over the arrangement space. It has the same completeness properties with the previous high-level algorithm under assumption 2.

Theorem 3. *The high-level task planner* `REARRANGE_PRM` *(Alg.11), using as:*

- `CONNECT_STATES`*: any primitive that is complete for monotone problems, where the initial and the final configuration of the robot is the same,*
- `SAMPLE_STABLE_ARRANGEMENT`*: a random sampler and*
- `NEAR_STATE`*: all roadmap nodes are returned as neighbors,*

will find a solution to the rearrangement problem $(\mathcal{W}, \mathcal{R}, \mathcal{O}, q_I, \alpha_I, q_F, \alpha_F)$ *with probability reaching 1 as the number of samples increases, if a solution exists, given that all the grasp states are reachable to the safe configuration* $q_s^{\mathcal{R}}$ *of the robot.*

Algorithm 11: REARRANGE_PRM($\mathcal{W}, \mathcal{R}, \mathcal{O}, q_s^{\mathcal{R}}, \alpha_I, \alpha_F$)

$\mathcal{G} \leftarrow \{\mathcal{V} \leftarrow \{\alpha_I, \alpha_F\}, \mathcal{E} \leftarrow \{\emptyset\}\}$;
while $((\Pi, v_{start}, v_{end}) \leftarrow$ FIND_PATH$(\mathcal{G}, \alpha_I, \alpha_F)) == \emptyset$ **do**
  $\alpha_{rand} \leftarrow$ SAMPLE_STABLE_ARRANGEMENT$(\mathcal{W}, \mathcal{O})$;
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{\alpha_{rand}\}$;
  $V_{near} \leftarrow$ NEAR$(\mathcal{G}, \alpha_{rand})$;
  for $\alpha_{near} \in V_{near}$ **do**
    $\pi \leftarrow$ CONNECT$(\mathcal{W}, \mathcal{R}, \mathcal{O}, q_s^{\mathcal{R}}, \alpha_{rand}, \alpha_{near})$;
    if $\pi \neq \emptyset$ **then**
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\alpha_{rand}, \alpha_{near}), \pi\}$;
return Π;

Proof. Given a prehensile rearrangement problem $(\mathcal{W}, \mathcal{R}, \mathcal{O}, q_I, \alpha_I, q_F, \alpha_F)$ consider a solution path π. The solution path can be decomposed into a sequence of segments $\{\pi_1, \pi_2, \ldots, \pi_m\}$, where each π_i corresponds to a monotone path, i.e., a path that can be discovered by the primitives of Section 5.1.

Each of these component paths will start and end with the safe configuration $q_s^{\mathcal{R}}$. Given assumption 2 all the grasps that are used in the solution path can be connected to the safe configuration. This means that the rearrangement primitive will be able to start and end from the safe configuration for the next segment π_i of the solution path.

The method using a random sampler will sample with positive probability states in the vicinity $\mathcal{B}_\delta$ of the states on the solution path. Given lemma 5 the rearrangement primitive solver that is being used in this high-level task planner will be able to connect these states. Then the configuration of the manipulator will end up in the safe configuration, $q_s^{\mathcal{R}}$. Moreover, given that the algorithm will try to connect a new sampled state with all the existing states will generate the solution path π in every case. □

Theorem 3 shows that if the available grasps for the algorithm are all connected to a safe configuration $q_s^{\mathcal{R}}$, then the algorithm will be able to find the solution. Considering only grasps that are reachable from $q_s^{\mathcal{R}}$ results in a computationally more efficient search.

5.3 Evaluation

This section evaluates the proposed algorithmic framework through the aid of a simulation environment. The experiments in the simulation environments focus on scalability issues and the ability of the algorithms to solve difficult non-monotone problems including in tight environments with limited free space. Furthermore, the methods are demonstrated on a physical platform. The physical experiment illustrates the ease with which the proposed methods can be used on a real system.

5.3.1 Physical Experiment

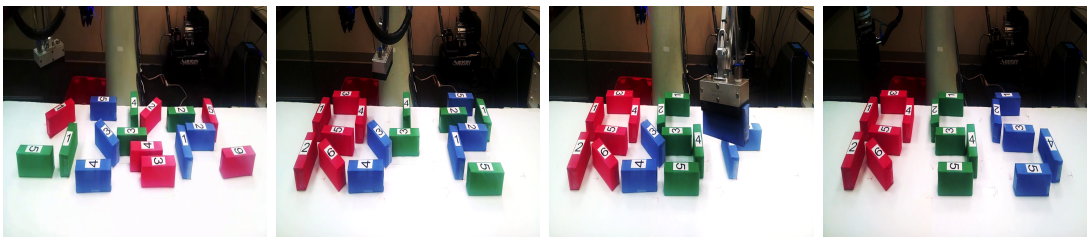

Figure 5.12: The "RSS challenge" solved by a single arm of a Yaskawa Motoman SDA10F robot equipped with a UniGripper vacuum tool. The images from left to right depict the process of rearranging the target objects so as to achieve the desired configuration of the "RSS challenge".

An arm of a Yaskawa Motoman SDA10F robot is used for the physical experiment. The arm has 7 joints, while the robot also provides a torsional degree of freedom. The arm is carrying a UniGripper vacuum tool. This end-effector provides an additional, wrist-like degree of freedom and is using suction to attach to and carry objects. The experiment involves sixteen cubic objects placed in front of the robotic arm on a flat surface. While the objects have the same geometry, they are uniquely identified. Initially, the objects are randomly placed on the table and a solution path is computed in order to transfer the objects to the desired target arrangement. For the example shown in Fig. 5.12, the target arrangement is to form the letters RSS with the sixteen objects.

The robot is provided the initial poses of the objects, the target arrangement as well as the static geometry. Then, a solution path is computed using the non-monotone algorithm (`plRS`). The robot executes the computed path by employing the architecture

described in the previous section. This allows to reason about the sequence of object arrangements that give rise to the solution as well as the motion of the arms. For the performed physical experiments the robot was able to solve the problem in an open-loop manner and engage/disengage the end-effector when necessary in order to move the objects to new locations.

The non-monotone algorithm (`plRS`) came up with a solution for this problem in 1.39 seconds. However, the execution time was about 18 minutes. Given that the path is executed in an open-loop manner the robot is using half of its maximum speed in order to be more robust. The robot had to move 20 objects, where 5 of them moved to an intermediate location before getting to their target.

The solution path has only one redundant step. In order to free the target pose of the moving object, the algorithm had to select an intermediate pose for the blocking object. The new intermediate pose was blocking the target pose of the next moving object. However, it is not possible for the algorithm to respect all the future objects when detects intermediate pose for a blocking object.

5.3.2 Simulated Setup

In terms of a more comprehensive evaluation of alternative approaches to rearrangement problems, three different simulated workspaces are considered where a model of a Baxter arm is used to move objects around. The Baxter arm has seven degrees of freedom and utilizes a parallel gripper in order to grasp and move objects. The "grid@tabletop" space is shown in Figure 5.13 (left), the "grid@shelf" space is shown in Figure 5.13 (right)) and the "`RSS` challenge" for the Baxter robot was illustrated in Figure 1.1.

The "RSS challenge" involves six, eleven or sixteen 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 four, ten and sixteen cubic objects on a tabletop. The "grid@shelf" challenge has two, six and ten cylinders placed on a shelf that limits the arm's reachability and does not allow overhand grasps. In both the "grid@tabletop" and the "grid@shelf" instances, the objective is to rearrange the objects from a random to a grid arrangement.

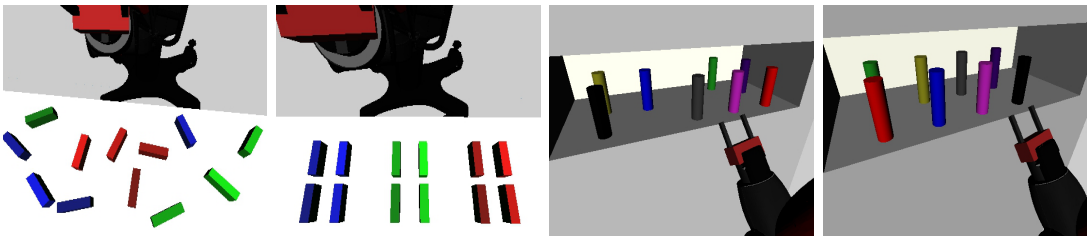

Figure 5.13: (left) The "grid@tabletop" problem. An initial and the final arrangement of the objects on the table. (right) The "grid@shelf" problem. An initial and the final arrangement of the objects inside the shelf.

Four different methods are evaluated in these setups. First, the rearrangement primitives: (a) `mRS` and (b) `plRS` are considered. Then, two versions of the hierarchical scheme are considered, where the function `CONNECT` of the high level `REARRANGE_PRM` is utilizing either the (c) `mRS` or the (d) `plRS` approach. The time limit provided to the methods was 10 mins. 10 experiments were performed for each combination of method and environment. All the statistics beyond success ratio are reported over the successful runs.

Benchmark: `RSS` Challenge for the Baxter robot

Figure 5.14 provide the results for the "`RSS` challenge" using a simulated Baxter robot for the four alternative methodologies. For six objects, all the methods with the exception of the monotone primitive were able to find a solution in all instances, Fig.5.14 top. As the number of objects increases, however, there is limited space for the objects to be placed and there is high probability for two objects to overlap each other's final positions. For that reason, the success rate of the monotone solver goes down quickly with additional objects. The non-monotone solver, however, manages to solve all instances. In terms of computation time, there is an increase for `plRS` as the number of objects increases but overall it performs satisfactorily across the board.

The hierarchical schemes based on `REARRANGE_PRM` resemble the computation times of the primitives they employ (Figure 5.14 middle). For the case with the six objects, all the examples were monotone and all the algorithms found the monotone solution. For eleven objects, the `REARRANGE_PRM` using the monotone primitive (`mRS`), improves

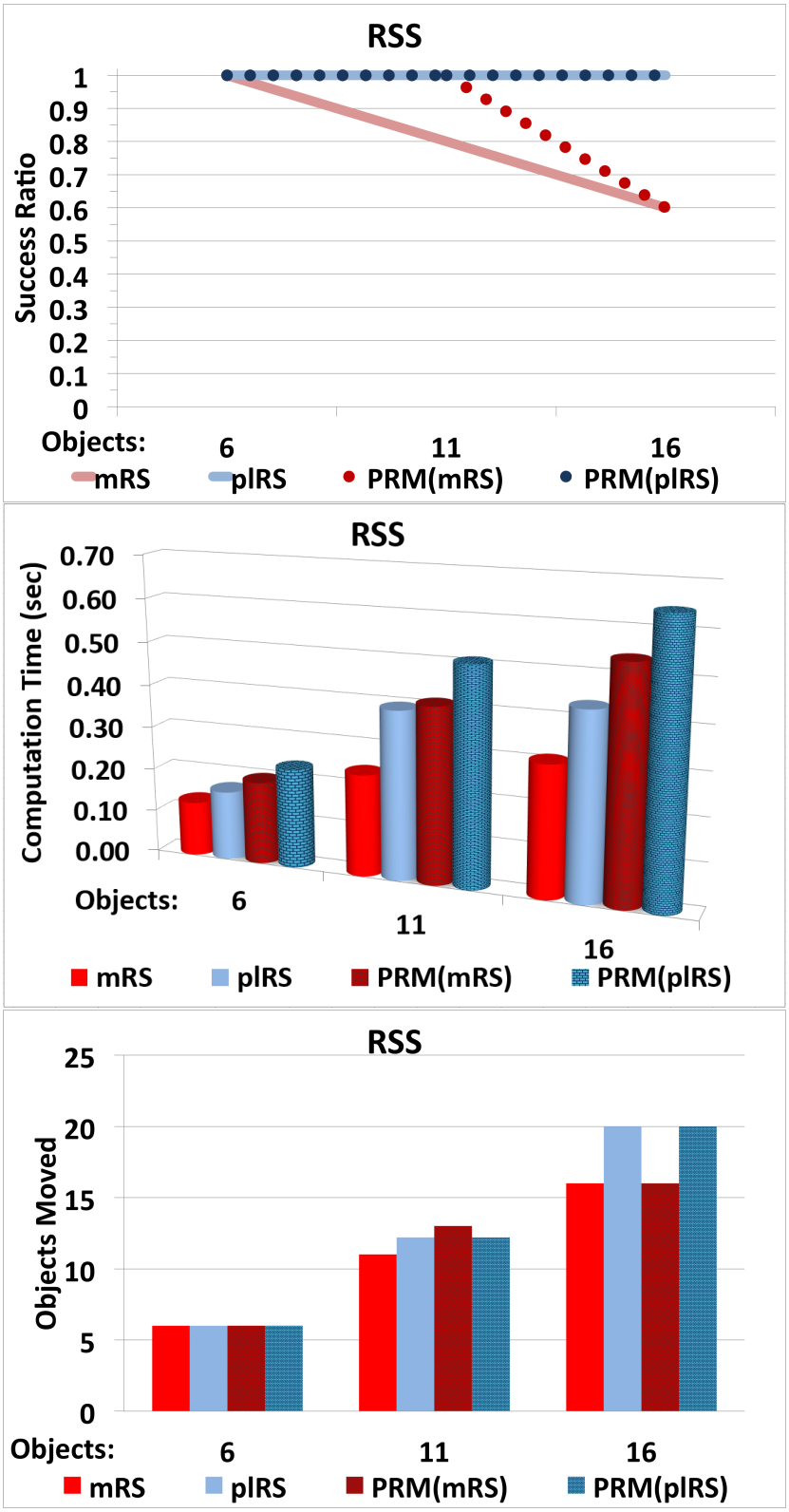

Figure 5.14: (Top) The success ratio, (Middle) the computation time , (Bottom) the number of object transfers for each algorithm out of 10 runs for the "`RSS` challenge" with the simulated Baxter robot. The results is the average number among 10 runs.

upon the success ratio of the primitive by itself. For sixteen objects, however, it does not have the time to solve all of the problems with sixteen objects as the monotone primitive cannot easily connect two randomly sampled arrangements resulting in a highly disconnected arrangement space. The non-monotone primitive is effective in this case and can quickly provide a solution. As a result, the `REARRANGE_PRM` using the non-monotone primitive (`plRS`) could solve all the problems and return a similar solution.

The number of objects that each algorithm had to transfer is shown in figure 5.14 bottom. It is easy, again, to realize that the examples with 6 objects were monotone problems because on average all the algorithms had to move 6 objects. For the other problems the `plRS` and the `REARRANGE_PRM` using `plRS` transferred more objects because the combination of the high-level planner `REARRANGE_PRM` and the primitive could solve the most difficult problems that require to detect multiple intermediate poses and move multiple objects to these intermediate poses.

Benchmark: "grid@tabletop"

Figure 5.15, provides the results for "grid@tabletop" for the four rearrangement algorithms. In this example, there are many partial overlaps between final poses and intermediate or initial poses. This affects the performance of the monotone method, which was not able to solve any problem with sixteen objects. In addition, the interactions between the poses affect the running time for the non-monotone primitive (`plRS`), which is successful across all challenges but needs to search over a large number of possible orders to find one that works.

