Manipulation Among Movable Objects for Pick-and-Place Tasks in Cluttered 3D Workspaces

Dhruv Mauria Saxena

CMU-RI-TR-23-76

September 2023

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

To the people that inspired this body of work, thank you for your genius.

Abstract

In cluttered real-world workspaces, simple pick-and-place tasks for robot manipulators can be quite challenging to solve. Often there is no collision-free trajectory that allows the robot to grasp and extract a desired object from the scene. This requires motion planning algorithms to reason about rearranging some of the "movable" clutter in the scene so as to make the task feasible. Our work focuses on solving these pick-and-place tasks in 3D workspaces where objects may tilt, lean on each other, topple, and slide. This problem setup can be used to describe common uses of robot manipulators tasked with packing boxes in warehouses, assembling structures in manufacturing industries, and assisting humans inside households. Robots operating in these environments will engage in contact-rich interactions with objects in the environment, and must be able to model and reason about the effect of these interactions on the environment. In this thesis we develop motion planning algorithms for robotic manipulation that make efficient use of a physics-based rigid-body simulator to solve pick-and-place manipulation tasks while accounting for the configuration of all objects in the environment and the effect that robot actions have on them. In particular, we require our planning algorithms to compute trajectories that satisfy all object-centric "interaction constraints". In the 3D workspaces we care about, these encode properties of objects such as whether the robot can make contact with an object, how far it can make objects tilt, whether objects are allowed to topple over, how fast they can be moved etc. The challenge in solving simple pick-and-place tasks in these environments lies in finding a solution trajectory that satisfies these constraints at all points in time.

In environments with relatively large object-free volumes, we show that it suffices to only model robot-object and any resultant object-object interactions near grasp poses given to the robot. These grasp poses are specified as intermediate goals for the robot from where the desired object may be grasped before being moved to its final goal (or "home") configuration. Our intuition in these workspaces suggests that we can exploit the large object-free volume to plan contact-free trajectories that get the robot near the grasp pose. Once such a trajectory has been found, the planner can forward simulate dynamically generated grasping primitives in the physics-based simulator to ensure that interaction constraints are not violated while grasping the target object.

With more clutter in the scene or in a more constrained workspace, we cannot assume the existence of a contact-free trajectory that gets close to the target object. In such cases, we have to reason about deliberately rearranging any movable clutter so that the object-of-interest (OoI) can be grasped and extracted. Such situations arise commonly when a robot must reach inside a cluttered shelf to retrieve a desired object. Manipulation planning algorithms for such scenes must be able to answer three fundamental questions – *which* movable objects should the robot rearrange, *where* should they be moved, and *how* should the robot move them while still satisfying all interaction constraints? This thesis presents a family of algorithms that draw on a unique connection between Multi-Agent Pathfinding (MAPF), the problem of finding paths for multiple robots that need to get from their start states to some desired goal states, and Manipulation Among Movable Objects (MAMO), solving manipulation tasks in environments where the robot is allowed to rearrange movable clutter. Our algorithms solve an appropriately constructed MAPF abstraction of MAMO to answer the first two questions of which objects should be moved and where to. We then convert the MAPF solution into rearrangement actions for the robot to rearrange movable objects and make progress towards solving the MAMO problem of retrieving the OoI from a cluttered shelf.

We test all algorithms developed as part of this thesis in the real-world with the PR2 robot. We show that we can execute trajectories returned by our algorithms on the PR2 to solve complex manipulation tasks that require rearranging objects on a refrigerator shelf before reaching inside to grasp and extract a desired object. Our work takes a step towards solving MAMO problems in realistic real-world workspaces and we conclude this thesis by discussing some possible directions for future work to develop even more capable and versatile MAMO planning algorithms.

Acknowledgments

The work in this thesis is the result of several years of effort during which I was supported by many exceptional people. My advisor, Maxim Likhachev, helped me find and refine the problems that interested me. He gave me the time and space I needed to explore ideas that sometimes led nowhere, put me in position to collaborate with other members of the lab on exciting projects, encouraged me to keep up my extra-curricular pursuits to maintain a healthy lifestyle, and most of all answered any questions I had and taught me the ins and outs of motion planning for real robot systems. By all accounts Max is the best advisor I could have hoped for for myself. He is super smart, very kind, and annoyingly funny. It was a pleasure working with him for six years and I am forever grateful for his advice.

I want to thank my thesis committee, all of whom have quite literally laid the groundwork for the work I did during my graduate studies. Chris Atkeson is a wealth of knowledge who always asks the right questions. In all my meetings with Chris, within minutes he gave me insights into the problem I had been working on for months. He is exceptional at helping you assess the utility of systems and algorithms you develop when it comes to deploying them on real robots, taking into account all the errors and uncertainties that show up in the real-world. Thank you Chris, for making me think more critically about my work and explain it in terms that helped me understand it better and improve it further. I first met Oliver Kroemer at a conference before he started as faculty at CMU. His passion for making robot arms do quite possibly everything you could imagine came through then as it does now. I was fortunate to have his advice throughout my research, and he always helped me break down difficult manipulation problems into pieces that I could feasibly solve. No research problem seems too daunting or too impossible to Oliver, which is extremely reassuring to a young researcher trying to finish their graduate studies. Thank you Oliver, for always reassuring me that I was capable of solving difficult problems. Finally, not too long ago Mehmet Dogar was a graduate student like me working on solving manipulation problems among movable objects. Mehmet has such great intuition about the work in this thesis that I never needed to explain the setup and motivation for my research to him. We were able to dive into the details of my algorithms and figure out both ways to address corner cases and ways to improve general performance. Thank you Mehmet, for the work you and your students have done that continues to inspire me, for believing in my work, and for teaching me the value of incremental progress towards solving challenging problems.

I was fortunate to be part of the Search-Based Planning Laboratory for the last six years. My time overlapped with many exceptional people and one Andrew Dornbush. I hope I have the privilege of working with someone as intelligent, proficient, and humble as Andrew everywhere I go. I want to thank Suhail in particular for sharing a lot of time, ideas, and meetings with me over the years – you directly contributed to a fair bit of work in this thesis and for that you forever have my gratitude. I want to thank Tushar, Shohin, Yash, and Oren for the work we did together. Shohin and I had an excellent trip to Japan a few years ago and I fondly remember all the food we ate, and also the cool work we did. To the other members of SBPL, old and new, thank you for always keeping the lab a fun place to work in. Thank you Venkat, John, Sung, Kalyan, Vinitha, Fahad, Anahita, Ishani, Aaron, Raghav, Wei, Allen, Aditya, Manash and Jacky – you left the lab a better place. Thank you to Ram, Shivam, Rishi, Itamar, Yorai, and Hanlan for giving me faith that the lab remains in good hands – my offer for a crash course on working with the PR2 stands as long as I stay in Pittsburgh.

I want to give the PR2 its own moment of glory. Its a fantastic robot, a bit temperamental at times, but I had a lot of fun working with it.

I feel an immense amount of gratitute and appreciation for all my friends who make me a better person every day. Arjav, you are the closest friend I have and our lives have been intertwined for over a decade at this point. I cannot thank you enough for the constant support you provide, through all the ups and downs, I can always count on you being there for me. There are a few friends that have been around since day one of grad school that I will always have a good time with, regardless of whether we are sitting at home doing nothing but chit-chatting, or whether we are out and about in the great outdoors, or whether we are out eating and drinking. Thank you to Achal for asking me deep questions early on and for sharing your wit and joy at all times; to Anirudh for always doing everything with a smile on your face, for answering many of my research questions, and for always making sure everyone is having a good time; and to Rosario for a seemingly infinite capacity of empathy, goodwill, humility and grace. I want to thank Vishal, Xuning and Ankit for many of the most memorable moments early on in grad school and many more times over the last decade. I want to thank Shaurya, Roberto and Alex for hours of fun and camaraderie on and off the football/soccer field. I want to thank Pragna, Abhijat, and Tanmay for giving me energy while I felt I was aging out of grad school. I want to thank the Hidden Harbor crew – Nick, Emily, Rob, Jenny, and Roman – for the most interesting, varied and informative topics of conversation, all while we drank many cups of rum week after week.

Finally I want to thank my love, Cara, for her support, encouragement, and love over so many years, and for all the years to come. You kept me sane and you kept me happy, and I hope I can make your life more joyful like you have mine. I would never have gotten this far if it was not for my family – my parents and my brother. In more ways than I can enumerate, they have allowed me to grow into the person I am today, they have inspired me to achieve more for myself than I knew I was capable of, and they have always wanted the best for me no matter the distance between us. I love you all very much.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Robots that manipulate objects in their environment are commonplace in industrial and warehouse automation, and are expected to achieve similar levels of integration in domestic scenarios. The core aspect of their deployments in these workspaces is that they act upon the environment in which they operate by manipulating the objects therein. This may happen in the form of a series of industrial arms assembling parts, a mobile manipulator packing boxes to be shipped in a warehouse, or a home robot retrieving cooking ingredients from pantry and refrigerator shelves. Manipulation planning algorithms for these tasks must be capable of physics-based reasoning to ensure safety and stability during robot operation. Intermediate sub-assemblies put together by robots and objects packed in boxes should create stable structures, while a home robot should be careful not to drop or break or tip over fragile containers. Since the effect of robot actions on reconfigurable environments is tied to the task at hand, planning algorithms must account for these effects to ensure the task is completed safely.

In order for a robot to solve the fundamental task in robot manipulation of *pick-and-place* – where the goal is for a robot to grasp a desired object and relocate it to a specified location – it must be able to deal with obstructing clutter because it is unrealistic to assume the robot will always have contact-free access to grasp the desired object. In cluttered environments, access to the desired object may be blocked

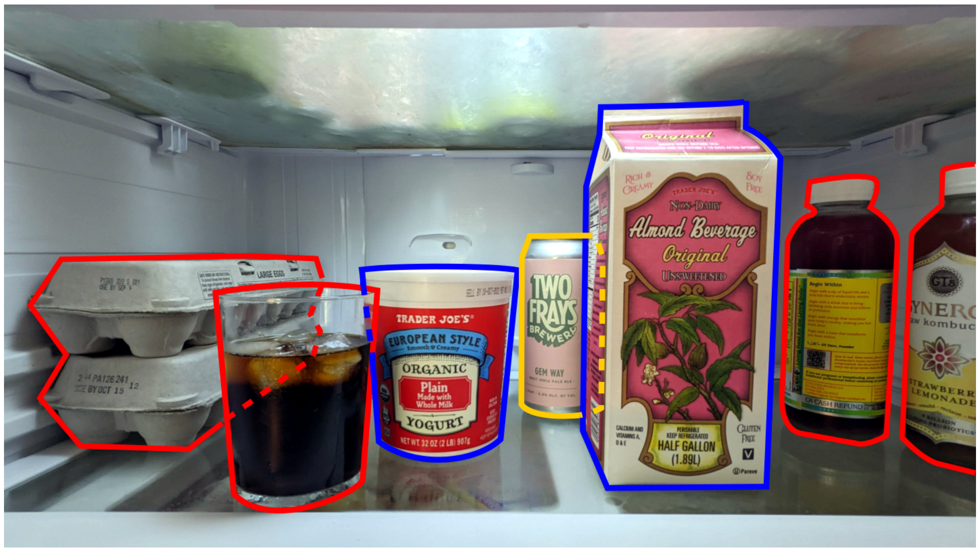


Figure 1.1: The goal is to retrieve the beer can (yellow outline). The yogurt and almond beverage are movable objects (blue outlines). All other objects in the scene are immovable obstacles (red outline).

by multiple other objects that need to be rearranged. The robot must reason about how best to rearrange this obstructing clutter in order to complete the pick-and-place task. Consider a robot that must extract the can of beer (outlined in yellow) from the cluttered refrigerator shelf in Figure 6.1. The robot cannot reach the can without moving aside a box of almond beverage and/or a tub of yogurt (both outlined in blue). However, it must take care in doing so because the shelf also contains fragile objects such as eggs, a glass of coffee, and two glass bottles (all outlined in red) that should not be made contact with by either the robot or other objects. The simply stated task of *"retrieve the can of beer"* suddenly becomes a lot more complex to solve. The motion planning algorithm must find a delicate sequence of rearrangements to clear enough room in the workspace to solve the task.

When robots make contact with their environment, they can do so via *prehensile* or *non-prehensile* actions [14]. Prehensile contacts stabilise the contacted object through the contact forces alone, independent of external forces like gravity. Grasping and holding an object is perhaps the simplest example of prehensile interactions with the environment. The physics of such interactions can be modeled accurately under the assumption that the object model is known, and the contact is 'rigid', i.e.

grasping an object rigidly attaches it to the robot arm at the point(s) of contact. If a robot could complete all tasks assigned to it in a reasonable amount of time via prehensile interactions only, existing planning algorithms for robot manipulation would go a long way to enabling their deployment across all scenarios.

However robots, like humans, are limited in the variety of objects they can grasp. Some objects may be too big or too bulky for a robot to interact with using only prehensile actions. Moreover, in many cases it is more time- and energy-efficient to manipulate objects in the environment via non-prehensile actions such as pushing and sliding, tilting, pivoting etc. For example, it may be simpler to push an object off to the side to reach another one than to grasp it, pick it up, move it elsewhere, place it down, and release it before going back to reach the second object.

In the case of our example in Figure 6.1, a planning algorithm could compute a sequence of pick-and-place rearrangements for the almond beverage and yogurt to solve the task. However, this is only possible if we assume that both objects can be grasped (i.e. fit inside the end-effector of the robot), we know good grasp poses to grasp them both from, the robot can exert enough torque to lift them, and we have knowledge of and access to stable locations to place the objects. Any of these assumptions may easily be violated in real-world scenarios and in such cases we would still like to be able to solve the problems of interest to us.

Instead of a sequence of prehensile rearrangements, our planners could alternatively solve the problem by rearranging the scene via a sequence of pushing actions. This requires us to be able to forward simulate the effect of robot actions on the configurations of objects in the scene – we would not want to inadvertently push the almond beverage into the glass bottles on the right or the yogurt into the coffee on the left. The mechanics of such non-prehensile interactions is not well understood, especially when we want to model complex multi-body interactions in 3D. There is work on creating analytical models of planar pushing of objects with simple geometries [97], but this becomes intractable for arbitrary 3D object models and multi-body interactions (e.g. when an object being pushed by a robot pushes other objects in turn).

It is still possible to use the simpler physics models to propagate the dynamics of complex multi-body interactions. Physics-based simulators [27, 95, 153] do this by maintaining a necessary set of physics parameters for all objects in the environ-

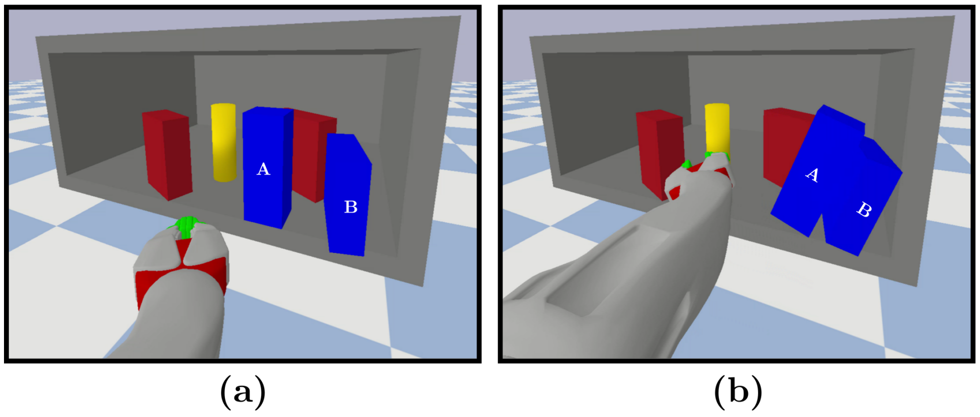


Figure 1.2: (a) The scene prior to the robot pushing object A to the right. (b) The resultant scene after the push.

ment. These parameters include the intrinsic object dynamics (e.g. mass, velocity, acceleration, frictional forces) and extrinsic interaction dynamics (e.g. contacts and contact forces). The complex multi-body interaction dynamics can then be simulated by integrating all simple physics models with a sufficiently small discretisation of time. This makes physics-based simulators computationally expensive to query and thus their use within planning algorithms introduces a challenging bottleneck.

The planning algorithms we present in this thesis use a physics-based simulator in-the-loop during planning to keep track of the state of the environment as the robot acts in it. Figure 1.2 shows the result of simulating a push action – when the robot decides to push object A towards the right, querying our physics-based simulator tells us the result of that action will lead to object A leaning on object B which in turn will lean on the right wall of the shelf. By keeping track of the state of all objects in the scene, our planners can find an appropriate sequence of actions that rearrange movable clutter and retrieve the object-of-interest (OoI) from the workspace. If the planner needs to satisfy object-centric "interaction constraints" on whether the robot can make contact with an object, how far it can make objects tilt, whether objects are allowed to topple over, how fast they can be moved etc. it can determine the validity of an action depending on the result obtained after querying the physics-based simulator. In this way, the planner would be able to solve the problem in Figure 6.1 by carefully pushing aside the movable objects (outlined in blue) just enough to clear up room to retrieve the OoI (outlined in yellow) without the movable objects or the robot making contact with "immovable" obstacles (outlined in red).

1.2 Thesis Overview

The algorithms we develop in this thesis attempt to solve exactly the kinds of problems described above. We would like robots to *solve manipulation planning tasks with a physics-based simulator in-the-loop to forward simulate the environment dynamics.* Our model-based planning algorithms use a physics-based simulator in-the-loop to propagate the multi-body interaction dynamics resulting from robot rearrangement actions executed in the environment. In particular, we require our planning algorithms to compute trajectories that satisfy all object-centric interaction constraints specified as part of the problem. In the 3D workspaces we care about, these encode properties of objects such as whether the robot can make contact with an object, how far it can make objects tilt, whether objects are allowed to topple over, how fast they can be moved etc. The challenge in solving simple pick-and-place tasks in these environments lies in finding a solution trajectory that satisfies these constraints at all points in time. This domain is called "Manipulation Planning Among Movable Obstacles", or MAMO [144, 165]. These algorithms must be efficient about when they query the simulator since that is the computational bottleneck. Typical manipulation planning problems may require the robot to evaluate hundreds of thousands of actions. Simulating all of them in problems that we care about would take an inordinate amount of time. We have developed algorithms that query simulators when absolutely necessary, and do so by reducing the general problem statement to a more tractable alternative, and adding constraints on which robot actions may be simulated.

We briefly state the contributions of the work included in this thesis in the sections below, with detailed technical exposition in the chapters to follow. Chapter 2 provides information on prior work in robot manipulation planning, MAMO and its predecessor "Navigation Among Movable Obstacles" (NAMO), and the use of physics-based simulators for planning. It also provides a basic introduction to some of the algorithmic techniques used in our work. Chapter 3 presents our work from [130] on simulation-based planning with adaptive motion primitives. The work in Chapters 4–6 relies on a connection between multi-agent pathfinding (MAPF) and MAMO to aid the planning algorithms in deciding which objects should be rearranged and where they should be moved. We introduce this connection in Chapter 4 to present our **M4M** algorithm from [128], use it to formulate a graph search-based solver E−**M4M** [127] for MAMO in Chapter 5, and generalise this further by introducing the use of prehensile rearrangements and object parameter uncertainty (imprecise knowledge of object masses and coefficients of friction) in Chapter 6. Finally Chapter 7 presents a summary of the ideas explored in this thesis and possible extensions to this line of work.

This thesis does not include details of two projects the author completed during his doctoral studies. We exclude our work on developing a general anytime, multi-heuristic, multi-resolution graph search algorithm AMRA* [131] from this thesis. AMRA* generalises several existing search algorithms [3, 40, 50, 87, 106, 119] into one unified algorithm. It is capable of searching a state space at multiple levels of discretisation, share information between multiple heuristics, and improve the quality of the solution found over time. We also exclude work done on learning autonomous driving policies in dense traffic [7, 129]. These works are not related to the contents of the thesis but were completed contemporaneously.

1.2.1 Simulation Constraints on Robot Actions

The first contribution of this thesis (Chapter 3) is to show that a large class of MAMO problems is solved by only allowing physics-based simulations for *adaptive motion primitives*. These actions are computed on-the-fly during the planning process to achieve some desired goal (or intermediate subgoal). We develop a strategy to leverage minimal precomputation prior to planning to reduce time spent querying a physics-based simulator by up to two orders of magnitude over state-of-the-art MAMO planning algorithms [130].

The class of MAMO problems solvable with this approach usually have a large volume of object-free space in the workspace. This is true for all tabletop manipulation scenes (since object-free volume above the table is unbounded), and also for shelves that are quite tall or shallow relative to the objects kept in them. It is not necessary to restrict ourselves to top-down grasps for pick-and-place tasks in these workspaces and in fact our experimental setup limits the robot to perform side-on grasps. The assumption about large object-free volume only helps avoid contact with objects until the robot is “close enough” to the grasp pose. Adaptive motion primitives help the

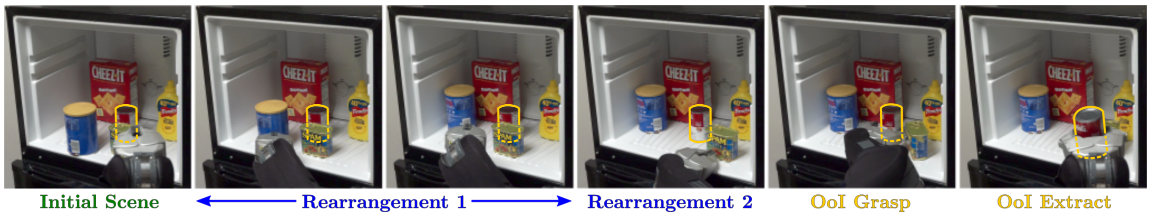


Figure 1.3: A PR2 robot executing a solution trajectory found by **M4M** for a real-world MAMO problem.

robot achieve the grasp pose from such a configuration, and our work shows that it suffices to restrict simulations to these actions only.

1.2.2 Abstracting MAMO Problems with Multi-Agent Pathfinding

Multi-agent pathfinding (MAPF) algorithms attempt to solve the problem of finding paths for a team of robots or 'agents' from their start locations to a set of goal locations. Chapter 4 presents our work from [128] where we develop the key insight that we can leverage existing MAPF solvers for the MAMO domain. This is done by pretending that the objects the robot is allowed to move in a MAMO problem are actuated agents themselves. We use MAPF solvers to compute a solution for this abstracted version of the MAMO problem, and develop an algorithm that is capable of 'realising' this solution in the real-world where only the robot is actuated.

The benefit of the MAPF abstraction is that – provided we can solve the abstract problem – if the movable objects were truly actuated, the robot trajectory computed prior to solving the MAPF problem solves the MAMO problem. In this way the MAPF solution helps guide us toward a rearrangement of the scene that lets us solve the MAMO problem. We present the *Multi-Agent Pathfinding for Manipulation Among Movable Objects* algorithm (**M4M**), a greedy algorithm for solving MAMO problems that uses the solution to an abstract MAPF problem to generate candidate rearrangement actions. Figure 1.3 shows a sequence of images from the solution trajectory found by **M4M** for the PR2 robot on a real-world MAMO problem.

1.2.3 A Graph Search Formulation for MAMO

One of the main challenges in solving MAMO problems lies in the combinatorial explosion of the search space with increasing numbers of movable objects in the scene. Since MAMO solvers are required to keep track of the configuration that all movable objects in the scene are in at any given point in time, the search space for solution grow with each movable object added to the scene. Our work in Chapter 4 dealt with this by "planning in the now" [65], an idea that commits to executing the first valid action found. This makes the algorithm greedy with respect to successful pushing actions that rearrange movable objects. However it precludes a systematic search over different orderings of movable object rearrangements, different ways to rearrange the same object, and different potential rearrangements of the same scene.

Chapter 5 addresses this issue by solving MAMO problems via a best-first graph search. The Enhanced-**M4M** algorithm [127] (E−**M4M**) for MAMO problems searches over orderings of object rearrangements, different rearrangements of the scene, and different ways to rearrange each object. Additionally we describe algorithmic improvements that help speed up the search by caching the result of all actions simulated during planning.

1.2.4 Solving MAMO Problems with Diverse Action Spaces and Object Parameter Uncertainty

The **M4M** and E−**M4M** algorithms as presented in Chapters 4 and 5 only make use of non-prehensile or pushing actions to rearrange movable objects. They also assume perfect knowledge of all object parameters, including their masses and coefficients of friction, in an attempt to minimise the *sim-to-real gap* [18, 62], something commonly encountered in works that try to use physics-based simulators as proxies for the real-world.

Given the graph search formulation for MAMO used by E−**M4M**, it is relatively straightforward to introduce a diverse action space that allows robots to rearrange the environment in different ways such as pick-and-place actions or the use of additional tools. Chapter 6 allows our robot to rearrange movable objects via prehensile actions in addition to the pushes we considered hitherto. This increases the computational cost to solve the same problem by increasing the branching factor of the graph search. We show that being "lazy" with respect to action simulations can help offset some of these costs.

There has been a lot of advancement in localising objects in a scene with great accuracy [155, 167], and some existing MAMO solvers are able to deal with localisation inaccuracies [35]. Yet there is little work in addressing uncertain measurements of physical properties of objects. In particular, we are interested in relaxing the assumption on perfect knowledge of object masses and coefficients and friction. Object masses are difficult to perceive and can change over time, while coefficients of friction are difficult to measure accurately and change with every surface the object is placed on. Our use of physics-based simulators allows us to simulate several actions in parallel and the extension to E−**M4M** in Chapter 6 leverages this to instantiate several copies of the same object with different sampled values for mass and coefficient of friction. We can then validate/invalidate actions based on the number of samples for which they succeed, thereby making us more robust to uncertain object parameters.

1.2.5 Discussion and Future Work

Chapter 7 includes a thorough discussion about the work in this thesis and tries to provide some insight on how it can be extended. We discuss ideas on incorporating heuristics into search algorithms for MAMO, interleaving planning and execution to deal with real-world uncertainties such as occlusions and errors in sensing, execution, and environment modeling, and finally on incorporating model-based rearrangement actions designed for manipulation tasks in clutter.

Chapter 2

Background

2.1 Related Work

The work in this thesis extends existing literature on “Manipulation Planning Among Movable Obstacles”, a domain where a robot arm has to perform a manipulation task (such as pick-and-place of a desired object) in a reconfigurable environment. This domain is closely related to “Navigation Among Movable Obstacles”, where the robot has to complete a navigation task in a reconfigurable environment. Several approaches to both problem domains make use of a physics-based simulator in the planning loop to keep track of the state of the environment. This section provides information on prior work in these fields.

2.1.1 Manipulation Planning Among Movable Obstacles

Wilfong [165] showed that when goal positions for the reconfigurable objects are specified, the MAMO problem is PSPACE-hard and is otherwise NP-hard. Much of the early work in this domain [5, 114, 149, 165], restricted itself to planar environments and geometric solutions. They relied on the ability to compute the joint configuration space of the robot and the movable objects. Furthermore, planning problems were intialised were known sets of grasp configuration for each object and restricted to prehensile manipulation of movable objects.

More recently, Stilman et al. [143, 144] have solved manipulation and navigation problems in 3D workspaces with movable obstacles with prehensile rearrangements only. These solvers search for solutions backwards from the goal state, and resolve conflicts relating to obstructing objects by incorporating prehensile manipulation plans for their rearrangement. When the robot has access to an intermediate "buffer" location to relocate obstructing objects, pick-and-place rearrangement of movable obstacles is possible [86, 163], however this is not always the case in the problems we consider in our thesis.

Most work that uses non-prehensile actions limits interactions to a plane [35, 70, 159]. The solution trajectories computed for these planar problem usually only plan for the robot end-effector. They are 'lifted' to a full joint-space trajectory by solving a constrained inverse dynamics problem, where the constraint enforces the end-effector move in a plane [15].

2.1.2 Non-prehensile Manipulation

Early work by Mason [97] on the mechanics of pushing established conditions for the movement of objects in a plane subject to frictional and external (pushing) forces. They showed that the direction of rotation of the object (clockwise or counter-clockwise) is determined based on the 'votes' of the external force vectors and the vectors that define the friction cone at the point of contact. This analysis was extended in [93] where a search-based planner for pushing polygonal objects to a goal configuration in a plane was introduced. Pivoting and toppling of objects in order to orient them into a known and observable or desired pose was explored in the context of automated parts feeding using conveyor belts or hoppers [4, 169] following the analysis of toppling manipulation in [92].

Dynamic non-prehensile manipulation, where the motion of an pushed object is not restricted to follow that of the object pushing it, can lead to complex behaviours such as rolling, sliding, and throwing objects [94, 123, 166]. Although we do not utilise such actions in our work, given accurate robot and object models, physics-based simulators can forward simulate the effect of these actions, albeit to a lesser degree of accuracy than quasi-static actions. Since we assume access to such models in our work, such actions can be integrated into the algorithms we develop in this thesis.

The reliance on accurate models can be relaxed by model-free learning-based approaches that help planning algorithms predict the effect of non-prehensile actions [82, 90, 115]. The learned models can help close the loop when it comes to planning a sequence of actions directly from sensor information, provided that they generalise well across all possible scenes observed by these sensors.

2.1.3 Rearrangement Planning

Rearrangement of movable objects in the scene is a defining characteristic of MAMO problems. In our work, we rearrange objects in the scene without any explicit information about desired goal poses for these objects since they only act as obstructing clutter for the pick-and-place style tasks we consider. However, the domain of rearrangement planning focuses on problems where goal poses are specified for some (or all) of the movable objects in the workspace [9, 78]. Given the right action space, sampling-based planners have been shown to solve 2D rearrangement planning problems while querying a physics-based simulator for the effects of non-prehensile actions [54, 69]. The ability to sample multiple actions from a state and roll them out in a simulator allows these algorithms to reason about modeling, sensing and execution uncertainty as well [71, 76].

There is a rich body of work that uses model-free learning to solve the rearrangement planning problem in different ways. A learned model can be used to determine the feasibility of robot actions by predicting collisions [30, 121]. They are trained by conditioning them on a specified goal configuration as input. Since these models are fast to query, they can quickly help eliminate infeasible actions and select feasible ones that help achieve the desired goal. Learned models have also been used to directly [172] or indirectly [174, 175] predict robot actions for rearrangement planning. Direct prediction of robot actions involves end-to-end learning of an execution policy from sensory input [172]. This can be hard to generalise to a variety of workspaces, objects, robots, and sensors. A more abstract learning model predicts parameters that inform the selection of robot actions. These parameters can take the form of desired displacements of specific objects in the scene [175], which may be achieved by local controllers the robot has access to. Alternatively, a learned model can predict spatial relations between the objects present in the scene as logical formulae [174] which can be passed to a symbolic planning algorithm for actually planning the sequence of robot actions given a goal specification.

2.1.4 Simulation-based Planning

Simulators are used within planning algorithms for problems when the effect of robot actions on the state of the environment cannot be computed easily. In such cases, physics-based simulators are a useful tool as they encapsulate simple physics models for all interactions between pairs of bodies in the scene. The notion of ‘push grasping’ and ‘negative goal regions’ was introduced for these problems in a line of work by Dogar et al. [33, 34, 35]. King studied rearrangement planning with non-prehensile actions extensively in their thesis [68] with a 2D physics simulator in their planning loop. Sucan and Kavraki [147] developed a randomised planning algorithm for cases with a computationally expensive transition model (querying a physics-based simulator is an example of such a model). This algorithm was extended in [101] to deal with uncertainty in some model parameters. Computing robust trajectories in the presence of parameter uncertainty has also been formulated as a multi-armed bandit problem [76], and a convergent planning problem [2] borrowing prior work from [64]. In other applications, simulators have been used to compute the effect of higher-level and longer-horizon robot ‘skills’ [176], and to determine low-level feasibility of a high-level discrete plan [118].

2.1.5 Contact-Based Trajectory Optimisation

Trajectory optimisation methods that are able to reason about contacts with the environment have received a lot of attention when planning trajectories for humanoid and quadruped robots [43, 99, 158]. These methods find a locally optimal solution around an initial trajectory that satisfies constraints on the sequence of contacts to be made and the forces applied through them while satisfying kinematic and dynamic limits of the robot. Typically it can be computationally expensive for these methods to converge to a solution since they require access to gradient-based information about good ‘descent’ directions to reach the local optimum. Contacts are discrete events that are difficult to detect for these continuous optimisation techniques and only help the optimiser take gradient descent steps towards the optimum while the specific contact is ‘active’. In practice, often a “soft”-contact model is used [107, 152] that provides information about virtual forces from a distance and can be used to converge to the optimum faster. Other works include contacts as complementarity constraints [120] which encode the property that contact forces between two bodies can be non-zero only when the two bodies are in contact, or recently as pressure fields [81] where contact forces are determined based on the volumetric overlap between rigid bodies with hydro-elastic properties.

Contact-based optimisation of a robot arm trajectory for manipulation also requires updating the state of other movable objects in the workspace and ensuring any interaction constraints are satisfied. For simple planar applications, analytical models describing the dynamics of contacts between a robot end-effector and a single object being pushed can be used to compute pushing trajectories [57, 100]. Analytical models have also been used to optimise trajectories for dexterous in-hand manipulation of objects [98], albeit at a high computational expense. If analytical models are inadequate at describing the dynamics of complex multi-body interactions in cluttered workspaces, an alternative approach numerically computes gradients via finite differencing and often uses a physics-based simulator in-the-loop to observe the state of the movable objects in the scene [73, 124].

2.1.6 Learning-based Methods for Manipulation in Clutter

The degree of difficulty in solving MAMO problems is dictated by the complex multi-body interactions between the robot and objects in the workspace that make it hard to find a valid rearrangement action that also satisfies all interaction constraints. Work on using machine learning to train models suited for this task hopes to generalise over the different configurations we might encounter in any MAMO problem. Existing work on learning non-prehensile pushing policies has thus far been limited to planar robot-object interactions [91, 117, 173] and does not capture the complexity of toppling objects [92]. Learned models have also been employed to solve simulated task-and-motion planning problems [112, 141] that require longer-horizon reasoning over both discrete decisions about which objects to manipulate and continuous trajectories for the robot.

Recent work has looked at solving more complicated rearrangement planning problems on real robots over long horizons [88, 122]. These methods rely on prehensile or pick-and-place rearrangement actions only as that simplifies the learning problem to one that need only figure out a good sequence of rearrangements and perhaps predict good grasp and placement poses for objects. Given a desired object to be rearranged, and grasp and placement poses for it, we can query a motion planner for robot trajectories to carry out the rearrangement. For the MAMO problems we consider in this thesis, we must reason about non-prehensile robot-object interactions in the 3D workspace, long sequences of rearrangements, and complex multi-body interactions, all while ensuring we do not violate interaction constraints. This is a particularly challenging problem to model, even for learning-based methods with powerful generalisation capabilities. However, the recent progress shown by methods for rearrangement planning, especially those that use graph neural networks and point clouds as input show promise for the complicated MAMO problems we target.

2.2 Discrete Graph Search

All of the algorithms used in this thesis are discrete search algorithms that find solutions for planning problems on an appropriate discrete graph representation of the search space. This section provides a brief overview of this category of algorithms at a high-level. Interested readers are encouraged to look up a rich history of literature in this field and details of many more discrete search algorithms in [41, 113].

Algorithm 1 Discrete Graph Search

```
1:  procedure SEARCH($v_{\text{start}}, V_{\text{goal}}, f$)
2:      $OPEN \leftarrow \emptyset$
3:      Insert $v_{\text{start}}$ into $OPEN$ with priority $f(v_{\text{start}})$
4:      while $OPEN$ is not empty and time remains do
5:          $v \leftarrow OPEN$.TOP()
6:          if $v \in V_{\text{goal}}$ then
7:              return RECONSTRUCTPATH($v$)
8:          for $v' \in$ GETSUCCESSORS($v$) do
9:              if EVALUATEACTION($v, v'$) then
10:                 Insert/update $v'$ in $OPEN$ with priority $f(v')$
11:     return $\emptyset$
```

Algorithm 1 contains a pseudocode for a discrete graph search algorithm. The

Search method is a basic graph search over a discrete graph $G = (V, E)$ where vertices $v \in V$ are search *states* (V is thus the search *space*) and edges $e = (u, v) \in E$ represent actions/transitions that take us from state u to state v. Search takes three input arguments – v_{start} is the start state or root of the search tree from where we would like to find a path, $V_{\text{goal}} \subset V$ is the set of goal states where a solution path may terminate, and f is a priority function that preferentially selects more promising states (usually in terms of "closeness" to V_{goal}) to grow the graph from.

$OPEN$ is a priority queue that contains states ordered according to the function f. The function GetSuccessors : $V \to \mathcal{P}(V)$ returns the set of all states that may be reached from the input state. As such, the set of states returned by GetSuccessors(v) can be considered successors/neighbours of v in G. In order to check whether the transition from v to one of its neighbours v' is valid, we call EvaluateAction(v, v'). If the transition is found to be valid, we may consider further growing the graph (towards V_{goal}) from v'. Once we reach a state in V_{goal}, we can backtrack from it to v_{start} along valid edges in the graph and return the solution path found.

Figure 2.1 shows a graphical illustration of a discrete graph search in four steps. Step (1) shows the graph $G = (V, E)$ representing the problem we want to solve. The search space $V = \{v_{\text{start}}\} \cup V_{\text{goal}} \subset V \cup \{v_i\}_{i=1}^{5}$. Edge set E is shown as gray arrows in the figure. In step (2) we evaluate edges to the successors of v_{start} and find both edges are valid (represented by green arrows). Step (3) proceeds to evaluate the successors of v_1 (preferred over v_2 due to the f-function) and finds two more valid edges, one invalid edge to v_5 (red arrow), and determines that the path to v_2 via v_1 is better than directly going from v_{start} to v_2 (indicated by the blue arrow $\overrightarrow{v_{\text{start}}, v_2}$). Finally in step (4) we solve the problem by finding a path from v_{start} to V_{goal} via v_1 and v_3.

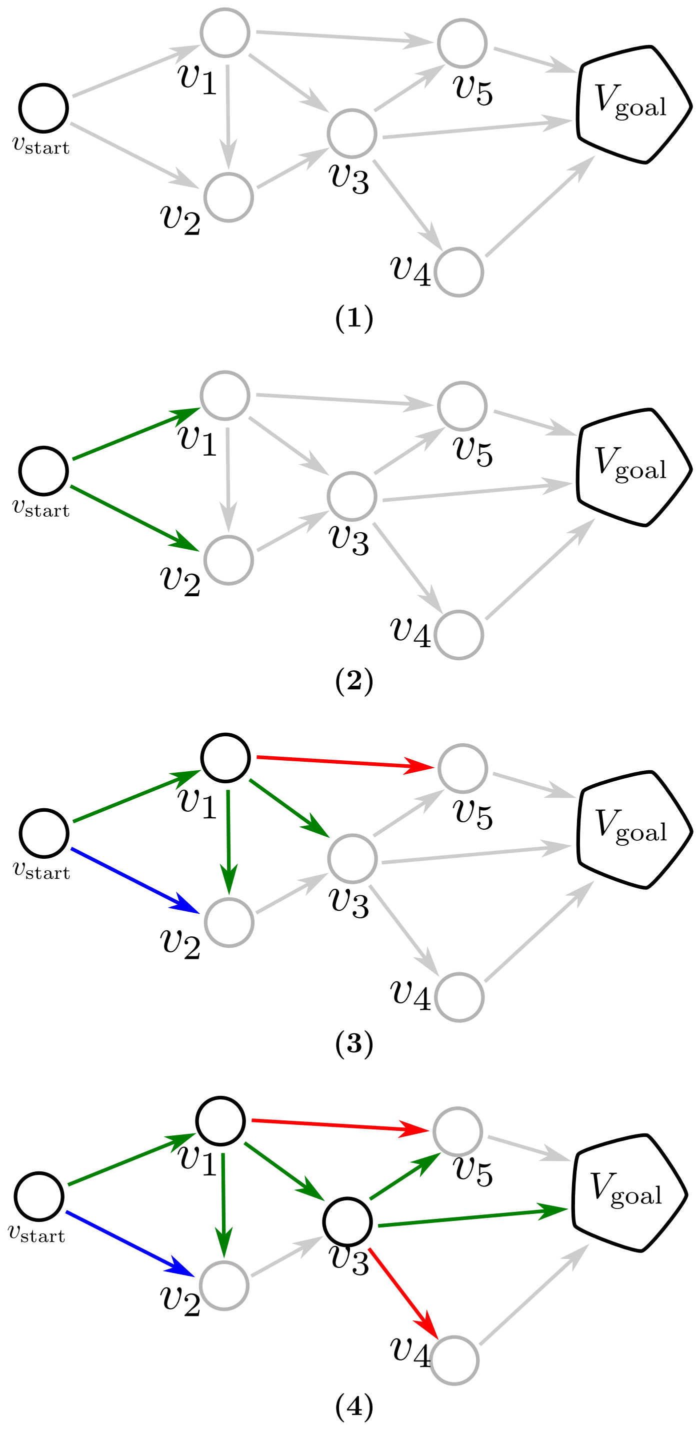


Figure 2.1: A high-level graphical illustration of a discrete graph search algorithm (Algorithm 1).

2.3 Problem Formulation

2.3.1 Search Space

In this thesis, we denote the robot manipulator as $\mathcal{R}$, and $\mathcal{X}_{\mathcal{R}} \subset \mathbb{R}^q$ as the configuration space for a q degrees-of-freedom (DoFs) manipulator. The set of objects in the scene is $\mathcal{O} = \{O_1, \ldots, O_n\}$, and the configuration of any object $\mathcal{X}_{O_i} \in SE(3)$ includes its 3D position and orientation. The search space for a planning problem in the MAMO domain is the Cartesian product of the robot and all object configuration spaces, denoted as $\mathcal{X} = \mathcal{X}_{\mathcal{R}} \times \mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$. We denote movable objects by $\mathcal{O}_M$ and immovable obstacles by $\mathcal{O}_I$ such that $\mathcal{O} = \mathcal{O}_M \cup \mathcal{O}_I$ and $\mathcal{O}_M \cap \mathcal{O}_I = \emptyset$.

2.3.2 Object Constraints

Each object is associated with a set of interaction constraints. For example, an 'immovable' obstacle (an object that cannot be interacted with, such as a wall) will contain a constraint function which is satisfied so long as neither the robot nor any other object makes contact with it. In our problems similar functions encode that movable objects cannot fall off the shelf, tilt too far (beyond 25°), or move with a high instantaneous velocity (above $1\,\mathrm{m\,s^{-1}}$). A state $x \in \mathcal{X}$ is valid if all constraints for all objects are satisfied at that state[1]. We denote the space of valid states by $\mathcal{X}_V$.

2.3.3 Problem Statement

A MAMO planning problem can be defined with the tuple $\mathcal{P} = (\mathcal{X}, \mathcal{A}, \mathcal{T}, c, x_S, \mathcal{X}_G)$. $\mathcal{A}$ is the action space of the robot, $\mathcal{T} : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$ is a deterministic transition function, $c : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ is a state transition cost function, $x_S \in \mathcal{X}_V$ is the start state, and $\mathcal{X}_G \subset \mathcal{X}, \mathcal{X}_G \cap \mathcal{X}_V \neq \emptyset$ is the set of goal configurations.

To solve MAMO problems, we would like to find the least-cost valid path π^* from start to goal i.e., a path made up of a sequence of valid states. Formally, we can write

[1]We omit the necessary robot kinematic and dynamic feasibility constraints from the definition of state validity for brevity.

this as:

$$\begin{aligned}
&\text{find } \pi^* = \operatorname*{argmin}_{\pi=\{x_1,\ldots,x_T\}} \sum_{i=1}^{T-1} c(x_i, x_{i+1}) \\
&\text{s.t. } x \in \mathcal{X}_V,\, \forall\, x \in \pi && \textit{(path of valid states)} \\
&\qquad x_1 = x_S, x_T \in \mathcal{X}_G && \textit{(start, goal constraints)} \\
&\qquad x_{i+1} = \mathcal{T}(x_i, a_i), a_i \in \mathcal{A}, \forall x_i, x_{i+1} \in \pi \\
&&& \textit{(transition dynamics)}
\end{aligned}$$

The cost of robot actions c is proportional to the distance traveled in $\mathcal{X}_\mathcal{R}$. The start state x_s includes a "home" robot configuration in $\mathcal{X}_\mathcal{R}$ and the initial poses of all objects. We assume $\mathcal{X}_G$ is defined in two parts – a grasp pose in $SE(3)$ for the OoI and a goal pose in $SE(3)$ where it must end up (while grasped by the robot). We will provide details about the action space $\mathcal{A}$ and transition function $\mathcal{T}$ as and when appropriate for the algorithms we present in this thesis.

Chapter 3

Planning with Physics-Based Adaptive Motion Primitives

Robot manipulation in cluttered scenes often requires contact-rich interactions with objects. It can be more economical to interact via non-prehensile actions, for example, push through other objects to get to the desired grasp pose, instead of deliberate prehensile rearrangement of the scene. For each object in a scene, depending on its properties, the robot may or may not be allowed to make contact with, tilt, or topple it. To ensure that these constraints are satisfied during non-prehensile interactions, a planner can query a physics-based simulator to evaluate the complex multi-body interactions caused by robot actions. Unfortunately, it is infeasible to query the simulator for thousands of actions that need to be evaluated in a typical planning problem as each simulation is time-consuming. The first contribution of this thesis shows that (i) manipulation tasks (specifically pick-and-place style tasks from a tabletop or a refrigerator) can often be solved by restricting robot-object interactions to *adaptive motion primitives* in a plan, (ii) these actions can be incorporated as subgoals within a multi-heuristic search framework, and (iii) limiting interactions to these actions can help reduce the time spent querying the simulator during planning by up to 40× in comparison to baseline algorithms. Our algorithm is evaluated in simulation and in the real-world on a PR2 robot using PyBullet as our physics-based simulator. The work included in this chapter was published in [130], and the accompanying video can be viewed at this link: `https://youtu.be/ABQc7JbeJPM`.

3.1 Introduction

Manipulation planning problems in domestic households, industrial manufacturing and warehouses require contact-rich interactions between a robot and the objects in the environment. As the amount of clutter in a scene increases, the likelihood of finding a completely collision-free trajectory for the manipulator decreases. This does not mean the task is impossible since we might still be able to complete it by moving the objects around. In these cases, non-prehensile interactions with objects can be much faster than deliberately rearranging the scene via a sequence of slow pick-and-place style prehensile maneuvers. In addition, each object in a cluttered scene is associated with constraints that define how a robot can manipulate it. For example, while we might be allowed to interact freely with a box of sugar, we might not be allowed to tilt or topple a cup of coffee.

We want to enable robots to grasp in clutter by using non-prehensile actions to interact with objects while satisfying any object-centric constraints, e.g. constraints that dictate whether or not we can make contact with, tilt, or topple an object. This domain of Manipulation Among Movable Obstacles (MAMO) [144] is derived from prior work on Navigation Among Movable Obstacles (NAMO) [143, 165]. Planning problems for NAMO aim to find a feasible path between start and goal states for a mobile robot navigating in an environment with reconfigurable obstacles[1]. We focus on the class of MAMO problems where the goal for the robot manipulator is a 6D pre-grasp pose of an object in $SE(3)$ without any constraints on the final poses of the movable objects. The task of planning the grasp itself is not addressed by our algorithm.

3.1.1 Challenges

Motion planning for MAMO is computationally costly because of two major challenges. First, we need to accurately model the dynamics of the robot-object and object-object interactions in the environment during planning. This requires the use of a high-fidelity physics-based simulator since hand-designed analytical models are hard to generalise for complex object geometries and cluttered scenes. The simulator is used

[1]In this work 'objects' may be movable, but 'obstacles' are immovable.

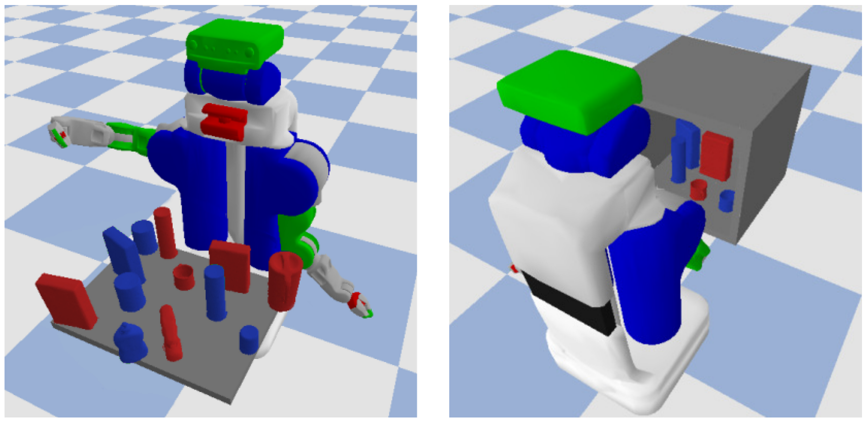


Figure 3.1: Manipulation tasks in cluttered tabletop (left) or refrigerator (right) workspaces require planners to account for complex multi-body interactions between the robot and objects (blue movable objects and red immovable obstacles). The goal is to pickup a randomly selected immovable obstacle.

to model the environment and predict the outcome of actions. The computational cost associated with running a simulator is high, which makes it infeasible to query the simulator for every action that needs to be evaluated. The second challenge is associated with the search space of the problem. Since the robot may interact with objects in the scene and reconfigure them, the search space needs to include the configuration space of all these objects. This makes the search space for a planning problem in this domain grow exponentially with the number of objects, and makes it computationally hard to find a solution.

3.1.2 Contribution

In this chapter, we make an observation that many MAMO problems can be solved effectively by restricting robot-object interactions to adaptive motion primitives and show how this observation can be exploited to structure an efficient search for a contact-rich motion. *Adaptive motion primitives* (AMPs, Section 3.3.3) are long-range actions generated on-the-fly such that they terminate in a valid goal state [24]. In our domain, they are straight lines in the configuration space of the robot, between two states whose end-effector Cartesian coordinates are within δ of each other in Euclidean norm. Since the goal in our domain is defined in the workspace of the robot, an AMP is computed by linearly interpolating between a state that satisfies the δ threshold condition and an inverse kinematics (IK) solution of the goal pose.

Our central assumption in this chapter limits robot-object interactions to the final action (an AMP) in a plan. This restriction limits the class of MAMO problems solvable by our algorithm to ones that require at most one robot action near the goal to make contact with the objects in the scene. We test our algorithm on random initialisations of the cluttered tabletop and refrigerator scenes from Figure 3.1. Empirically our results in Section 3.5 show that even with this restriction, our algorithm solves many MAMO planning problems and is up to $40\times$ faster than competitive baselines in our experiments.

Our main contributions towards robot manipulation planning among movable obstacles include:

- a *two-stage planning approach* where we first sample promising AMPs in parallel, and then systematically use them in our planning algorithm to find a solution.
- the use of these AMPs as *subgoals* within a multi-heuristic search algorithm.
- an action evaluation scheme that minimises the time spent querying a simulator during planning.

3.2 Related Work

The domain of Manipulation Among Movable Obstacles (MAMO) is closely tied to prior work in the field of Navigation Among Movable Obstacles (NAMO) [143, 165]. Lynch and Mason [93] studied the mechanics of pushing maneuvers and used them in a planner to solve early NAMO problems. Past works in the MAMO domain have taken one of two popular approaches - either solving problems via a sequence of pick-and-place style maneuvers [144], or limiting solutions to only planar robot-object interactions [35, 70, 159]. The rearrangement planning problem was extensively studied by King [68] in their thesis which focused on non-prehensile interactions. Manipulation planning in clutter with non-prehensile interactions has also seen solutions that make use of model predictive control [66], human guidance [110], and reinforcement learning [116].

We use a physics-based simulator in-the-loop during planning to account for dynamics of robot-object and object-object interactions in the scene. Plaku et. al. [118]

decompose the planning problem into a high-level discrete space, and a low-level sampling-based planner with a complex dynamics model (physics-based simulator). Zickler and Veloso [176] attempt to solve physics-based planning problems with the help of high-level, long-range robot behaviours. Dogar et. al. [36] simulate and cache multiple robot-object interactions, and use their result during planning to find feasible solutions. However, they do not allow any object-object interactions, which can be unavoidable in cluttered MAMO scenes. Similar to our work, the idea of planning till the proximity of the goal and using a more expensive specialized maneuver from within this proximity can be seen in [157] in the context of grasp planning. Most relevant to our work in this paper are: a sampling-based planning algorithm KPIECE [147] and a search-based planning algorithm Selective Simulation [148]. Each uses different approaches to incorporate a simulator in-the-loop during planning. KPIECE is a sampling-based planner for applications with complex dynamics that uses an importance function over discretised cells of the robot workspace to guide exploration, but calls the simulator for all action evaluations. This is very computationally expensive for MAMO and in our experimental comparisons with KPIECE we show that intelligently limiting the number of simulator queries can improve quantitative performance. Selective Simulation iteratively plans with simulations in a reduced search space (accounting for some objects) and executes the plan in a simulator (with all objects). If any object constraints are violated upon execution, it decides on one of these objects to be added to the search space for the next planning iteration. However, in the original paper Selective Simulation was evaluated for simple constraints of contact or toppling off the table. In addition to these, we consider constraints on how far obstacles can be tilted and how much velocity can be imparted to them. Selective Simulation is also prone to repeated simulations of similar actions which is time consuming and something we explicitly account for in our work with soft duplicate detection. Our experimental analysis also includes comparison with Selective Simulation.

3.3 Approach

In this section we present our work on solving MAMO problems formulated in Section 2.3. Our solution involves simulating only goal-directed AMPs, and deferring simulation of these actions until absolutely necessary.

3.3.1 Graph Representation

We solve MAMO planning problems using a search-based planning algorithm in $\mathcal{X}$. Our graph representation contains two types of actions in $\mathcal{A}$ - *simple primitives* and *adaptive motion primitives* (AMPs). Each simple primitive changes one joint angle of a robot by a small amount[2]. In comparison, an AMP is computed on the fly and can change all coordinates in $\mathcal{X}_{\mathcal{R}}$. A consequence of our core assumption (Section 3.3.4) is that for simple primitives $a_{\text{s}} \in \mathcal{A}$, a valid transition $x' = \mathcal{T}(x, a_{\text{s}})$ implies that the state x and the successor state x' only differ in the robot configuration (in $\mathcal{X}_{\mathcal{R}}$). To be precise, for $x, x' \in \mathcal{X}, a_{\text{s}} \in \mathcal{A}$ and $x' = \mathcal{T}(x, a_{\text{s}})$, x and x' differ only in one robot DoF in $\mathcal{X}_{\mathcal{R}}$. For AMPs $a_{\mathsf{AMP}} \in \mathcal{A}$, a valid transition $x' = \mathcal{T}(x, a_{\mathsf{AMP}})$ can lead to differences in object configurations ($\mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$) in addition to a difference in all robot DoFs in $\mathcal{X}_{\mathcal{R}}$.

We search over the graph $G = (V, E)$ where the vertex set $V \subset \mathcal{X}_{\mathcal{R}}$ and edges $e = (x_i, x_j) \in E$ correspond to actions $a \in \mathcal{A}$ such that $x_j = \mathcal{T}(x_i, a)$. Let $\texttt{FK} : \mathcal{X}_{\mathcal{R}} \rightarrow SE(3)$ be the forward kinematics function for the robot. Similarly let $\texttt{IK} : SE(3) \rightarrow \mathcal{X}_{\mathcal{R}}$ be the inverse kinematics function. For $x \in \mathcal{X}_R$ and some user-defined threshold $\delta > 0$, if the Cartesian end-effector distance to the goal $\|\texttt{FK}(x) - \mathcal{X}_G\| < \delta$, we add AMP edges $e = (x, \texttt{IK}(\mathcal{X}_G))$ to our graph. All other edges correspond to simple motion primitives $a_{\text{s}} \in \mathcal{A}$.

3.3.2 Action Evaluation

Following the ideas outlined in [148], we decompose our action evaluation scheme into a relatively fast collision checking routine and a much slower physics-based simulation. Collision checking involves checking for volumetric overlaps between the collision models of the robot and objects. This is computationally a relatively cheap operation that can be easily implemented with the use of a distance field. If and when necessary, an action that passes this collision checking phase might need to be simulated to determine whether or not it violates any object constraints.

[2]In our implementation, $4°$ or $7°$ depending on the joint.

The set of objects $\mathcal{O}$ in a scene can be separated into two subsets - *movable* objects $\mathcal{O}_M$ that the robot is allowed to interact with, and *immovable* obstacles $\mathcal{O}_I = \mathcal{O} \setminus \mathcal{O}_M$. We assume that this separation is known a priori.

Definition 1 (Phase 1 validity)**.** We say an action $a \in \mathcal{A}$ from state $x \in \mathcal{X}_V$ is *Phase 1 valid* if it does not make contact with any immovable obstacle $O \in \mathcal{O}_I$.

Definition 2 (Phase 2 validity)**.** We say an action $a \in \mathcal{A}$ from state $x \in \mathcal{X}_V$ is *Phase 2 valid* if it is Phase 1 valid and it does not result in any object constraint violations.

The fast collision checking routine is used to determine Phase 1 validity of an action as it can quickly detect overlaps with immovable obstacles. This can also determine Phase 2 validity if there is no overlap with any object. Note that for $a \in \mathcal{A}$, $x \in \mathcal{X}_V$, $x' = \mathcal{T}(x, a)$, we call the collision checking routine for all intermediate states between x and x', including x' but not x. In the case when an action is Phase 1 valid, but also makes contact with some movable object(s), Phase 2 validity can only be determined after simulating the action. This is because we need to account for the complex multi-body interactions that might result upon executing the action. These interactions might violate object constraints due to a movable object-immovable obstacle contact, or the robot violating other movable object constraints such as tilting or toppling. We emphasise that determining Phase 1 validity of an action is computationally much cheaper (around 30ms per action evaluation for a 7 DoF manipulator) than determining Phase 2 validity which requires simulating the action (e.g., around 1.5s per action of a 7 DoF manipulator in PyBullet).

3.3.3 Adaptive Motion Primitives

Adaptive motion primitives (AMPs) are IK-based motion primitives that are generated on-the-fly as part of our algorithm [24] and included in $\mathcal{A}$. For any robot configuration $x \in \mathcal{X}_\mathcal{R}$, if the robot's 3D Cartesian end-effector pose is within $\delta = 0.2$m of the goal end-effector pose $\mathcal{X}_G$, we generate an AMP that tries to connect x to $\mathcal{X}_G$. This is done by obtaining an IK solution $x_G = \texttt{IK}(\mathcal{X}_G) \in \mathcal{X}_\mathcal{R}$, and linearly interpolating between x and x_G. We choose this value of δ based on basic domain knowledge like the size of our workspaces and obstacles therein. We did not tune this value to improve performance. The 3D Cartesian distance between end effector poses $\|\texttt{FK}(x) - \mathcal{X}_G\|$ is computed while accounting for immovable obstacles. This prevents AMPs from being

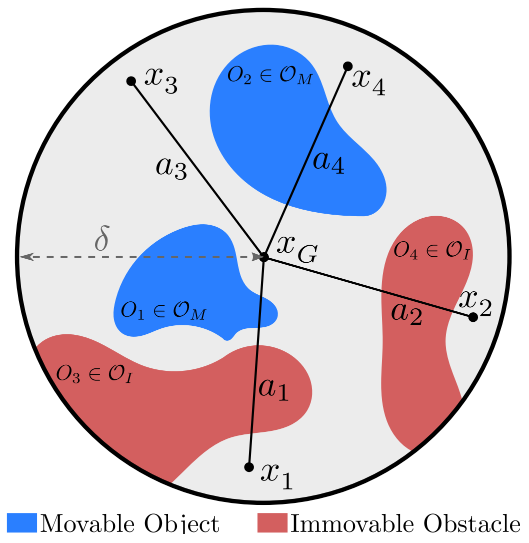


Figure 3.2: For a particular goal state configuration $x_G \in \mathcal{X}_\mathcal{R}$, we can generate AMPs from several states x_i within distance δ from it. This δ-sphere in configuration space $\mathcal{X}_\mathcal{R}$ might be occupied by both movable (blue) objects and immovable (red) obstacles. This can lead to invalid actions a_1, a_2, valid action a_3, and Phase 1 valid action a_4 whose Phase 2 validity will be determined after simulation.

generated from poses close to the goal that have an immovable obstacle in the way. We refer to the set of robot configurations from which an AMP can be generated as the "δ-sphere in configuration space" (around goal $\mathcal{X}_G$).