The `REARRANGE_PRM` method using the monotone primitive `mRS` could solve all the problems up to ten objects, but found solutions only for 20% of the problems with sixteen objects (Figure 5.15 top). The non-monotone solvers run slightly slower in the smaller examples. This is because they spend a little bit more effort to find intermediate poses for a non-monotone solution instead of searching if there is an easy monotone solution. For the same reason, the `REARRANGE_PRM` using the monotone primitive `mRS`

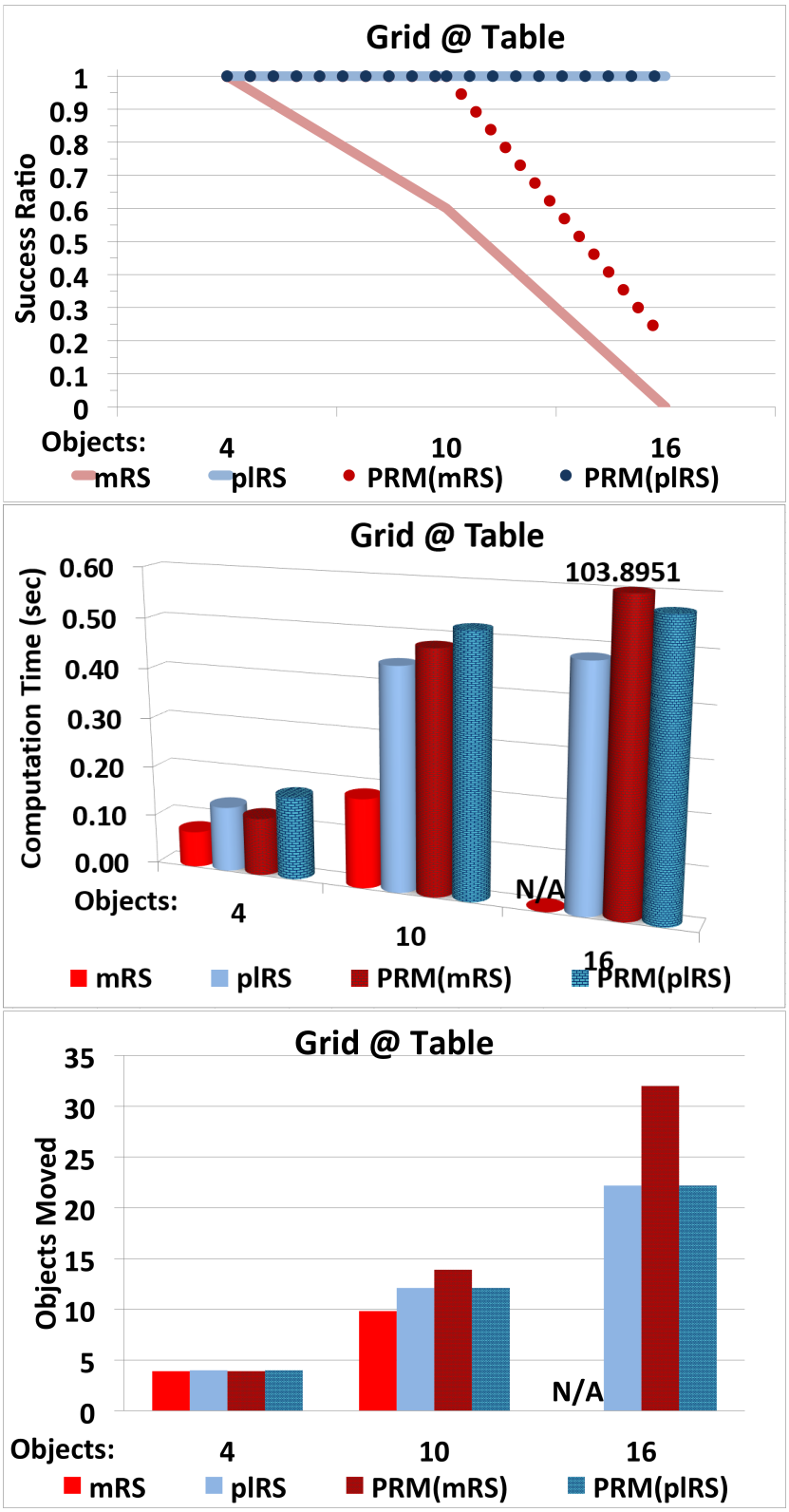

Figure 5.15: (Top) The success ratio, (Middle) the computation time (Bottom) he number of object transfers for each algorithm out of 10 runs for the "grid@tabletop challenge" with the simulated Baxter robot. The results is the average number among 10 runs.

is faster than the REARRANGE_PRM that uses the non-monotone primitive plRS. Nevertheless, the REARRANGE_PRM with mRS is significantly slower when there are sixteen objects, where it could solve only 20% of the problems. These are the easiest problems. Restricting the comparison only in this set, the REARRANGE_PRM using plRS can solve them faster and more efficiently in terms of solution quality.

Similar to the previous benchmark the non-monotone primitive and the high-level planner REARRANGE_PRM using the non-monotone primitive (plRS) could connect immediately the initial arrangement with the target arrangement. These algorithms found a solution for all the problems by moving some extra objects to intermediate poses. However, the monotone solver could solve all the problems with the four object, but could not solve any problem with sixteen objects. The REARRANGE_PRM using the monotone primitive also has low success ratio and had to move more objects than the non-monotone algorithms before finding a solution to the problems with sixteen objects.

The non-monotone solvers run slightly slower in the smaller examples (Fig.5.15 middle). The 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 using non-monotone methods tracks those of the primitives as they typically succeed because first connection works, given that the non-monotone algorithm solved all the problems.

The high number of transfers for the REARRANGE_PRM using mRS (Fig.5.15 bottom) is because the algorithm had to solve many sub-problems until it finds a solution. For the same reason the computation time is three orders of magnitude more than the non-monotone methods, Fig.5.15 middle. plRS on average had to move 6 objects on an intermediate pose before returns a solution to a problem with 16 objects for the "grid@tabletop challenge".

Benchmark: "grid@shelf"

The results for the last challenge, "grid@shelf", are shown in the Figure 5.16. Here the arm has reduced reachability and is not able to use overhand grasps. As the number of objects increases, the monotone solvers' success ratio goes down very quickly, because there are more occluded objects that need to be moved to intermediate poses in order

to find a solution. The non-monotone primitive has better success ratio despite the difficulty of the challenge and the limited reachability. Nevertheless, its success ratio and running time deteriorate as well, especially with 10 objects in the shelf. Moreover, the `plRS` primitive has to search a lot before identifying that it cannot return a solution.

Using the non-monotone primitives within `REARRANGE_PRM` results in higher success rate (Figure 5.16 top). When the algorithm tries to find a solution for the examples with eight objects in the shelf, then only the version with the `plRS` manages to solve all the problems but one. The monotone primitive failed to report a solution in almost all of the problems. With regards to running time, the non-monotone primitive `plRS` appears to be faster than the `REARRANGE_PRM` scheme using the `plRS` (Fig.5.16 middle), but it can solve only half of the problems in the shelf. These are the easier examples and for that reason, these examples need less number of transfers (Fig.5.16 bottom). Harder problems, which only the proposed integration of the sampling-based planner with the non-monotonic primitive could solve, require more search in the arrangement space and more transfers in order to get a solution.

For the last benchmark, the `REARRANGE_PRM` using the non-monotone primitive could solve almost all the problems. This algorithm is moving the most objects out of all the other algorithms (Fig.5.16 bottom) because it could solve the most difficult problems that require to detect multiple intermediate poses and move multiple objects to these intermediate poses. For the problems with six objects, the `REARRANGE_PRM` that uses the monotone primitive could solve all the problems, but had to move more objects in order to find the solution. The reason for these extra moves is the intermediate nodes in the high-level graph that the algorithm has to visit. The problems are not monotone and the algorithm had to generate more nodes that are connected with a monotone path to the initial arrangement and the final arrangement.

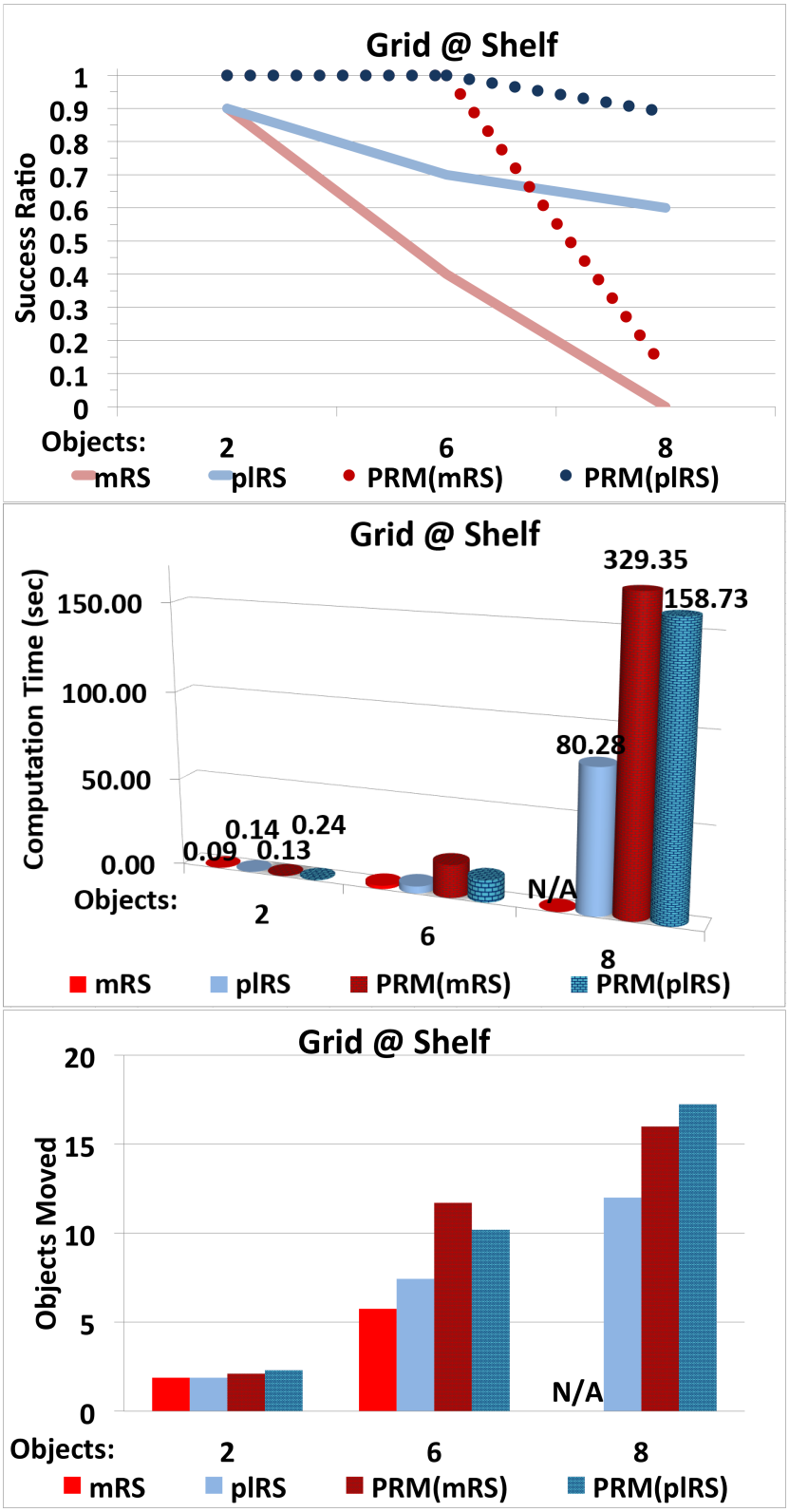

Figure 5.16: (Top) The success ratio, (Middle) the computation time (Bottom) he number of object transfers for each algorithm out of 10 runs for the "grid@shelf challenge" with the simulated Baxter robot. The results is the average number among 10 runs.

5.4 Discussion

The focus of the current work has been on showing the potential of combining sampling-based algorithms with strong local primitives to solve hard object rearrangement problems. The first step was a new algorithmic primitive for object rearrangement that is able to practically solve many non-monotone instances. It extends a previously developed technique for monotone problems [110] and utilizes principles from the multi-robot planning literature [85, 71] as well as efficient solutions to the Minimum Constraint Removal path problem [46, 47, 67]. This non-monotone primitive is integrated with a higher-level planner in the context of a hierarchical framework, which operates to a similar function to a probabilistic roadmap (`PRM`). This work shows that the proposed integration achieves probabilistic completeness by searching the space of possible object arrangements. It uses the proposed non-monotone primitive as a local planner to locally connect pairs of object arrangements, which constitute nodes of the high-level `PRM` planner. Simulated experiments show that the proposed primitive solves many non-monotone challenges by itself. The integration with the higher-level task planner results in an improved solution in terms of success ratio, path quality and scalability, especially in setups in tight spaces and with limited reachability for the manipulator, such as shelves. Typically, few arrangements are sampled by the high-level planner to solve relatively hard instances when the primitive is failing to find a solution directly.

The provided simulation experiments make use of objects of similar geometry for which it is easy to quickly identify grasps. This allows performing helpful precomputation towards fast resolution of rearrangement queries. The framework itself, however, does not depend on such assumptions and can be applied in general setups with objects of varying geometry given a primitive for effectively detecting appropriate grasps. Furthermore, there are many different algorithmic choices for integrating sampling-based planners with local primitives in the context of object arrangement. For instance, instead of sampling different object arrangements and trying to connect them with a local primitive, similar to the way a `PRM` algorithm works, one can consider an incremental approach, similar to a bidirectional `RRT`. In this case, the local primitive does not have

to exactly connect two object arrangements. Instead, it can partially extend an initial arrangement towards a target one. Preliminary indications with a bidirectional `RRT` and a fast, partial extension primitive, show promise in significantly reducing computation time while utilizing a similar hierarchical framework [69]. In the context of such incremental approaches, it is easy to apply heuristics for guiding the search process and solving the problem faster. For the hierarchical framework proposed in this work, heuristics can be introduced in the form of a biased sampling process at the high-level `PRM` planner.

Another useful objective in the context of object rearrangement is to identify the set of problems which are directly addressable by primitives similar to the non-monotone one proposed here and which do not require the proposed more general, hierarchical approach. One can look for inspiration in methods rooted in multi-robot planning and especially those, which can provide optimality guarantees [118, 105, 123]. This can lead to the development of solutions, which under certain conditions, can provide high-quality paths. In the context of the proposed, hierarchical and sampling-based framework, it is similarly important to consider the conditions under which it can asymptotically converge to optimal solutions. There has been recent work in the area of asymptotic optimality for manipulation task planning [117], which utilizes sampling-based algorithms. This work can indicate a direction of how the proposed framework can achieve such an objective while remaining computationally efficient.

Future efforts should also focus on the computation of robust rearrangement trajectories, given actuation and observation noise, as different manipulation operations have a different probability of being successful in practice. There has been some work in hierarchical decision-theoretic planning tools for related problems [82], which motivate further work in this area. Similarly, tight integration with perception for detecting objects is necessary for the real-world adoption of rearrangement algorithms in target applications. Furthermore, the current work has not considered rearrangement problems where physics can play a critical role, such as stacking challenges [113]. At a combinatorial level, the algorithmic primitives developed as part of this work can still be helpful in such setups but need to be integrated with processes that reason about

the feasibility of manipulation actions, given physical constraints.

An interesting extension of the framework would involve the use of two arms. In this case, one arm can grasp a blocking object in order to clear the scene, while the second object can reach for a previously blocked object. The current setup can also be integrated with non-prehensile primitives. Operations, such as pushing, can be considered as the actions for transferring the objects in the environment [25, 34]. Similarly, it can also be extended to the case of mobile manipulation [40], where the robot is able to make use of additional free space that is accessible only once the robot navigates to it. Cloud computing can be considered as a practical way to improve performance through parallelization and share knowledge between different solutions [9].

Chapter 6

A Fast Incremental Search Framework for General Object Rearrangement

In the previous chapter, it was shown that an extension of the monotone solver `mRS`, could solve non-monotone problems. The method `plRS` could deal with harder problems, but could not solve all the problems in limited time. Probabilistic completeness for object rearrangement was proven by using a task planner that builds a graph in the space of object arrangements. The task planner works similar to `PRM`* [58, 61] and used the `plRS` algorithm for connecting pairs of object arrangements [68]. Building such a `PRM`-like roadmap requires that the local planner can frequently connect two randomly sampled arrangements. But this is often not possible given the size of the search space even with a powerful local planner.

This chapter focuses on finding more efficient solutions for hard instances of general object rearrangement. The first criterion is to use **fast rearrangement primitives** that can solve many rearrangement problems. By taking advantage of recent algorithmic insights [115, 47, 67] a "fast monotone Rearrangement Solver" (`fmRS`) and "fast non-monotone Rearrangement Solver" (`fplRS`) avoid backtracking search are proposed. While the procedure is incomplete, even for monotone instances, it has a high success ratio and significant computational benefits.

The second criterion in order to have fast solutions is to **maintain good connectivity** for the graph in the arrangement space. The rearrangement primitives can be adapted and return a partial solution when no complete solution is found. This involves moving as many objects as possible towards the target arrangement. Given the partial formulation of the rearrangement primitives, it is possible to use them in a high-level task planner as an expansion step from an initial arrangement instead of

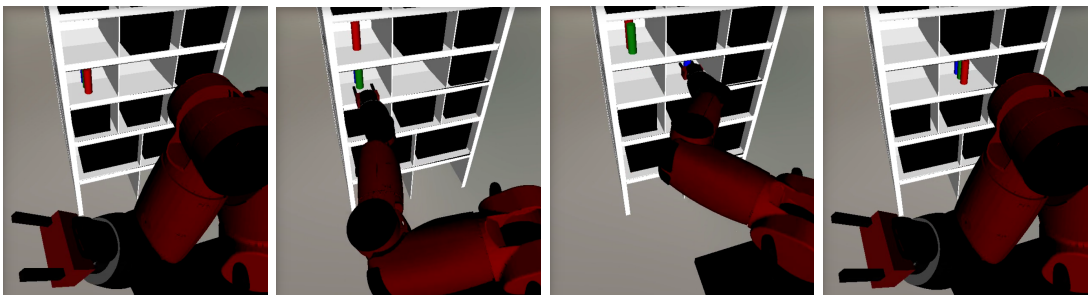

Figure 6.1: A toy example of the towers of Hanoi. The manipulator needs to transfer all the objects with the same order from the left shelf to the center shelf. It can also use as extra free area the top shelf.

a connection two arrangements. This allows the efficient use of such methods in the context of higher-level search procedures similar to `Bi-RRT` [80], where two tree data structures originating at the start and goal arrangements are built.

6.1 Discovering the Constraint Graph

Solving a rearrangement challenge means finding the combination of paths that will move the objects to their final poses without colliding with the other objects. However, finding these paths requires searching over all possible orders of moving the objects one after the other. This is how the original `mRS` and the extension `plRS` achieved via backtracking search. This chapter proposes fixing the paths of the objects to the "minimum constraint removal" (`MCR`) paths [47] for constructing the "constraint graph". An `MCR` path minimizes the number of collisions with other objects. The benefit of such paths is that they introduce the minimum number of constraints in the "constraint graph". While the `MCR` problem itself is hard, reasonable and efficient approximations are available [47, 70, 67].

Definition 18 (Constraint graph)**.** *The nodes of the constraint graph correspond to objects $o \in \mathcal{O}$ and an edge $(o_i \rightarrow o_j)$ expresses the constraint that o_j needs to be moved before o_i in the execution sequence. In particular, constraints are defined in the following way:*

- *If object o_i's initial pose $\alpha_I[o_i]$ collides with the* `MCR` *path $\Pi[o_j]$ for object o_j, then o_i has to move first, i.e., $o_j \rightarrow o_i$.*

- *If object o_i's final pose $\alpha_F[o_i]$ collides with the* `MCR` *path $\Pi[o_j]$ for object o_j, then o_j has to move first, i.e., $o_i \rightarrow o_j$*

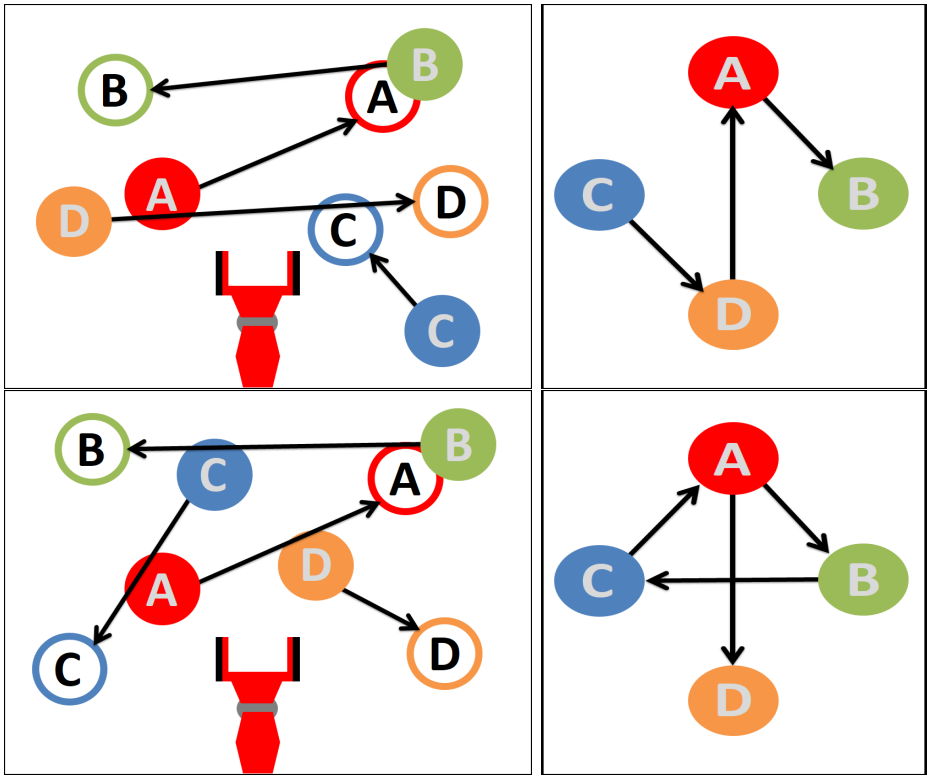

Figure 6.2: (left) An example of two arrangements of four objects (Initial: filled disk, final: empty circle). (right) The constraint graph generated by these examples given the MCR paths. Top case: A monotone solution has been found. Bottom case: A cycle arises.

If the "constraint graph" has no cycles given the `MCR` transfer paths, it means that the problem is monotone and the order with which the objects can be moved, without collisions, is given by a topological sorting on the "constraint graph" (Fig.6.2(top)). If the graph contains cyclical dependencies (Fig.6.2(bottom)) then the problem is non-monotone and the `fmRS` will not be able to give a solution for this instance. The "fast non-monotone Rearrangement Solver" (`fplRS`) will try to decouple the objects that causing cyclical dependencies, by moving some of these objects to intermediate poses.

6.2 Fast Approximate Rearrangement Primitives

This section presents two rearrangement 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. 6.1. 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 [40, 106]. Exhaustively searching for all possible orders can result in long solution times and does not scale well with the number of objects in the scene. There is a way, however, to approximate this process, while avoiding backtracking search. This section presents approximate versions for both the monotone and the non-monotone primitives.

6.2.1 Fast, Approximate Monotone Rearrangement Primitive (`fmRS`)

Exhaustively searching for all possible orders can result sometimes in long solution times and does not scale well as the number of objects increases in the scene. The proposed method, "fast monotone Rearrangement Solver" (fmRS), avoids backtracking search given the following observation: if the paths for transferring the objects are fixed, then one can easily compute the sequence of moving objects without collisions - if one exists [115]. In particular, the sequence is the output of topological sorting on a "constraint graph" which is a directed graph that defines the monotone execution sequence of moving objects $\mathcal{O}$ so as to solve a rearrangement problem.

Algorithm 12 shows the new rearrangement primitive that avoids the backtracking search while solves many monotone rearrangement problems. As mentioned above this is an approximate method that if it can find a solution to a monotone problem then it will find it faster than the method using backtracking search (`mRS`). Moreover, the new method will be able to return failure of finding a solution faster than `mRS`.

First, for each pair of initial $\alpha_I[o]$ and final $\alpha_F[o]$ poses for an object $o \in \mathcal{O}$, the new faster approach computes the "minimum constraint removal" (`MCR`) path [47, 70] (line 1,2). The corresponding routine computes the path with the minimum set of constraints that need to be removed from the problem to find a good quality solution,

Algorithm 12: `fmRS`$(\mathcal{W}, \mathcal{R}, \mathcal{O}, q, \alpha_I, \alpha_F)$

```
1  for each o ∈ O do
2      Π[o] ← MCR_PICK_AND_PLACE(o, q, α_I[o], α_F[o], α_I[O \ o] ∪ α_F[O \ o]);
3  G_c ← (V ← {O}, E ← {∅});
4  for each o_i ∈ V do
5      for each o_j ∈ V \ o_i do
6          if (α_I[o_i] ∈ Π[o_j]) then
7              E ← E ∪ {o_j → o_i};
8          if (α_F[o_i] ∈ Π[o_j]) then
9              E ← E ∪ {o_i → o_j};
10 if (G_c is DAG) then
11     L ← TOPOLOGICAL_SORT(G_c);
12     return ⨁_{o∈L} Π[o];
13 return ∅;
```

for the manipulator to grasp the object from its initial position $\alpha_I[o]$ and transfer it to its final position $\alpha_F[o]$. When computing such a path for an object $o \in \mathcal{O}$, the constraints correspond both to the initial and final placements of all other objects in $\mathcal{O} \setminus \{o\}$. Given such paths for all the objects, it is then possible to compute a "constraint graph" [115] (lines 3-9), as explained above.

If there is no cycle in the constraint graph (Fig.6.2 top), then it is a Directed Acyclic Graph (DAG), (line 10) and it is possible to compute an ordering, using topological sort (line 11). For that ordering, the committed `MCR` paths can be used to solve the monotone problem without the need for search (line 12). If it is not a DAG (Fig.6.2 bottom), then it is not possible to solve the problem with `MCR` paths, by moving the objects at most once.

6.2.2 A Faster Non-Monotone Rearrangement Primitive (`fplRS`)

The idea in this chapter is to explore the space of arrangement using a higher level task planner. In order to achieve that a fast primitive for connecting the nodes is needed. Speed is not the only criterion though, the primitive needs to successfully connect two arrangements most of the times. Although the previously described algorithm can find a solution, while avoiding the backtracking search, it still moves the objects

monotonically. As it is mentioned before, the monotone primitive frequently cannot find a solution even for simple tabletop examples (fig.5.3).

However, there is a way to approximate the non-monotone solution using the same idea of constructing a "constraint graph". The main difference with the monotone primitive is that the non-monotone algorithm will not give up if there is a cycle in the "constraint graph". Instead, it will try to decouple and move some of the objects to intermediate positions, until the cycle is resolved and the graph is a Directed Acyclic Graph. The Algorithm 13 describes the methodology that has been followed to solve this problem.

Algorithm 13: $\texttt{fplRS}(\mathcal{W}, \mathcal{R}, \mathcal{O}, q, \alpha_I, \alpha_F)$

```
1  {Π, G_c, Π_MCR} ← fmRS(W, R, O, q, α_I, α_F);
2  if (G_c is not DAG) then
3      G_SC ← STRONG_COMPONENTS(G_c);
4      L ← TOPOLOGICAL_SORT(G_SC);
5      α_C ← α_I;
6      for each w ∈ L do
7          if (|w| == 1) then
8              π' ← MCR_PICK_AND_PLACE(w.O_w, q, α_C[w.O_w], α_F[w.O_w], α_C[O ∖ w.O_w]);
9          else
10             α'_F[O ∖ w.O_w] ← α_C[O ∖ w.O_w];
11             α'_F[w.O_w] ← α_F[w.O_w];
12             π' ← SOLVE_CYCLE(O, G_c, ∅, q, w, α_C, α'_F, Π_MCR);
13         if (π' == ∅) then
14             return ∅;
15         Π ← Π ⊕ π';
16         α_C[w.O_w] ← α_F[w.O_w];
17 return Π;
```

First, the algorithm will check if the problem is monotone, where a call to fast monotone solver will be enough. The version of the monotone method that it is used here will also return the "constraint graph" and the set of the minimum constraint paths that are computed for all the objects (line 1). If the monotone method could not find a solution, because the graph is not a DAG, the new non-monotone method will try to resolve the conflicts between the objects that are forming a cycle of dependencies

in the "constraint graph". Intermediate poses will be selected for the objects that are forming the cycle of dependencies until the cycle is resolved (line 2-16). In case that the graph is a DAG, but the `fmRS` could not solve the problem, then the fast non-monotone method will not be able to find a path either. This is because both of these rearrangement primitives are fast approximate methods that will not guarantee to find a solution if one exists.

Definition 19 (single node)**.** *When a strong component in the constraint graph contains an individual object,* $|w| = 1$.

Definition 20 (coupled node)**.** *A cycle in the "constraint graph" will form a strong component. A coupled node will contain all the objects involved in the cycle of dependencies,* $|w| > 1$.

The function `STRONG_COMPONENTS` creates a new graph, $\mathcal{G}_{\mathcal{SC}}$, where each node will be either a "single node" with an individual object or a "coupled node" which will contain more than one object (fig.6.3 right) (line 3). Using the topological sort function a new order is created, where the algorithm will use to expand the nodes (line 4). If the new node w is a single node (line 7), a new `MCR` path is computed for the corresponding object. This path respects the current poses of all objects (lines 8). Given that it is time for this object to move and that is in a single node, it means that the path to its final pose π already exists in the $\Pi_{\texttt{MCR}}$, returned by `fmRS`. However, this path had to respect both the initial and the final poses of all the objects. The current path is computed with respect the current pose of each object.

To compute paths for the objects on a "coupled node", the method first sets as the final pose of all objects not belonging to the "coupled node" their current pose (line 10). For all the objects in the "coupled node", the final pose remains the same (line 11). For the objects inside the "coupled node", the function `SOLVE_CYCLE` is used to try to resolve the cycle (line 12). The paths for the objects in the "coupled node" are now less constrained than those computed by the original `fmRS` (where paths are constrained both by the initial and final object poses) because the algorithm considers as obstacles only the current positions of those objects not in the "coupled node".

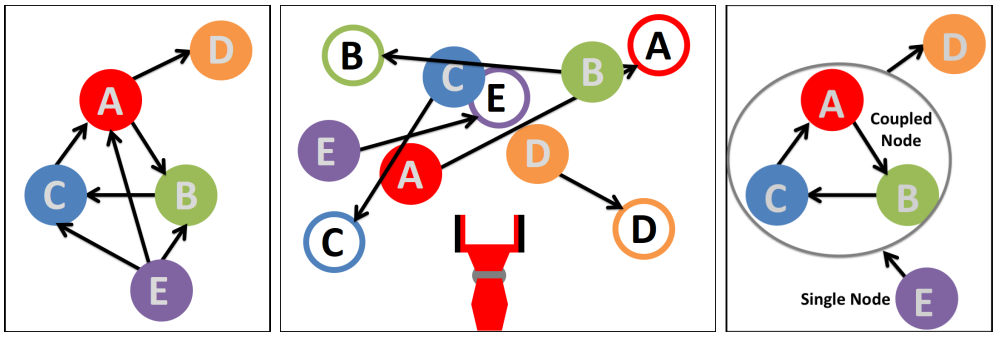

Figure 6.3: Left: A constraint graph with 5 objects. Objects A, B and C are forming a cyclical dependency, as a result this graph is not a DAG. Center: The non-monotone case that is causing this constraint graph. Right: A new constraint graph where all the objects that forming a cycle are included in a "set" node.

If there is a path to move the objects inside the "coupled node", then the method will aggregate the final path and will move the objects, in this "coupled node", to their final pose (lines 15-16). Otherwise, the algorithm could not detect a path to solve the cycle, thus will not be able to solve the problem either and it will return $\emptyset$ (lines 13-14).

In order to deal with non-monotone cases, the method has to resolve the cycles in the "constraint graph". Algorithm 15, that is described later in this section, tries to find a way to resolve such cycles and move all the objects inside a cycle to their final pose. In order to achieve that it will have to decouple the objects by moving the object with the most constraints to an intermediate pose, and it will continue until the cycle is resolved. Eventually, the objects that are placed on an intermediate pose will be moved to their final pose.

Algorithm 14: DECOUPLE$(\mathcal{O}, \mathcal{O}_P, o, q, w, \alpha_C, \alpha_F, \Pi_{\text{MCR}})$

```
1 O_R ← w \ {o ∪ O_P};
2 P ← AVAILABLE_POSES(α_C[O] ∪ α_F[O_R] ∪ Π_MCR[O_R]);
3 for (each p ∈ P) do
4     if (PICK_AND_PLACE(o, q, α_C[o], p, α_C) ≠ ∅) then
5         if (PICK_AND_PLACE(o, q, p, α_F[o], α_F[O_R] ∪ α_C[O \ O_R]) ≠ ∅) then
6             return p;
7 return ∅;
```

The DECOUPLE function (Alg.14) will detect the intermediate pose for the given object o. $\mathcal{O}_P$ are the objects inside the cycle that are already in an intermediate pose,

while $\mathcal{O}_R$ represents the objects inside the cycle that will be moved after the object o, future objects (line 1). The set of available poses for the object o are all the poses that do not interfere with:

- The current poses of all the objects, $\alpha_C[\mathcal{O}]$,
- The final poses from all the future objects inside the cycle, $\alpha_F[\mathcal{O}_R]$, and
- The computed paths for moving the future objects to their final pose, $\Pi_{\texttt{MCR}}[\mathcal{O}_R]$.

Algorithm 15: SOLVE_CYCLE$(\mathcal{O}, \mathcal{G}_c, \mathcal{O}_P, q, w, \alpha_C, \alpha_F, \Pi_{\texttt{MCR}})$

```
1  Π ← ∅;
2  L ← CONSTRAINT_ORDER(w);
3  for each o ∈ L do
4      p_I ← DECOUPLE(O, O_P, o, q, w \ o, α_C, α_F, Π_MCR);
5      if (p_I ≠ ∅) then
6          p_prev ← α_C[o];
7          π' ← PICK_AND_PLACE(o, q, α_C[o], p_I, α_C[O \ o]);
8          α_C[o] ← p_I;
9          π ← ∅;
10         if (w \ o is DAG) then
11             L ← TOPOLOGICAL_SORT(w);
12             π ← ⊕_{o∈L} Π_MCR[o];
13         else
14             π ← SOLVE_CYCLE(O, G_c, O_P ∪ o, q, w \ o, α_C, α_F, Π_MCR);
15         if (π ≠ ∅) then
16             π'' ← PICK_AND_PLACE(o, q, p_I, α_F[o], α_C[O \ o]);
17             if (π'' ≠ ∅) then
18                 α_C[o] ← α_F[o];
19                 return π' ⊕ π ⊕ π'';
20         α_C[o] ← p_prev;
21 return ∅;
```

Given these constraints, the algorithm will detect the available intermediate poses for the object o to move (line 2). Out of all these poses, the first pose p that the manipulator will be able to:

- Pick the object o from its initial pose, place it to this intermediate pose, while respects the current poses of all the other objects (line 4).

- Pick the object from the intermediate pose and place it to its final pose, while respects the final poses of the future objects ($\alpha_F[\mathcal{O}_R]$), that will be moved after the object o is transferred to its intermediate pose and the current pose of all the other objects ($\alpha_C[\mathcal{O} \setminus \mathcal{O}_R]$) (line 5).

will be the intermediate pose for the blocking object o (line 6). If such a pose does not exist, then the object o cannot be decoupled, given these minimum constraint removal paths (line 7).

As mentioned before the Algorithm 15 recursively tries to resolve a cycle. First of all the algorithm has to detect the object o inside the cycle with the most constraints. The function `CONSTRAINT_ORDER` will return an order of objects where the object with the higher number of constraints will be first (line 2). Following the order of this list and using the `DECOUPLE` function the algorithm will detect an intermediate pose for the selected object (line 4). If the there is no intermediate pose for this specific object the algorithm will check the next object in the list.

If there is an intermediate pose, the object o will be moved there (lines 6-9). If the "constraint graph" is a DAG, after this move, then the algorithm will move all the objects to their final pose (lines 10-12) and finally will move the object o to its final pose (lines 15 - 19). If the "constraint graph" is not a DAG after moving the object o to the intermediate pose, the algorithm recursively will try to solve the new cycle (lines 13-14). If eventually the cycle is resolved and a path is returned with the objects on their final pose, the objects on the intermediate poses will also move to their final pose (lines 15-19).

The algorithm will search over all the possible orders in order to solve the cycle. If all the orders fail the algorithm will return an empty path which means that the algorithm could not find a solution.