The validity of an AMP (Phase 1 or Phase 2) is dependent on checking all interpolated states between x and x_G. Figure 3.2 shows our action evaluation strategy from Section 3.3.2 for AMPs. Since AMPs terminate in a goal state, they can only be included in valid paths as the final action.

3.3.4 Assumptions

We make one assumption for solving MAMO planning problems in this chapter which is closely related to AMPs and their use in our search algorithm. In this subsection we hope to provide an intuitive justification for this assumption.

Assumption 1. *We only need to simulate AMPs to find a valid solution for a MAMO planning problem.*

For grasping and reaching in cluttered scenes like those we consider in this chapter,

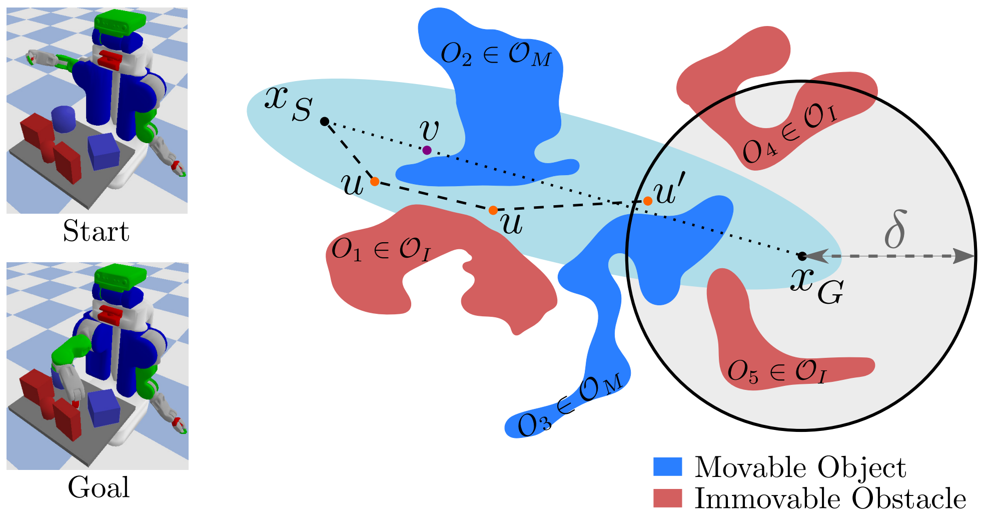


Figure 3.3: Consider planning between start and goal configurations shown on the left. A "greedy" shortest-path search algorithm in the MAMO domain would proceed along the dotted path, exploring states in the light blue region, and spend a lot of time simulating interactions with object O_2 near the purple state v. Due to the assumptions we make, our search algorithm SPAMP proceeds along the dashed path via orange states u, and only starts simulating AMPs from states beyond u' in the δ-sphere around goal x_G.

there is a large volume of object-free space between the start configuration and goal region. Interactions with objects are necessary when the robot is in a region with a high degree of clutter. The tabletop and refrigerator workspaces we consider in this chapter contain clutter near the goal which is often the most pertinent for finding a feasible plan. Based on this observation we delay interacting with objects until the robot reaches a configuration near the goal.

For a preset value of δ, we restrict robot-object interactions until the end-effector is within δ of the goal pose. Since AMPs are long-range actions contained inside this δ-sphere, interactions that are vital to the success of a plan are often the terminal AMPs in a plan (in comparison to the short-range simple primitives which do not lead to meaningful interactions). This leads us to Assumption 1. Consequently, since AMPs are terminal actions, the valid solution paths we find only contain a single action which interacts with obstacles. It is important to note that restricting interactions to a single action does not limit the number of objects the robot can interact with. We illustrate the effect of Assumption 1 in Figure 3.3 by comparing

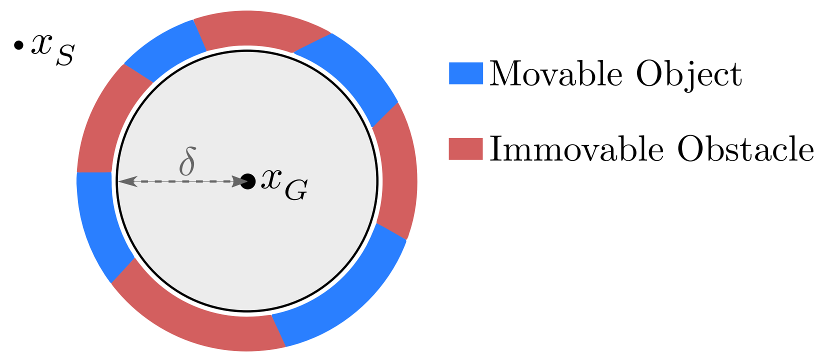


Figure 3.4: In case there is no interaction-free path between start state x_S and the δ-sphere around any goal state x_G, our assumption of only simulating terminal AMPs will cause our algorithm to not return a solution. A greater value of δ in this case could make this problem solvable by our algorithm.

our algorithm against a naive search algorithm.

This assumption restricts the space of MAMO planning problems solvable by our algorithm to those that require at most one AMP to interact with objects near the goal configuration. Our success rates from Section 3.5 suggest that this assumption is not restrictive for the scenes we consider in this chapter (Figure 3.1). In cases where no such configuration near the goal is achievable by the manipulator as shown in Figure 3.4, our algorithm will fail to find a solution. It might still be possible to find solutions in these cases by dynamically changing δ to find a valid AMP, but we have not explored this yet.

Assumption 1 helps us deal with the two major computational challenges for MAMO:

1. The number of calls to the simulator go down significantly, as we now only simulate terminal AMPs that interact with movable objects as opposed to any action that interacts with movable objects.
2. Since only terminal AMPs might be simulated, we can plan from x_S to inside the δ-sphere around some goal configuration purely in $\mathcal{X}_\mathcal{R}$. This means that the search space for finding a path from x_S to x_{T-1} (the penultimate state in a solution path) reduces from $\mathcal{X} = \mathcal{X}_\mathcal{R} \times \mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$ to $\mathcal{X}_\mathcal{R}$, thereby tackling the issue of a prohibitively large search space in cluttered environments.

3.3.5 Subgoals

We solve MAMO planning problems using a search-based planning algorithm. The performance of these algorithms is dependent on the quality of the heuristic functions used. As it is often infeasible to create a single heuristic that can perfectly guide the search from start to goal in all scenarios, it is common practice in high-dimensional spaces to use a multi-heuristic framework. Multiple heuristics guide the search along multiple promising directions, which can help overcome local minima associated with any one heuristic.

Given the fact that simulations are the computational bottleneck in our domain, and the assumption that we only simulate AMPs, it is helpful to guide a search-based planning algorithm to regions of the search space where a valid AMP likely exists. Our two-stage planning approach discussed in Section 3.4 first finds AMPs that are Phase 1 or Phase 2 valid, and generates heuristic functions that guide the search to the beginning of these actions. For an AMP from state $x_{T-1} \in \mathcal{X}_V$, the corresponding heuristic function is a simple Euclidean distance from x_{T-1} in $\mathcal{X}_\mathcal{R}$[3]. This first stage is run in parallel across multiple simulator instances, one per AMP sampled. The second stage runs a multi-heuristic search to find a path from start state x_S to a goal state in $\mathcal{X}_G$.

The beginning of the AMP x_{T-1} can be thought of as a *subgoal* for the planner since we guide the search towards it. While it is not necessary to reach the subgoal, the use of a heuristic centered on the subgoal implies that exploring the search space near it can help find a solution.

If an AMP a_{AMP} is Phase 2 valid, and the subgoal x_{T-1} is reachable without making contact with any object, we do not need to simulate actions on the way to x_{T-1}. It suffices to find a collision-free path in $\mathcal{X}_\mathcal{R}$ from start x_S to x_{T-1}, and append $x_T = \mathcal{T}(x_{T-1}, a_{\mathsf{AMP}})$ for a valid MAMO solution.

If we only have access to Phase 1 valid subgoals, we allow our planning algorithm to simulate any Phase 1 valid AMPs that are generated during the search. To account for the rare case when a subgoal is Phase 2 valid and unreachable (perhaps due to kinematic limits or a scenario like Figure 3.4), we allow our planner to simulate Phase

[3]We refer to the start state of an AMP as x_{T-1} since AMPs are necessarily terminal actions in any potential solution plan between x_{T-1} and $x_T \in \mathcal{X}_G$

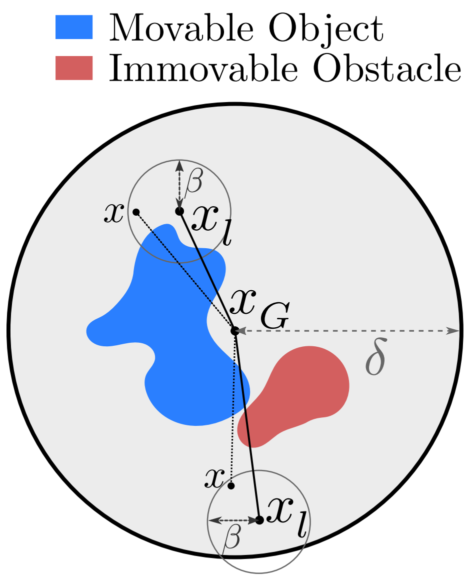


Figure 3.5: If an AMP from any $x_l \in L$ is found to be invalid, we postpone the simulation of AMPs from any other state x that is within β distance from it by inflating the heuristic value of that action.

1 valid AMPs after time t has elapsed during planning (Algorithm 2, Line 3)[4]. This is implemented by maintaining a priority queue of all Phase 1 valid AMPs generated by the search, and simulating them in order after time t.

3.3.6 Soft Duplicate Detection for Action Evaluation

Assumption 1 states that we only simulate Phase 1 valid AMPs during planning. However, since the number of such AMPs in cluttered environments can be very large, we further optimise the calls to the simulator by employing a soft duplicate action detection scheme [39].

Soft duplicate detection estimates the similarity between an action that needs to be evaluated in simulation and an action that has already been simulated and deemed invalid (one which violated object constraints). If they are very similar then it is likely that the new action would also violate some of the same constraints and be invalid. The idea of preferring promising actions based on the validity of similar actions can also be found in [55].

[4]We set $t = 30$s for our experiments.

We maintain a list of states L from which an AMP has been simulated and found to be invalid. For any new state $x \in \mathcal{X}_V$ from which we find a Phase 1 valid AMP, we first compute its Euclidean distances in $\mathcal{X}_\mathcal{R}$ to states in L. Given a preset threshold β, if $\|x - x_l\|_2 < \beta$ for any $x_l \in L$ we postpone the simulation of the AMP from x by artificially inflating the heuristic value of x and re-inserting it into the priority queue it was expanded from. This ensures that x might be re-expanded from the same priority queue at a later time, at which point we would simulate the AMP from it to a goal configuration. Figure 3.5 shows how this process implicitly creates β-spheres in configuration space around states that lead to invalid AMPs due to any constraint violation.

3.4 Algorithm

Algorithm 2 Simulation-based Planning with AMPs (SPAMP)

Input: Planning problem $\mathcal{P}$, number of AMP subgoals N, number of AMP samples M, simulation start time t_{sim}, planning timeout $t_{\max}$
Output: solution path π

```
1: procedure SPAMP($\mathcal{P}, N, M, t_{\text{sim}}, t_{\max}$)
2:     $H \leftarrow$ GetValidSubgoals($N, M$)          ▷ Phase 1 or Phase 2 valid subgoals.
3:     $t \leftarrow t_{\text{sim}}$                     ▷ Simulations are allowed starting from time $t$.
4:     if $|H| = 0$ or $|$IsPhase2Valid$(H)| = 0$ then
5:         $t \leftarrow 0$                              ▷ $|\cdot|$ is the set cardinality operation.
6:     $OPEN \leftarrow$ InitialiseHeuristics($H$)
7:     $\pi \leftarrow$ Plan($\mathcal{P}, H, OPEN, t, t_{\max}$)
8:     return $\pi$
```

Algorithm 2 is a high-level overview of our two-stage planning pipeline. We call our algorithm Simulation-based Planning with AMPs (SPAMP). The GetValidSubgoals subroutine in Line 2 selects subgoals via rejection sampling. M Phase 1 valid AMPs are randomly sampled and simulated in parallel. N Phase 2 valid subgoals are returned (if available), else a combination of Phase 1 and Phase 2 valid make up the N returned subgoals (first stage). This section goes into more details about our specific planning algorithm from Line 7 of Algorithm 2 (second stage).

3.4.1 Multi-Heuristic Framework for MAMO

We use Multi-Heuristic A* (MHA*) [3] as our search algorithm in this work. MHA* maintains multiple priority queues, one for each heuristic that is used. It was originally developed under the assumption that all action evaluations take roughly the same amount of time. This meant priority queues could be selected round robin for state expansions to equitably distribute computational resources across the queues. However, our action evaluations have varying time complexity (checking for Phase 1 vs. Phase 2 validity). Expanding a state from which an AMP has to be simulated is far more expensive than expanding a state from which we do not have to simulate any action. Since some queues might need many simulator calls, and some queues might never query the simulator, a simple round robin strategy would lead to an uneven distribution of computational resources across the queues. For this reason, we prioritise state expansions from queues that the search has spent the least time expanding states from thus far. This time-based prioritisation of queues in MHA* leads to a much more equitable allocation of computational resources for MAMO.

3.4.2 Planning Algorithm

Algorithm 3 contains details from the preceding sections to provide a more in-depth look at our planning algorithm. Details of the MHA* planning algorithm can be found in the original publication [3]. In our algorithm, $OPEN$ is a set of priority queues, each of which is defined by a heuristic function. Following standard MHA* terminology, our *anchor* heuristic is a 3D Breadth-First Search (BFS) heuristic computed from the specified Cartesian goal end-effector position [23], taking into account the immovable obstacles in the scene. Every other heuristic is defined by a corresponding AMP subgoal as per Section 3.3.5. The action set $\mathcal{A}$ is made up of the simple primitives and AMPs via the function `GenerateAMP` which can dynamically generate AMPs if the requisite conditions are satisfied. Simple primitives and AMPs are denoted by a_s and a_AMP respectively (Section 3.3.1).

SPAMP terminates if the next-best state to expand x is in the goal set or we have already found a Phase 2 valid AMP from it. For every other x, the Expand function generates and evaluates all possible successor states of x. These necessarily include successors $x' = \mathcal{T}(x, a_\mathrm{s}) \,\forall\, a_\mathrm{s} \in \mathcal{A}$ (Line 13). In addition, we may generate

Algorithm 3 SPAMP Planner

```
1: procedure Plan($\mathcal{P}, H, OPEN, t, t_{\max}$))
2:     Insert($OPEN, x_S$)                                    ▷ Add to all queues.
3:     while $OPEN$ is not empty do
4:         $h \leftarrow$ BestQueue($OPEN$)                    ▷ Time-based selection.
5:         $x \leftarrow$ BestState($h$)                       ▷ Pop from all queues.
6:         if $x \in \mathcal{X}_G$ then
7:             return ExtractPath($x$)
8:         if $\exists$ Phase 2 valid AMP from $x \in H$ then
9:             $a_{\mathsf{AMP}} \leftarrow$ Phase 2 valid AMP from $x \in H$
10:            return ExtractPath($x$) $\cup \{\mathcal{T}(x, a_{\mathsf{AMP}})\}$
11:        Expand($x, OPEN, t$)
12: procedure Expand($x, OPEN, t$)
13:     for $a_{\text{s}} \in \mathcal{A}$ do                  ▷ Simple primitives only.
14:         $x' \leftarrow \mathcal{T}(x, a)$.
15:         if $x' \in \mathcal{X}_V$ then
16:             Insert($OPEN, x'$)
17:     if $\|$FK$(x) - \mathcal{X}_G\| \leq \delta$ then     ▷ 3D end-effector poses.
18:         if $x$ has soft duplicate then
19:             InflateHeuristic($x$)
20:             Insert($OPEN, x$)
21:             return
22:         $x_G \leftarrow$ IK($\mathcal{X}_G$)               ▷ Inverse kinematics.
23:         $a_{\mathsf{AMP}} \leftarrow$ GenerateAMP($x, x_G$)
24:         if IsPhase1Valid($a_{\mathsf{AMP}}$) then
25:             if IsInteraction($a_{\mathsf{AMP}}$) and $t_{\text{elapsed}} > t$ then
26:                 $x' \leftarrow$ Simulate($a_{\mathsf{AMP}}$)
27:                 if $x' \in \mathcal{X}_V$ then
28:                     Insert($OPEN, x'$)
29:                 else
30:                     Insert($L, x$)
31:             else if not Interaction($a_{\mathsf{AMP}}$) then
32:                 $x' \leftarrow \mathcal{T}(x, a_{\mathsf{AMP}})$.
33:                 Insert($OPEN, x'$)
```

Table 3.1: Quantitative evaluation of simulated tabletop MAMO experiments

		Planning Algorithms					
Metrics	**Scenario**	**SPAMP**	K1	K2	K3	SS	SS2
Success %	Overall	**99%**	91%	92%	85%	87%	91%
Planning Time (s)	Easy	**5 ± 5**	267 ± 301	339 ± 351	211 ± 369	6 ± 20	8 ± 26
	Difficult	**11 ± 16**	318 ± 310	464 ± 416	210 ± 327	22 ± 41	38 ± 55
Simulation Time (s)	Easy	**0 ± 0**	267 ± 301	339 ± 351	42 ± 88	4 ± 18	4 ± 14
	Difficult	**1 ± 7**	318 ± 310	463 ± 415	90 ± 131	4 ± 14	16 ± 31

and evaluate an AMP from x to $x_G \in \mathcal{X}_{\mathcal{R}}$, an inverse-kinematics solution for $\mathcal{X}_G$ (Line 22). This evaluation (Line 23 onwards) occurs if x passes the end-effector distance check (Line 17) and soft duplicate check (Line 18). The `IsInteraction` function returns true if a_{AMP} 'collides' with a movable object during the Phase 1 validity check (Line 24).

3.5 Experimental Results

We run all our experiments on the PR2 robot and use PyBullet [27] as our physics-based simulator. We run experiments in two different workspaces - a tabletop and a refrigerator. The objects in a scene are divided into immovable and movable subsets prior to planning. The robot is allowed to interact with the movable objects but cannot tilt them excessively, cause them to fall over or outside the workspace (table or refrigerator), or impart high velocities. Neither the robot nor any movable object can make contact with immovable obstacles.

3.5.1 Comparative Quantitative Evaluation in Simulation

We compare the performance of our algorithm Simulation-based Planning with AMPs (SPAMP) against relevant state-of-the-art baseline algorithms that can be applied to the MAMO domain - KPIECE [147] and Selective Simulation (SS) [148], which were described in Section 3.2. Three variants of KPIECE were tested. The first two use use the KPIECE implementation from OMPL [146], but differ in how goal biasing is implemented. K1 precomputes a set of 3 valid goal configurations by running IK

Table 3.2: Quantitative evaluation of simulated refrigerator MAMO experiments

		Planning Algorithms					
Metrics	**Scenario**	**SPAMP**	K1	K2	K3	SS	SS2
Success %	Overall	93%	51%	38%	**94%**	87%	91%
Planning Time (s)	Easy	**3 ± 3**	116 ± 293	210 ± 443	167 ± 222	7 ± 22	14 ± 57
	Difficult	**8 ± 13**	58 ± 119	128 ± 335	249 ± 425	57 ± 126	63 ± 157
Simulation Time (s)	Easy	**0 ± 0**	116 ± 293	208 ± 441	50 ± 89	3 ± 18	8 ± 35
	Difficult	**1 ± 6**	57 ± 118	137 ± 334	44 ± 92	20 ± 82	24 ± 62

before planning. K2 runs IK online (with random seeds) every time the search tree is grown towards the goal. K3 is our implementation of KPIECE[5]. The projective space for all KPIECE algorithms is the 3D Cartesian coordinate for the end-effector (via robot forward kinematics). In addition, we implemented a modified version of Selective Simulation (SS2) which includes soft duplicate detection on top of SS.

A planning problem is initialised with objects selected at random from the YCB Object Dataset [16]. Objects are placed in the workspace at random. The goal for the PR2 is to reach a pre-grasp pose for an immovable obstacle. Object masses are obtained from the YCB dataset, and their PyBullet friction coefficients are randomly sampled from the interval $[0.5, 1.1]$ as the dataset does not provide any friction coefficient values. We run all planners on 180 randomly initialised planning problems in both workspaces.

Tables 3.1 and 3.2 show quantitative results for all planners. We use 12 objects on the tabletop (6 movable and 6 immovable), and 5 objects in the refrigerator (3 movable and 2 immovable)[6]. Sample initialisations of these workspaces are shown in Figure 3.1. SPAMP uses $N = 3$ subgoals from $M = 8$ samples in GetValidSubgoals. All planners were given a maximum planning time of 1800s. We divide all planning problems into two scenarios based on how long it takes a Naive planner (a vanilla MHA* algorithm with only the 3D BFS heuristic to the goal) to solve them. Naive uses the action evaluation scheme from Section 3.3.2 and simulates all Phase 1 valid AMPs. Problems solved by Naive in less than 100s are 'Easy', and the rest are

[5]Per [147], we implement multiple levels of discretisation, goal biasing, and add all intermediate states of 'motions' to the search tree (something OMPL does not do). Additionally, we only simulate Phase 1 valid motions.

[6]The tabletop is 0.6m × 0.8m, and refrigerator is 0.6m × 0.6m × 0.6m.

'Difficult'.

SPAMP achieves the highest success rate across all algorithms which shows that our assumption from Section 3.3.4 is not too restrictive for the MAMO domain. In terms of planning times, SPAMP is $30 - 50\times$ faster than KPIECE, and $2 - 8\times$ faster than Selective Simulation. SS is most competitive in terms of planning times, but it is still $2 - 7\times$ slower than SPAMP for difficult planning problems. By design, SPAMP spends most of the simulation time finding valid subgoals for the search which is still comparable to SS, and one or two orders of magnitude less than the other baseline algorithms.

KPIECE by default simulates all actions in the search tree, spends almost all of its planning time in simulation, and in OMPL samples a random state in $\mathcal{X}_\mathcal{R}$ 95% of the time ($\mathcal{X}_\mathcal{R} \subset \mathbb{R}^7$ for a PR2 arm). Since each simulation takes around 1.5s, and mostly random point-to-point exploration of $\mathcal{X}_\mathcal{R}$ (as implemented in OMPL) is wasteful, planning times grow quickly with the number of actions KPIECE evaluates.

3.5.2 Runs on a Physical Robot

We setup the tabletop workspace experiment with the PR2 robot in our laboratory. We also set up a rudimentary experiment to calculate the coefficient of static friction as the tangent of the incline angle of the table at which the objects start sliding. We used this friction coefficient in our simulator in an attempt to minimise the sim-to-real gap. We selected 6 objects at random to initialise our scene and used a search-based object localisation algorithm [1] on an NVidia TITAN X GPU to detect the object poses for simulator initialisation. SPAMP was given a 30s planning timeout, and objects were instantiated in the simulator as movable with 75% probability.

The quantitative data from execution of 39 plans on the physical robot is shown in Table 3.3. A success rate of 82% means that in 7 out of 39 of the executions, the robot violated an obstacle constraint in the real-world. Since the plan found by the robot must have been valid in simulation, constraint violations in the real-world are due to a mismatch between the simulator and the real-world. This could occur due to modeling errors for the obstacles, execution errors on the robot, or perception errors in object localisation.

A qualitative assessment of the 39 executions indicates that the two main sources

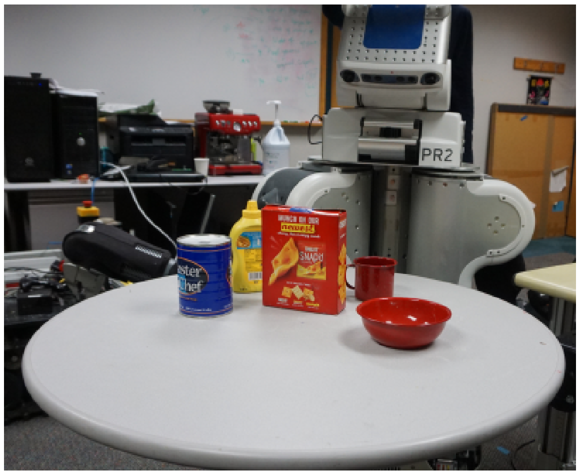


Figure 3.6: Experimental setup for a PR2 robot in front of a tabletop workspace for MAMO.

Table 3.3: Quantitative Performance for Real-World Experiments

	Metrics		
Algorithm	Success Rate	Planning Time (s)	Simulation Time (s)
SPAMP	82%	2 ± 4	0.8 ± 0.6

of error in our experiment were inaccurate friction coefficients and object localisations. Figure 3.6 shows an image of our experimental setup. A supplemental video (`https://youtu.be/ABQc7JbeJPM`) also includes successful executions by the PR2 in a refrigerator and cabinet workspace.

3.5.3 In-Depth Analysis of SPAMP in Simulation

To get a better understanding of the quantitative performance of SPAMP, we conducted experiments to highlight the effect of various components. For our first experiment, we highlight the effect of subgoals and soft duplicate detection. We consider four different planning algorithms - Naive is a vanilla MHA* algorithm with only the 3D BFS heuristic to the goal; Naive+DD uses soft duplicate detection on top of the Naive planner, everything else being the same; SubG uses one randomly sampled Phase 1 valid AMP as a subgoal in MHA*; and SubG+DD uses soft duplicate detection on top of the SubG planner. Problems in this experiment are initialised

Table 3.4: Effect of Subgoals and Soft duplicate detection

Metrics	Scenario	Planning Algorithms			
		NAIVE	NAIVE+DD	SUBG	SUBG+DD
Success Rate	Overall	90%	92%	95%	**96%**
Planning Time (s)	Easy	13 ± 22	9 ± 21	7 ± 14	**6** ± **10**
	Difficult	433 ± 388	188 ± 264	58 ± 108	**39** ± **97**

Table 3.5: Quantitative Performance of SPAMP Variants (Tabletop)

Metrics	Scenario	Planning Algorithms			
		NAIVE	NAIVE+DD	PHASE1	**SPAMP**
Success Rate	Overall	85%	92%	97%	**99%**
Planning Time (s)	Easy	16 ± 22	14 ± 37	13 ± 101	**5** ± **5**
	Difficult	524 ± 439	379 ± 461	48 ± 218	**11** ± **16**
Simulation Time (s)	Easy	7 ± 13	4 ± 7	5 ± 11	**0** ± **0**
	Difficult	130 ± 292	63 ± 136	18 ± 62	**1** ± **7**

with 8 movable objects on the tabletop. Table 3.4 shows the quantitative benefits of subgoals and soft duplicate detection individually, and that in tandem they can greatly improve performance over the NAIVE planner, which can be considered a lower-bound on performance for any planning algorithm in this domain.

In a second experiment, we compare the performance of SPAMP against three related variants on the same tabletop workspace experiment from Section 3.5.1. We compare against NAIVE, NAIVE+DD, and also a planner (PHASE1) which randomly samples $N = 3$ Phase 1 valid AMPs without simulation for use as subgoals. All three of these baselines are allowed to simulate AMPs from within the δ-sphere of a goal configuration from the outset. Table 3.5 shows that while simply using Phase 1 valid subgoals has clear benefits over not using them, the necessary reliance on simulating all Phase 1 valid AMPs leads to higher simulation times, and thereby higher planning times, as compared to SPAMP.

All planners in this section utilise Assumption 1 from Section 3.3.4. If a planning problem is unsolvable due to the assumption being violated, that failure is common to all planners. All other failures for the experiments in this section are due to planners exceeding the timeout (1800s).

3.6 Discussion & Future Work

In this chapter we present SPAMP, an algorithm for Simulation-based Planning with Adaptive Motion Primitives for the MAMO domain. We use AMPs as subgoals within a multi-heuristic search framework to solve manipulation planning problems in cluttered scenes. SPAMP improves planning times by up to $40 - 80\times$ over KPIECE, and up to $2 - 8\times$ over Selective Simulation, two state-of-the-art baselines for the MAMO domain. SPAMP also reduces simulation times by up to $20 - 40\times$ in comparison to these baselines. We show that our assumption of restricting robot-object interactions to terminal AMPs in a plan is not restrictive since we solve $93 - 99\%$ of all problems. In ongoing and future work as part of this thesis, we relax our assumption that interactions may only occur during terminal AMPs in order to solve all planning problems in the MAMO domain.