6.3 An Incremental Search Approach

Both of these algorithms are fast but approximate methods. This means that it is easy to generate problems where none of these methods will be able to give a solution. For

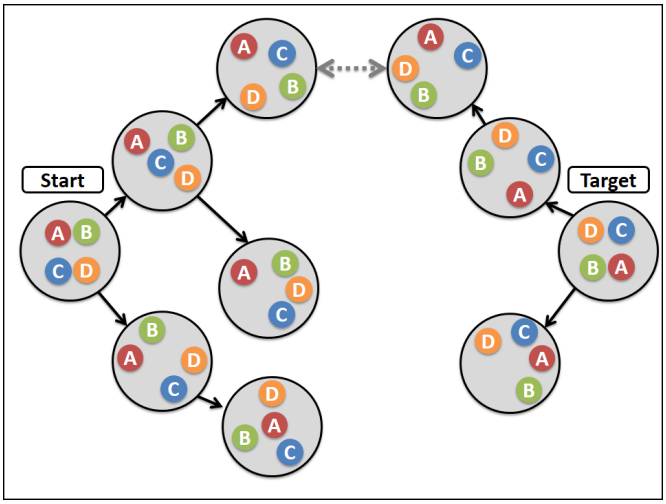

Figure 6.4: A bi-directional tree in the space of object arrangements.

the monotone method it is easy to generate a problem that will make the algorithm to fail, e.g. the initial pose of an object blocks the final pose of a different object and vice versa (fig.5.3). The non-monotone algorithm may also fail to solve a non-monotone case. This is because the methods are using a pre-specified set of paths for all the objects.

An idea in a previous work of the authors [68] was to use rearrangement primitives, like the `fmRS` method, as local planners within higher-level processes that search the space of object arrangements. The benefit of such monotone solvers is that they can provide a way to connect an arrangement to a larger set of neighboring arrangements relative to a pick and place action, which is the typical low-level primitive for manipulation task planning [40, 106].

The high-level search process can be performed in many different ways, e.g., following a general heuristic search approach, as in related work [40], or in a `PRM`-like fashion [68], by building a graph of object rearrangements. In the latter case, edges could be constructed by calling a rearrangement primitive to connect pairs of arrangements. A drawback for a `PRM`-like task planner is that for hard problems, i.e. multiple objects that could reduce the maneuverability of the manipulator and its ability to grasp all the objects, the probability of connection between two arbitrary arrangements using a rearrangement primitive is still low.

It is preferable, if after the approach samples an arrangement, a local path between a pair of arrangements is generated. To achieve this, instead of trying to exactly connect a new sampled arrangement with existing ones in the graph, it is better to extend an existing arrangement towards a new sampled point. This idea is closer to the operation of incremental sampling-based tree planners, such as a Bi-directional RRT ($\texttt{Bi}-\texttt{RRT}$) [80], which is summarized in Alg. 16 for use in rearrangement planning.

Algorithm 16: $\texttt{Bi}-\texttt{RRT}(\mathcal{W}, \mathcal{R}, \mathcal{O}, q, \alpha_I, \alpha_F)$

1 $\mathcal{T}_I \leftarrow \{\mathcal{V}_I \leftarrow \{\alpha_I\}, \mathcal{E}_I \leftarrow \{\emptyset\}\}$;
2 $\mathcal{T}_F \leftarrow \{\mathcal{V}_F \leftarrow \{\alpha_F\}, \mathcal{E}_F \leftarrow \{\emptyset\}\}$;
3 **while** $(\Pi \leftarrow \texttt{FIND\_PATH}(\mathcal{T}_I, \mathcal{T}_F, \alpha_I, \alpha_F)) == \emptyset$ **do**
4 $\quad \alpha_{rand} \leftarrow \texttt{SAMPLE\_STABLE\_ARRANGEMENT}()$;
5 $\quad \alpha_{near} \leftarrow \texttt{NEAR\_STATE}(\mathcal{T}_I, \alpha_{rand})$;
6 $\quad \{\pi, \alpha_{partial}\} \leftarrow \texttt{CONNECT\_STATES}(\mathcal{W}, \mathcal{R}, \mathcal{O}, q, \alpha_{near}, \alpha_{rand})$;
7 $\quad \mathcal{E}_I \leftarrow \mathcal{E}_I \cup \{(\alpha_{near}, \alpha_{partial}), \pi\}$;
8 $\quad \alpha_{near} \leftarrow \texttt{NEAR\_STATE}(\mathcal{T}_F, \alpha_{partial})$;
9 $\quad \{\pi, \alpha\} \leftarrow \texttt{CONNECT\_STATES}(\mathcal{W}, \mathcal{R}, \mathcal{O}, q, \alpha_{near}, \alpha_{partial})$;
10 $\quad \mathcal{E}_F \leftarrow \mathcal{E}_F \cup \{(\alpha_{near}, \alpha), \pi\}$;
11 **return** Π;

The high-level planner builds a bi-directional tree (as in Fig.6.4), where nodes correspond to object arrangements. The two trees start with the initial and final arrangements α_I and α_F correspondingly as their only nodes (line 1-2). Edges correspond to local rearrangement paths that will be computed by a rearrangement primitive, such as `fmRS`. While the problem is not solved (line 3), the method samples new random arrangements α_{rand} (line 4). Then, based on a distance estimate in the space of arrangements, the closest neighboring arrangement α_{near} in the starting tree ($\mathcal{T}_I$) is returned (line 5). In order to compute distances between arrangements, the sum of distances between the placement of objects in the two arrangements is used. A connection is attempted between the neighbor and the random arrangement (lines 6) with a rearrangement primitive search algorithm.

In the spirit of the original $\texttt{Bi}-\texttt{RRT}$ algorithm, it is not necessary to exactly connect α_{near} with α_{rand}. Instead, it is sufficient if the method makes some progress away from α_{near} in the space of arrangements and a new $\alpha_{partial}$ is linked to the tree (line 7). The closest neighboring arrangement α_{near} is discovered in the other tree ($\mathcal{T}_F$) (line 8) and

a similar extension of this tree is also performed (lines 9-10).

By extending a new arrangement from α_{near} and not connect it to α_{rand}, the algorithm is able to frequently generate edges. The first time the two trees connect, the algorithm will return a solution. It is beneficial to consider such a bi-directional solver rather than a single-tree expansion. It may be that the problem is more constrained at the initial arrangement and the tree from the target arrangement will be able to expand easier. A bi-directional tree tends to achieve better and faster coverage of the space.

6.3.1 Partial Solutions

The rearrangement primitives have been defined so that they connect exactly two arrangements. Here, this requirement is relaxed so that these processes can be used as effective extension operations for the high-level $\texttt{Bi}-\texttt{RRT}$ task planner. This means it should be possible to return a best effort partial solution where just some of the objects are moved. This section describes how all the rearrangement primitives could return a partial solution.

It is easy to make `mRS` [68] return a partial solution by remembering the maximum depth the algorithm has managed to achieve during the backtracking search. In each iterative call, the algorithm needs to keep track of the maximum depth and the corresponding sequence. At the end of the search process, the algorithm either succeeds to connect the two arrangements or a partial solution corresponding to the best effort to connect the two arrangements is returned together with the resulting arrangement. However, it is not trivial to return a partial solution from a non-monotone rearrangement primitive. The non-monotone rearrangement primitive, `plRS` [68], is trying to solve the problem by searching through possible ways of moving the objects to intermediate poses, in order to come up with a solution. Thus, there is no a criterion when the search should stop and return a partial solution.

There is also a way to get a partial solution from both the fast primitives, i.e. `fmRS` and `fplRS`. For instance, in Fig. 6.2 (bottom), object D can be moved to its target regardless of the cycle (A, B, C). The proposed `partial_fmRS` starts from sink nodes of the constraint graph, moves the corresponding objects to their target, removes the nodes

from the graph and continues for as long as it is possible to move objects. Furthermore, it tries to decouple cycles whenever possible given that higher priority objects have moved and different MCR paths can be computed. Alg. 17 describes the algorithm, which extends the previous fmRS, follows similar idea with fplRS and operates over the resulting constraint graph $\mathcal{G}_c$.

Algorithm 17: partial_fmRS$(\mathcal{W}, \mathcal{R}, \mathcal{O}, \mathcal{G}_c, q, \alpha_C, \alpha_F)$

```
1  {Π, G_c, Π_MCR} ← fmRS(W, R, O, q, α_I, α_F);
2  if (G_c is not DAG) then
3      G_SC ← STRONG_COMPONENTS(G_c);
4      L ← TOPOLOGICAL_SORT(G_SC);
5      Π ← ∅;
6      for each w ∈ L do
7          if (|w| == 1) then
8              π' ←
                MCR_PICK_AND_PLACE(w.O_w, q, α_C[w.O_w], α_F[w.O_w], α_C[O ∖ w.O_w]);
9          else
10             α'_F[O ∖ w.O_w] ← α_C[O ∖ w.O_w];
11             α'_F[w.O_w] ← α_F[w.O_w];
12             π' ← fmRS(W, R, O, q, α_C, α'_F);
13         if (π' ≠ ∅) then
14             Π ← Π ⊕ π';
15             α_C[w.O_w] ← α_F[w.O_w];
16 return Π;
```

First, the algorithm will try to solve the problem using fmRS (line 1). If the "constraint graph" is not DAG this algorithm will not be able to solve the problem, but it is allowed to return a partial solution (line 2). This allows turning $\mathcal{G}_c$ into a directed acyclic graph (DAG), where each node will be a strongly connected component. These nodes may include many interdependent objects (line 3), e.g., objects (A, B, C) in Fig. 6.2 (bottom) will be in a "coupled node". It is then possible to apply topological sorting over the resulting DAG (line 4). The final path is initialized to be empty (line 5). Following similar ideas as fplRS algorithm, move the objects inside each node w of the graph $\mathcal{G}_{\mathcal{SC}}$ based on the topological order $\mathcal{L}$ (line 6). Each of these nodes can be one of the following:

- A "single node" (line 7), where a new MCR path is computed for the corresponding

object (line 8). This path respects the current poses of all objects. Objects with higher priority in the `DAG`'s topological order can be in either their initial or final poses, depending on whether the partial solution managed to transfer them or not. All the objects with lower priority must be at their initial poses.

- A "coupled node" (line 9), where a new set of paths for each object in this node has to be computed. The method first sets as the final pose of all objects not belonging to the component their current pose (line 10). For all the objects in the component, the final pose remains the same (line 11). Given this new final arrangement, the algorithm will call again `fmRS` to try to solve the cycle. The objects inside the "coupled node" are less constrained to move than the original `fmRS` (where paths are constrained both by the initial and final object poses), because the algorithm considers as obstacles only the current positions of those objects not in the "coupled node".

In either case, if a path is found for the current node (line 13), the path is appended to the current solution (line 14) and the objects are transferred to their final pose (line 15). Otherwise, the objects corresponding to w remain at their current pose. If there is at least one object moved to its final pose, then the method returns the corresponding partial solution.

6.4 Evaluation

Experimental Setup: The methods have been tested in 2 simulated workspaces with a model of a Baxter arm: "grid@tabletop" (Fig. 6.5 (left)), "grid@shelf" (Fig. 6.5 (right)). The "grid@tabletop" benchmark places 4, 10 and 16 objects on a tabletop. The "grid@shelf" challenge has 2, 6 and 10 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. Eleven different methods are tested, in two different test cases. First, single runs of the rearrangement primitives are tested: (a) `mRS` [68], (b) `plRS` [68], (c) `fmRS` [69], (d)`fplRS` (Top figures of Fig.6.6-Fig.6.11) then all the primitives replace the function `CONNECT` of the high

level REARRANGE_PRM (Middle figures of Fig.6.6 - Fig.6.11). Finally, the rearrangement primitives that can return partial solutions (a) mRS, (b) fmRS, (c) PICK_AND_PLACE replace the function CONNECT of an incremental search high level planner $Bi-RRT$ (Bottom figures of Fig.6.6 - Fig.6.11). The time limit provided to the methods was 10 mins. 10 experiments were performed for each combination of method and environment. All the results are collected over the successful runs.

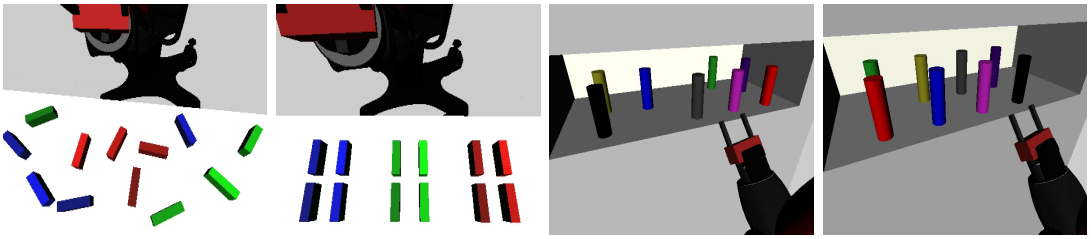

Figure 6.5: (top) The "grid@tabletop" problem. An initial and the final setup. (bottom) The "grid@shelf" problem. An initial and the final setup.

Results for the "grid@tabletop challenge": Figure 6.6 provides the success ratio of all the methods for the "grid@tabletop" challenge.In this example, there are many partial overlaps between final poses and intermediate or initial poses. This affects the results of the monotone methods, where they could not solve any problem with 16 objects. In addition, affects the results for the fplRS. Although, it is an approximate method found a solution to all the problems except one case with 14 and one case with 16 objects on the table. This is because the minimum constraint paths are generated only ones and the algorithm does not search over different paths or orders. In figure 6.6 middle figure shows the success ratio when the REARRANGE_PRM method is using the rearrangement primitives. The REARRANGE_PRM method found solutions for all the problems up to sixteen objects, however, the success ratio of the method while using the monotone primitive (mRS) drops quickly as the number of objects increases. As a result, found solutions only for 20% of the problems with sixteen objects. The bottom figure (fig. 6.6) provides the success of the incremental search algorithm ($Bi-RRT$) while using as CONNECT function a rearrangement primitive. The success ratio drops only when the algorithm is using the monotone primitive (mRS). The rest of the rearrangement primitives were able to solve all the instances.

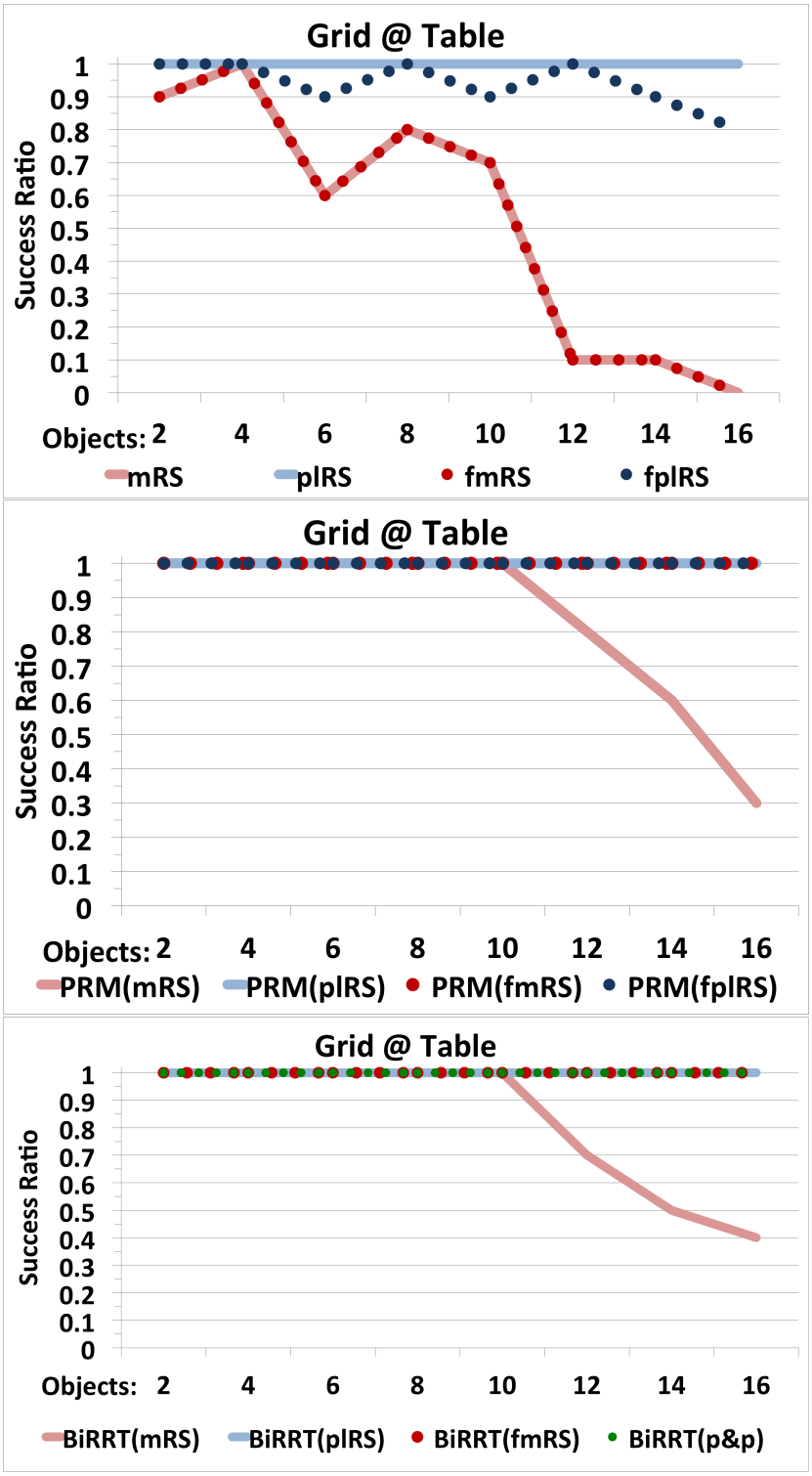

Figure 6.6: The success ratio for each approach out of 10 runs for the "grid@tabletop challenge".