Chapter 4

Multi-Agent Pathfinding for Manipulation Among Movable Objects

The work in Chapter 3 of this thesis relied on the key insight that for a certain class of MAMO problems, we can defer robot-object interactions until the robot is close to the goal. The defining factor of this class of MAMO problems is a relatively high volume of object-free space near the goal region. In such scenes it is possible for the robot to get close to the goal (within the δ-sphere) without making contact with any object. This assumption is easily violated when (i) there is significant clutter between the robot and the goal, and (ii) a top-down grasp of the desired object is impossible due to tight workspace constraints. Such situations can arise in cluttered shelves in domestic refrigerators and cupboards, and warehouse bins as seen in the Amazon Picking Challenge [26]. Together, these two conditions ensure that a robot manipulator needs to reason about rearranging the clutter in the scene in order to grasp and extract an object-of-interest. Our goal in this chapter is to build this type of reasoning into a manipulation planning algorithm for pick-and-place tasks in densely cluttered environments.

Real-world manipulation problems in heavy clutter require robots to reason about potential contacts with objects in the environment. We focus on pick-and-place style tasks to retrieve a target object from a shelf where some 'movable' objects must be rearranged in order to solve the task. In particular, our motivation is to allow the robot to reason over and consider non-prehensile rearrangement actions that lead to complex robot-object and object-object interactions where multiple objects might be moved by the robot simultaneously, and objects might tilt, lean on each other, or topple. To support this, we query a physics-based simulator to forward simulate these interaction dynamics which makes action evaluation during planning computationally very expensive. To make the planner tractable, we establish a connection between the domain of **M**anipulation Among Movable Objects and **M**ulti-Agent Pathfinding that lets us decompose the problem into two phases our **M4M** algorithm iterates over. First we solve a multi-agent planning problem that reasons about the configurations of movable objects but does not forward simulate a physics model. Next, an arm motion planning problem is solved that uses a physics-based simulator but does not search over possible configurations of movable objects. We run simulated and real-world experiments with the PR2 robot and compare against relevant baseline algorithms. Our results highlight that **M4M** generates complex 3D interactions, and solves at least twice as many problems as the baselines with competitive performance. This chapter was first published as a paper in [128].

4.1 Introduction

Manipulation Among Movable Objects (MAMO) [144] defines a broad class of problems where a robot must complete a manipulation task in the presence of obstructing clutter. In heavily cluttered scenes, there may be no collision-free trajectory that solves the task. This does not make the problem unsolvable since MAMO allows rearrangement of some objects *a priori* designated as 'movable'. In addition, MAMO may associate each object with constraints on how it can be interacted with – it is undesirable to allow robots to carelessly push or throw objects around.

In this chapter, we consider MAMO problems for pick-and-place manipulation tasks where the robot needs to retrieve a target object from a cluttered shelf, cabinet, fridge, or a similar structure. Figure 4.1 (a) shows an example of such a scene where two movable objects must be rearranged in order to retrieve the desired object, while ensuring they do not topple and no contacts are made with an immovable obstacle.

Solving such MAMO problems requires answers to three difficult questions: *which*

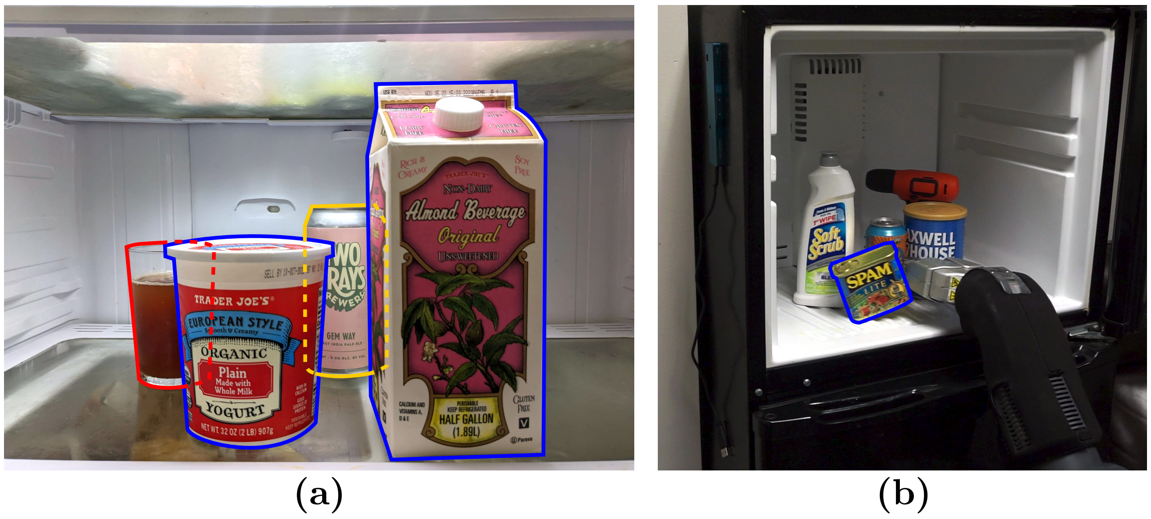


Figure 4.1: (a) An example MAMO problem to retrieve the beer can (yellow outline). Access is blocked by the movable box of milk and tub of yogurt (blue outlines). In order to retrieve the can, they must be rearranged out of the way without toppling them, and without anything making contact with the glass of juice (red outline). (b) A complex non-prehensile action that tilts the movable potted meat can (blue outline) to rearrange it.

objects to move, *where* to move them, and *how* to move them. Thus MAMO problems assign the robot a goal with respect to the overall task and object-of-interest (OoI), without any additional goal specifications for other objects except for satisfying their associated interaction constraints; while MAMO solutions exist in a composite configuration space that includes the configuration of the robot arm and all objects in the scene. The search for a solution is computationally challenging since the size of this space grows exponentially with the number of objects.

We are interested in non-prehensile rearrangement actions since they allow robots to manipulate objects that may be too big or too bulky or otherwise ungraspable. In many cases it is more time- and energy-efficient to push an object off to the side than to grasp it, pick it up, move it elsewhere, place it down, and release it before proceeding. Furthermore, we allow the robot to move multiple objects simultaneously with the same push action, and we allow objects to tilt, lean on each other, and slide (an example is shown in Figure 4.1 (b)). Planning with these actions requires the ability to predict the effect of robot actions on the configuration of objects, typically through computationally expensive forward simulations of a rigid-body

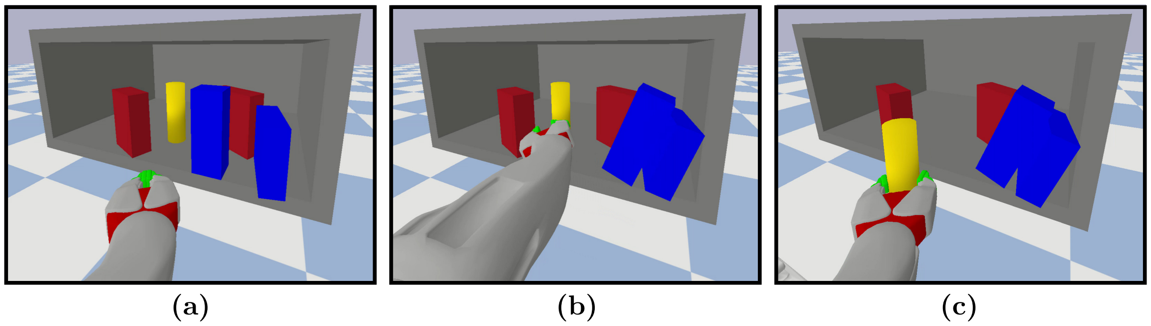


Figure 4.2: Sequence of images showing a solution found by our **M4M** algorithm for a simple MAMO scene. From *left* to *right*: (a) initial scene, (b) rearranged scene after one push action, (c) successful OoI retrieval. Movable objects are blue, immovable obstacles are red, and the OoI is yellow.

physics simulator.

Our key insight in this work draws a connection between the MAMO domain and Multi-Agent Pathfinding (MAPF) to decompose the problem into two parts. First, we treat the movable objects as artificially actuated agents tasked with avoiding collisions with (i) our robot arm retrieving the OoI, (ii) each other, and (iii) immovable obstacles. A solution to this abstract MAPF problem searches over potential rearrangements of objects *without* the need to query a physics simulator. Next, we use the MAPF solution to compute informed push actions to rearrange movable objects *without* searching over their possible configurations. These actions are forward simulated with a physics model to ensure validity. The decomposition helps us keep track of object configurations in the full $SE(3)$ space and generate informed push actions that lead to realistic multi-body interactions in the 3D workspace as shown in Figure 4.1 (b). Fig 4.2 shows a complex and interesting solution found by our algorithm for one of the simpler scenarios in our test data.

The main contributions of our work in this chapter for solving MAMO planning problems are:

- Enable reasoning over and usage of complex non-prehensile interactions that may push multiple objects in tandem and produce object-object interactions like leaning and toppling (Figure 4.1 (b)).
- MAPF abstraction for computing suitable rearrangements for MAMO planning problems, without using a simulation-based model.

- An efficient algorithm to solve MAMO problems that iterates between calls to an MAPF solver (to determine *which* objects to move *where*) and a push planner (to verify *how* to move the objects).
- A thorough experimental evaluation of our approach in simulation and in the real-world on a PR2 robot.

We provide details of relevant works from MAMO literature in Section 4.2. Section 4.3 formalises the MAMO planning problem. Section 4.4 presents our iterative planning algorithm **M4M**, including the abstraction from MAMO to MAPF (Section 4.4.1) and a non-prehensile push planner (Section 4.4.2). We provide extensive quantitative evaluation against relevant MAMO baselines in simulation in Section 4.5 along with real-world results of our algorithm on the PR2 robot. Section 4.6 discusses the benefits, limitations, and future extensions of this work.

4.2 Related Work

MAMO generalises Navigation Among Movable Obstacles (NAMO) where a mobile robot must navigate from start to goal in a reconfigurable environment [5, 143, 165]. It is also related to the rearrangement planning problem [12, 108] which explicitly specifies desired goal configurations for movable objects. Latombe [83] provides an excellent review of early work in these domains which were limited to planar environments and geometric solutions. Wilfong [165] showed that rearrangement planning is PSPACE-hard, and MAMO problems are NP-hard to solve. Non-prehensile actions were introduced in MAMO planning algorithms [93] based on analysis done by Mason [97] on the mechanics of pushing.

MAMO problems can be formulated as *task and motion planning problems* [20, 65, 67, 103] where a high-level search reasons about all allowed rearrangements of the workspace such that the manipulation task of OoI retrieval may be completed. The success of these methods depends on how the high-level actions are parameterised. Parameterisations that include more information – such as stable configurations and grasp poses of objects – potentially lead to easier motion planning problems, provided the parameters are sampled intelligently. In contrast, we rely on a MAPF abstraction to guide our search towards a suitable rearrangement for completing the manipulation task, and dynamically generate push strategies based on the MAPF solution.

Many existing MAMO and *rearrangement planning* solvers make use of prehensile actions [79, 80, 86, 135, 144, 163]. This simplifies planning since grasped objects behave as rigid bodies attached to the robot, but assumes access to known stable configurations of and grasp poses for objects [79, 80, 135, 144]. In some cases a "buffer" location to place grasped objects is required [86, 163]. In particular, [80] and [135] utilise the concept of "pebble graphs" [75, 140] from MAPF literature to find prehensile actions for rearrangement planning. Their formulation restricts the motion of the movable objects (pebbles) on a precomputed roadmap of robot arm trajectories via prehensile actions. This limits the possible configurations of objects they consider since motions are limited to poses from where they can be grasped and to those where they can be stably placed. Since we utilise non-prehensile pushes for rearrangement and a physics-based simulator for action validation, our planner explores a richer space of robot-object and object-object interactions in the 3D workspace.

Allowing *non-prehensile interactions* with objects typically requires access to a simulation model to obtain the result of complex interaction dynamics [35, 59, 70, 148, 159, 161]. Of these approaches, only Selective Simulation [148] considers realistic interactions in the 3D workspace and is one of our comparative baselines in Section 4.5. Others rely on planar robot-object interactions which fail to account for object dynamics in $SE(3)$ where they might tilt, lean, or topple. In Section 4.5, we adapt the MAMO solver from [35] to use our push actions that lead to 3D robot-object interactions and require a physics simulator during planning. Originally their work was limited to interacting with a single object at a time, and used an analytical motion model in $SE(2)$ to propagate the effect of the push on the planar configuration of the object being pushed (tilting and toppling was not considered in [35, 59, 70, 159, 161]).

Querying *physics-based simulators* for the result of an action is much more expensive than collision checking it. KPIECE [147] is a randomised algorithm for planning with a computationally expensive transition model (querying a physics-based simulator is an example of such a model). We compare against KPIECE in our experiments in Section 4.5. In our own prior work on MAMO planning [130] from Chapter 3, we find a collision-free trajectory to a region near the OoI grasp pose, and simulate goal-directed non-prehensile actions only within this region. The assumption that such a collision-free trajectory exists is easily violated in the cluttered MAMO workspaces we instantiate in our experiments (see Figs. 4.1 (a), 4.6, and 4.7 for example).

4.2.1 Multi-Agent Pathfinding

Multi-agent Pathfinding is a family of planning problems that tries to find paths for a team of robots from a set of start locations to a set of goal locations. This general formulation can give rise to many different types of problems based on the homogeneity of the robot team, a centralised or decentralised planning algorithm, the number of robots in the team and the number of goals to be achieved, and whether the goals are labeled (specific goals for specific robots) or not. Additional factors that can be important to consider include whether the environment is static or dynamic, known or unknown, and whether there are inter-robot communication latencies. A general introduction to multi-agent pathfinding can be found in [84, 134, 142].

One class of algorithms that solve MAPF problems assigns priorities to the robots and solves a sequence of single-agent planning problems based on this prioritisation [42, 137]. This *prioritised planning* scheme trades off algorithmic incompleteness and solution suboptimality for practical efficiency. Turpin et al. [156] present a resolution complete prioritised MAPF solver for the specific case when robots are interchangeable.

Conflict-based search (CBS) [132, 134] and M* [162] are complete and optimal MAPF solvers that use different techniques to provide strong theoretical guarantees. CBS searches a tree of all possible solutions in a best-first manner by resolving robot conflicts into constraints on robot motion. For two robots that collide at any instant, either one can be in that location in the final solution but not both. CBS enumerates all such possibilities for all potential conflicts until a solution with no conflict is found. M* resolves robot conflicts by combining two conflicting robots and treating them as one agent until the conflict between them is resolved. This idea has also been adopted in the CBS family of algorithms [133].

While the focus of MAPF is typically limited to finding paths on discrete graphs, multi-robot motion planning tries to compute feasible trajectories for a team of robots. This has been formulated as an integer program in prior work [171], and techniques from sampling-based planning have been used to quickly search the continuous composite space [32, 136]. However, in this work we solve an abstract MAPF problem with artificially actuated agents that move on a discrete grid. This simplifies the MAPF problem by discretising it, and we then attempt to realise the MAPF solution in the continuous 3D workspace via our non-prehensile push planner.

4.2.2 Abstract Planning

Abstract search spaces or *abstractions* of planning problems refer to simpler, reduced forms of the *full space* of the original problem. The use of these search abstractions is related to the field of hierarchical planning [125] which attempts to use abstract space solutions as partial plans or heuristics in the full space. Pearl [113] contains a thorough review of these techniques. An abstract solution can sometimes be refined to obtain a full solution by removing the simplifications used to construct the abstraction hierarchy [6, 51]. Another common approach is to use abstract space distances to abstract goals as a heuristic in the full space [17, 56, 58]. Pattern databases generalise this idea for multiple abstract space states or "subgoals", and store cost-to-subgoal values for use as a heuristic in the full search [28]. Guided optimisation in the computer graphics and animation community [158], postulates a "hand of god" which applies an external balancing torque to help find a control trajectory for bipedal locomotion. The work done by this "hand of god" is reduced over time to guide the solver towards a good optimum. Philosophically this idea is the same as the artificial actuation of movable objects in our work.

The idea of solving a simplified version of the planning problem as a means to finding a solution in the full space is related to *curriculum learning*. Curriculum learning was explored in the context of robotics by Sanger [126], and since then a more formal theoretical basis for machine learning methods has been established [13]. A brief overview specifically for robot manipulation can be found in [77].

4.3 Problem Setup

The work in this chapter follows the general problem setup from Section 2.3.3. For ease of readability, we will reiterate some of the salient aspects of our formulation.

A MAMO planning problem can be defined with the tuple $\mathcal{P} = (\mathcal{X}, \mathcal{A}, \mathcal{T}, c, x_S, \mathcal{X}_G)$. $\mathcal{A}$ is the action space of the robot, $\mathcal{T} : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{X}$ is a deterministic transition

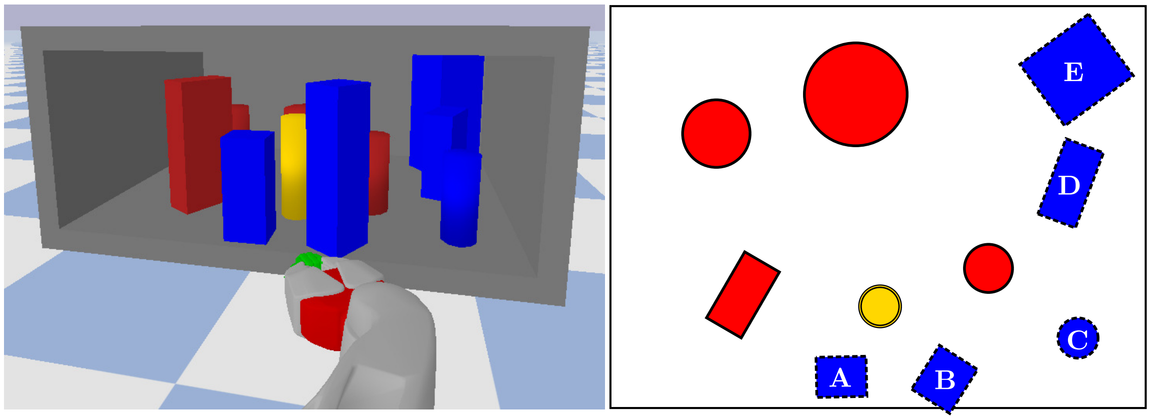


Figure 4.3: MAMO workspace (*left*) and its 2D projection labeled with movable object IDs. Movable objects are in blue, immovable obstacles in red, and the object-of-interest to be retrieved in yellow.

function, $c : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ is a state transition cost function, $x_S \in \mathcal{X}_V$ is the start state, and $\mathcal{X}_G \subset \mathcal{X}, \mathcal{X}_G \cap \mathcal{X}_V \neq \emptyset$ is the set of goal configurations. For the work in this chapter we discretise the action space of the robot $\mathcal{A}$ to include "simple motion primitives" that independently change each robot joint angle by a fixed amount (described in more detail earlier in Section 3.3.1) and dynamically generated "push actions" described in Section 4.4.2. For transition $x_{i+1} = \mathcal{T}(x_i, a_i)$, action $a_i \in \mathcal{A}$ can affect object configurations between x_i and x_{i+1} only if a_i is a push action or the OoI has been grasped. Our solution to MAMO problems is a sequence of arm trajectories in the robot configuration space $\mathcal{X}_\mathcal{R} \subset \mathbb{R}^q$ ($q = 7$ for the PR2 robot) that (i) rearrange movable clutter and (ii) retrieve the OoI. Figure 4.3 shows an example of the MAMO problems we consider in this chapter, along with its 2D projection. Red objects are immovable obstacles $\mathcal{O}_I$, blue objects are initial movable objects $\mathcal{O}_M^{\text{init}}$, and the goal for the robot arm is to extract the yellow OoI from the shelf. There is no collision-free trajectory for the arm to extract the OoI from the shelf. Upon rearrangement of some movable objects (A and B in particular), such a trajectory may be found.

4.3.1 Classical Multi-Agent Pathfinding

Classical MAPF planning problems seek to find non-conflicting paths for a set of agents $\{r_1, \ldots, r_n\}$ on a discrete graph $G = (V, E)$ in discrete time. Each robot r_i has a designated start state $s_i \in V$ and a desired goal state $g_i \in V$. Robot r_i has access to an action space $\mathcal{A}_i$, which includes an action to *wait* at the current state. An edge $(v, v') \in E$ implies that some action $a^j \in \mathcal{A}_i$ takes robot r_i from vertex v to v'. All actions are assumed to take the same amount of time such that traversing an edge $(v, v') \in E$ takes one unit of time. A *single-agent solution path* for r_i is a sequence of states $\pi_i = \{v_0 = s_i, \ldots, v_T = g_i\}$ where the subscripts denote time indices. Two single-agent solution paths π_i and π_j are *conflict-free* if robots r_i and r_j never collide as they traverse their respective paths. The solution to a MAPF problem with n robots $\{r_1, \ldots, r_n\}$ is a set of n *mutually* conflict-free paths, i.e. π_i and π_j must be conflict-free for any $1 \leq i, j \leq n\,, i \neq j$.

Although a thorough review of MAPF literature is beyond the scope of this work, we would like to highlight that the MAPF problem is NP-hard to solve optimally [170]. We use Conflict-Based Search (CBS) [134], a complete and optimal MAPF algorithm, to solve our abstract MAPF problem formulated in Section 4.4.1.

4.4 The **M4M** Planning Algorithm

We call our algorithm **M4M**: Multi-Agent Pathfinding for Manipulation Among Movable Objects. **M4M** is given access to a physics-based simulator (PyBullet [27]) to ensure that no interaction constraints defined in the MAMO problem are violated. We note that a MAMO problem $\mathcal{P}$ to retrieve the OoI with $\mathcal{O}_M \neq \emptyset$ is solvable *iff* the simpler problem $\hat{\mathcal{P}}$ without any movable objects i.e., $\mathcal{O}_M = \emptyset$ can be solved. We denote a solution trajectory to $\hat{\mathcal{P}}$ as $\hat{\pi}_{\mathcal{R}}$. Let $\mathcal{V}(\hat{\pi}_{\mathcal{R}})$ denote the volume occupied by the robot arm in the workspace during execution of $\hat{\pi}_{\mathcal{R}}$. $\mathcal{V}(\hat{\pi}_{\mathcal{R}})$ specifies a "negative goal region" (NGR) [35] for the movable objects. A NGR is a sufficient volume of the 3D workspace which, if there are no objects inside it, allows the robot arm to retrieve the OoI without other contacts. If all movable objects can be rearranged such that they are outside $\mathcal{V}(\hat{\pi}_{\mathcal{R}})$, the robot can execute $\hat{\pi}_{\mathcal{R}}$ to retrieve the OoI. Figure 4.4 shows a NGR $\mathcal{V}(\hat{\pi}_{\mathcal{R}})$ for the problem from Figure 4.3.

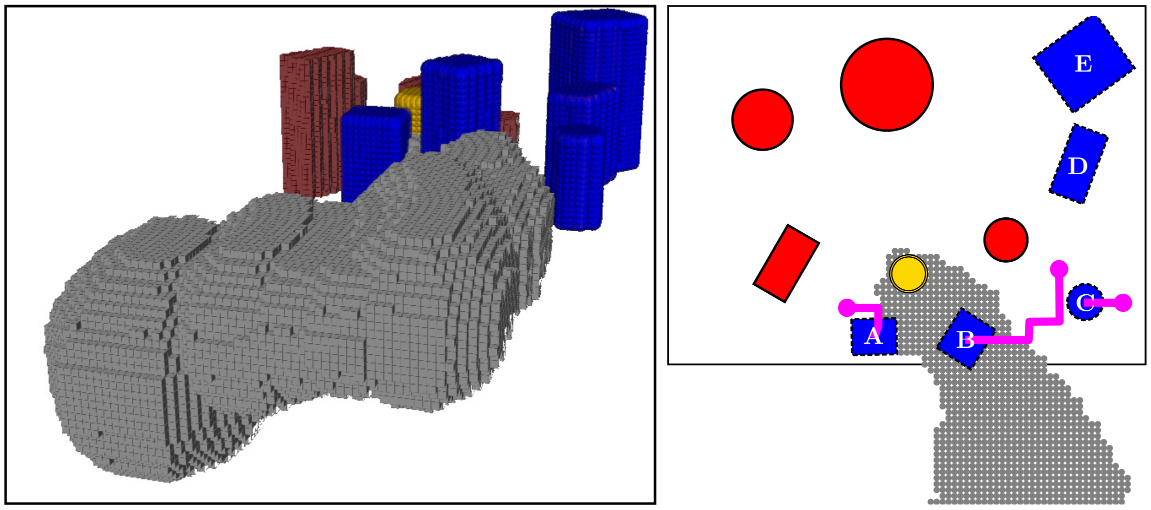


Figure 4.4: The negative goal region (NGR) $\mathcal{V}(\hat{\pi}_{\mathcal{R}})$ in gray for the MAMO problem from Figure 4.3. (*left*) 3D volumes of the NGR and all objects at their initial poses (we omit the shelf for ease of visualisation). (*right*) 2D projection of the NGR and the workspace, overlayed with the solution to the abstract MAPF problem from Section 4.4.1 formulated for this scene. Objects A and B need to move outside the NGR, and object C needs to move to allow A to reach its goal. MAPF solution paths are shown in pink.

Algorithm 4 contains the pseudocode for **M4M**. At a high-level, **M4M** first computes $\hat{\pi}_{\mathcal{R}}$ (Line 4) and the NGR $\mathcal{V}(\hat{\pi}_{\mathcal{R}})$ (Line 5). It then iterates over two steps:

1. Section 4.4.1: Compute a solution to the abstract Multi-agent Pathfinding (MAPF) problem where each movable object is treated as an agent that needs to escape the NGR without colliding with other agents using Conflict-Based Search (CBS) [134], a complete and optimal MAPF algorithm.
2. Section 4.4.2: Pick a movable object to be rearranged according to the MAPF plan computed in 1 and find a valid non-prehensile push for it by forward simulating potential pushes using a physics-based simulator.

Algorithm 4 uses `replan` to ensure CBS is only called to solve new MAPF problems. After the first CBS call, `replan` triggers subsequent CBS calls once a valid push has been found i.e., at least one object has been moved (Line 8). This leads to a different MAPF problem with new object poses. Until a valid push is found, we sample and simulate pushes for all objects that move in the MAPF solution. Only when we find a valid push for an object that needs to move per the MAPF solution do we call the

MAPF solver again for the resulting scene. Until such time we sample and simulate pushes for all objects that move as part of the MAPF solution. After successfully moving an object, the next call to the MAPF solver can take this change in object state into account to find a better solution for rearrangement. In this way **M4M** is greedy with respect to valid pushes that it finds and plans "in the now" [65].

Algorithm 4 Multi-Agent Pathfinding for Manipulation Among Movable Objects

```
1: procedure M4M($\mathcal{O}_M^{\text{init}}, \mathcal{O}_I$)
2:     $\mathcal{O}_M \leftarrow \mathcal{O}_M^{\text{init}}$                ▷ Rearranged object positions
3:     $\Psi \leftarrow \emptyset$                ▷ Sequence of arm trajectories
4:     $\hat{\pi}_\mathcal{R} \leftarrow$ PlanRetrieval($\mathcal{O}_I$)                ▷ OoI retrieval trajectory
5:     Compute $\mathcal{V}(\hat{\pi}_\mathcal{R})$
6:     replan ← true, done ← false
7:     while time remains do
8:         if replan then
9:             $\pi_\mathcal{R} \leftarrow$ PlanRetrieval($\mathcal{O}_I \cup \mathcal{O}_M$)
10:            if $\pi_\mathcal{R}$ exists then
11:                $\Psi \leftarrow \Psi \cup \{\pi_\mathcal{R}\}$, done ← true
12:                break
13:            $\{\pi_{o_m}\}_{o_m \in \mathcal{O}_M} \leftarrow$ CBS($\mathcal{O}_M, \mathcal{O}_I, \mathcal{V}(\hat{\pi_\mathcal{R}})$)
14:            replan ← false
15:        for $o_m \in \mathcal{O}_M$ do
16:            if $\pi_{o_m} = \emptyset$ then
17:                continue
18:            $\psi \leftarrow$ PlanPush($o_m, \pi_{o_m}, \mathcal{O}_M, \mathcal{O}_I$)
19:            (valid, $o'_m$) ← SimulatePush($\psi$)
20:            if valid then
21:                $\Psi \leftarrow \Psi \cup \{\psi\}$, replan ← true
22:                UpdatePose($\mathcal{O}_M, o'_m$)
23:                break
24:    if ¬done then
25:        return $\emptyset$
26:    return $\Psi$
```

The PlanRetrieval function takes as input a set of objects to be considered as immovable obstacles for the robot and runs Multi-Heuristic A* [3] to find an arm trajectory in $\mathcal{X}_\mathcal{R}$ to retrieve the OoI.

CBS is called in Line 13 with the latest known movable object poses in $SE(3)$ to obtain a set of paths that ensure they all satisfy the NGR $\mathcal{V}(\hat{\pi}_{\mathcal{R}})$. This searches over all possible rearrangements of the scene from the current state, without ever querying a physics simulator, by assuming that movable objects are artificially actuated agents (Section 4.4.1).

We then loop over all objects that need to be rearranged (from Line 15) and try and find a valid push for them (Section 4.4.2). Details of our push planner are provided in Section 4.4.2. If a valid push is found (Line 20), it is added to the final sequence of arm trajectories to be executed Ψ, and the pose of that object is updated for future iterations.

M4M terminates either when the allocated planning budget expires, or we successfully find a trajectory to retrieve the OoI in the presence of all objects $(\mathcal{O}_I \cup \mathcal{O}_M)$ as obstacles in Line 9. Although this trajectory $\pi_{\mathcal{R}}$ may be different from $\hat{\pi}_{\mathcal{R}}$ (Line 4), it will still retrieve the OoI successfully since it is guaranteed to not make contact with any object (immovable or movable). The sequence of trajectories Ψ can then be executed in order to rearrange the movable objects (if required) and finally ending in successful OoI retrieval.