Figure 6.7 provides the computation time for all the methods for the "grid@tabletop" challenge, while figure 6.8 provides the path quality of the methods. The top figure shows the computation time for the rearrangement primitives. The `plRS` method was successful across all the instances but needs to search over a large number of possible orders to find one that works. On the other hand, `fplRS` could return solutions faster, when it was possible to find a solution. In addition, figure 6.8 (top) shows that `fplRS` solutions are efficient in terms of path quality than `plRS`. The middle figure at fig.6.7 shows the computation time for the `REARRANGE_PRM` method. The fast monotone primitive `fmRS` could search the rearrangement space more, given that it could return a solution faster. As a result, it could solve all the problems up to 16 objects on the table. The non-monotone solvers run slightly slower in the smaller examples. This is because they spend a little bit more effort to find intermediate poses for a non-monotone solution instead of searching if there is an easy monotone solution. For the same reason, the `REARRANGE_PRM` using the monotone primitive `mRS` is faster than the `REARRANGE_PRM` that uses the non-monotone primitive `plRS`. This is not the same when the `REARRANGE_PRM` uses the fast rearrangement primitives. Using the `fmRS` primitive is the slowest option because the algorithm will fail to connect more arrangements, as a result the `REARRANGE_PRM` will need to search more for possible solutions in the rearrangement space. The best option for `REARRANGE_PRM` is to use the `fplRS` primitive, which could detect the solution of a non-monotone problem faster and more efficiently in terms of solution quality than `plRS`. The bottom figure (Fig.6.7) shows the results from the $\texttt{Bi} - \texttt{RRT}$ method. Using a simple pick and place primitive gave solutions to all the problems, however, the fast rearrangement primitive (`fmRS`) could return a solution faster and more efficiently in terms of how many objects had to be moved. These results are more obvious in larger-scale examples, where the $\texttt{Bi} - \texttt{RRT}$ using `fmRS` manages to solve all the hard problems in less than 2.7 seconds on average.

Results for the "grid@shelf challenge": Figures 6.9, 6.10 and 6.11 provides the results of all the methods for the "grid@shelf" challenge. Here the arm has reduced reachability and is not able to use overhand grasps. As the number of objects increases, the monotone solvers' success ratio goes down quickly. The non-monotone primitives

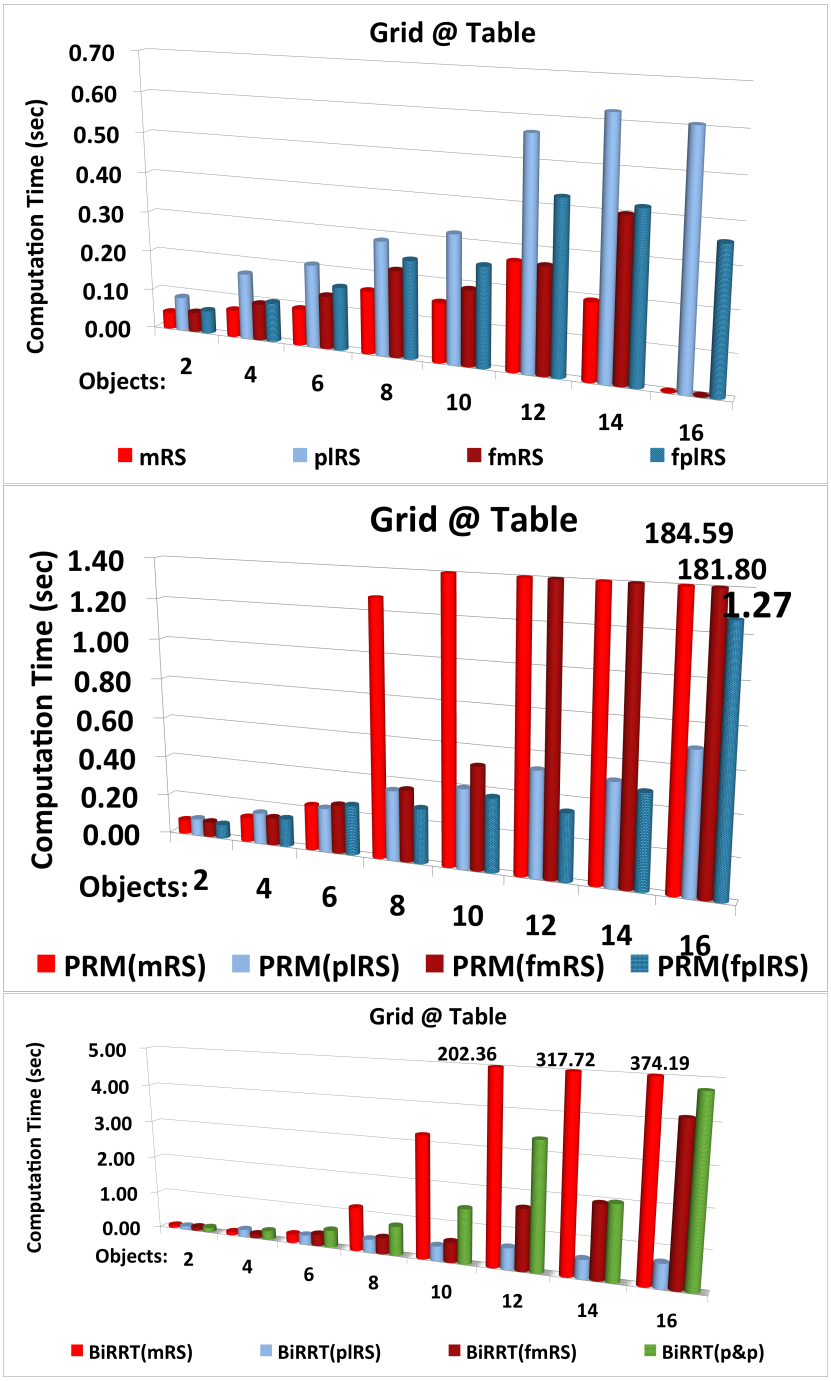

Figure 6.7: The computation time for each approach out of 10 runs for the "grid@tabletop challenge".

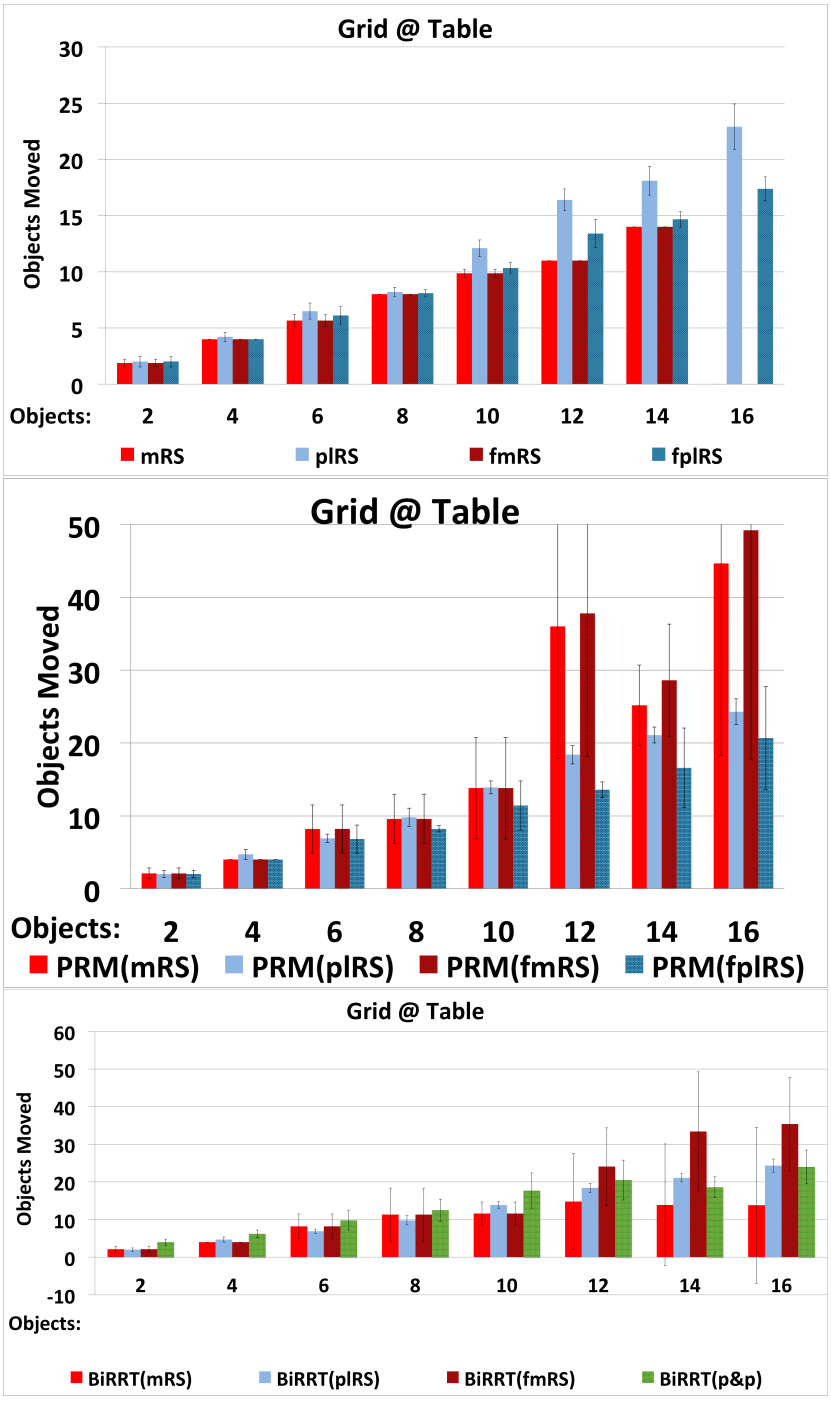

Figure 6.8: The number of transfers the manipulator had to execute in order to solve the problem, for each approach out of 10 runs for the "grid@tabletop challenge".

have better success ratio despite the difficulty of the challenge and the limited reachability. However, the ratio and the running time are not that good especially with 10 objects in the shelf. Moreover, the `plRS` has to search a lot before return that cannot find a solution.

Using the non-monotone primitives within `REARRANGE_PRM` results in even better success rate (figure 6.9 middle). When the algorithm tries to find a solution for the examples with 10 objects in the self, then only the version with the `fplRS` manages to solve all the problems. The simple `plRS` algorithm found solution only for half of the cases, and run out of time for the rest of them, while the monotone primitives failed to report a solution. In regards to running time, the `fplRS` is an order of magnitude faster than the common `plRS`. The reason for this behavior is, that it takes more time for `plRS` to find a solution and connect two arrangements in the arrangement space. The `REARRANGE_PRM` that uses as primitive the `fplRS` algorithm does not spend a lot of time to connect two arrangement, as a result, can search the space faster than the method that uses as primitive the `plRS` method. In addition, the suggested method returns better paths with less number of transfers. The higher number of moves at the "grid@shelf challenge" with 10 objects is because this method manages to solve harder problems that require more transfers in order to get a solution.

The bottom picture at the figures 6.9, 6.10 and 6.11 shows the results from the $\texttt{Bi}-\texttt{RRT}$ method while using the monotone rearrangement primitives, i.e. `mRS` and `fmRS`, the non-monotone primitive, `plRS` and a simple pick and place primitive. The combination of $\texttt{Bi}-\texttt{RRT}$ with `fmRS` could solve up to 12 objects in the shelf. The other method that could solve 20% of these hard problems is the combination of $\texttt{Bi}-\texttt{RRT}$ with `PICK_AND_PLACE`. In addition, while `fmRS` is in use, the $\texttt{Bi}-\texttt{RRT}$ method could find a solution faster with better path quality. At the bottom graph in figure 6.10, it appears that $\texttt{Bi}-\texttt{RRT}$ using `fmRS` is slower than $\texttt{Bi}-\texttt{RRT}$ with simple pick and place, but this is because the first combination could solve harder instances that need more time to search the arrangement space before return a solution.

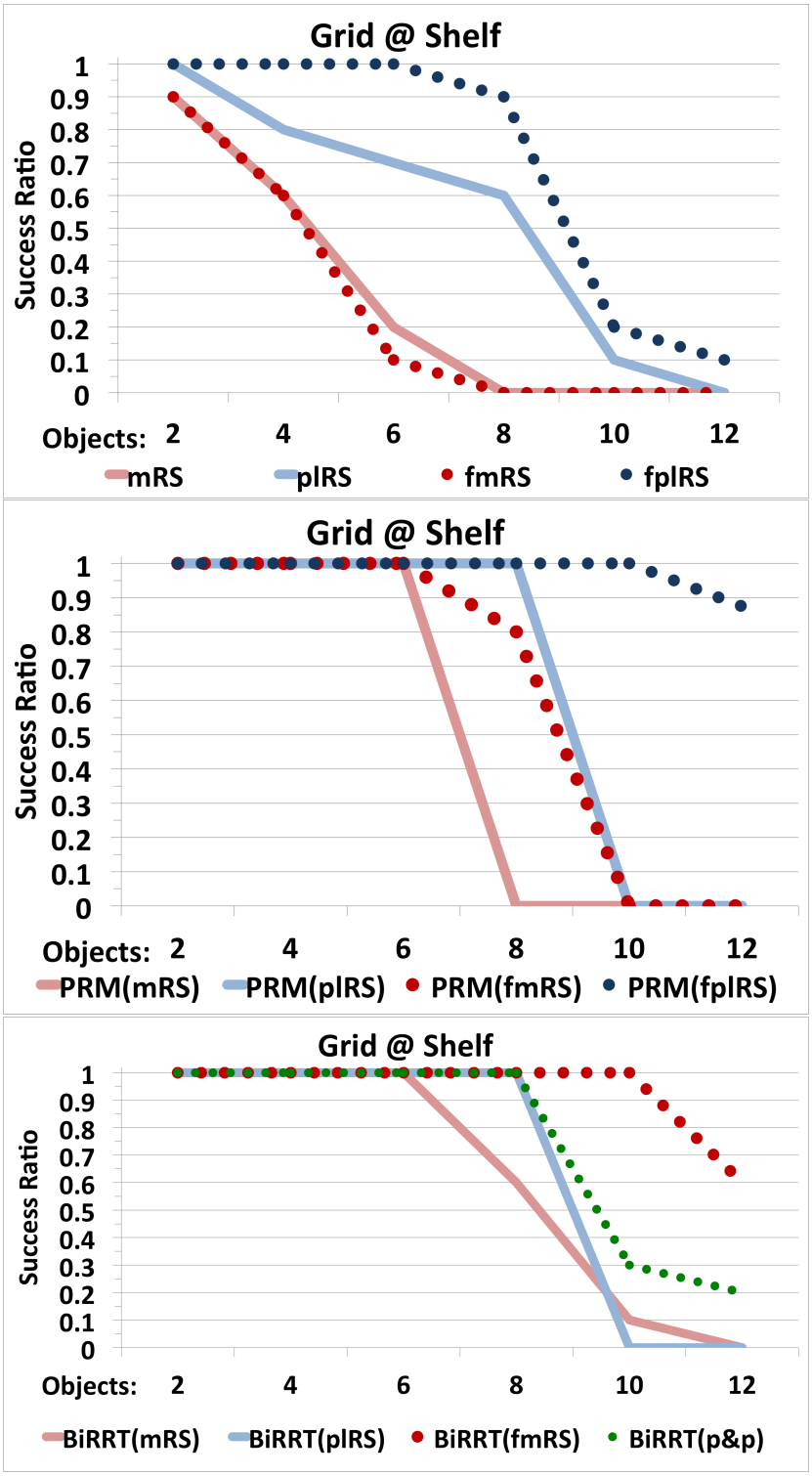

Figure 6.9: The success ratio for each approach out of 10 runs for the "grid@shelf challenge".