4.4.1 MAPF Abstraction for Manipulation

A fundamental challenge to solving MAMO problems requires determining *which* objects need to be rearranged and *where* they should be moved. The key idea in this chapter uses an existing MAPF solver to search over potential rearrangements of the scene which lead to successful OoI retrieval. Importantly, the MAPF solver does not require access to a physics simulator for this purpose – it only relies on 3D collision checking. Our MAPF abstraction includes all movable objects $o_m \in \mathcal{O}_M$ as agents. We check for collisions between agents in space and time in their full $SE(3)$ configuration space. All agents have a discrete action space corresponding to a four-connected grid on the $(x, y)-$plane of the shelf. We assume each action takes unit time and either the agent remains in place, or the $x-$ or $y-$coordinate of the agent pose changes by 1 cm.

Agent start configurations are determined by their latest pose in $SE(3)$ prior to the MAPF call (Algorithm 4, Line 13). Each agent o_m in the MAPF problem has a set of possible goals that include all states where the agent satisfies the NGR by being "outside" it. We call CBS to obtain a solution, shown in Figure 4.4, to this MAPF abstraction. CBS runs a two-level search to solve MAPF problems. The high-level of CBS searches a tree of all possible solutions in a best-first manner by resolving agent conflicts into constraints on agent motion. For two agents that collide along their paths, either one can be in the location where and when they collide in the final solution but not both. The low-level of CBS runs single-agent searches for each agent on a discrete graph $G_{\text{CBS}} = (V, E)$ where vertices $V \subset SE(3)$ are object poses. The solution returned by CBS is a set of paths for movable objects $\{\pi_{o_m}\}_{o_m \in \mathcal{O}_M}$ whose final states $\pi_{o_m}^{end}$ satisfy the NGR, and suggests a rearrangement strategy in terms of *which* objects to move and *where*. If we can rearrange all $o_m \in \mathcal{O}_M$ to their respective $\pi_{o_m}^{end}$ poses, we know that the trajectory $\hat{\pi}_{\mathcal{R}}$ will successfully retrieve the OoI, thereby solving the MAMO problem.

4.4.2 Generating Non-Prehensile Push Actions

Given a path π_{o_m} for $o_m \in \mathcal{O}_M$ from the MAPF solution, PlanPush (Algorithm 4, Line 18) determines *how* an object may be rearranged (Figure 4.5). We would like to move the object to $\pi_{o_m}^{end}$, which is known to satisfy the NGR. To compute a push trajectory, we first shortcut π_{o_m} (taking into account collisions with immovable obstacles $\mathcal{O}_I$) into a series of straight line segments defined by points $\{x^1 = \pi_{o_m}^{start}, \dots, x^n = \pi_{o_m}^{end}\}$. We also compute the point of intersection x^{aabb} of the ray from x^1 along the direction $\overrightarrow{(x^2, x^1)}$ with the axis-aligned bounding box of o_m.

PlanPush computes a collision-free path between successive pushes by planning in $\mathcal{X}_{\mathcal{R}}$ with all objects $\mathcal{O}_I \cup \mathcal{O}_M$ as obstacles to a point x_{push}^0 sampled around x^{aabb}[1]. If this path is found, PlanPush similarly samples points x_{push}^i around each x^i in the shortcut path. It runs inverse kinematics (IK) in sequence for each segment of the push action between points $\left(x_{\text{push}}^{i-1}, x_{\text{push}}^i\right), i = \{1, \dots, n\}$. If all IK calls succeed, we return the full push trajectory by concatenating π_0 with all push action segments.

This push action, informed by the MAPF solution about *which* object to move *where*, is forward simulated with a physics model to verify whether it satisfies all interaction constraints for all objects. If so, it is queued into the sequence of

[1]We sample (x, y) coordinates for x_{push}^0 from $\mathcal{N}(x^{aabb}, \sigma I)$, $\sigma = 2.5\,\text{cm}$. The $z-$coordinate is fixed at 3 cm above the shelf for the entire push action.

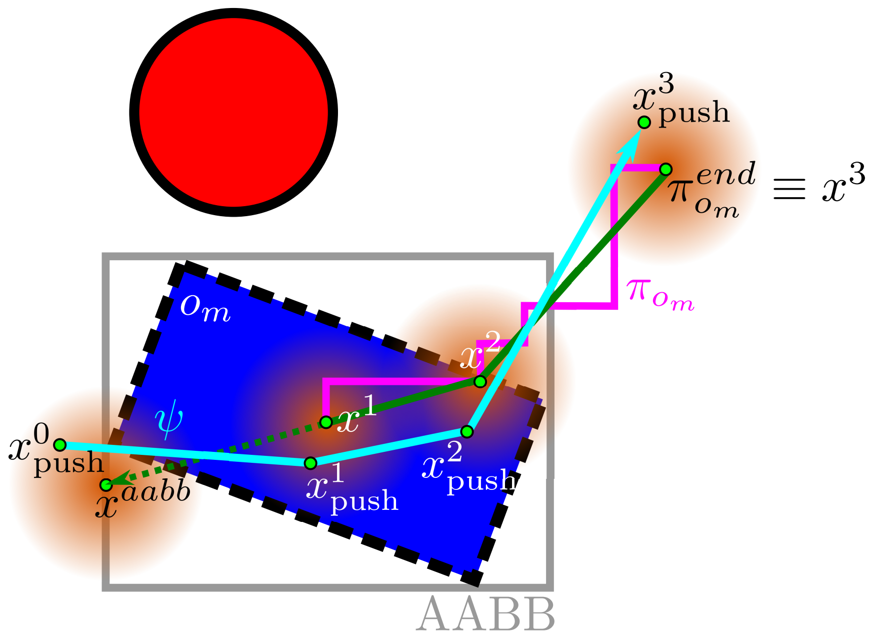


Figure 4.5: 2D illustration of our push planner. Given a movable object o_m (blue) and its MAPF solution path π_{o_m} (pink), we shortcut π_{o_m} while accounting for immovable obstacles $\mathcal{O}_I$ (red) to get the green path of straight line segments. After computing x^{aabb} by intersecting the $\overrightarrow{(x^2, x^1)}$ ray with the axis-aligned bounding box for o_m, the push action (cyan) is computed via inverse kinematics between sampled points $x^i_{\text{push}} \sim \mathcal{N}(x^i, \sigma I)$, $i = \{0, \ldots, n\}$, $x^0 := x^{aabb}$.

rearrangements that will be executed as part of the MAMO solution returned by **M4M** (Algorithm 4, Line 20).

4.5 Experimental Results

4.5.1 Simulation Experiments

We run our simulation experiments in MAMO workspaces of three difficulty levels shown in Figure 4.6. Each workspace has one OoI (yellow), four immovable obstacles (red), and different numbers of movable objects (blue). Objects are cylinders and cuboids with random sizes, initial poses, masses, and coefficients of friction. We assume perfect knowledge of the initial workspace state and all object parameters. We set a planning timeout of 120 s for 100 randomly generated MAMO problems at each level. Our analysis includes two versions of our algorithm – **M4M** refers to Algorithm 4, and $\widehat{\textbf{M4M}}$ refers to a version which only calls CBS once (after Line 5) and does not iterate between calling CBS and finding a valid push in simulation.

Baselines: We compare the performance of **M4M** against three types of baselines for solving MAMO problems with non-prehensile interactions. The first is a standard implementations of a sampling-based algorithm KPIECE [147] from OMPL [146] that searches the entire MAMO state space $\mathcal{X}$ by randomly sampling robot motions.

The second baseline, Selective Simulation [148] (SelSim), is a search-based algorithm that interleaves a 'planning' phase and a 'tracking' phase. The former queries the physics-based simulator for interactions with a set of 'relevant' movable objects identified so far. The latter executes the solution found by the planning phase in the presence of all objects in simulation and, if any interaction constraints are violated, it adds the 'relevant' object to the set. It only uses the simple motion primitives described in Section 4.3.

Our final baseline is the work from Dogar et al. [35] (Dogar) which introduced the idea of a negative goal region (NGR) we use in **M4M**. Dogar recursively searches for a solution backwards in time, similar to [144]. It first finds an OoI retrieval trajectory ignoring all movable objects. The NGR induced by this trajectory helps identify a set of objects to be rearranged, and the OoI is added as an obstacle. If an object is successfully rearranged, the NGR and set of objects still to be rearranged are updated with the trajectory found, and the rearranged object is added as an obstacle at its initial pose. This process continues until no further objects need to be rearranged. Our implementation of Dogar finds the same OoI retrieval trajectory as **M4M**, and uses the same push actions (Section 4.4.2) to try and rearrange objects. Notably, Dogar only has information about *which* objects to move but not *where* to move them. Our implementation finds the closest cell outside the latest NGR for an object and samples points around this location to try to move the object towards.

Table 4.1: Simulation Study for MAMO Planning in Cluttered Scenes - success rates and *min/median/max* planning and simulation times

Metrics	**Level**	**Planning Algorithms**				
		M4M	$\widehat{\text{M4M}}$	Dogar [35]	SelSim [148]	KPIECE [147]
Success Rate (%)	1	92	79	40	33	48
	2	73	54	20	21	33
	3	62	36	6	16	17
Total Planning Time (s)	1	1.0 / 2.6 / 102.5	1.0 / 2.4 / 103.8	0.1 / 0.9 / 115.3	0.004 / 0.02 / 0.03	7.4 / 23.4 / 117.8
	2	1.2 / 6.6 / 115.4	1.3 / 2.6 / 100.3	0.3 / 0.5 / 113.5	0.002 / 0.008 / 0.2	9.3 / 28.2 / 112.0
	3	1.3 / 7.2 / 116.1	1.6 / 2.4 / 72.6	0.2 / 0.4 / 55.0	0.004 / 0.01 / 0.03	10.6 / 32.0 / 98.5
Simulation Time (s)	1	0 / 0 / 58.6	0 / 0 / 20.1	0 / 0 / 42.0	27.3 / 35.0 / 43.6	0 / 10.6 / 99.0
	2	0 / 0.4 / 75.9	0 / 0 / 37.0	0 / 0 / 20.9	36.7 / 44.1 / 58.3	0 / 16.1 / 95.4
	3	0 / 0.4 / 55.1	0 / 0 / 24.3	0 / 0 / 20.0	47.3 / 55.7 / 76.0	0 / 18.3 / 79.3

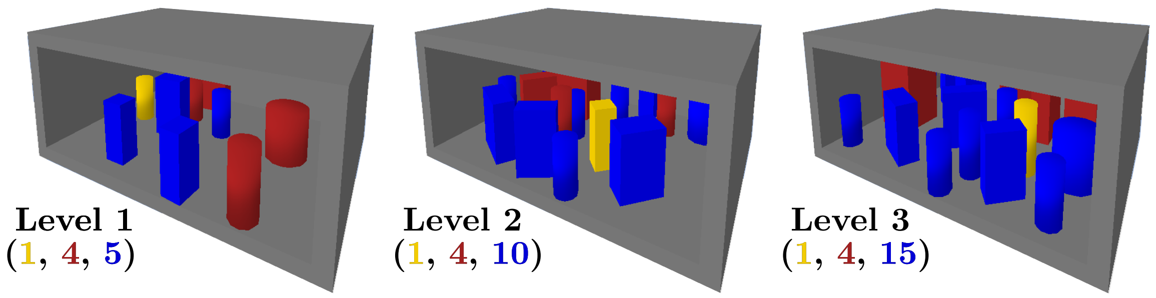


Figure 4.6: MAMO problems of differing complexity. From *left* to *right*, Levels 1, 2, and 3 have 5, 10, and 15 movable objects respectively. Each Level has 1 OoI and 4 immovable obstacles.

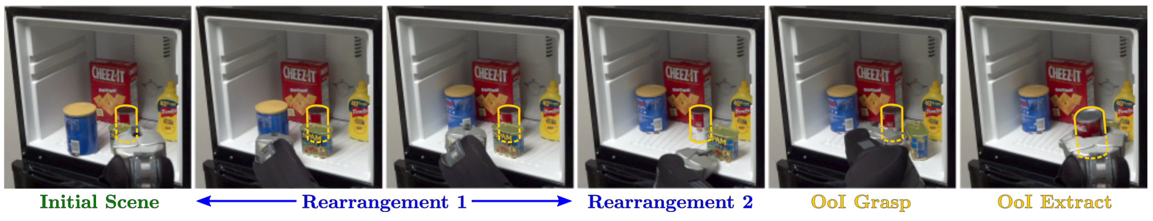


Figure 4.7: A MAMO solution generated by **M4M**. The tomato soup can (yellow outline) is the OoI, all other objects are movable.

Results: Table 4.1 shows the result of our experiments where we present the *min*/*median*/*max* values for total planning time and simulation time of successful runs only. Experiments were run on a 4 GHz Intel i7-4790K CPU with 28 GB 1600 MHz DDR3 RAM.

Both versions of **M4M** solve the most problems across all difficulty levels. For Levels 1, 2, and 3, the **M4M** solution successfully executed 0.8, 1.9, and 3.1 push actions on average. The difference in performance between **M4M** and $\widehat{\textbf{M4M}}$ highlights the benefit of the iterative nature of **M4M**. Since MAPF paths are usually not precisely replicated in simulation via pushes, querying the solver repeatedly with an updated workspace configuration leads to more informed future paths for objects, instead of trying to forcibly push them to the first goal configuration suggested by MAPF.

All baseline algorithms from Table 4.1 suffer due to poor exploration over the space of rearrangements. Our approach benefits from the MAPF abstraction to produce guidance on *where* to move each object to free up the NGR. The stochastic sampling of push actions used by our push planner leads to complex, multi-body non-prehensile interactions that satisfy interaction constraints in the final solution. In contrast Dogar naively samples pushes to be simulated, and necessarily tries to ensure there is no overlap between the NGR and movable objects, even if a slightly different collision-free path can be found to retrieve the OoI (Algorithm 4, Line 9). This strategy suffers when sampled points are near immovable obstacles, and limits the possible rearrangements considered since movable objects that are rearranged successfully are treated as immovable obstacles. Dogar also never executes a potential trajectory until there is no overlap between the NGR and movable objects, unlike SelSim which simulates all trajectories found during planning. In fact, all SelSim successes in Table 4.1 correspond to scenes where the very first planned trajectory succeeds in OoI retrieval in simulation. This is only true when there is minimal overlap between the NGR and movable objects. When any movable object needs to be rearranged, SelSim suffers from its poor action space – the simple motion primitives are ineffective at causing meaningful robot-object interactions in the workspace. KPIECE benefits significantly from goal biasing in simpler scenes where either little to no robot-object interactions are required or the objects that need to be moved have nice physical properties (large supporting footprint, low center-of-mass, low coefficient of friction).

4.5.2 Real-World Performance on the PR2

We ran **M4M** on a PR2 robot where we used a refrigerator compartment as our MAMO workspace (Figure 4.7). We placed five objects from the YCB Object Dataset [16] in the refrigerator. Four of these were movable and the tomato soup can was the object-of-interest. Objects were localised using a search-based algorithm [1] run on a NVidia Titan X GPU. Figure 4.8 shows an image of this setup in our lab. We gave **M4M** a total planning timeout of 120 s.

Out of 16 perturbations of the initial scene from Figure 4.7, 12 runs successfully retrieved the OoI. Across the successful runs the planner took 56.41 ± 27.29 s to compute a plan of which 49.26 ± 24.21 s was spent simulating pushes. Failures were due to interaction constraints being violated during execution by the PR2. Since **M4M** returns a solution that does not violate constraints in simulation, failures are due to modeling errors between the simulator and the real-world. Specifically, accurately computing coefficients of friction is difficult and can lead to differing

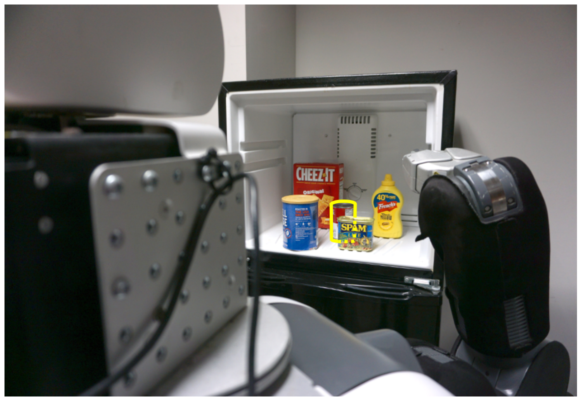


Figure 4.8: Real-world setup for pick-and-place MAMO experiments with a PR2. The tomato soup can (yellow outline) is the object-of-interest.

contact mechanics in simulation than the real-world. Figure 4.7 shows the solution to a MAMO problem being executed by the PR2. It moves the coffee can out of the way, pushes the potted meat can slightly aside, and finally the OoI (tomato soup can) is extracted while also nudging the potted meat can.

4.5.3 Comparison of MAPF Solvers

M4M uses a resolution complete MAPF solver (CBS) to ensure it does not miss any potential rearrangement of a scene (with respect to the graph G_{CBS} of the low-level CBS searches). Another class of MAPF solvers assigns priorities to agents and solves a sequence of single-agent planning problems based on this prioritisation [42]. This *prioritised planning* (PP) scheme trades off algorithmic incompleteness and solution suboptimality for practical efficiency. For all the problems solved by **M4M** in Table 4.1, we compare the performance of CBS against PP in terms of success rates for an initial solution and planning times.

Being provably complete, CBS succeeds in finding an MAPF solution 100% of the time. Given the same 30 s timeout as CBS, PP failed to find solutions for 2 Level 1 problems, 9 Level 2 problems, and 9 Level 3 problems. The bottleneck for MAPF is collision checking between objects in $SE(3)$. CBS only collision checks solution trajectories returned by the low-level searches of the corresponding agents. In addition to being incomplete, PP turns out to also be much slower than CBS. This is because PP collision checks every state expanded by the low-level search against corresponding states of higher priority agents, which is slow when many agents collide with each other along their solution paths. We compare the ratio of planning times for CBS to those for PP across the three levels. The median value of this ratio $T_{\text{CBS}}/T_{\text{PP}}$ for Level 1 is 1.22 (PP is at least 22% faster in half the problems). For Levels 2 and 3, this value is 0.89 (PP is at least 11% slower in half the problems) and 0.63 (PP is at least 37% slower in half the problems). The ability to solve all problems and in less time when the MAMO problem is more complicated highlights the benefit of using CBS over PP.

4.6 Conclusion and Discussion

This chapter presents **M4M**: Multi-Agent Pathfinding for Manipulation Among Movable Objects, an algorithm to plan for manipulation in heavy clutter that considers complex interactions such as rearranging multiple objects simultaneously, and tilting, leaning and sliding objects. These MAMO problems include interaction constraints that define how the robot is allowed to interact with objects. **M4M** decouples the search over all rearrangements of movable objects from the need to query a physics-based simulator. It first constructs and solves an appropriate MAPF abstraction for MAMO to search over all rearrangements without the need to query a physics-based simulator. This MAPF solution helps the push planner generate informed non-prehensile rearrangements that are simulated for interaction constraint verification. We recognise that if we artificially actuate these movable objects and solve an appropriately constructed abstract MAPF problem, the solution informs us of a suitable rearrangement for completing the original MAMO task. The MAPF formulation searches over object configurations without a simulator, and upon returning a solution, **M4M** computes non-prehensile push actions to realise the suggested rearrangement within the simulator. Thus, **M4M** is able to find a suitable rearrangement of the MAMO workspace without querying a physics-based simulator and attempts to rearrange movable clutter without needing to search the space of all rearrangements of the scene for an appropriate one to complete the MAMO task. It dramatically outperforms alternative approaches that do not reason about such interactions efficiently.

The key contribution of this chapter, an application of MAPF solvers within MAMO, leads to several distinct areas of future research. **M4M** greedily commits valid pushes found to its sequence of rearrangement trajectories. This greedy behaviour makes **M4M** incomplete (it may not find a solution to a MAMO problem even if one exists), given that it has no ability to backtrack from this decision. It is important to address this incompleteness of **M4M** by developing an algorithm that considers (i) all feasible pushes for an object that needs to be rearranged to a specific location, (ii) all orderings of all feasible push actions to realise a particular rearrangement for a set of objects, and (iii) all possible rearrangements for a set of objects. Our work in Chapter 5 makes progress towards such an algorithm. Additionally, the MAPF solver used in **M4M** should be modified to use a cost function which has information about robot kinematics and pushing dynamics so as to compute and thus simulate better push actions. Using a model-based push planner, even for simple straight-line pushes like those used by **M4M**, will greatly reduce the time **M4M** currently spends stochastically sampling and simulating valid pushes. **M4M** currently terminates with success when it has found a complete sequence of rearrangement actions that help solve the MAMO task. Modifying it to interleave planning and execution can help introduce robustness by allowing **M4M** to replan from the current rearrangement of the scene. We can also augment the MAPF solver with information from past execution steps that constrain the motions of some movable objects.

Chapter 5

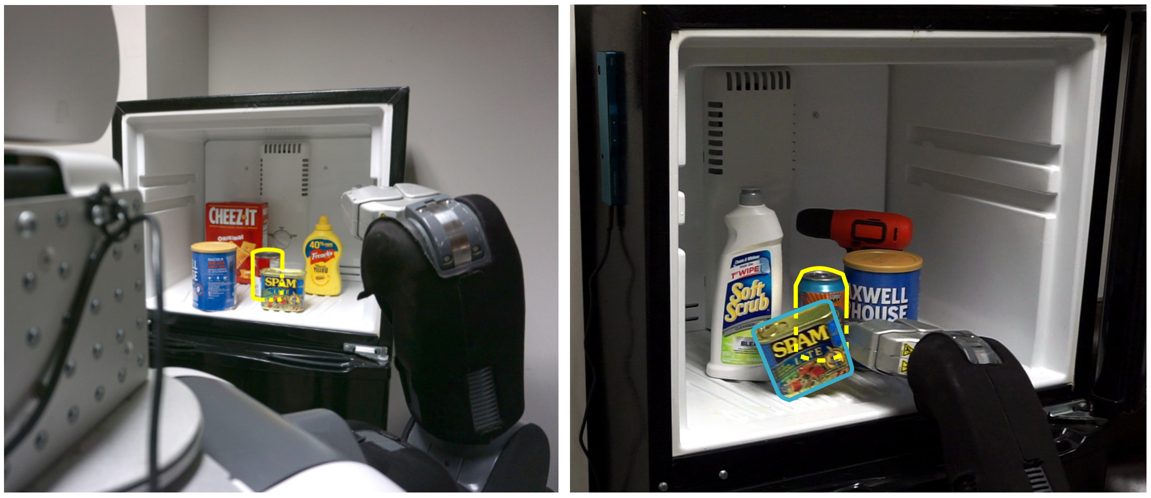
A Graph Search Formulation of Manipulation Among Movable Objects

In this thesis we are interested in pick-and-place style robot manipulation tasks in cluttered and confined 3D workspaces among movable objects that may be rearranged by the robot and may slide, tilt, lean or topple as the robot interacts with them. The algorithm we presented earlier in Chapter 4, **M4M**, determines *which* objects need to be moved and *where* by solving a Multi-Agent Pathfinding (MAPF) abstraction of this problem. It then utilises a non-prehensile push planner to compute actions for *how* the robot might realise these rearrangements and a rigid body physics simulator to check whether the actions satisfy physics constraints encoded in the problem. However, **M4M** greedily commits to valid pushes found during planning, and does not reason about orderings over pushes if multiple objects need to be rearranged. Furthermore, **M4M** does not reason about other possible MAPF solutions that lead to different rearrangements and pushes. This chapter extends **M4M** and we present Enhanced-**M4M** (E−**M4M**) – a systematic graph search-based solver that searches over orderings of pushes for movable objects that need to be rearranged and different possible rearrangements of the scene. We introduce several algorithmic optimisations to circumvent the increased computational complexity, discuss the space of problems solvable by E−**M4M** and show that experimentally, both on the real robot and in simulation, it significantly outperforms the original **M4M** algorithm, as well as other state-of-the-art alternatives when dealing with complex scenes. To address the higher computational complexity associated with searching for a solution in this much larger space, E−**M4M** stores information about all successful and unsuccessful pushes found during planning. The former are used to avoid simulations of similar pushes seen previously, and the latter help us feedback information to the MAPF solver to efficiently search the space of rearrangements of the scene. The work in this chapter was first published in [127].

5.1 Introduction

Simple pick-and-place robot manipulation tasks can be difficult to solve for motion planning algorithms that do not reason about how 'movable' objects in the confined workspace might need to be rearranged in order to find a feasible solution path. Such situations are commonly encountered when robot arms have to grasp and extract desired objects from cluttered shelves or pack several objects in a box. Solving these "Manipulation Among Movable Objects" (MAMO) problems [5, 144] requires a planning algorithm to decide *which* objects should be moved [52], *where* to move them, and *how* they may be moved. For the scene shown in Figure 5.1 (a), the tomato soup can is the "object-of-interest" (OoI) to be retrieved. In order to do so, the PR2 robot must first move the coffee can and potted meat can out of the way so that the grasp pose for the OoI becomes reachable.

Existing state-of-the-art approaches in literature commonly assume prehensile (pick-and-place) rearrangement actions, e.g. Stilman et al. [144], Wang et al. [164] and/or planar robot-object and object-object interactions, e.g. van den Berg et al. [159], Vieira et al. [161]. Prehensile rearrangements not only preclude manipulation of big, bulky and otherwise ungraspable objects, but also assume access to known grasp poses for all movable objects and availability of stable placement locations for them in a cluttered and confined workspace. The planar world assumption does not account for realistic physics interactions between objects in a real-world scene. In contrast to these, our emphasis is on solving MAMO problems (i) in a 3D workspace where robot actions can lead to complex multi-body interactions where objects tilt, lean on each other, slide, and topple (Figure 5.1 (b)); and (ii) with non-prehensile



(a) (b)

Figure 5.1: (a) The tomato soup can (yellow outline) is the object-of-interest (OoI) to be retrieved. The potted meat can and coffee can in front of it must be rearranged out of the way in order to retrieve the OoI and solve the MAMO problem. (b) Trying to retrieve the beer can (OoI, yellow outline) leads to a complex interaction with the movable potted meat can being tilted by the robot arm.

push actions for rearranging the clutter in the scene. With this we allow for more seamless and natural manipulation that rearranges objects aside without picking them up while considering complicated toppling, sliding, and leaning effects.

The MAMO problem definition includes information about which objects are *movable* and which are static or *immovable* obstacles. All objects have a set of *interaction constraints* associated with them that define valid robot-object and object-object interactions in the workspace. Interaction constraints encode that neither the robot nor any other object can make contact with immovable obstacles (an object that cannot be interacted with, such as a wall), and movable objects cannot fall off the shelf, tilt too far (beyond 25°), or move with a high instantaneous velocity (above $1\,\mathrm{m\,s^{-1}}$). These constraints help model realistic and desirable robot-object interactions since we want to prevent robots from carelessly hitting, pushing or throwing objects around. In order to forward simulate the effect of non-prehensile robot pushes on the objects, we use a rigid body physics simulator to evaluate the interaction constraints and determine the resultant state of the workspace.

In recent work [128], we proposed the **M4M** algorithm for MAMO to answer the questions of *which* objects to move *where*, and *how*. **M4M** ("Multi-Agent Pathfinding for Manipulation Among Movable Objects") relies on an MAPF abstraction of MAMO problems where the movable objects are artificially actuated agents with the goal of avoiding collisions with (i) the robot arm as it retrieves the OoI, (ii) each other, and (iii) immovable obstacles. A solution to this MAPF abstraction informs **M4M** of *which* objects must be moved and *where* so that the OoI can be retrieved to solve the MAMO problem. **M4M** then samples non-prehensile pushes to try and realise the rearrangements suggested by the MAPF solution in the real-world, thereby addressing the third question of *how* objects may be moved.

However, **M4M** is greedy and can fail to find solutions in many cases where one may exist. It is greedy in three different ways. First, **M4M** does not search over all possible orderings of object rearrangements if multiple movable objects need to be moved. It greedily commits to the first valid push it finds and continues searching for a solution from the resultant state of that push. Second, **M4M** never reconsiders solving the MAPF problem again for a different solution that might require objects to be rearranged differently. As such, it does not search over all possible rearrangements of the scene. This is important because in cases where an object cannot be rearranged successfully as per the MAPF solution (perhaps due to robot kinematic limits, the presence of immovable obstacles, interaction constraint violations etc.), we must replan the MAPF solution and consider a different way to rearrange the scene that may indeed be feasible. Finally, even in cases where we successfully rearrange an object to a particular location, **M4M** never reconsiders moving that object differently, which may be required if no solution can be found from the resultant state of the valid push.

This chapter extends **M4M** and presents Enhanced-**M4M** (E−**M4M**), an algorithm that addresses all three shortcomings of **M4M** and does so by searching a much larger space for solutions. It considers different orderings for rearranging objects, replans MAPF solutions as and when required, and considers different ways to rearrange any particular object. Although the search space for E−**M4M** grows tremendously as a result, E−**M4M** exploits the information gained during its execution to reduce redundant exploration of the solution space. There is redundancy in considering the same or similar pushing actions for an object in different nodes of E−**M4M** search tree such that if one action succeeds or fails (i.e. its validity is determined by forward simulating it for interaction constraint verification), it is likely that the other actions will succeed or fail as well. We exploit this by introducing caching of positive and negative simulation results and learning a probabilistic estimate of solving a particular subtree of the search, and use these within E−**M4M** to bias its exploration. We make the following contributions as part of our E−**M4M** algorithm:

1. A best-first graph search for MAMO problems that searches over orderings of object rearrangements, different rearrangements of the scene, and different ways to rearrange each object.
2. Caching results of successful (valid) pushes to avoid simulations of similar pushes repeatedly.
3. Caching results of unsuccessful (invalid) pushes to feedback information to the MAPF solver to efficiently search the space of rearrangements of the scene.
4. A learned probabilistic model for solving a particular subtree to bias exploration of the best-first search.
5. Significant quantitative improvements over **M4M** and several other state-of-the-art MAMO baselines.

5.2 Related Work

In recent years, MAMO planning algorithms have continued to rely on at least one of two simplifying assumptions. The first limits the action space of the robot to prehensile or pick-and-place rearrangements of movable objects [79, 80, 86, 103, 135, 144, 164]. This simplifies the planning problem as grasped objects behave as rigid bodies attached to the robot end effector, and rearrangement paths can be computed by avoiding collisions with other objects in the scene. It is important to note that these paths can only be found if we assume (i) all objects that may need to be rearranged are graspable by the robot, (ii) we have known grasp poses for all graspable objects, (iii) the existence of stable placement locations for these objects in the cluttered workspace, and (iv) a relatively large volume of object-free space so that collision- and contact-free rearrangement paths with a grasped object exist. In our work, we make none of these assumptions, instead relying on non-prehensile pushing actions to rearrange the scene. This allows us to manipulate a much larger set of objects, and also lets us rearrange multiple objects simultaneously. However, using these actions within a planning algorithm necessitates the ability to accurately predict their effect on the configurations of movable objects in order to compute the resultant state of the world after the push. In case the assumptions stated earlier are indeed true for the problems being solved, the E-**M4M** algorithm can be easily modified to include pick-and-place rearrangement actions as part of the action space of the robot within its graph search, something we will include in E-**M4M** in Chapter 6.

The second simplification assumes planar robot-object and object-object interactions, while allowing non-prehensile push actions for rearrangement. This planar assumption halves the size of the configuration space of movable objects from $SE(3)$ to $SE(2)$. To predict the effect of push actions on the scene, some existing algorithms make use of simple analytical or learned physics models [35, 59, 159], while others use computationally cheap 2D physics simulators [60, 68]. Assuming planar interactions does not capture the complex multi-body physics of the 3D real-world where objects may tilt, lean on each other, topple etc., something we account for in the E-**M4M** algorithm. As part of our experiments, we compare against an implementation of the algorithm from [35] that uses the same non-prehensile push planner as E-**M4M** in conjunction with a 3D rigid body physics simulator. The original algorithm is not viable for our MAMO problems as it uses a 2D analytical model to predict the result of planar robot-object interactions, and only allows the robot to rearrange one object at a time.

Existing work which uses a full 3D rigid body physics simulator to forward simulate the effect of robot actions on the scene does not account for the difficult interaction constraints we include in our MAMO problems which makes it harder to find a feasible solution. Instead, they either only deal with simple constraints [130, 148] where objects are not allowed to fall off the workspace shelf, or do not include any such constraints [111, 161] and ignore cases where objects topple. We include a comparison against [148] in our experiments, albeit with the full interaction constraint set that E-**M4M** considers.

We also include comparisons against two general purpose sampling-based planning algorithms, KPIECE [147] and RRT [85], and our own recent **M4M** algorithm developed for this MAMO domain. KPIECE is a randomised algorithm developed for planning problems where it is expensive to determine the resultant state of an action (like querying a physics-based simulator for the effect of robot pushes). M4M, as discussed earlier, decouples the search for a solution to the MAMO problem between solving an abstract MAPF problem that reasons about the configuration of movable objects but does not require forward simulating a simulation-based model, and solving a simulation-based arm motion planning problem that does not need to search over the possible configurations of movable objects.

5.3 Problem Formulation

We are interested in solving MAMO problems with a q degrees-of-freedom robot manipulator $\mathcal{R}$ whose configuration space $\mathcal{X}_{\mathcal{R}} \subset \mathbb{R}^q$. The workspace is populated with objects $\mathcal{O} = \{O_1, \ldots, O_n\}$ whose configuration spaces $\mathcal{X}_{O_k} \equiv SE(3)$. We assume we know which objects $\mathcal{O}_M \subset \mathcal{O}$ are *movable* and which objects $\mathcal{O}_I \subset \mathcal{O}$ are *immovable*. Each object O_k is associated with *interaction constraints* described earlier that help determine whether any state x in the search space $\mathcal{X} \coloneqq \mathcal{X}_{\mathcal{R}} \times \mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$ is valid or not. The planning algorithm is provided the initial configurations of all movable objects (denoted as $\mathcal{O}_M^{\text{init}}$) and immovable obstacles ($\mathcal{O}_I$), information about which object is the "object-of-interest" (OoI), desired grasp pose for the OoI, and a "home" configuration outside the workspace shelf where the OoI must be moved. Our goal is to find a path of valid states in $\mathcal{X}$ that successfully retrieves the OoI from the cluttered and confined workspace shelf.

Conceptually this path rearranges some movable objects if necessary and retrieves the OoI while ensuring (i) neither the robot nor any movable object makes contact with immovable 'obstacles', and (ii) movable objects are rearranged satisfactorily (without making them topple, fall off the workspace shelf, move very fast etc.). We make no assumptions about the MAMO problem being *monotone* where each movable object may only be moved once, and we allow the robot to rearrange several movable objects at the same time. We do assume access to a rigid body physics simulator to evaluate the effect of robot actions on the states of the objects in the workspace.

5.4 E–M4M

This chapter presents the E–**M4M** algorithm, an enhanced version of our previous **M4M** algorithm. In this section we provide details about E–**M4M**, the MAPF abstraction and non-prehensile push planner used within it, and discuss when and why E–**M4M** will solve a MAMO problem (or not).

In order to solve the MAMO problems of interest to us, E–**M4M** must answer questions about *which* objects should be moved, *where* they may be moved, and *how* the robot can move them. Like **M4M**, it relies on two modules to answer these questions – an MAPF solver [134] is used to answer the first two questions, while our non-prehensile push planner uses the MAPF solution to try and answer the third. The push planner also updates the feasible solution space for the MAPF solver if it determines that certain rearrangements are infeasible due to interaction constraint violations. Unlike **M4M** however, E–**M4M** runs a best-first search over a graph $G = (V, E)$ with the help of these two modules. The vertices $v \in V$ represent a set of configurations (alternatively a *rearrangement*) of the movable objects $\mathcal{O}_M$ in the scene. Since the configurations of immovable obstacles are known prior to planning and cannot be changed by virtue of the definition of interaction constraints, we do not explicitly store them in each vertex v. Edges $e = (u, v) \in E$ represent a successful rearrangement action changing the configuration of *at least* one $O_m \in \mathcal{O}_M$ between u and v. If a rearrangement action from vertex v is unsuccessful due to any interaction constraint violation or reachability constraint, E–**M4M** will use this information to generate a different rearrangement action from v at a later point in the search. Thus given enough time, for any vertex $v \in V$, E–**M4M** will evaluate all possible rearrangement actions for all objects $O_m \in \mathcal{O}_M$. The overall E–**M4M** search expands vertices in an order dictated by some priority function $f : V \to \mathbb{R}_{\geq 0}$. In contrast **M4M** (i) greedily commits to the first valid push found (it does not search over orderings of rearrangements of multiple movable objects like E–**M4M**), (ii) only obtains a single MAPF solution for each rearrangement it sees (it never replans the MAPF solution based on the result of push actions like E–**M4M**), and consequently (iii) only tries to rearrange a movable object along a single MAPF solution path for each rearrangement (it does not consider alternate ways to push an object for the same rearrangement like E–**M4M**).

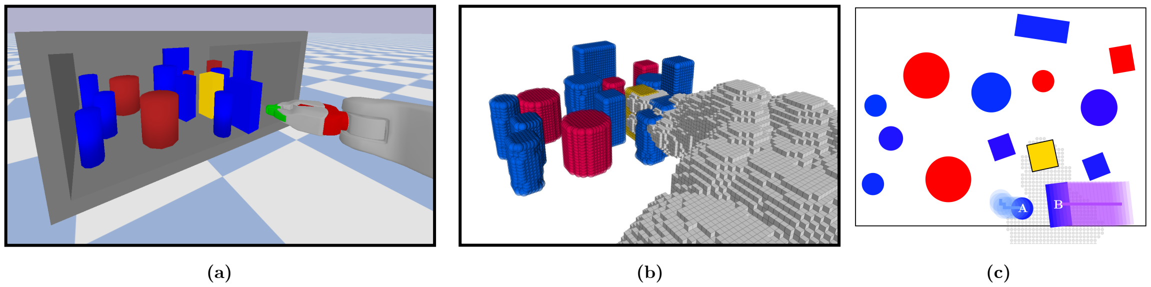


Figure 5.2: (a) A MAMO problem with ten movable objects and four immovable obstacles, (b) the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$ found for this scene (in gray), and (c) a 2D projection of the scene with the MAPF solution paths in pink. This MAPF solution suggests that the objects labeled A and B should be rearranged as per the pink paths to be outside $\mathcal{V}(\gamma_{\text{OoI}})$.

5.4.1 Main Algorithm

Algorithm 5 contains the pseudocode for E-**M4M**. Initially, E-**M4M** computes a trajectory $\gamma_{\text{OoI}} \subset \mathcal{X}_{\mathcal{R}}$ for the robot to grasp and extract the OoI while pretending no movable objects $\mathcal{O}_M$ exist in the scene (Line 28). The argument for the PlanRetrieval function is the set of objects to be considered as obstacles during planning. The volume occupied by the robot arm and OoI during execution of γ_{OoI}, written as $\mathcal{V}(\gamma_{\text{OoI}})$, creates a *negative goal region* (NGR) [35]. We define an NGR parameterised with a robot trajectory as some volume in the workspace that, if free of all objects, will lead to successful retrieval of the OoI upon execution of that robot trajectory. Note that if the trajectory γ_{OoI} cannot be found, the overall MAMO problem as specified is unsolvable. It may be solvable given a different grasp pose for the OoI, however grasp planning is beyond the scope of this work. Once the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$ has been computed (Line 29), E-**M4M** executes a best-first search using a priority queue ordered by f. Figure 5.2 shows a simulated MAMO problem, the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$ for the scene, and a 2D projection of the scene which shows the MAPF solution found.

Every time a vertex v is expanded from this queue during the search (Line 36), E-**M4M** calls an MAPF solver (Line 14). The set of solution paths returned by the MAPF solver is then used by our non-prehensile push planner to generate and evaluate successor rearrangement states (the loop from Line 18). For each object O_m that "moves" in the MAPF solution, our push planner tries to compute a trajectory

Algorithm 5 E–M4M

```
1: procedure CreateVertex($\mathcal{O}_M, v, \gamma$)
2:     $v'.\mathcal{O}_M \leftarrow \mathcal{O}_M, v'.parent \leftarrow v, v'.\gamma \leftarrow \gamma$
3:     return $v'$
4: procedure Done($v$)
5:     if $v.\mathcal{O}_M \cap \mathcal{V}(\gamma_{\text{OoI}}) = \emptyset$ then
6:         return true
7:     $\hat{\gamma}_{\text{OoI}} \leftarrow$ PlanRetrieval($v.\mathcal{O}_M \cup \mathcal{O}_I$)
8:     if $\hat{\gamma}_{\text{OoI}}$ exists then
9:         $\gamma_{\text{OoI}} \leftarrow \hat{\gamma}_{\text{OoI}}$
10:        return true
11:    return false
12: procedure ExpandState($v$)
13:    $\kappa \leftarrow$ InvalidGoals($v$)
14:    $v.\{\pi_m\}_{m \in \mathcal{O}_M} \leftarrow$ RunMAPF($v, \kappa, \mathcal{V}(\gamma_{\text{OoI}})$)
15:    if MAPF failed then
16:        Remove $v$ from $OPEN$
17:        return
18:    for $m \in v.\mathcal{O}_M$ do
19:        if $v.\pi_m \neq \emptyset$ then
20:            $\gamma_m \leftarrow$ PlanPush($v.\pi_m, v.\mathcal{O}_M$)
21:            $\mathcal{O}'_M$, valid $\leftarrow$ IsValid($\gamma_m$)
22:            if valid then
23:                $v' \leftarrow$ CreateVertex($\mathcal{O}'_M, v, \gamma_m$)
24:                Insert $v'$ into $OPEN$ with priority $f(v')$
25:            else
26:                Add final state in $v.\pi_m$ to InvalidGoals($v$)
27: procedure Main($\mathcal{O}_M^{\text{init}}, \mathcal{O}_I$)
28:    $\gamma_{\text{OoI}} \leftarrow$ PlanRetrieval($\mathcal{O}_I$)
29:    Compute $\mathcal{V}(\gamma_{\text{OoI}})$
30:    $OPEN \leftarrow \emptyset$, $v_{\text{start}} \leftarrow$ CreateVertex($\mathcal{O}_M^{\text{init}}, \emptyset, \emptyset$)
31:    Insert $v_{\text{start}}$ into $OPEN$ with priority $f(v_{\text{start}})$
32:    while $OPEN$ is not empty and time remains do
33:        $v \leftarrow OPEN$.top()
34:        if Done($v$) then
35:            return ExtractRearrangements($v$)
36:        ExpandState($v$)
37:    return $\emptyset$
```

$\gamma_m \subset \mathcal{X}_\mathcal{R}$ to push O_m along its MAPF solution path π_m (Line 20). If γ_m is found, it is forward simulated in a rigid body physics simulator for interaction constraint verification (Line 21). If γ_m successfully rearranges the scene, i.e. at least one object is moved and no constraints are violated, E−**M4M** generates a successor state v' with the resultant rearrangement and adds it to the queue (Lines 23 and 24).

A vertex v is *closed* in Line 16 and never re-expanded again *iff* the MAPF solver fails to return a solution in Line 14 as this implies there are no more rearrangement actions for us to try given the configurations of movable objects in that vertex v. Otherwise when a vertex v is re-expanded, we ensure that the MAPF solver returns a different solution than one obtained during any previous expansion of v by including a set κ of invalid goals (Line 13). κ contains configurations for each movable object $O_m \in v.\mathcal{O}_M$ that *cannot* be the final state in the path π_m found by the MAPF solver. This helps E−**M4M** search over different MAPF solutions for the same rearrangement $v.\mathcal{O}_M$, thereby helping it search over different ways to rearrange $v.\mathcal{O}_M$. Invalid goals are populated in κ in two ways – all push actions γ_m found to be invalid lead to the final state of the corresponding π_m being included as an invalid goal for O_m (Line 26); additionally if no such invalid pushes remain to be added to κ, the final states of valid pushes found previously from v are “hallucinated” as invalid goals.

E−**M4M** terminates with v as the goal state when the OoI can be successfully retrieved given the rearrangement $v.\mathcal{O}_M$. This may be achieved in one of two ways. If the movable objects $\mathcal{O}_M$ in v have been successfully rearranged to be outside the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$, we know the robot can execute γ_{OoI} to retrieve the OoI (Line 6). Alternatively, if a different trajectory $\hat{\gamma}_{\text{OoI}}$ (and therefore its NGR $\mathcal{V}(\hat{\gamma}_{\text{OoI}})$) can be found in the presence of all objects (movable and immovable) as obstacles, the robot can execute $\hat{\gamma}_{\text{OoI}}$ to retrieve the OoI without making contact with any other object (Line 10). Thus E−**M4M**, like **M4M**, implicitly makes the assumption that the OoI retrieval trajectory will only make contact with the OoI.

Figure 5.3 shows the entire graph constructed by E−**M4M** to solve the problem from Figure 5.2. E−**M4M** finds a sequence of four pushes to rearrange the scene in order to retrieve the OoI. Note that in the solution both pushes that were computed to rearrange object B also move object C. This is allowed since E−**M4M** places no restriction on the number of movable objects that may be moved during a pushing action.

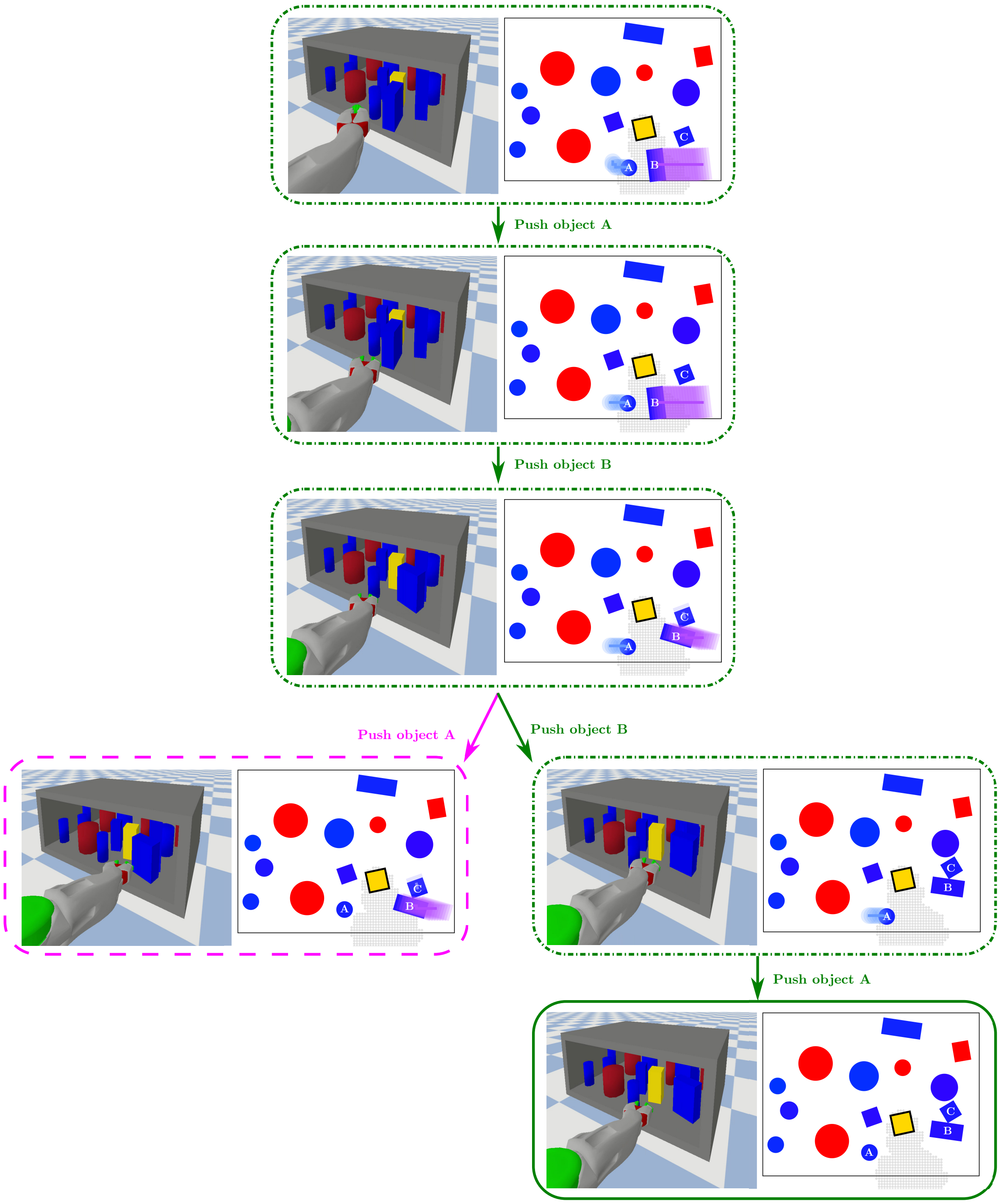


Figure 5.3: The graph constructed by E−M4M to find a solution to the MAMO problem from Figure 5.2. Within each graph vertex we show an image of the 3D scene in simulation and its 2D projection to visualise the MAPF solution found by CBS.

5.4.2 MAPF Abstraction

E−**M4M** relies on the same key observation as **M4M** (Section 4.4.1) that by solving a carefully constructed MAPF problem with movable objects as agents, we can obtain information about which objects our MAMO planner should consider rearranging and where. As before, we use Conflict-Based Search (CBS) [134] as the solver for our abstract MAPF problem. For a vertex v in E−**M4M**, we include all movable objects as agents in CBS starting at their current poses in $v.\mathcal{O}_M$. Each agent is assigned a goal of being outside the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$, while avoiding collisions with each other and all immovable obstacles $\mathcal{O}_I$. Although agent states in CBS specify their configuration in $\mathcal{X}_{O_m} \equiv SE(3)$, agents use a 2D action space on a four-connected grid in the MAPF abstraction that only changes the $x-$ or $y-$coordinates of their state. The CBS solution for this MAPF problem is a set of paths $\{\pi_m\}_{m \in \mathcal{O}_M}$ such that O_m ends up outside $\mathcal{V}(\gamma_{\text{OoI}})$ after following π_m.

In order to search over all possible ways to rearrange $v.\mathcal{O}_M$, E−**M4M** includes a set of invalid goals κ when it calls CBS in Line 14. For each vertex v, κ includes information about where each object $O_m \in v.\mathcal{O}_M$ *cannot* end up in any future CBS solution. During a previous expansion of v, if path π_m led to an invalid push γ_m, the final state of π_m is added as an invalid goal for $O_m \in v.\mathcal{O}_M$ since we failed to rearrange O_m along π_m (Line 26). As a result, the next solution from CBS would return a new path $\pi'_m \neq \pi_m$ which in turn would cause our push planner to consider a different rearrangement action. When all invalid pushes from v have been included in the previous call to CBS, we also include the final states of valid pushes found previously as invalid goals in κ so as to ensure we consider all possible ways to rearrange $v.\mathcal{O}_M$. If CBS fails to find a solution, or if no new invalid goals can be added to κ from the last call to CBS, we *close* vertex v and stop it from being re-expanded (Line 16) since no new way to rearrange $v.\mathcal{O}_M$ can be found. Figure 5.4 (a) shows another simulated MAMO problem and the effect that invalid goals κ can have on the MAPF solution – when both pushes γ_A and γ_B computed as per the MAPF solution in (b) failed, adding the final states of π_A and π_B to κ results in a different MAPF solution in (c).

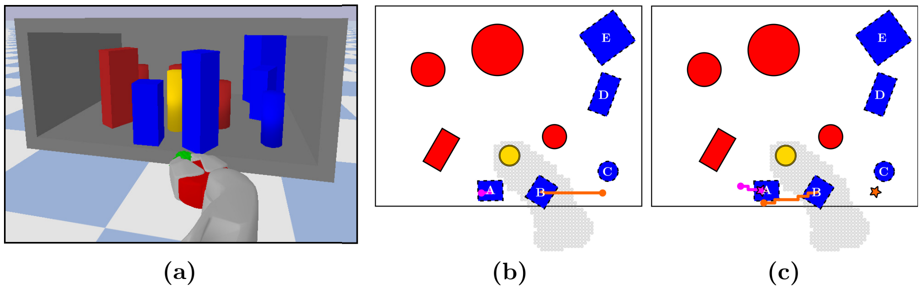


Figure 5.4: (a) A MAMO problem with five movable objects and four immovable obstacles, (b) First MAPF solution that led to invalid pushes γ_A and γ_B, (c) Adding the final states of π_A and π_B (colour coded stars) to κ leads to a new MAPF solution.

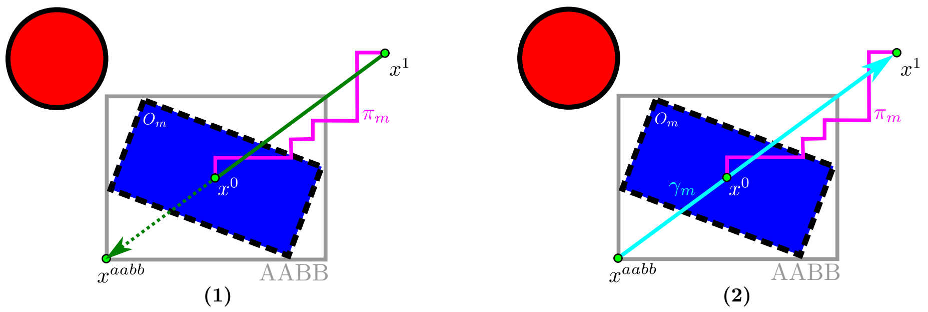


Figure 5.5: 2D illustration of our push planner. Movable object O_m is blue, and an immovable obstacle is drawn in red. The green path is obtained after shortcutting the pink MAPF solution path π_m. x^{aabb} is the point-of-intersection between the first segment (x^1, x^2) and the axis-aligned bounding box for O_m. The cyan segments depict the path along which inverse kinematics is used to obtain γ_m^i.

5.4.3 Non-prehensile Push Planner

The goal for our push planner is to find robot trajectories $\gamma_m \subset \mathcal{X}_\mathcal{R}$ that rearrange a movable object along the path π_m returned by our MAPF solver. If we are able to precisely rearrange each O_m to the final state of π_m, we know the robot can execute γ_{OoI} to solve the MAMO problem. In order to do so, we assume the push planner is provided a shortcut path π_m (accounting for immovable obstacles $\mathcal{O}_I$) in Line 20.

Each call to PLANPUSH stochastically generates a robot trajectory γ_m, as discussed earlier in Section 4.4.2. For ease of understanding, Figure 5.5 shows a simplified, deterministic version of our push planner. The push planner first tries to compute a trajectory to a end-effector pose in $SE(3)$ from where we would like to push the intended object. In the simplest version of our push planner, the push start pose is the point of intersection x^{aabb} between the axis-aligned bounding box of O_m and the ray $\overrightarrow{(x^1, x^0)}$ (from the final state in π_m to the initial state). If the motion planner for the robot arm succeeds in finding this trajectory $\gamma_m^0 \subset \mathcal{X}_\mathcal{R}$ to x^{aabb} in the presence of all objects (movable and immovable) as obstacles – depicted by Step (1) in Figure 5.5 – we will go on to compute the pushing action in Step (2). We use an inverse kinematics solver for our robot $\mathcal{R}$ to compute a sequence of joint configurations that take the end-effector of $\mathcal{R}$ in a straight-line from x^{aabb} to x^1. If we succeed in computing the pushing action γ_m^1 in this way, we append it to γ_m^0 to obtain the full pushing trajectory. However, since γ_m^0 is known to be contact-free by construction, we only forward simulate γ_m^1 in our physics-based simulator to detect if any interaction constraints are violated during its execution and get the resultant rearrangement of the scene. In addition, by ensuring that we retract the robot arm to the final configuration in γ_m^0 after executing the push action γ_m^1, we can guarantee that the robot can move from one push to the next entirely in free space without making contact with any object.

5.4.4 What E–M4M Can and Cannot Solve

Solutions to MAMO problems lie in a space $\mathcal{X} = \mathcal{X}_\mathcal{R} \times \mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$ that grows exponentially with the number of movable objects. There are several subtle reasons due to which E–**M4M** might fail to find solutions to complicated MAMO problems in this space. A rigorous theoretical analysis of E–**M4M** requires careful consideration of three different graphs that it searches – the high-level graph G with vertices that denote rearrangements of movable objects and edges that encode rearrangement actions from our non-prehensile push planner, the graphs with vertices in $SE(3)$ with edges corresponding to a 2D action space on a four-connected grid used by the single-agent searches within our MAPF solver CBS, and finally a graph with vertices in $\mathcal{X}_\mathcal{R}$ and edges representing changes in one of the q degrees-of-freedom that is used when computing any robot trajectory (within the PlanRetrieval and PlanPush functions in Algorithm 5).

When trying to rearrange an object to a specific location, E–**M4M** only considers moving it along the particular path π_m found by CBS and not along all such paths. E–**M4M** does not compute all ways to push an object to a particular location, i.e.

it does not compute all paths (and by extension pushes) that end in the same final state, instead only computing a push γ_m for the particular π_m found by CBS. Its reliance on CBS and our push planner also means that E−**M4M** might fail to find interesting non-monotone solutions where it must rearrange an object O_a partially along its solution path π_a before moving O_b along π_b and finally going back to move O_a the remainder of the way along π_a. The MAPF abstraction itself uses a simple 2D action space which fails to capture all possible rearrangements of the scene in $\mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$. Finally, E−**M4M** does not actively search over all OoI retrieval trajectories $\gamma_{\text{OoI}} \subset \mathcal{X}_{\mathcal{R}}$, and consequently the same NGR is used to specify goals for movable objects in all MAPF calls.

Despite these limitations, by relying on our MAPF abstraction and non-prehensile push planner, E−**M4M** does search over 'allowed' (i) orderings of movable object rearrangements, (ii) potential rearrangements for the set of movable objects, and (iii) ways to rearrange the same movable object. In doing so it makes progress towards a complete MAMO planning algorithm that uses non-prehensile actions for rearrangement in a 3D workspace with complex multi-body interactions where movable objects may tilt, lean, topple etc. Our quantitative analysis shows that E−**M4M** performs better than many state-of-the-art planning algorithms for these MAMO problems.

5.5 Speeding up the Algorithm

This section discusses three algorithmic optimisations we propose as part of E−**M4M** that significantly improve its quantitative performance. We discuss caching information from successful and unsuccessful rearrangement actions to speed up the E−**M4M** search, and a data-driven priority function that learns a naive probabilistic estimate of the subtree at a vertex v in the E−**M4M** graph being solvable based on features of the rearrangement $v.\mathcal{O}_M$.

5.5.1 Caching Unsuccessful Push Actions

The goal set for movable objects in the MAPF abstraction includes any configuration outside the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$ that is free of collision from all other objects. Given vertex v, consider an object O_m in $v.\mathcal{O}_M$ and its corresponding path π_m which achieves this goal. If the resulting push γ_m is invalid, the next call to CBS from v will lead to a path $\pi'_m \neq \pi_m$. However, naively including the last state in π_m as an invalid goal state for CBS will likely lead to the new path π'_m ending in a neighbouring state of the invalid goal (since CBS is an optimal MAPF solver). This in turn will lead to a push $\gamma'_m \approx \gamma_m$ that is also likely to be invalid.