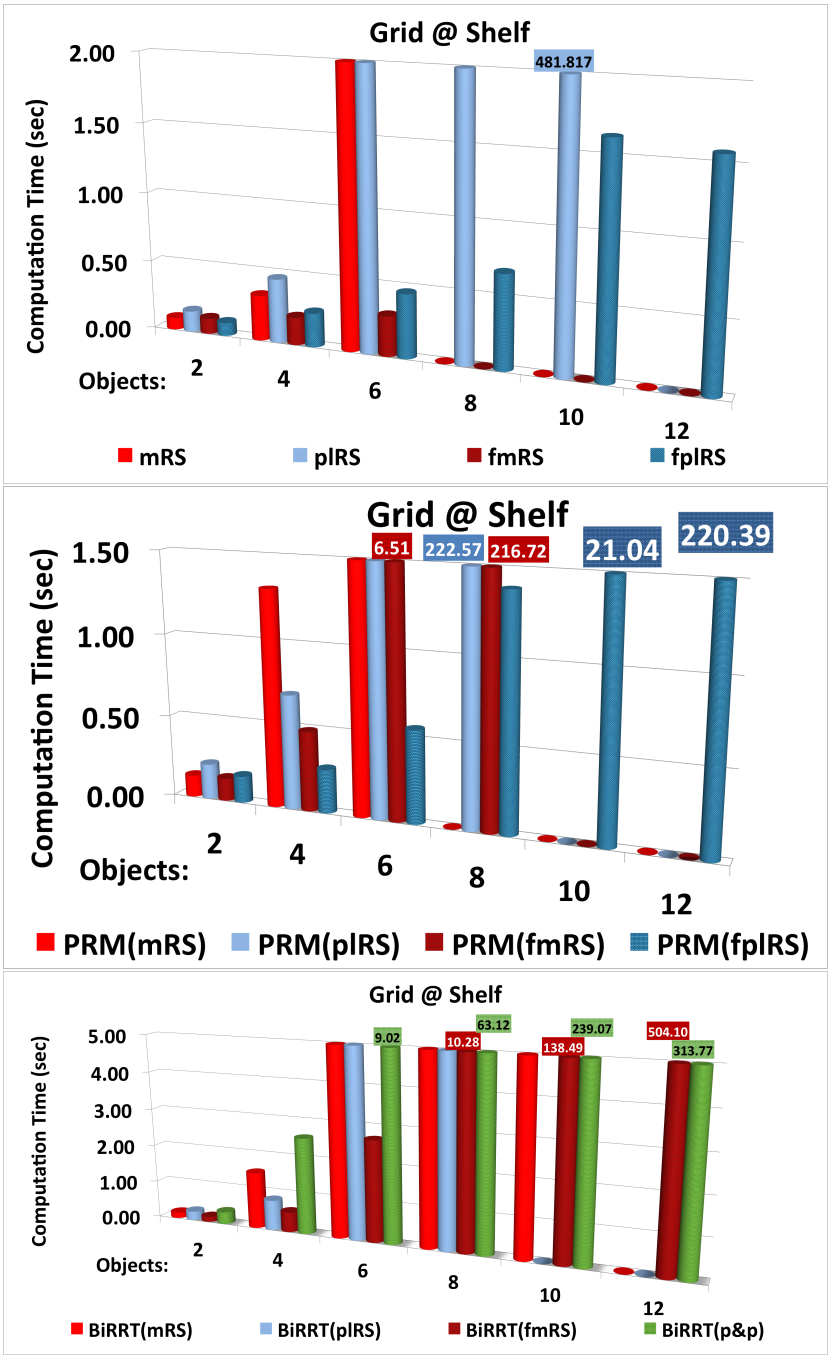

Figure 6.10: The computation time for each approach out of 10 runs for the "grid@shelf challenge".

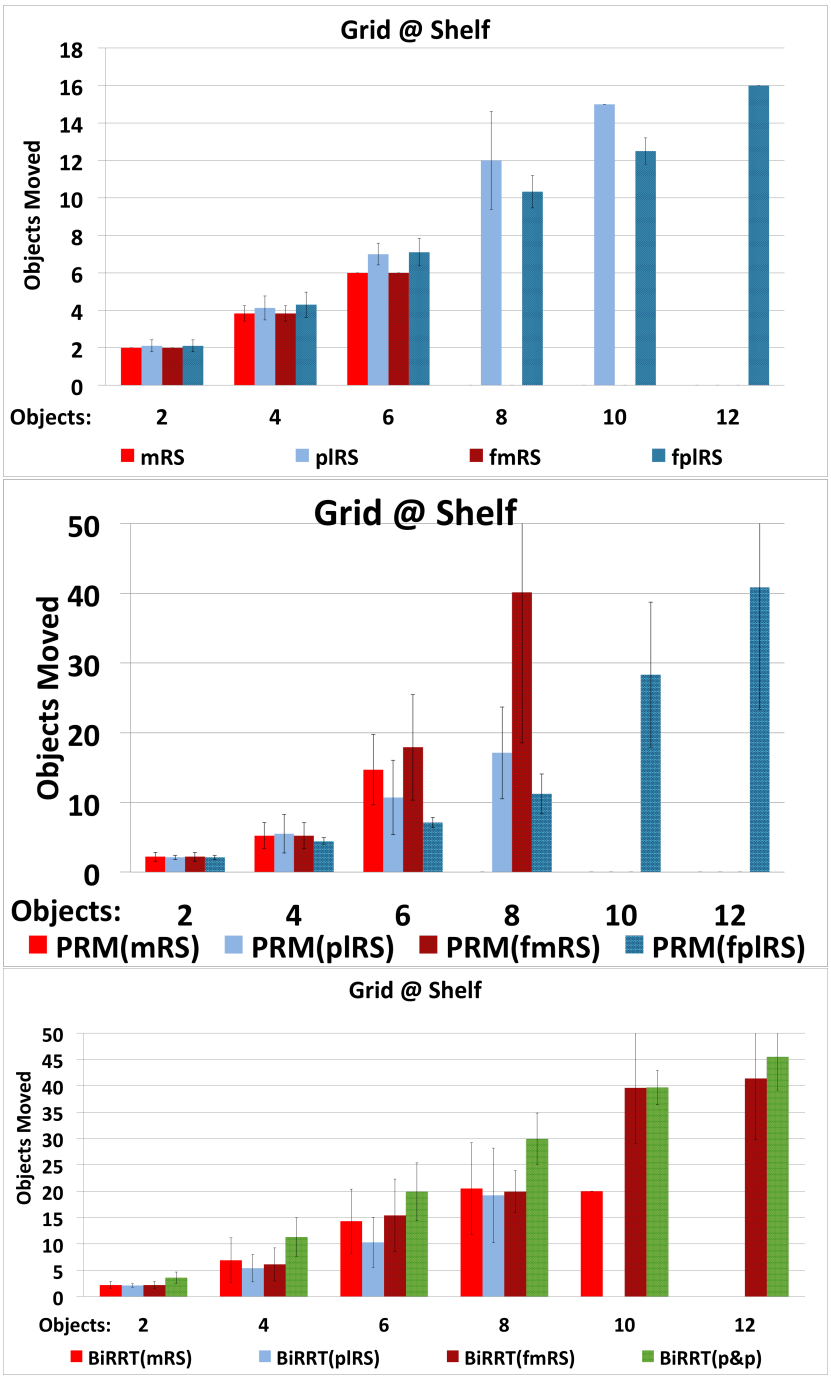

Figure 6.11: The number of transfers the manipulator had to execute in order to solve the problem for each approach out of 10 runs for the "grid@shelf challenge".

6.5 Discussion

This chapter proposes a primitive for rearrangement, which provides improved connectivity among object arrangements relatively to pick-and-place actions. The method extends an existing technique [110] 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 `plRS`'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 arrangement 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 [25, 34] and mobile manipulation [40]. 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 [9]. An alternative but important direction is to adapt complete but efficient multi-robot planning algorithms in the context of manipulation [118, 105]. Future efforts should also focus on the computation of robust rearrangement trajectories given actuation and observation noise.

Chapter 7

Computationally Efficient Implementation

Object rearrangement is a challenging problem as it involves combinatorially large, continuous $\mathcal{C}$-spaces with multiple movable bodies and complex kinematic constraints. To significantly speed up online query resolution, it is possible to pre-built the transfer and transit roadmaps for the arm given the static geometry. These graphs can be used during the online query resolution in order to compute collisions with other objects. The high-level task planner will work as a traditional probabilistic roadmap, where a correct implementation of sampling the nodes and finding the closest neighbors will result in a more efficient search process. While important in many applications, the Minimum Constraint Removal path (`MCR`) is harder than searching for the shortest path. In this work an approximate bounded path length approach of `MCR` is used [67] that is computationally more efficient and returns paths with number of constraints close to the exact approach [47, 46].

7.1 Implementation of the Manipulation Planning Framework

In this work, a core manipulation framework has been implemented and is used from all the algorithms (figure 7.1). This manipulation framework knows how to move the manipulator in the space in order to resolve tasks. The coordinator of the manipulation framework is the manipulation task planner. Under the manipulation task planner, there is a set of motion planners that can detect valid paths for the manipulator whenever the task planner has a request. One motion planner is used for the transit paths, where the motion planner can find collision-free paths for the manipulator in $\mathbb{X}^0$, where the manipulator does not hold anything, and k motion planners for the different transfer modes, one for each different type of objects. The motion planners for the transfer

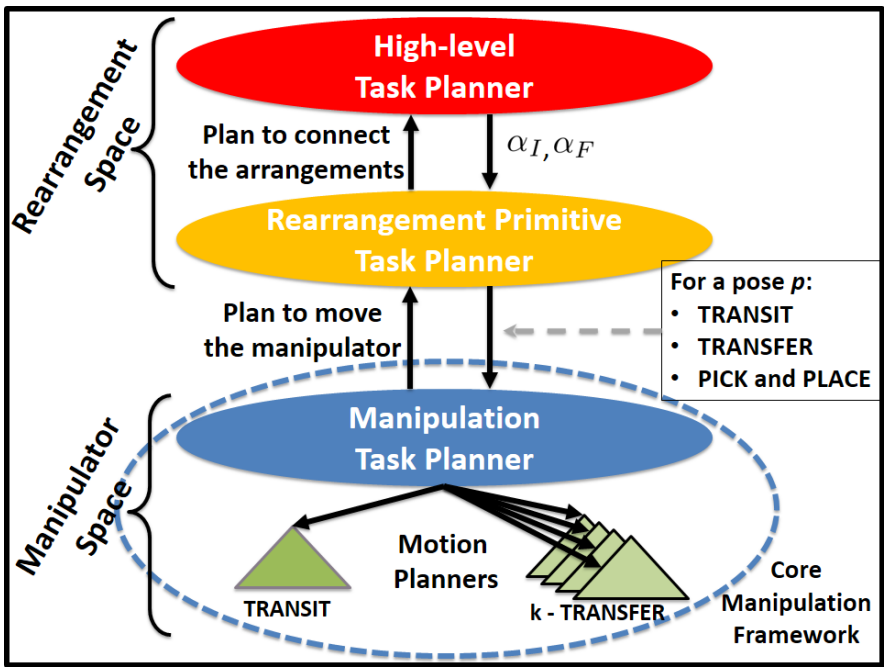

Figure 7.1: Implementation of manipulation planning framework. The core of this framework consists from the manipulator task planner that controls the transit and the k transfer motion planners. For the experiments, the core manipulation framework is always the same and the rearrangement primitive is the proposed monotone/non-monotone algorithms. The high-level task will use the rearrangement primitives in order to solve the problem. In this work `REARRANGE_PRM` is taking the role of the high-level task planner.

modes can find collision-free paths inside $\mathbb{X}^i$ for each object o_i that the manipulator is grasping.

The core manipulation framework is capable of answering queries about the different motion planning primitives, `TRANSIT`, `TRANSFER`, `PICK_AND_PLACE`. Given the motion planning primitive, the manipulation task planner will use the correct motion planners in order to compose a plan for the manipulator. The returned path will be a collision-free path that solves the task that the planner above requested.

In this work, above the core manipulation framework lives the rearrangement primitive that represents the proposed algorithms, `mRS`, `plRS`. The rearrangement primitive works in the rearrangement space and will ask from the core manipulation framework queries such as transit the arm, transfer this object to this position p or even pick the object from the pose p_0 and place it to the position p_1. The rearrangement primitive has as input an initial arrangement and a final arrangement for the objects, were based

on the strategy of each individual algorithm it will move the objects from the initial to the final arrangement.

The high-level task planner can be any search algorithm, over the arrangement space. A state point in the arrangement space corresponds to an arrangement of the objects in the workspace. The high-level task planner will select which two arrangement would like to connect and it will request from the rearrangement primitive to find a path that brings the objects from the initial to the final arrangement. In this work, the sampling based algorithm `REARRANGE_PRM`, described in section 5.2.4, is used as the high-level.

As mentioned before all the proposed algorithms in this work will be using the core manipulation framework in order to move the manipulator to an appropriate location. The algorithm on-line will be asking for solutions to motion planning primitives. 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.

7.1.1 Preprocessing

The first step is the sampling of a discrete set of useful stable poses $\hat{\mathcal{P}}$ 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 [58]. 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 transfer roadmaps are generated, plus one transit roadmap (fig. 7.1) 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 on-line phase. Objects are placed in the poses $\hat{\mathcal{P}}$ and then for each edge of the roadmap, the set of object poses that lead into collisions is discovered and stored. This type of precomputation is similar to the "conditional reachability graph" data structure [40].

7.1.2 Query Resolution

During the on-line operation of the methods, every time that a `TRANSIT`, a `TRANSFER` or a `PICK_AND_PLACE` primitive is executed, a multi-goal A^* algorithm is executed on the corresponding precomputed roadmaps [27, 31]. The multiple goals for the A^* 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 is tested against the poses in the current arrangement α_C. If there is any overlap, then the edge is in collision. In the case of the non-monotone primitives, 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 off-line phase, then no actual collision checks need to be performed during on-line 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 probabilistic 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 on-line.

7.2 Implementation Aspects of the Search Process

Similar to the traditional probabilistic roadmap method, the performance of the `REARRANGE_PRM` algorithm depends on the implementation of the following primitives:

7.2.1 Sampling nodes

The top-level algorithm splits the process of sampling nodes into two steps. First, a stable collision-free arrangement α_{rand} is found for the objects and then a collision-free grasping arm configuration $q(\alpha_{rand}[o])$ for one of the objects o is identified for the arm.

The simplest way to select object poses in the arrangement α_{rand} is via uniform sampling. A biased sampling process can also be used in order to accelerate the solution to a specific challenge. For instance, object poses either from the initial a_I or the goal a_F arrangement can be selected with a certain frequency in order to increase the probability of connecting arrangements with the initial or goal states. The accompanying implementation uses this biased sampling process with a 5% frequency of selecting the pose of an object as it appears in the initial or the goal arrangement.

Given an arrangement α_{rand}, there could be multiple arm configurations of the form $q(\alpha_{rand}[o])$, which achieve to grasp an object $o \in \mathcal{O}$, for a redundant manipulator. Inverse kinematics solvers can be used in order to identify different solutions and generate different such grasping configurations. The accompanying implementation considers cylindrical objects and specifies a certain type of relative end effector grasp e so that the arm grasps the center of the cylinder. All grasps that are symmetrical to the cylinder's long axis are valid.

7.2.2 Finding neighboring states

The `NEAR` primitive computes the neighboring states V_{near} to the newly sampled state v_{new}. This function depends on the selection of an appropriate distance metric between transition rearrangement states as well as the identification of the number of neighbors that should be returned. For the distance metric, an estimation of the number of necessary transfers between two input arrangements can be potentially used. For instance,

the problem can be relaxed and the number of overlapping poses between the initial and goal arrangement can be counted. In addition, if the rearrangement primitive can solve only monotone problems, then states with overlapping poses, where a swap of two objects has to happen (Fig.7.2), will not be returned as neighboring states. These problems have a solution only if an object moves to an intermediate pose, something that a monotone rearrangement primitive cannot detect. The accompanying implementation uses the sum of the distances between poses as an estimation of selecting the near neighboring states.

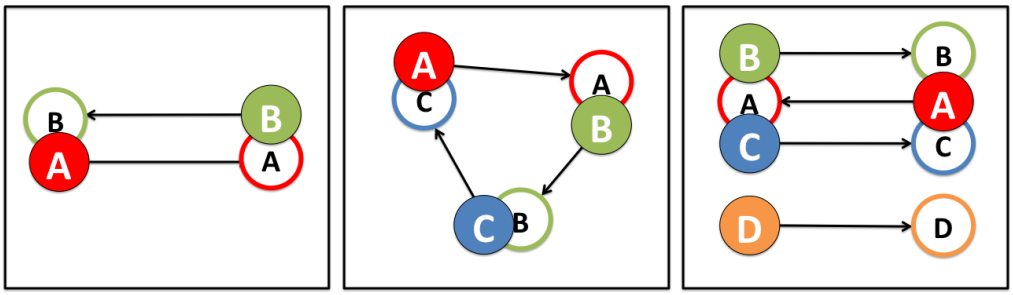

Figure 7.2: If the light colored positions correspond to an initial arrangement, while the dark colored positions correspond to the final arrangement, then a monotone solver will not be able to solve any of these examples. In this case if the high-level task planner is using a monotone rearrangement primitive the `NEAR` function will not return these nodes as neighboring states.

The value from `PRM`* [58] and a linear search for closest neighbors was used. If path quality is not a priority, then if two arrangements are already in the same connected component, then 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 [86].

An efficient implementation of this top-level approach will also consider of directly attempting to connect start and goal states before generating intermediate states to solve problems. This is not shown in Algorithm 1 for brevity purposes.

7.3 Implementation of Minimum Constraint Removal Algorithm

While important in many applications, the `MCR` challenge is harder than searching for the shortest path. The issue arises because `MCR` paths do not satisfy dynamic programming properties. In particular, a subset of an optimal solution is not necessarily an optimal

solution itself. In the process of computing a MCR path, a configuration q will be visited by different paths because each path can have a different combination of violated obstacle regions.

In most of the applications, and specifically in the manipulation, the search space is significantly larger than the space that the approach needs to search in order to find a reasonable MCR path. The method for computing the MCR path can waste a lot of time searching away from the solution before moving towards the target. In this work, a variation of bounded path length approach [67, 70] for computing MCR is used as an alternative. This methodology bounds the length of paths that it searches given a multiple of the shortest path length in a constraint-free version of the state space.