To mitigate this, for every CBS call from a vertex v, for each object O_m, E-**M4M** caches the goals that were previously determined to be invalid in a nearest neighbour data structure. We use this cached information to bias the solutions produced by CBS to avoid moving objects to states close to known invalid goals for the respective objects. This biasing is done by assigning penalties to each potential goal location for each object O_m, and then during each low-level search within CBS finding the solution that minimises the summation of getting to a goal plus the penalty associated with the goal. This can be done by introducing one single 'pseudogoal' that the search searches towards and connecting all the potential goal locations to this 'pseudogoal' with edges whose cost is proportional to the respective penalty. This helps penalise paths to states close to known invalid goals, and lets E-**M4M** search the space of allowed rearrangements of $v.\mathcal{O}_M$ more efficiently. For the same MAMO problem as in Figure 5.4, Figure 5.6 (a) shows the new MAPF solution when we include invalid goals naively – the new paths π'_A and π'_B are very similar to π_A and π_B (from Figure 5.4 (b)) and end in final states very close to the invalidated goals. However, using the above explained approach that modifies the single-agent search to use our nearest neighbour data structure for invalid goals, we get a very different MAPF solution in Figure 5.6 (b). This allows us to search the space of possible rearrangements much more efficiently.

5.5.2 Caching Successful Push Actions

During the search, CBS solutions in different vertices of the graph E-**M4M** may contain the same path π_m for object O_m. This can occur if the best rearrangement for O_m is unaffected by the different configurations of other objects in these vertices. In such cases, if we have computed the push γ_m in one of these vertices and found it to be valid in simulation while generating the successor rearrangement state, we

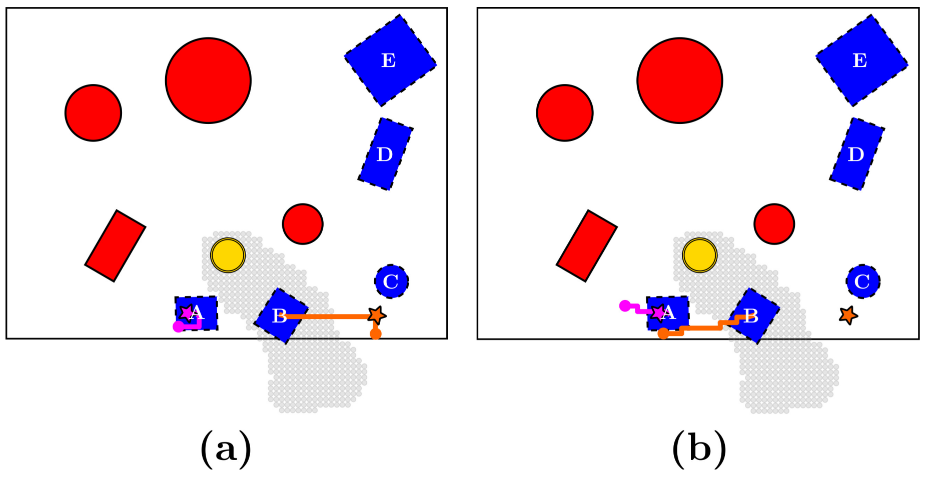


Figure 5.6: (a) First MAPF solution that led to invalid pushes γ_A and γ_B, (b) Adding the final states of π_A and π_B (colour coded stars) to κ leads to a new MAPF solution.

would like to reuse this result whenever possible since simulating a push action is computationally expensive (around $2-6$s per action simulation). This reuse of successful push actions is enabled by storing the successful pushes in a database.

If a push γ_m from path π_m for object O_m rearranged the scene from $v.\mathcal{O}_M$ to $v'.\mathcal{O}_M$, we index into this database with the key (O_m, π_m). While any push is simulated, we keep track of objects that are *relevant* for that push – these are all objects whose configurations are changed between $v.\mathcal{O}_M$ and $v'.\mathcal{O}_M$. For each (O_m, π_m) tuple, the database stores the value $(v.\mathcal{O}_M, v'.\mathcal{O}_M, \gamma_m, \textit{relevant}$ objects$)$. During the expansion of some other vertex u, if CBS returns the same path π_m for O_m, we try to reuse the result of the stored push γ_m to generate the successor state u' corresponding to rearranging O_m. However, this reuse is only possible if all *relevant* objects are in the same configurations in $v.\mathcal{O}_M$ and $u.\mathcal{O}_M$, and all other 'irrelevant' objects in $u.\mathcal{O}_M$ are in configurations that will not be affected by γ_m. If both these conditions are true, we can simply reuse the result of γ_m stored in the database to say that the *relevant* objects in $u.\mathcal{O}_M$ will be rearranged to their respective configurations $v'.\mathcal{O}_M$, and the 'irrelevant' objects will remain unaffected.

5.5.3 Learned Priority Function

E-M4M searches an extremely large space for MAMO solutions since it searches over orderings of object rearrangements, different rearrangements of the scene, and different ways to rearrange each object. In an attempt to focus its search effort on more promising vertices of the search tree, we learn to predict the probability of a particular vertex leading to a solution for the MAMO problem. The learned function is used as the priority function f in the best-first E-M4M search. We predict the probability of a vertex v leading to a solution based on features of the rearrangement $v.\mathcal{O}_M$ – the percentage volume of the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$ occupied by movable objects (ϕ_1), the number of movable objects inside the NGR (ϕ_2), and for each such object the product of its mass, coefficient of friction and percentage volume inside the NGR (ϕ_3). These features indicate how difficult it is to clear the NGR and therefore solve the MAMO problem. Our predictive model f makes the Naive Bayes assumption [63] that these features are conditionally independent of the others. Thus if E is the event that vertex v leads to a MAMO solution,

$$\begin{aligned} f(v) = P(E) \times P(\phi_1 \,|\, E) \times P(\phi_2 \,|\, E) \\ \times \Pi_{O_m \in \mathcal{V}(\gamma_{\text{OoI}})} P(\phi_3 \,|\, E) \end{aligned}$$

We model $P(\phi_1 \,|\, E)$ as a beta distribution, and $P(\phi_2 \,|\, E)$ and $P(\phi_3 \,|\, E)$ are both exponential distributions. Their parameters are estimated via maximum likelihood estimation from a dataset of self-supervised MAMO problems. We generate this set of problems by running E-M4M breadth-first on 20 MAMO scenes with a 10 min planning timeout. We store each vertex generated during these E-M4M runs as a separate MAMO problem, and then run E-M4M depth-first with a 60 s timeout on these problems to get our training dataset of 2400 datapoints.

5.6 Experimental Analysis

This section compares the performance of E-M4M against several MAMO baselines, studies the effect of the algorithmic improvements we propose in an ablation study, and provides results from real-world experiments on a PR2 robot.

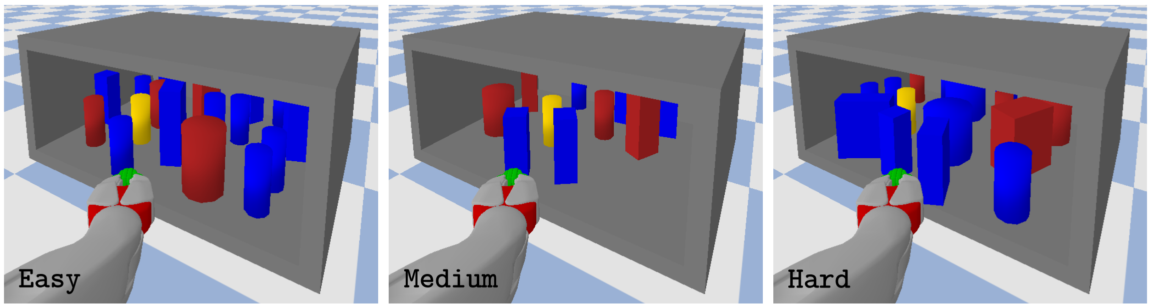


Figure 5.7: Example `Easy`, `Medium`, and `Hard` scenes.

5.6.1 Simulation Experiments Against MAMO Baselines

Our simulation experiments randomly generate MAMO workspaces with one OoI, four immovable obstacles, and five, ten or fifteen movable objects. All object properties (shapes, sizes, mass, coefficient of frictions, initial poses) are randomised and known to each planner prior to planning. We categorise these workspaces into three difficulty levels based on the number of movable objects inside the initial NGR $\mathcal{V}(\gamma_{\text{OoI}})$. Problems are `Easy`, `Medium`, or `Hard` depending on whether there are one, two, or more than two movable objects overlapping with the initial NGR. Figure 5.7 shows sample MAMO workspaces of each difficulty level. We test the performance of all algorithms on 98 `Easy`, 63 `Medium`, and 39 `Hard` problems with a 5 min planning timeout.

In addition to (1) **M4M**, we compare the performance of E−**M4M** against four other baselines. (2) We re-implement DOGAR [35] to use our push planner with a physics-based simulator. It recursively searches backwards in time for objects that need to be rearranged outside the most recent NGR. If it rearranges an object successfully, the volume spanned by the rearrangement trajectory is added to the previous NGR and the recursion continues. However, DOGAR only has information about which objects need to be rearranged but not where they should be moved. Our implementation randomly samples points outside the latest NGR as goal locations for our push planner. (3) SELSIM [148] interleaves planning a trajectory while simulating interactions with 'relevant' objects with tracking the found trajectory in the presence of all objects. If tracking violates interaction constraints, the 'culprit' object is identified and added to the set of relevant objects for the next iteration. SELSIM uses simple motion primitives that change only one of the q degrees-of-freedom of the robot,

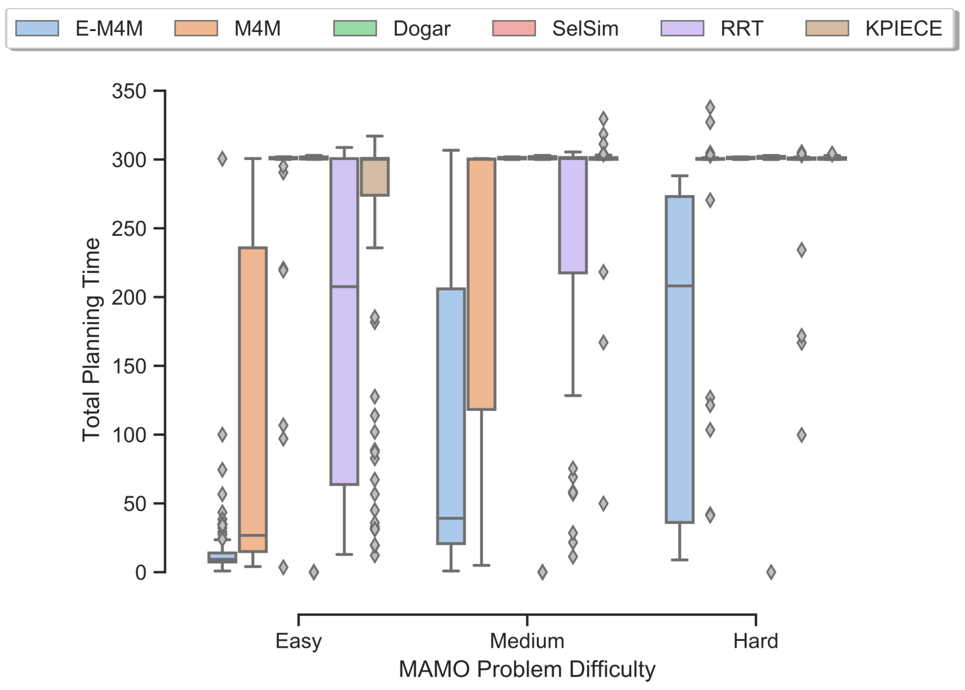
Table 5.1: Number of problems solved by various MAMO planning algorithms in simulation experiments

Difficulty	**Planning Algorithm**					
	E-**M4M**	**M4M**	Dogar	SelSim	RRT	KPIECE
`Easy` (98)	97	78	7	16	33	16
`Medium` (63)	45	25	0	8	7	1
`Hard` (39)	15	7	0	1	1	0

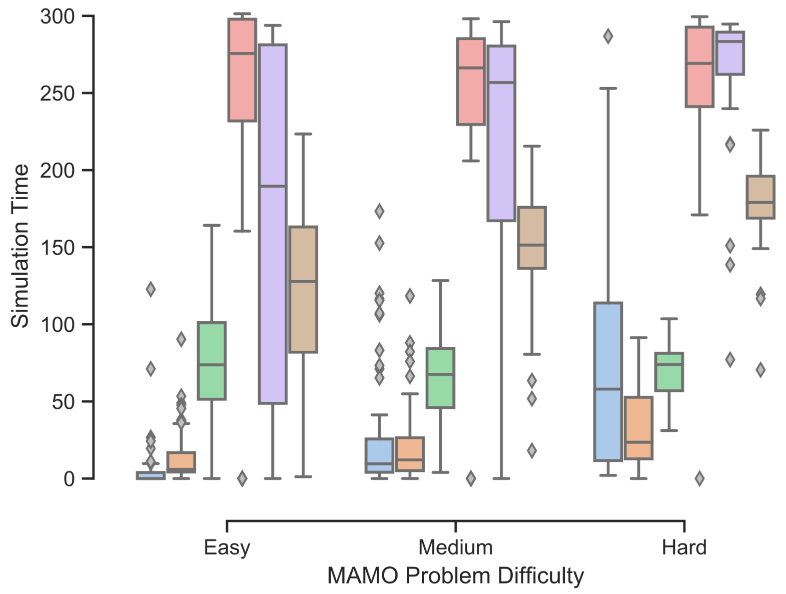
which does not lead to meaningful robot-object interactions in this domain. Finally, we compare against the standard OMPL [146] implementations of general-purpose sampling-based planning algorithms (4) RRT [85] and (5) KPIECE [147].

Table 5.1 contains the number of problems solved by each planning algorithm for the different difficulty levels. It is apparent that E-**M4M** far outperforms all other algorithms, and that all baselines struggle to solve `Medium` and `Hard` problems.

The quantitative performance of all algorithms in terms of total planning time and time spent simulating robot actions is shown in Figure 5.8. E-**M4M** is able to achieve a good balance of time spent computing robot actions (with the MAPF solver and push planner) and forward simulating them for interaction constraint verification. Since **M4M** never replans the MAPF solution, it spends most of its time trying to sample push actions to be simulated. Dogar repeatedly fails to find solutions, even for simple problems, because it (i) has no information about where objects should move, choosing to randomly sample uninformed pushes instead, (ii) only considers pushes to be successful if they rearrange an object to be completely outside the NGR, and (iii) never considers rearranging an object more than once. The performance of SelSim is particularly interesting. It is only able to solve problems where the first planned path succeeds when tracked without any interaction constraints being violated, resulting in negligible planning times for its successes (and no time spent in simulation). Otherwise, owing to its primitive action space, it spends most of its planning budget in simulation trying to rearrange the scene with small robot actions that are incapable of significantly changing object configurations. RRT and KPIECE perform well as they are able to sample long robot motions that can rearrange objects with favourable physical properties (low masses and coefficients of friction) with high likelihood.



(a)



(b)

Figure 5.8: (a) Total planning time and (b) time spent querying a physics-based simulator for MAMO planning algorithms across planning problems with varying difficulty levels.

Table 5.2: Number of problems solved E-**M4M** ablations

Difficulty	**Ablation**				
	E-**M4M**	NEG-DB	POS-DB	NO-DB	TIEBREAK
`Easy` (98)	97	87	86	82	85
`Medium` (63)	45	24	29	25	36
`Hard` (39)	15	7	8	7	13

5.6.2 E-M4M Ablation Study

To study the effect of the algorithmic improvements we propose as part of E-**M4M**, we compare four different versions of E-**M4M**. NEG-DB only caches information from unsuccessful push actions to modify the low-level CBS single-agent search; POS-DB only caches information from successful push actions and tries to reuse their results whenever possible; NO-DB does not cache any information from push actions; and TIEBREAK assigns priorities by lexicographically tiebreaking E-**M4M** vertex feature vectors $(\phi_1, \phi_2, \sum_{O_m \in \mathcal{V}(\gamma_{\text{OoI}})} \phi_3)$. NEG-DB, POS-DB, and NO-DB all use the learned priority function like E-**M4M**, while TIEBREAK caches information from unsuccessful and successful push actions like E-**M4M**.

Table 5.2 shows that each of these E-**M4M** ablations solve fewer MAMO problems in comparison to E-**M4M** which combines all of them. Quantitatively, their performance can be compared from the plots in Figure 5.9. While there is no significant difference between the different ablations for `Easy` problems, for `Medium` and `Hard` problems we can see that performance degrades as we remove cached information. POS-DB performs worse than NEG-DB since it is not as likely for E-**M4M** to find the same push multiple times during a search as it is for it to require several different MAPF solutions. Finally, even with a naive tiebreaking based priority function, TIEBREAK performs only slightly worse than E-**M4M** for `Hard` problems. This suggests that the learned priority function (using the Naive Bayes assumption) is not as useful for these problems, perhaps due to the 60 s timeout imposed during data collection being insufficient to result in a rich set of datapoints for `Hard` problems.

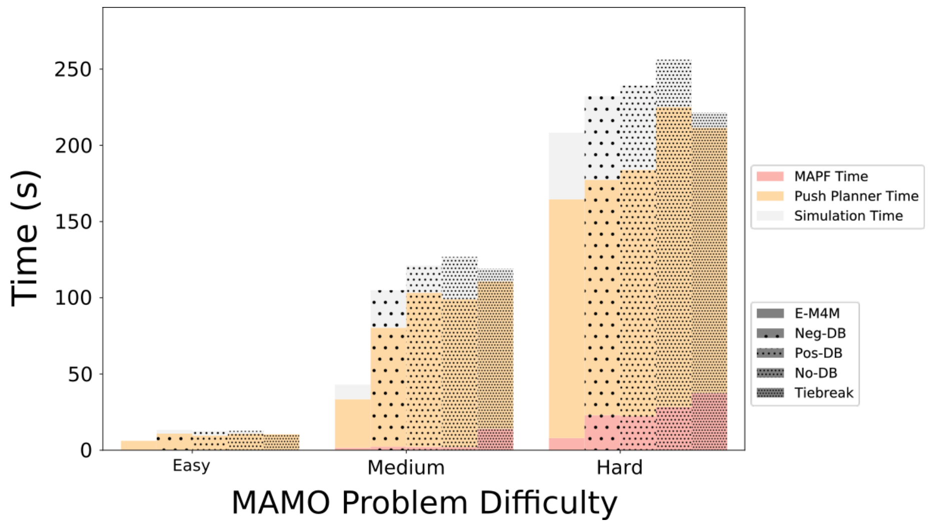


Figure 5.9: Median statistics for time spent calling the MAPF solver, push planner, and simulator for different E−**M4M** ablations.

5.6.3 Real-World Experiments

We ran experiments with the PR2 robot with a refrigerator compartment as our MAMO workspace (Figure 5.1). Problems were initialised with five objects from the YCB Object Dataset [16]. The tomato soup can was always our OoI, while all other objects were initialised as movable. Their initial poses were localised with a search-based algorithm [1] run on a NVidia Titan X GPU.

We ran E−**M4M** on 20 perturbations of the scenes in Figure 5.1 with a 5 min planning timeout. 15 runs resulted in successful OoI retrieval, with the others failing due to unforeseen discrepancies between simulated and real-world robot-object interactions. Four failures were due to inaccurate computation of coefficients of friction of movable objects. One failure was the result of an object getting stuck in ridges in the real-world refrigerator shelf that were not modeled in simulation. These discrepancies highlight the sim-to-real gap that E−**M4M** can suffer from, since it blindly relies on the result of the physics based simulator used in the algorithm. On average, for the 15 successful retrievals, E−**M4M** took a total time of 39.3 ± 28.2 s of which 0.9 ± 1.2 s was spent calling the MAPF solver, 33.5 ± 25.8 s was spent planning pushes, and 7.1 ± 5.3 s was spent simulating them.

5.7 Conclusion and Discussion

The Enhanced-**M4M** algorithm presented in this chapter builds upon our prior work on Multi-Agent Pathfinding for Manipulation Among Movable Objects [128]. E-**M4M** utilises an MAPF abstraction of MAMO, a non-prehensile push planner, and a rigid body physics simulator within a best-first graph search for solving MAMO problems that require determining *which* movable objects should be moved, *where* to move them, and *how* they can be moved. E-**M4M** searches over different orderings of object rearrangements, different rearrangements of the workspace, and different ways to rearrange the same object.

Formulating a graph search for MAMO allows us to lean on a rich history of literature on improving the efficacy of search-based planning algorithms [113]. First and foremost, it becomes trivial to expand the action space $\mathcal{A}$ available to the robot for rearrangement of movable objects. In theory we can add any number of rearrangement actions to this action space – prehensile or pick-and-place style rearrangements, use of tools accessible by the robot, full-body and/or bi-manual manipulation capabilities etc. In practice however, each additional action provided to the robot increases the branching factor of the graph we implicitly construct as part of E-**M4M**, which in turn can increase the computational burden on the planning algorithm as it would need to evaluate more actions per state. Evaluating these edges (Algorithm 1, Line 9) is more often than not the most time consuming step of a graph search-based planning algorithm [22, 102, 105]. In Chapter 6, we will introduce the use of prehensile or pick-and-place style rearrangements in addition to the non-prehensile pushing actions we have focused on thus far.

We can offset some of the computational burden introduced by increasing the branching factor by making our graph search "eagerly lazy" [25]. This is the idea that we only need to evaluate edges (u, v) to vertex v (for any u such that $(u, v) \in E$), when we want to grow the graph beyond vertex v. In our case, this would let us defer running expensive arm motion planning queries and even more expensive forward simulations in our physics-based simulator until absolutely necessary. We modify E-**M4M** to exploit this idea in Chapter 6.

Finally, the work in this thesis until now has assumed that prior to planning we have perfect knowledge of all object parameters – their poses in the environment, dimensions, masses, and coefficients of friction. This is an unrealistic assumption to make for real-world deployment of the algorithms we have developed. In Chapter 6 we will relax the assumption on perfect knowledge of object masses and coefficients of friction in particular as this has received relatively little attention in literature and is extremely important for algorithms that utilise non-prehensile rearrangements. Relaxing this assumption complements ongoing work on accurate localisation of objects in the environment [155, 167] and allowing MAMO solvers to be robust in any errors in object localisation [35].

Chapter 6

Manipulation Among Movable Objects With Diverse Actions and Parameter Uncertainty

In Chapter 5 we introduced E−**M4M** [127], a graph search-based solver for solving Manipulation tasks Among Movable Objects (MAMO). We focused on rearrangement by non-prehensile or pushing actions in 3D workspaces where objects may tilt, topple, lean etc. The E−**M4M** algorithm searches over different orderings of object rearrangements, different rearrangements of the workspace, and different ways to rearrange the same object. In this chapter we make several improvements to E−**M4M** – we introduce the use of prehensile or pick-and-place rearrangement actions in addition to pushes; we show that by running it as a depth-first search improves performance and alleviates the shortcomings of the learned heuristic function used by E−**M4M**; we show how the search can be run "eagerly lazily" to only simulate actions in a physics-based simulator when necessary; finally we relax the assumption that we require perfect knowledge of the physical properties of objects (mass and coefficient of friction in particular) and leverage parallelised simulations to make E−**M4M** robust to uncertainty in these parameters. The improved version of E−**M4M** presented in this chapter, I−**M4M**, is a faster and more versatile MAMO solver with a rich action space. We discuss the impact of the improvements we make in an extensive simulation study and show previously unachievable results on a real-world PR2 robot.

6.1 Introduction

"Manipulation Among Movable Objects" (MAMO) [5, 144] defines a class of *hybrid* or *multi-modal* planning problems [53, 138]. The difficulty in solving multi-modal planning problems arises from the need to jointly optimise over discrete decisions (which objects to rearrange) and continuous trajectories (for the robot arm to rearrange objects) in a search space that grows with the number of movable objects in the workspace of the robot. In our work, the configuration space of the robot $\mathcal{X}_\mathcal{R} \subset \mathbb{R}^7$ since we use the 7 degree-of-freedom manipulator on a PR2 robot. The configuration space of each movable object O_i is $\mathcal{X}_{O_i} \equiv SE(3)$ since we keep track of their 3D pose (position and orientation). The search space for MAMO solutions in a workspace with n movable objects is the cross-product space $\mathcal{X} = \mathcal{X}_\mathcal{R} \times \mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$. As an example, Figure 6.1 shows a MAMO problem a robot may be asked to solve. The task can be stated very simply – retrieve the can of beer (object-of-interest or OoI, outlined in yellow) from the refrigerator shelf. The complexity in solving this problem arises from the fact that the robot is only allowed to move the yogurt and almond beverage (movable objects outlined in blue). Furthermore, the solution must not violate object-centric "interaction constraints" specified in the problem. These may encode properties desired in the MAMO solution such as whether the robot is allowed to make contact with certain objects, how far it can tilt them, whether they can topple, and how fast they can be moved at all times along the solution trajectory. In the problems we consider, the robot cannot make movable objects tilt beyond 25° along any axis, topple, or make contact with the (fragile) immovable obstacles outlined in red in Figure 6.1 (eggs, cup of coffee, and glass bottles).

There are several approaches to make it computationally tractable to solve such problems. *Task and motion planning* (TAMP) [31, 47, 65] decomposes the MAMO problem to first compute a sequence of abstract, high-level, symbolic actions that solve the problem (without taking into account any kinematic or dynamic constraints of the problem). Using this "task plan" as guidance, they try and solve for a sequence of robot trajectories in the continuous space that does take all necessary constraints of the problem. Other solvers compute solution trajectories in the joint space $\mathcal{X}$ either by sampling random control sequences and validating them by taking into account all problem constraints [8, 69, 160] or using a discretised robot action space

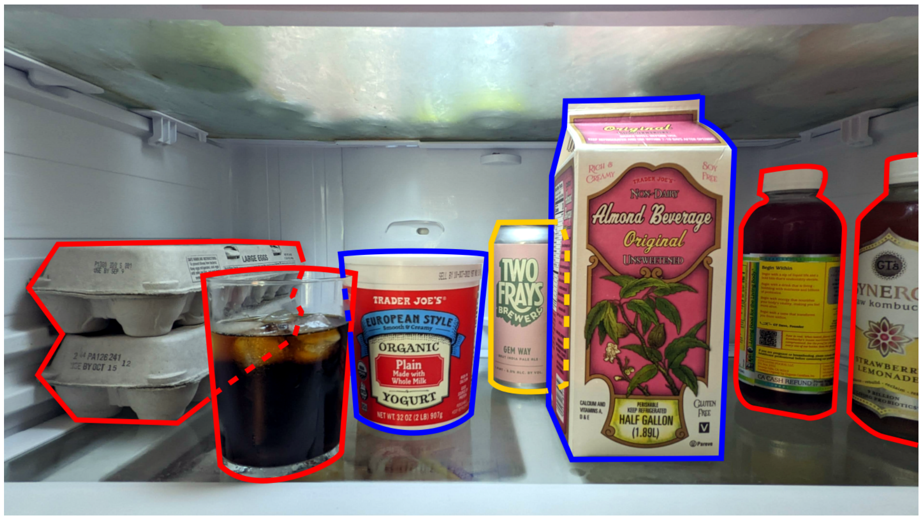


Figure 6.1: The goal is to retrieve the beer can (yellow outline). The yogurt and almond beverage are movable objects (blue outlines). All other objects in the scene are immovable obstacles (red outline).

in a graph search [35, 144].

These methods however usually simplify the MAMO problem further to make it easier to find solutions. In some cases the robot is allowed to only rearrange the scene via prehensile or pick-and-place actions [79, 80, 86, 103, 135, 164]. This simplifies motion planning as we can rely on collision-free motion planning to find prehensile rearrangement trajectories by assuming that once an object is grasped it becomes rigidly attached to the robot kinematic chain. Other approaches allow non-prehensile rearrangement actions such as pushing, but only consider non-prehensile interactions in a planar world where objects do not tilt, topple, lean etc. [35, 59, 60, 68, 111, 159, 161]. Often MAMO solvers will make additional assumptions about access to intermediate or "buffer" locations where objects may be rearranged [86, 163], or that the robot can only interact with and rearrange one object at a time and that an object may not be rearranged more than once [35, 144, 160].

In prior work [127, 128] we have developed the E−**M4M** algorithm – a solver for MAMO problems that draws a connection between the MAMO domain and Multi-Agent Pathfinding (MAPF). E−**M4M** first solves an MAPF abstraction of the MAMO problem to determine which rearrangement actions it should generate as candidates from a particular state. These are then converted into robot trajectories for validation in a physics-based simulator. Unlike TAMP solvers, E−**M4M** does not need access to a prescribed set of parameterised abstract/symbolic actions. Instead it generates these on-the-fly by solving the MAPF problem. E−**M4M** formulates a graph search for MAMO problems and searches over different rearrangements of the scene, different orderings of rearrangements for movable objects in a scene, and different ways to rearrange a movable object. However E−**M4M** is limited to the use of non-prehensile or pushing actions to rearrange movable objects, uses a learned heuristic function that does not capture the complexity of difficult-to-solve problem instances[1], and assumes accurate estimates of movable object masses and coefficients of friction in order to minimise the *sim-to-real* gap [18, 62].

In this chapter we make several changes to E−**M4M** and present an **I**mproved version of the algorithm, I−**M4M**. Each of the modifications are intended to create a more versatile and/or efficient solver for MAMO problems. To be precise, we make the following contributions with the I−**M4M** algorithm:

1. We introduce the use of prehensile rearrangement actions to allow the robot to deliberately pick up objects and place them down elsewhere. This makes significant progress towards the MAMO solution that may otherwise have required multiple pushes.
2. We implement I−**M4M** as a depth-first search with respect to valid rearrangement actions found. In doing so we are able to address inaccuracies in the learned heuristic function used by E−**M4M** that would lead to re-expansions of states from which the algorithm had already made progress.
3. With the inclusion of prehensile rearrangements I−**M4M** uses a richer action space that increases the branching factor of the search and hence the computational complexity. We propose an "eagerly lazy" version of I−**M4M** that only validates rearrangement actions when necessary.
4. We parallelise simulations across multiple instances of the physics-based simulator to relax the assumption of accurate estimates of object masses and coefficients of friction that make I−**M4M** more robust to modeling errors in the real-world.

[1]Problems that are difficult to solve typically require the robot to rearrange many objects, each with multiple actions.