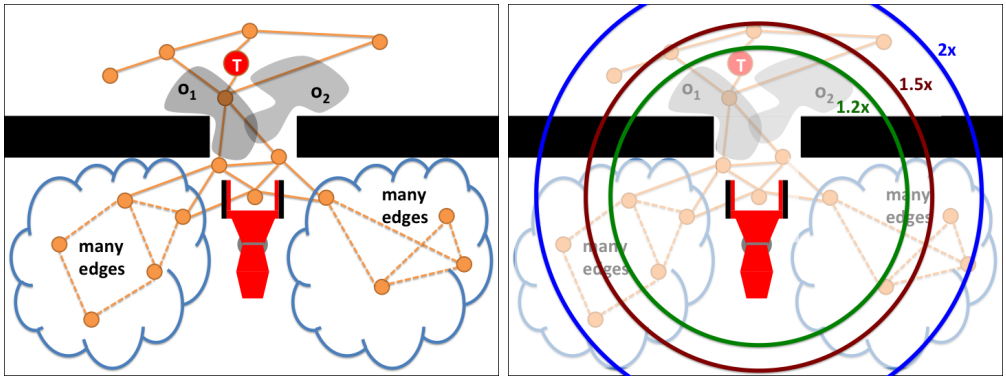

Figure 7.3: Left: A case where the Bounded-Length algorithm will not waste a lot of time to check the collision free areas, before start moving towards the target. Right: If the algorithm has more time to search for the optimal MCR, it will increases the depth limit of the search relative to the shortest path.

This work is using a bounded-Length version of the exact approach method [67] for computing the MCR path. However, this is an approximate method that it is faster than both the exact and greedy algorithms, with some tradeoffs on the number of constraints computed. The approximate version is shown in Algorithm 18. This version allows to incrementally increase the depth limit of the search until there is convergence to the true optimal MCR.

Alg.18 works similar to the "exact" approach. The method uses a priority queue, which prioritizes search elements that have a low number of constraints $|\mathsf{c}|$. A small evaluation function f is utilized in order to break ties between the nodes. While the

Algorithm 18: BL_MCR($\mathcal{G}(\mathcal{V},\mathcal{E}), q_s, q_f, threshold$)

```
1  Q ← NEW_ELEMENT(q_s, ∅, ∅, 0);
2  while Q not empty do
3      u_top ← Q.pop();
4      if u_top.⊑ == q_f then
5          return u_top.π;
6      for each v_neigh ∈ Adj(G, u_top.⊑) do
7          π ← u_top.π | e(u_top.⊑, v_neigh);
8          if |π| < threshold then
9              c ← u_top.c ∪ e(u_top.⊑, v_neigh).c;
10             if IS_NEW_SET(v_neigh.C, c) then
11                 f ← |π| + h(v_neigh, q_f);
12                 Q ← NEW_ELEMENT(v_neigh, π, c, f);
13     return ∅;
```

queue is not empty (line 2), the algorithm will pop the highest priority search element u_{top} (line 3). If this node is the target node q_f (line 4), then the algorithm will return the path stored in the search element (line 5). Otherwise, all the adjacent graph nodes $\mathtt{v}_{neigh}$ of $u_{top}.\sqsubseteq$ are considered (line 6). The main difference from the "exact" approach is that the BL_MCR approach first checks if the path has length greater than the threshold (lines 7-8). If the path is within the threshold, then the function IS_NEW_SET (line 10) will check:

- If there is a constraint set $\mathtt{c}' \in \mathbf{C}$, where $\mathtt{c}' \subset \mathtt{c}$ then the algorithm returns false and the node is not added in Q.
- If there is a constraint set $\mathtt{c}' \in \mathbf{C}$, where $\mathtt{c} \equiv \mathtt{c}'$, then the algorithm will compare the f values. If the new f' value is smaller than the existing one, IS_NEW_SET returns true.
- If there is a constraint set $\mathtt{c}' \in \mathbf{C}$, where $\mathtt{c} \subset \mathtt{c}'$ then the set $\mathtt{c}'$ will be replaced by c and the method returns true.
- If none of the above is true, then the algorithm will return true and c will be added in the $\mathbf{C}$ list of the graph node.

If IS_NEW_SET detects that the constraints set includes new constraints, then the

algorithm will create a new search element and update the corresponding node, using the `NEW_ELEMENT` function (lines 11-12). For speed purposes, each graph node can keep track of the different sets of constraints, $\mathbf{C}$, for each path that has reached the node, and the corresponding evaluation functions f.

`BL_MCR` with a small threshold will be able to search through short length paths that violate constraints fast without concentrating time searching through constraint-free nodes far from the shortest paths. Of course, the algorithm cannot guarantee the optimal `MCR` path, but in practice, it returns good solutions, close to those of the exact approach for `MCR` and with paths that are of known length relative to the shortest.

Chapter 8

Conclusions

This work is presented as a framework for solving object rearrangement using a robotic manipulator. The focus has been on hard instances of general object rearrangement, which correspond to:

- Non-monotone problems,
- Unique, labeled objects,
- In tight and cluttered environments.

Moreover, this work showing the potential of combining sampling-based algorithms with strong local primitives to solve the hard object rearrangement problems. This work makes contributions towards a fully automated warehouse where robots could rearrange and pick objects from a shelf. The methods provide formal guarantees on the success ratios, solution times and path quality. The final result of this work is a scalable and fast framework which utilizes sampling-based motion planning algorithms with powerful rearrangement primitives in order to probabilistically complete methods for solving hard instances of general object rearrangement instances.

8.1 Statement of Contributions

The main contributions of this work are spread across the chapters 4, 5, 6 and 7. It begins with solving unlabeled rearrangement problems, continues by solving labeled general rearrangement problems and ends with showing that it is possible to have efficient and fast methods with probabilistic completeness guarantees for solving rearrangement problems.

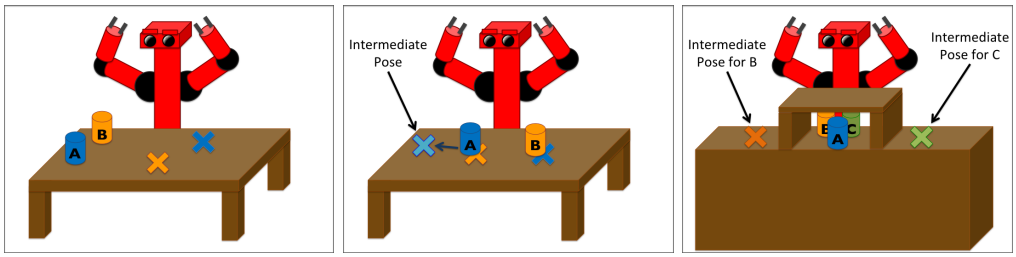

Figure 8.1: Barky is able to solve all the rearrangement problems using the methods described in this work. Left: a monotone problem. Middle: A non-monotone problem, Right: Pick the object 'A' located behind the objects 'B' and 'C'

Beginning in chapter 4, the main idea was to reduce the continuous, high-dimensional rearrangement problem into several discrete rearrangement challenges on "manipulation pebble graphs" (MPGs). This idea was inspired by work in algorithmic theory on "pebble graphs" [6] and related contributions in multi-robot motion planning [104]. However, the transfer of the pebble graph idea was not trivial. The presence of a manipulator induces additional constraints. These extra constraints required the development of different solutions for the connection of MPGs. Formally, the main contributions of chapter 4 can be summarized as follows :

- Integration of combinatorial multi-robot planning and manipulation planning for efficient computation of manipulation paths to rearrange similar objects in cluttered spaces.
- Continuous rearrangement challenges are abstracted as discrete pebble graph problems, i.e. Manipulation Pebble Graphs (MPGs).
- This method is able to solve non-monotone problems, where objects can be grasped more than once.

The experimental results showed the scalability and the class of difficult non-monotone problems this method was able to solve. Although this method was able to deal with the problem of dimensionality and solve cluttered problems, including multiple movable objects and a high degree of freedom robotic manipulator, it suffered by the limitation of solving labeled rearrangement problems. Meaning that the solution could not enforce specific poses for each moving object. While that was a known problem, it motivates

the contributions of the rest of the work.

In particular, chapter 5 aims to still provide solutions for non-monotone problems in clutter environments but doing that for labeled cases as well. The first step was a new algorithmic primitive for object rearrangement that is able to practically solve many non-monotone instances. It extends a previously developed technique for monotone problems [110] and utilizes principles from the multi-robot planning literature [85, 71] as well as efficient solutions to the Minimum Constraint Removal path problem [46, 47, 67, 70] This non-monotone primitive is integrated with a higher-level planner in the context of a hierarchical framework, which operates to a similar function to a probabilistic roadmap (`PRM`). This work shows that the proposed integration achieves probabilistic completeness by searching the space of possible object arrangements. It uses the proposed non-monotone primitive as a local planner to locally connect pairs of object arrangements, which constitute nodes of the high-level `PRM` planner. Formally, the main contributions of chapter 5 can be summarized as follows:

- Extends the monotone solver [110] to solve some non-monotone problems.
- Achieve completeness by integrating these rearrangement primitives with a higher-level planner in the context of a hierarchical framework.

Simulated experiments show that the proposed primitive solves many non-monotone challenges by itself. The integration with the higher-level task planner results in an improved solution in terms of success ratio, path quality and scalability, especially in setups in tight spaces and with limited reachability for the manipulator, such as shelves. Typically, few arrangements are sampled by the high-level planner to solve relatively hard instances when the primitive is failing to find a solution directly.

Given the idea of the hierarchical framework where the rearrangement primitives are utilized as local planners in a high-level task planner, chapter 6 proposes a faster approach for solving general object rearrangement problems. The two important findings were the requirement of a fast rearrangement primitive and the maintenance of good connectivity for the graph structure of the high-level planner. The proposed solution in chapter 6 trades completeness for efficiency by avoiding exhaustive search. Instead,

it commits to using only "minimum constraint removal" paths for the objects, based on which it is possible to quickly figure out whether a monotone solution exists. This trade-off proves to be beneficial when using rearrangement primitive as a subroutine in a `Bi-RRT` task planner in the space of general object arrangements. The integration of the fast primitive with this task planner results in a method that a) is probabilistically complete, b) exhibits higher success ratios and computational performance relative to the alternatives, as well as c) better solution paths. The contribution of the chapter 6 can be summarized as follows:

- Define a fast primitive for solving monotone rearrangement problems.
- Extend the fast primitive for solving and non-monotone rearrangement problems.
- Allow the monotone rearrangement primitives to return partial solutions.
- Use the primitive as a local planner in an incremental search algorithm for general rearrangement planning (similar to Bi-RRT) to solve general problems and guarantee probabilistic completeness, efficiency and high success ratios.

The provided simulation experiments make use of objects of similar geometry for which it is easy to quickly identify grasps. This allows performing helpful precomputation towards fast resolution of rearrangement queries. The framework itself, however, does not depend on such assumptions and can be applied in general setups with objects of varying geometry given a primitive for effectively detecting appropriate grasps. Furthermore, there are many different algorithmic choices for integrating sampling-based planners with local primitives in the context of object arrangement. Chapter 7 shows all these implementation details and how the framework is implemented in order to take advantage of the similarity of the objects. While it can easily be shown that the limitation of using similar objects, it is not a framework limitation but an implementation limitation. Along with the details about the implementation chapter 7 presents and the contributions for the "minimum constraint removal" path, where this work is massively using. Finding the "minimum constraint removal" paths (`MCR`) is a computationally hard problem in the general case as such paths do not exhibit dynamic programming

properties. This work presents and uses approximate solutions that bound the path length of the considered path seem to provide a desirable trade-off in terms of returning solutions with a low number of constraints, relatively short path lengths and low computation time.

8.2 Important Open Questions for Future Work

This work provides several contributions towards having a fast and efficient framework that could deal with all the cases of rearrangement. However, there are still many open avenues for investigation and further improvements. 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 the integration of `RRT` with the fast rearrangement primitives is also desirable. The high-level planners are selecting the nearest arrangement based on the distance between the poses, i.e. L-infinity in order to compute the accumulative distance between the poses of the two arrangements. More investigation about a good distance function between two arrangements is needed and it will improve the connectivity of the high-level planner.

Another useful objective in the context of object rearrangement is to identify the set of problems which are directly addressable by primitives similar to the non-monotone one proposed here and which do not require the proposed more general, hierarchical approach. One can look for inspiration in methods rooted in multi-robot planning and especially those, which can provide optimality guarantees [118, 105, 123]. Moreover, recent development of a method that is dealing with high-quality paths can already minimize the number of pick-and-place actions and minimize also the distance that is traveled by the end-effector [102]. This can lead to the development of solutions, which under certain conditions, can provide high-quality paths with optimality guarantees while solving object rearrangement problems. In the context of the proposed, hierarchical and sampling-based framework, it is similarly important to consider the conditions under which it can asymptotically converge to optimal solutions. There has been recent work in the area of asymptotic optimality for manipulation task planning [117], which utilizes sampling-based algorithms. This work can indicate a direction of

how the proposed framework can achieve such an objective while remaining computationally efficient.

Future efforts should also focus on the computation of robust rearrangement trajectories under the presence of uncertainty, which can arise from pose estimation processes [97], as different manipulation operations have a different probability of being successful in practice. There has been some work in hierarchical decision-theoretic planning tools for related problems [82], which motivate further work in this area. Similarly, tight integration with perception for detecting objects is necessary for the real-world adoption of rearrangement algorithms in target applications.

Other interesting directions include: a) *Multiple arms collaborating* [32] to rearrange objects. Using multiple arms could be helpful as one of the arms can grasp an object to clear the scene for the other arm to perform a transfer. This way problems as in figure 5.3 can be solved as monotone problems, as one of the hands can be used as an intermediate pose. b) *Non-prehensile actions* [25, 34]. Non-prehensile actions can also be useful, especially in clutter environment where the free space might not be sufficient for the robotic arm to grasp an object. In such cases, the robot will have to push some objects in order to create more space. c) *Mobile manipulation* [40]. Although this extension is irrelevant with the current work's problem setup, it is still a valid extension for object rearrangement problems. This work is trying to solve rearrangement problems in cluttered environment. With a mobile base, the manipulator can utilize more space and the solution to a rearrangement problem became easier, but more time-consuming given that the robot will have to travel. d) *Dealing with symbolic* goal conditions, where conditions should be satisfied before the robot is able to find a solution. This can involve stacking challenges [26, 113] where the robot has to place objects on top of each other and physics can play a critical role. At a combinatorial level, the algorithmic primitives developed as part of this work can still be helpful in such setups but need to be integrated with processes that reason about the feasibility of manipulation actions given physical constraints. It is interesting to evaluate how different object placements may result in different probability of success during real-world execution. From a system's point of view, it is interesting to investigate how to quickly compute such rearrangement

paths through the use of cloud-based computation [9].

References

[1] R. Alami, J.-P. Laumond, and T. Siméon. Two Manipulation Planning Algorithms. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*. A. K. Peters, Wellesley, MA, 1997.

[2] R. Alami, T. Siméon, and J.-P. Laumond. A Geometrical Approach to Planning Manipulation Tasks. In *Proc. of International Symposium on Robotics Research*, pages 113–119, 1989.

[3] R. Alterovitz, S. Patil, and A. Derbakova. Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, 2011.

[4] M. S. Apaydin, D. L. Brutlag, C. Guestrin, D. Hsu, J.-C. Latombe, and C. Varm. Stochastic Roadmap Simulation: An Efficient Representation and Algorithm for Analyzing Molecular Motion. *Journal of Computational Biology*, 10:257–281, 2003.

[5] B. Aronov, M. de Berg, A. F. van den Stappen, P. Svestka, and J. Vleugels. Motion Planning for Multiple Robots. *Discrete and Computational Geometry*, 22(4):505–525, 1999.

[6] V. Auletta, A. Monti, D. Parente, and G. Persiano. A Linear Time Algorithm for the Feasibility of Pebble Motion on Trees. *Algorithmica*, 23:223–245, 1999.

[7] J. Barraquand and J.-C. Latombe. Robot Motion Planning: A Distributed Representation Approach. *IJRR*, 10(6):628–649, Dec. 1991.

[8] J. Basch, L. J. Guibas, D. Hsu, and A. T. Nguyen. Disconnection proofs for motion planning. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1765–1772, 2001.

[9] K. E. Bekris, R. Shome, A. Krontiris, and A. Dobson. Cloud Automation: Precomputing Roadmaps for Flexible Manipulation. *IEEE Robotics and Automation Magazine (Special Issue on Emerging Advances and Applications in Automation)*, 2015.

[10] O. Ben-Shahar and E. Rivlin. Practical Pushing Planning for Rearrangement Tasks. *IEEE Transactions on Robotics and Automation*, 14(4), August 1998.

[11] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. J. Kuffner. Grasp Planning in Complex Scenes. In *Proc. of IEEE-RAS International Conference on Humanoid Robots*, 2007.

[12] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner. Manipulation Planning on Constraint Manifolds. In *Proc. of the IEEE Intern. Conf. on Robotics and Automation (ICRA)*, 2009.

[13] D. Berenson, S. S. Srinivasa, and J. J. Kuffner. Task Space Regions: A Framework for Pose-Constrained Manipulation Planning. *The International Journal of Robotics Research (IJRR)*, 30(12):1435–1460, 2012.

[14] A. Botea, M. Muller, and J. Schaffer. Using Abstraction for Planning in Sokoban. *Computers and Games*, pages 360–375, 2002.

[15] T. Bretl, S. Lall, J.-C. Latombe, and S. Rock. Multi-step Motion Planning for Free-Climbing Robots. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2004.

[16] R. Brooks and T. Lozano-Pérez. A Subdivision Algorithm in Configuration Space for Findpath with Rotation. In *IJCAI*, pages 799–803, 1983.

[17] G. Calinescu, A. Dumitrescu, and J. Pach. Reconfigurations in Graphs and Grids. *SIAM Journal on Discrete Mathematics*, 22(1):124–138, 2008.

[18] S. Cambon, R. Alami, and F. Gravot. A Hybrid Approach to Intricate Motion, Manipulation, and Task Planning. *International Journal of Robotics Research*, (28), 2009.

[19] J. Canny. *The Complexity of Robot Motion Planning*. PhD thesis, MIT, Cambridge, MA, 1988.

[20] P. C. Chen and Y. K. Hwang. Practical Path Planning Among Movable Obstacles. In *Proc. of the IEEE Intern. Conf. on Robotics and Automation*, pages 444–449, 1991.

[21] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston, MA, 2005.

[22] M. Ciocarlie, K. Hsiao, G. E. Jones, S. Chitta, R. B. Rusu, and I. A. Sucan. Towards Reliable Grasping and Manipulation in Household Environments. In *International Symposium on Experimental Robotics (ISER)*, 2010.

[23] J. B. Cohen, S. Chitta, and M. Likhachev. Search-based Planning for Manipulation with Motion Primitives. In *Proc. of the IEEE Intern. Conf. on Robotics and Automation (ICRA)*, 2010.

[24] J. B. Cohen, S. Chitta, and M. Likhachev. Single- and dual-arm motion planning with heuristic search. *International Journal of Robotic Research*, 33(2):305–320, 2014.

[25] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman. Push Planning for Object Placement on Cluttered Table Surfaces. In *Proc. of the IEEE Intern. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[26] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems (RSS)*, 2016.

[27] D. Davidov and S. Markovitch. Multiple-Goal Heuristic Search. *Journal of Artificial Intelligence Research*, pages 417–451, 2006.

[28] L. De Moura and N. Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.

[29] L. De Silva, M. Gharbi, A. K. Pandey, and R. Alami. A new approach to combined symbolic-geometric backtracking in the context of human-robot interaction. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3757–3763. IEEE, 2014.

[30] E. Demaine, J. O'Rourke, and M. L. Demaine. Pushpush and push-1 are NP-hard in 2D. In *Proc. of the 12th Canadian Conf. on Computational Geometry*, pages 211–219, 2000.

[31] A. Dobson and K. E. Bekris. Improved Heuristic Search for Computing Sparse Data Structures for Motion Planning. In *Symposium on Combinatorial Search (SoCS)*, Prague, Czech Republic, 2014.

[32] A. Dobson and K. E. Bekris. Planning Representations And Algorithms For Prehensile Multi-Arm Manipulation. In *IROS*, 2015.

[33] A. Dobson, A. Krontiris, and K. E. Bekris. Sparse Roadmap Spanners. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2012.

[34] M. R. Dogar and S. S. Srinivasa. A Framework for Push-Grasping in Clutter. In *Robotics: Science and Systems (RSS)*, 2011.

[35] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel. Semantic attachments for domain-independent planning systems. In *Towards service robots for everyday environments*, pages 99–115. Springer, 2012.

[36] C. Dornhege, M. Gissler, M. Teschner, and B. Nebel. Integrating Symbolic and Geometric Planning for Mobile Manipulation. Denver, CO, 2009. IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR), IEEE.

[37] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras. Combining High-Level Causal Reasoning with Low-Level Geometric Reasoning and Motion Planning for Robotic Manipulation. In *IEEE Internation Conference on Robotics and Automation (ICRA)*, pages 4575–4581, 2011.

[38] L. H. Erickson and S. M. LaValle. A Simple, but NP-Hard Motion Planning Problem. In *AAAI Conference on Artificial Intelligence*, 2013.

[39] K. Erol, J. Hendler, and D. S. Nau. Htn planning: Complexity and expressivity. In *AIII*, volume 94, pages 1123–1128. Springer, 1994.

[40] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. Ffrob: An efficient heuristic for task and motion planning. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2014.

[41] S. Ge and Y. Cui. Dynamic Motion Planning for Mobile Robots using Potential Field Method. *Autonomous Robots*, 13:207–222, 2002.

[42] M. Gharbi, R. Lallement, and R. Alami. Combining symbolic and geometric planning to synthesize human-aware plans: toward more efficient combined search. In *IROS*, pages 6360–6365. IEEE, 2015.

[43] M. Gobelbecker, T. Keller, P. Eyerich, M. Brenner, and B. Nebel. Coming up with good excuses: What to do when no plan can be found. In *International Conference on Automated Planning and Scheduling*, 2010.

[44] G. Goraly and R. Hassin. Multi-Color Pebble Motion on Graphs. *Algorithmica*, 58(3):610–636, 2010.

[45] D. Halperin, J.-C. Latombe, and R. H. Wilson. A General Framework for Assmbly Planning: the Motion Space Approach. *Algorithmica*, 26(3-4):577–601, 2000.

[46] K. Hauser. The Minimum Constraint Removal Problem with Robotics Applications. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2012.

[47] K. Hauser. Minimum Constraint Displacement Motion Planning. In *Robotics: Science and Systems (RSS)*, 2013.

[48] K. Hauser. The minimum constraint removal problem with three robotics applications. *The International Journal of Robotics Research*, 2013.

[49] K. Hauser and J.-C. Latombe. Multi-Modal Planning in Non-Expansive Spaces. *International Journal of Robotics Research (IJRR)*, 29(7):897–915, 2010.

[50] K. Hauser and V. Ng-Thow-Hing. Randomized Multi-Modal Motion Planning for a Humanoid Robot Manipulation Task. *International Journal of Robotics Research*, 2011.

[51] G. Havur, G. Ozbilgin, E. Erdem, and V. Patoglu. Geometric Rearrangement of Multiple Moveable Objects on Cluttered Surfaces: A Hybrid Reasoning Approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[52] D. Hsu, L. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On Finding Narrow Passages with Probabilistic Roadmap Planners. In *WAFR*, Houston, TX, 1998.

[53] D. Hsu, J.-C. Latombe, and R. Motwani. Path Planning in Expansive Configuration Spaces. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 3, pages 2719–2726, Albuquerque, NM, Apr. 1997.

[54] D. Hsu, J.-C. Latombe, and R. Motwani. Path Planning in Expansive Configuration Spaces. *Int. Journal of Computational Geometry and Applications*, 9(4-5):495–512, 1999.

[55] Y. K. Hwang and N. Ahuja. A Potential Field Approach to Path Planning. *TRA*, 8(1):23–32, Feb. 1992.

[56] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical Task and Motion Planning in the Now. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[57] S. Kambhampati and L. S. Davis. Multiresolution Path Planning for Mobile Robots. *JRA*, 2(3):135–145, Sept. 1986.

[58] S. Karaman and E. Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. *International Journal of Robotics Research (IJRR)*, 30(7):846–894, June 2011.

[59] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe. Analysis of Probabilistic Roadmaps for Path Planning. *IEEE TRA*, 14(1):166–171, 1998.

[60] L. E. Kavraki and J.-C. Latombe. *Probabilistic Roadmaps for Robot Path Planning*, pages 33–53. John Wiley, 1998.

[61] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[62] O. Khatib. Real-time Obstacle Avoidance for Manipulators and Mobile Robots. *IJRR*, 5(1):90–98, 1986.

[63] J. King, M. Klingensmith, C. Dellin, M. Dogar, P. Velagapudi, N. Pollard, and S. S. Srinivasa. Pregrasp Manipulation as Trajectory Optimization. In *Robotics: Science and Systems (RSS)*, 2013.

[64] D. Koditschek. Robot Planning and Control via Potential Functions. In *The Robotics Review 1*, pages 349–367. MIT Press, 1989.

[65] G. Konidaris, L. Kaelbling, and T. Lozano-Perez. Constructing Symbolic Representations for High-Level Planning. In *Association for the Advancement of Artificial Intelligence (AAAI) conference*, 2014.

[66] D. Kornhauser, G. Miller, and P. Spirakis. Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications. In *Foundations of Computer Science (FOCS)*, pages 241–250, 1984.

[67] A. Krontiris and K. E. Bekris. Computational Tradeoffs of Search Methods for Minimum Constraint Removal Paths. In *Symposium on Combinatorial Search (SoCS)*, Dead Sea, Israel, 2015.

[68] A. Krontiris and K. E. Bekris. Dealing with difficult instances of object rearrangement. In *Robotics: Science and Systems (RSS)*, Rome, Italy, July 2015.

[69] A. Krontiris and K. E. Bekris. Efficiently solving general rearrangement tasks:a fast extension primitive for an incremental sampling-based planner. In *IEEE International Conference on Robotics and Automation (ICRA)*, Sweden, 2016.

[70] A. Krontiris and K. E. Bekris. Tradeoffs in the computation of minimum constraint removal paths for manipulation planning. *Advanced Robotics Journal (accepted)*, 2017.

[71] A. Krontiris, R. Luna, and K. E. Bekris. From feasibility tests to path planners for multi-agent pathfinding. In *Symposium on Combinatorial Search (SoCS - 2013)*, Leavenworth, WA, USA, 07/2013 2013.

[72] A. Krontiris, R. Shome, A. Dobson, A. Kimmel, and K. E. Bekris. Rearranging similar objects with a manipulator using pebble graphs. In *IEEE Humanoids*, Madrid, Spain, 2014.

[73] A. M. Ladd and L. E. Kavraki. Measure Theoretic Analysis of Probabilistic Path Planning. *IEEE TRA*, 20(2):229–242, Apr. 2004.

[74] F. Lamiraux and L. E. Kavraki. Planning Paths for Elastic Objects Under Manipulation Constraints. *International Journal of Robotics Research*, 20(3):188–208, 2001.

[75] F. Lamiraux and J.-P. Laumond. On the Expected Complexity of Random Path Planning. In *ICRA*, pages 3306–3311, 1996.

[76] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[77] S. LaValle. Rapidly-exploring random trees: A new tool for path planning, 1998.

[78] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[79] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. WAFR, 2000.

[80] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research (IJRR)*, 20:378–400, May 2001.

[81] S. Leroy, J.-P. Laumond, and T. Siméon. Multiple Path Coordination for Mobile Robots: A Geometric Algorithm. In *International Joint Conference on Artificial Intelligence*, pages 1118–1123, 1999.

[82] M. Levinh, J. Scholz, and M. Stilman. Hierarchical Decision Theoretic Planning for Navigation Among Movable Obstacles. In *Proc. of the Workshop on the Algorithmic Foundations of Robotics*, 2012.

[83] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, pages 108–120, 1983.

[84] R. Luna and K. E. Bekris. Efficient and Complete Centralized Multi-Robot Path Planning. In *International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[85] R. Luna and K. E. Bekris. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. In *International Joint Conferences in Artificial Intelligence (IJCAI-11)*, pages 294–300, Barcelona, Spain, July 2011.

[86] J. Marble and K. E. Bekris. Asymptotically Near-Optimal Planning with Probabilistic Roadmap Spanners. *IEEE Transactions on Robotics*, 29(2):432–444, 2013.

[87] Z. McCarthy, T. Bretl, and S. Hutchinson. Proving Path Non-Existence using Sampling and Alpha Shapes. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2563–2569, 2012.

[88] P. Mike, V. Hwang, S. Chitta, and M. Likhachev. Learning to Plan for Constrained Manipulation from Demonstrations. In *Robotics: Science and Systems (RSS)*, 2013.

[89] S. Nedunuri, S. Prabhu, M. Moll, S. Chaudhuri, and L. E. Kavraki. Smt-based synthesis of integrated task and motion plans from plan outlines. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 655–662. IEEE, 2014.

[90] C. Nielsen and L. E. Kavraki. A Two-Level Fuzzy PRM for Manipulation Planning. In *IEEE/RSJ IROS*, pages 1716–1722, Japan, 2000.

[91] D. Nieuwenhuisen, A. Frank van der Stappen, and M. H. Overmars. An Effective Framework for Path Planning amidst Movable Obstacles. In *Proc. of the Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2006.

[92] J. Ota. Rearrangement Planning of Multiple Movable Objects. In *Prof. of the IEEE Intern. Conference on Robotics and Automation (ICRA)*, 2004.

[93] A. G. Pereira, M. R. P. Ritt, and L. S. Buriol. Finding Optimal Solutions to Sokoban Using Instance Dependent Pattern Databases. In *Symposium on Combinatorial Search (SoCS)*, 2013.

[94] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki. Sampling-Based Roadmap of Trees for Parallel Motion Planning. *IEEE TRA*, 21(4):587–608, 2005.

[95] E. Plaku and G. Hager. Sampling-based Motion Planning with Symbolic, Geometric, and Differential Constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[96] J. H. Reif. Complexity of the Generalized Mover's Problem. In *FOCS*, pages 421–427, 1979.

[97] C. Rennie, R. Shome, K. E. Bekris, and A. F. De Souza. A Dataset For Improved Rgbd-Based Object Detection And Pose Estimation For Warehouse Pick-And-Place. *IEEE RA-L - also at ICRA*, 2016.

[98] L. I. Reyes Castro, P. Chaudhari, J. Tumova, S. Karaman, E. Frazzoli, and D. Rus. Incremental sampling-based algorithm for minimum-violation motion planning. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 3217–3224. IEEE, 2013.

[99] E. Rimon and D. Koditschek. Exact Robot Navigation Using Artificial Potential Functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, Oct. 1992.

[100] G. Sánchez and J.-C. Latombe. On Delaying Collision Checking in PRM Planning: Application to Multi-Robot Coordination. *International Journal of Robotics Research*, 21(1):5–26, 2002.

[101] F. Schwarzer, M. Saha, and J.-C. Latombe. Adaptive Dynamic Collision Checking for Single and Multiple Articulated Robots in Complex Environments. *IEEE Transactions on Robotics*, 21(3):338–353, 2005.

[102] H. Shuai, N. Stiffler, A. Krontiris, K. E. Bekris, and J. Yu. High-quality tabletop rearrangement with overhand grasps: Hardness results and fast methods. In *Robotics: Science and Systems (RSS)*, Cambridge, MA, 07/2017 2017. [Best Student Paper Award Finalist], [Best Student Paper Award Finalist].

[103] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani. Manipulation Planning with Probabilistic Roadmaps. *International Journal of Robotics Research (IJRR)*, (23), 2004.

[104] K. Solovey and D. Halperin. k-Color Multi-Robot Motion Planning. In *Proceedings of the 10th International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 191–207, 2012.

[105] K. Solovey, O. Salzman, and D. Halperin. Finding a Needle in an Exponential Haystack: Discrete RRT for Exploration of Implicit Roadmaps in Multi-Robot Motion Planning. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2014.

[106] 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 *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[107] A. Stentz. The Focused D* Algorithm for Real-Time Replanning. 1995.

[108] M. Stilman and J. Kuffner. Navigation among Movable Obstacles: Realtime Reasoning in Complex Environments. In *Journal of Humanoid Robotics*, pages 322–341, 2004.

[109] M. Stilman and J. J. Kuffner. Planning Among Movable Obstacles with Artificial Constraints. In *Proc. of the Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2006.

[110] M. Stilman, J. Schamburek, J. J. Kuffner, and T. Asfour. Manipulation Planning Among Movable Obstacles. In *IEEE International Conference on Robotics and Automation*, 2007.

[111] S. Sundaram, I. Remmler, and N. M. Amato. Disassembly Sequencing Using a Motion Planning Approach. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1475–1480, Washington, D.C., May 2001.

[112] P. Švestka. *Robot Motion Planning using Probabilistic Road Maps*. PhD thesis, Utrecht University, the Netherlands, 1997.

[113] M. Toussaint. Logic-Geometric Programming: An Optimization-based Approach to Combined Task and Motion Planning. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 2015.

[114] J. van den Berg and M. Overmars. Prioritized Motion Planning for Multiple Robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 430–435, 2005.

[115] J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha. Centralized Path Planning for Multiple Robots: Optimal Decoupling into Sequential Plans. In *Robotics: Science and Systems (RSS)*, 2009.

[116] J. van den Berg, M. Stilman, J. J. Kuffner, M. Lin, and D. Manocha. Path Planning Among Movable Obstacles: A Probabilistically Complete Approach. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2008.

[117] W. Vega-Brown and N. Roy. Asymptotically optimal planning under piecewise-analytic constraints. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[118] G. Wagner, M. Kang, and H. Choset. Probabilistic Path Planning for Multiple Robots with Subdimensional Expansion. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012.

[119] Y. Wang, N. Dantam, S. Chaudhuri, and L. Kavraki. Task and motion policy synthesis as liveness games. In *International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI, 2016.

[120] G. Wilfong. Motion Planning in the Presence of Movable Obstacles. In *Proc. of the 4^{th} Annual Symp. of Computational Geometry*, pages 279–288, New York City, NY, USA, 1988. ACM.

[121] R. H. Wilson and J.-C. Latombe. Geometric Reasoning about Mechanical Assembly. *Artificial Intelligence Journal*, 71(2):371–396, 1994.

[122] J. Yu and S. M. LaValle. Multi-agent Path Planning and Network Flow. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2012.

[123] J. Yu and S. M. LaValle. Optimal multi-robot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016.

[124] L. Zhang, Y. Kim, and D. Manocha. A Simple Path Non-Existence Algorithm using C-Obstacle Query. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2008.

[125] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. A. Bagnell, and S. S. Srinivasa. CHOMP: Covariant Hamiltonian Optimization for Motion Planning. *International Journal of Robotics Research (IJRR)*, 2013.