Section 6.2 of this chapter contains a review of work related to MAMO. Section 6.3 contains technical details of the major improvements we incorporate in the I-**M4M** algorithm. These are thoroughly evaluated in simulation and on real-world runs on the PR2. The results of these experiments are presented in Section 6.4. Section 6.5 provides a discussion about the work in this chapter and how it can be extended in future work.

6.2 Related Work

There are two broad categories of MAMO solvers that can be differentiated by whether they use a discretised action space or whether they search for solutions in the joint configuration space of the robot and all movable objects.

Task and motion planning (TAMP) solvers [20, 31, 46, 47, 65, 67] use a discrete set of parameterised symbolic actions to compute "task plans" – abstract, high-level action sequences that solve the MAMO problem but need to be refined into continuous trajectories executable by a robot. They differ in whether a complete task plan to the goal state is refined or whether each abstract action the algorithm considers adding to the task plan is verified by a motion planner when generated. In our work, we present an algorithm that implicitly generates these high-level actions by solving an appropriately constructed MAPF abstraction to MAMO and does not rely on a predefined set of symbolic actions. Other MAMO solvers and algorithms for *rearrangement planning* keep track of object configurations as part of the search graph they construct with a discrete set of rearrangement actions. To avoid an exhaustive search over the entire configuration space $\mathcal{X}$, these algorithms restrict themselves to the use of prehensile or pick-and-place style rearrangement actions that change the configuration of at most one object at a time [79, 80, 86, 103, 135, 144, 159, 163, 164]. Pick-and-place rearrangement actions can be computed with collision-free motion planning where the robot grasps an object, rigidly attaching the object to its kinematic chain, and relocates it to the desired location. This does not require querying a computationally expensive model (such as a physics-based simulator in our case) to forward simulate the effect of robot actions on the configurations of all other movable objects, something typical of algorithms that use non-prehensile robot-object interactions.

Since non-prehensile robot-object interactions can change the configuration of several movable objects during the same action, usually a physics-based simulator is integrated into the motion planning algorithm. The simulator is used to forward simulate the effect of robot actions on the configuration of objects in the environment. Since querying such a simulator is computationally expensive, existing literature limits non-prehensile interactions to a planar world [59, 60, 68, 69, 111, 161]. This makes it easier to find valid pushing actions as the only interaction constraint to be satisfied is ensuring objects remain within workspace bounds. Our work does not make the planar world assumption and keeps track of object configurations in $SE(3)$ (3D position and orientation) and satisfies a more challenging but realistic set of interaction constraints – neither the robot nor *movable objects* can make contact with *immovable obstacles*; movable objects cannot be tilted beyond 25° along any axis, they cannot be toppled or leave workspace bounds, and they cannot be moved faster than $1\,\mathrm{m\,s^{-1}}$. There is existing work that allows the robot to manipulate objects in the workspace via both prehensile and non-prehensile rearrangement actions [8, 35, 45]. However they also limit non-prehensile interactions to a planar world and only allow the robot to manipulate a single object at a time. We do not impose these restrictions in our work, nor do we limit each object to be rearranged once.

The second category of algorithms that search the joint configuration space of the robot and all objects are able to rely on sampling algorithms to generate candidate robot actions [8, 10, 44, 69, 72, 150, 151, 160]. They are aided by additional guidance or biased sampling that provides heuristic guidance to the algorithm about which (if any) object should be manipulated and how. Additionally they may rely on constrained projections of robot motions to ensure that the robot maintains appropriate contacts when evaluating actions.

6.3 Improvements to the E–M4M Algorithm

This chapter introduces several improvements to the E–M4M that contribute to a faster MAMO solver with a richer action space. We call this improved version of the algorithm Improved-M4M or I–M4M. This section discusses four improvements we introduce in this chapter.

6.3.1 Addition of Prehensile Rearrangement Actions

Given the structure of the E-**M4M** graph search, it is straightforward to extend it to obtain a richer MAMO solver with a diverse action space.In this chapter we introduce the use of pick-and-place rearrangement actions as part of the I-**M4M** algorithm. For objects that are "graspable", i.e. those that will fit inside the end-effector of our robot, we generate two outgoing edges from vertex v to vertices v'_1 and v'_2 in I-**M4M** corresponding to rearranging an object by a push and pick-and-place action respectively. The edge (v, v'_1) corresponding to a push action is evaluated as described in Section 5.4.3. To validate edges (v, v'_2) corresponding to pick-and-place rearrangement, we first compute a pick-and-place trajectory, and if one is found, validate it in the simulator.

We assume access to known grasp poses for each object. While the MAPF solution contains entire paths for objects to be rearranged as shown in Figure 5.2, if we want to rearrange an object with a pick-and-place action we select the final state in its MAPF solution path as the placement pose and discard the rest. The rearrangement action can be broken down into four parts – a trajectory to reach the grasp pose, the grasp maneuver, a trajectory to reach the placement pose, and the placement action. Grasping and placement actions are hard-coded movements of the end-effector relative to the object from the grasp pose and placement pose respectively. The trajectories to the grasp and placement poses are computed in $\mathcal{X}_\mathcal{R}$ by calling a motion planner for the robot arm. For prehensile rearrangements, we impose the restriction that the entire trajectory must be collision-free with *all* objects (movable and immovable). If such a collision-free trajectory is found, we simulate the grasping and placement actions in the simulator to ensure that no interaction constraints are violated. We do not need to simulate any other parts of the trajectory since by construction they are collision-free. Figure 6.2 shows a MAMO solution found by I-**M4M** with one non-prehensile and one prehensile rearrangement.

6.3.2 Depth-First I-M4M

E-**M4M** was proposed as a prioritised best-first search on graph G described in Section 5 where the priority function was learned offline to predict the likelihood a particular state lay on a path to the goal state for the MAMO problem. This

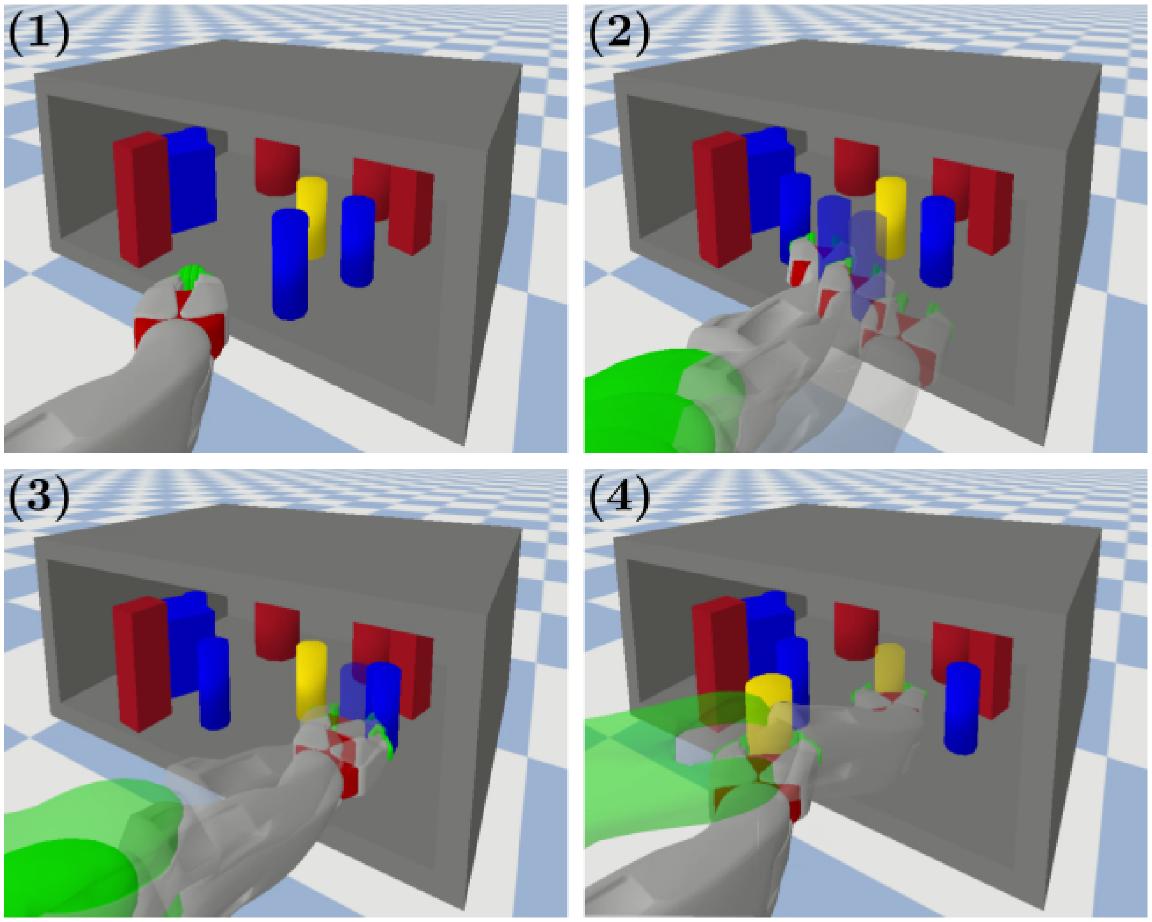


Figure 6.2: (1) The initial scene; (2) Pushing the movable cylinder in the front to the left; (3) Prehensile rearrangement of the movable cylinder in the back; (4) Retrieving the OoI from the scene.

was shown to be particularly ineffective in disambiguating between, i.e. accurately prioritising, scenes in which several objects overlap with the negative goal region. This leads to `E`−**M4M** (re-)expanding vertices in the graph from which it is difficult to make further progress.

Instead `I`−**M4M** is run as a depth-first search to continue the search for solutions from states which we have reached via valid rearrangement actions. This is closely related to the work of Garrett et al. [45] which proposes a hill-climbing solver for TAMP problems, and that of Stilman et al. [144] and Dogar et al. [35] who propose a depth-first solver for MAMO problems that searches backwards from the goal state. However all of these works are restricted to manipulating a single object at a time, and limit all non-prehensile interactions to a plane so that objects do not tilt, topple or lean. We still allow `I-`**M4M** to re-expand states if it is unsuccessful in validating rearrangement actions, and it can backtrack to other vertices in the graph as and when necessary.

`I-`**M4M** orders the priority queue lexicographically based on the pair (`actions`, `reexpands`) where `actions` is the number of valid rearrangement actions found on the partial path to a vertex and `reexpands` is the number of times that vertex has been re-expanded. We preferentially prioritise greater values of `actions` and lesser values of `reexpands`. Thus if $u \prec v$ implies that u appears in the priority queue *ahead* of v, and consequently u will be expanded *before* v, `I-`**M4M** will contain the following ordering of states: $(3,0) \prec (3,1) \prec (2,4) \prec (1,0) \prec (0,2)$.

6.3.3 Eagerly Lazy Evaluation of Rearrangement Actions

The virtue of lazy search algorithms is well-understood [49, 96, 102]. *Laziness* in the property where the evaluation of an edge is postponed until it becomes necessary. For MAMO problems edge-evaluations require computing a robot arm trajectory in $\mathcal{X}_\mathcal{R}$ and in some cases simulating it in a physics-based simulator if such a trajectory is found. This is a particularly time-consuming operation executed for every rearrangement action we evaluate during the search (taking on average $3-4$s per action). Lazy `I-`**M4M** can thus save considerable computational effort by evaluating actions as and when required.

Conventional lazy search algorithms assume that during the search the existence of an edge (v, v'), and by extension that of the successor state v', cannot be refuted and only its validity is determined upon evaluation. However in our domain the edge is generated as an abstract action based on the MAPF solution that when evaluated will almost surely achieve a successor state $v'' \neq v'$ if valid. This leads to a slightly modified implementation of the Lazy Weighted A* (LwA*) algorithm from [25] that we propose in this chapter.

Until an edge (v, v') has been evaluated, we say that v' is an "unevaluated" state and afterwards it is a "fully evaluated" state. LwA* maintains duplicates of all states in the priority queue with different parents. When unevaluated states are expanded, LwA* evaluates the edge from its best parent. When fully evaluated states are expanded[2] LwA* grows the graph by adding unevaluated successor states. To deal with the fact that evaluating an edge to an unevaluated state can lead to a different fully evaluated state in Lazy I-M4M, our priority queue is allowed to contain duplicates of unevaluated states with different parents but not of fully evaluated states. This is because we cannot discard any unevaluated state on the basis of a different edge to it having been evaluated before and we need only keep track of the best parent/path to fully evaluated states. To implement this we store fully evaluated states in a hash table, and we do not hash unevaluated states.

6.3.4 Parallelised Simulations for Robustness to Parameter Uncertainty

E-M4M assumes that prior to planning we can perfectly localise all objects in the scene and we know their respective masses and coefficients of friction. This is difficult to satisfy in the real-world even with accurate localisation algorithms [155, 167] since masses can change over time and cannot be perceived by vision sensors, while coefficients of friction are difficult to compute accurately and change with the surface an object is placed on. If we do not know one or both of these parameters accurately, even with the use of a physics-based simulator, we cannot accurately simulate the result of a pushing action. Figure 6.3 (a) shows a graphical 2D illustration of the potential results of pushing an object whose mass and coefficient of friction are not known accurately. At the end of the push, it may end up in a different configuration depending on the value of these parameters.

Fortunately, our use of a physics-based simulator for action evaluation makes it easy to make E-M4M (or any modified version of it) robust to these uncertain parameters. Taking inspiration from existing work [2, 64**?**], we utilise parallelised simulations of the same trajectory in scenes where each object is instantiated with different values of their mass and coefficient of friction. We assume known, bounded uncertainty on each of these parameters and given some budget on the number of parallelised simulators we are allowed to launch, we instantiate multiple copies of each object within each simulator. For each object copy, we sample its mass and coefficient of friction from within the known bounded uncertainty we assumed for

[2]Unevaluated copies of this state can be ignored/discarded in the future.

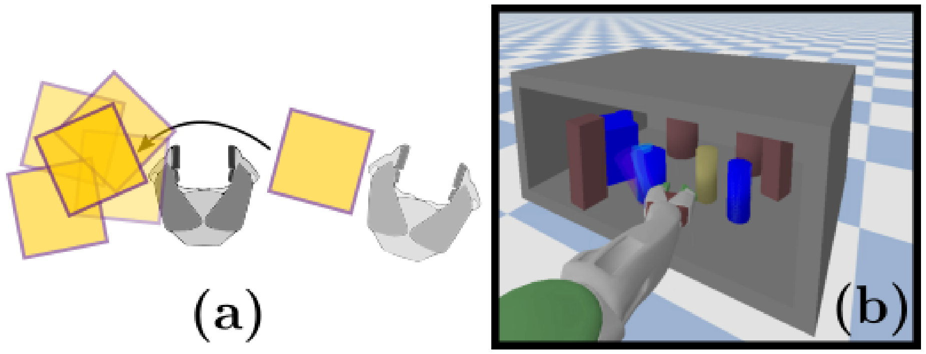


Figure 6.3: (a) If the robot end-effect pushes an object whose mass and coefficient of friction is not known precisely, it may end up in one of many different possible final poses. (b) A screenshot from our simulator where multiple copies of an object with different parameters are being pushed. Irrelevant colours have been desaturated for ease of viewing. Different object copies can be seen in blue (opaque), and cyan and magenta (partially transparent)

that parameter. If the true mass and coefficient of friction for an object are m and μ respectively, for each object copy we sample:

$$\hat{m} = \min\left(2m, \max\left(\frac{m}{2}, \mathcal{N}(m, 0.2m)\right)\right)$$
$$\hat{\mu} = \min\left(2\mu, \max\left(\frac{\mu}{2}, \mathcal{N}(\mu, 0.5\mu)\right)\right)$$

. $\mathcal{N}(x, y)$ represents a sample from a 1D Gaussian distribution with mean x and standard deviation y.

We ensure that contacts and collisions within the simulator are computed appropriately – object copies do not collide with each other, but they do collide with one copy each of all other objects. An action is said to be valid or not based on some user-defined threshold $\delta \in (0, 1]$ for success. Given N parallel simulators and M copies of each object, we say that an action is valid if $\max(1, \lfloor \delta NM \rfloor)$ samples are valid in simulation. Figure 6.3 (b) shows a screenshot of a push being evaluated in our physics-based simulator with multiple object copies. Each copy of the objects being moved by this push can be seen behaving differently than the others with two out of three copies on the verge of interaction constraint violation by toppling.

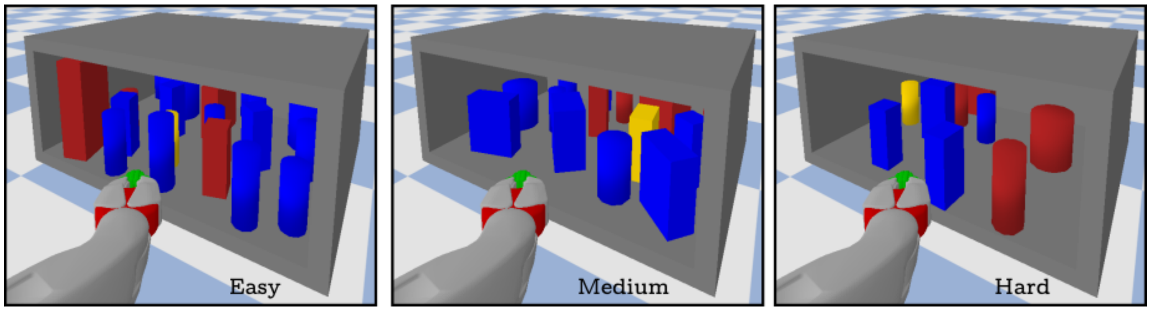


Figure 6.4: Example `Easy`, `Medium`, and `Hard` scenes.

6.4 Experimental Results

6.4.1 Simulation Study

We ran all algorithms on 75 randomly generated MAMO problems categorised into three difficulty levels. Problems are `Easy`, `Medium`, or `Hard` depending on whether there are one, two, or more than two movable objects overlapping with the initial NGR. Each problem has 1 OoI, 4 immovable obstacles, and 5, 10, or 15 movable objects. Objects are randomly generated as cuboids or cylinders and their dimensions, mass, coefficient of friction, and initial configuration are all randomly initialised. Figure 6.4 shows sample MAMO workspaces of each difficulty level. We test the performance of all algorithms on 25 `Easy`, 25 `Medium`, and 25 `Hard` problems with a 5 min planning timeout and use PyBullet [27] as our physics-based simulator.

Tables 6.1 and 6.2 contain the number of problems solved by all algorithms, and for solved problems only the minimum/median/maximum total planning time and time spent simulating actions. The first experiment (Table 6.1) compares performance of `E`–M4M against: (i) `I`–M4M – depth-first search with pushing actions only, (ii) `I`–M4M-PnP – `I`–M4M with pick-and-place actions, (iii) Lazy `I`–M4M – eagerly lazy version with pushes only, and (iv) Lazy `I`–M4M-PnP – eagerly lazy version with pick-and-place actions added. The second experiment (Table 6.2) compares the performance of Robust `I`–M4M that uses parallelised simulations against $\widehat{\texttt{I}\text{–M4M}}$ – a version of `I`–M4M that instantiates a single copy of each object with mass and coefficient of friction sampled as described in Section 6.3.4. For this experiment a problem is successfully solved if the solution trajectory found by the planner does not violate any interaction constraints when executed in simulation with the true parameters for all objects. We only used pushing actions for this experiment since they are acutely affected by parameter uncertainty. Robust I–M4M used $N = 6$ parallel simulators each with $M = 2$ object copies and with $\delta = 0.9$ an action must be valid for 10 samples to be added to the graph.

All I–M4M versions solve more problems than E–M4M. Basic I–M4M solves problems faster and spends less time in simulation than E–M4M. This is because the planner tries to exploit the "goal-directed" rearrangement actions suggested by our MAPF solver that make deliberate progress towards clearing the NGR and solving the MAMO problem. The addition of prehensile rearrangement actions in I–M4M-PnP increases the branching factor of the search which increases planning and simulation times when compared against E–M4M. The major difference between the solutions found by I–M4M and I–M4M-PnP is observed in the number of rearrangement actions in their solution. I–M4M solves problems with 1.8 ± 0.8 (Easy), 2.9 ± 1.3 (Medium), and 6.2 ± 3.0 (Hard) pushes while I–M4M-PnP requires 1.7 ± 1.1 (Easy), 2.4 ± 0.9 (Medium), and 4.5 ± 1.8 (Hard) rearrangement actions in comparison. This is because if we find a valid pick-and-place rearrangement action, given the strict conditions required of it (Section 6.3.1), we exactly achieve the desired configuration of a movable object as suggested by our MAPF solver. This results in much greater progress towards the goal of solving the MAMO problem in comparison to a push action which is subject to the complex multi-body contact dynamics that are not modeled by the MAPF solver and can lead to a successor state much different than the one the MAPF solver wanted to achieve. As expected, Lazy I–M4M and Lazy I–M4M-PnP outperform their non-lazy versions in all metrics (the statistics for the length of the solution found remain similar in comparison). Laziness is particularly effective in combating the computational overhead of a greater branching factor caused by adding prehensile actions as the median planning time is almost halved between I–M4M-PnP and Lazy I–M4M-PnP.

Table 6.1: Quantitative Comparison between E-**M4M** and several I-**M4M** variants

Metrics	**Difficulty**	**Planning Algorithms**				
		E-M4M	I-M4M	I-M4M-PnP	Lazy I-M4M	Lazy I-M4M-PnP
Solved Problems	**Easy**	22	23	23	25	24
	Medium	20	24	22	25	23
	Hard	17	22	21	23	22
Total Planning Time (s)	**Easy**	0.5 / 18.6 / 79.3	0.5 / 12.1 / 46.3	15.8 / 27.3 / 279.0	0.4 / 9.9 / 53.6	6.2 / 11.6 / 266.7
	Medium	0.5 / 41.4 / 195.7	0.4 / 21.1 / 84.4	0.6 / 47.2 / 221.0	0.5 / 20.8 / 179.9	0.4 / 24.7 / 132.1
	Hard	9.8 / 55.3 / 188.5	13.8 / 35.6 / 212.0	9.6 / 61.2 / 206.2	9.3 / 29.7 / 144.6	12.0 / 40.9 / 187.7
Simulation Time (s)	**Easy**	0 / 6.4 / 58.0	0 / 5.9 / 28.0	0 / 15.8 / 196.5	0 / 5.9 / 40.5	0 / 6.5 / 119.7
	Medium	0 / 23.5 / 82.8	0 / 13.4 / 46.7	0 / 30.1 / 71.6	0 / 10.6 / 70.8	0 / 11.2 / 58.3
	Hard	11.3 / 34.9 / 154.9	6.8 / 25.8 / 139.9	5.2 / 36.6 / 120.4	12.5 / 21.5 / 98.5	5.7 / 23.1 / 76.1

Table 6.2: Experiment Quantifying the Effect of Being Robust to Physics Parameter Uncertainty

Metrics	Difficulty	Planning Algorithms	
		Robust I−M4M	$\widehat{\text{I−M4M}}$
Solved Problems	Easy	23	18
	Medium	23	17
	Hard	22	11
Total Planning Time (s)	Easy	8.1 / 11.3 / 284.9	7.4 / 14.3 / 177.1
	Medium	1.1 / 60.3 / 216.7	1.9 / 54.3 / 128.8
	Hard	19.2 / 83.6 / 295.9	22.9 / 104.7 / 257.4
Simulation Time (s)	Easy	0 / 6.2 / 253.8	0 / 5.4 / 77.3
	Medium	0 / 29.8 / 127.2	0 / 20.8 / 64.7
	Hard	10.5 / 51.5 / 200.5	8.6 / 41.9 / 138.1

The timing statistics for Robust I−M4M and $\widehat{\text{I−M4M}}$ are comparable. However Robust I−M4M solves many more problems than $\widehat{\text{I−M4M}}$., i.e. the solution found by Robust I−M4M does not violate any interaction constraint when executed in simulation with the true object masses and coefficients of friction. Robust I−M4M ensures that each action is considered valid only if it is valid for 90% of the samples which makes the solution more likely to be valid for the true parameters as opposed to $\widehat{\text{I−M4M}}$ which hopes to get lucky during execution because the solution found by the planner was only valid for one sampled value of all parameters.

6.4.2 Real-World Experiments

We solved 10 MAMO problems in the real-world with our PR2 robot using I−M4M-PnP. Figure 6.5 shows the robot executing a solution where the PR2 first pushes the coffee can aside, then moves the tomato soup can via a prehensile action, before finally retrieving the OoI potted meat can. We used objects from the YCB dataset [16] for our experiments and they were localised using PERCH 2.0 [1]. We initialised each MAMO problem with either the tomato soup can or the potted meat can as the OoI and $2-4$ other movable objects. Although I−M4M-PnP found solutions to all problems, 3 out of 10 executions failed. In two of the runs, the sim-to-real gap led to

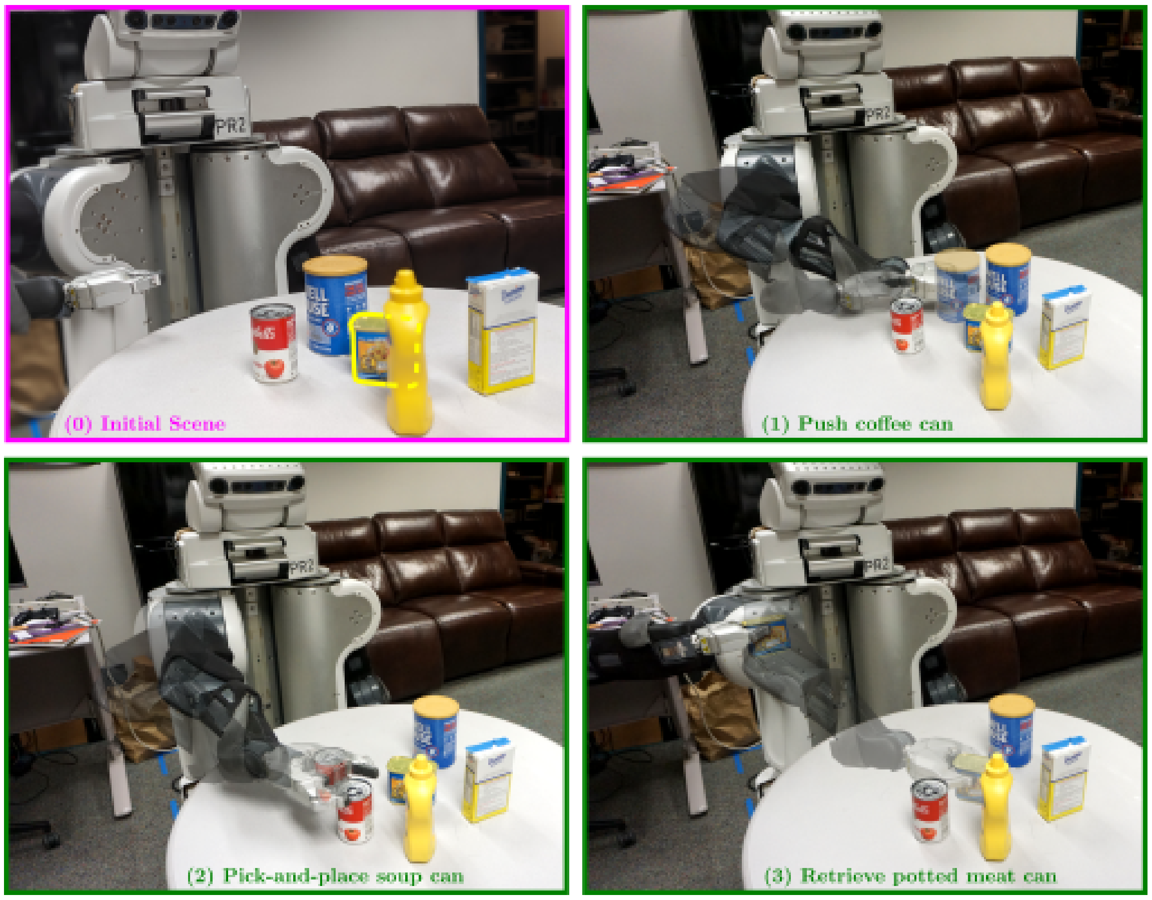


Figure 6.5: A real-world MAMO problem being solved by the PR2 using I-M4M-PnP. The potted meat can is the OoI (outlined in yellow), all other objects are movable.

discrepancies between the result of a pushing action. In one case an object toppled over when it did not in simulation, and another time an object was pushed into the OoI which is an interaction constraint violation that did not occur during planning. The third failure was due to the robot failing to grasp an object during a prehensile rearrangement action. This could be due to either small object localisation errors or due to trajectory execution errors.

6.5 Conclusion and Discussion

This chapter extends [127] and presents I-M4M, a fast and versatile MAMO solver – it utilises a diverse action space consisting of both non-prehensile and prehensile rearrangement actions, can be run lazily, and can leverage parallelised simulations to make it robust to inaccurate estimates of physics parameters of objects – for solving manipulating tasks among movable objects in cluttered and constrained 3D workspaces where these objects tilt, topple, and lean on each other.

`I-`**M4M** spends the majority of its time simulating actions since it is difficult to satisfy all interaction constraints on contact, tilting, toppling etc. in our domain. To improve performance, we would like to incorporate learned rearrangement skills that are aware of the physical properties of all objects in the scene. We are also interested in extending `I-`**M4M** to deal with occlusions in the scene while maintaining the property that the solution will satisfy all interaction constraints at all times.

Chapter 7

Discussion and Future Work

This chapter concludes the work in this thesis by discussing some directions for future work to develop even more capable MAMO solvers that we can deploy on robots in the real-world.

7.1 Discussion

7.1.1 Theoretical Analysis on Completeness of Planning Algorithms

A motion planning algorithm is considered *complete* if it is guaranteed to find a solution when one exists, and return failure in finite time otherwise [48]. In the case of discrete search algorithms like the ones we propose as part of this thesis, we typically want algorithms to be *resolution complete*, i.e. they will find a solution, if one exists, given a particular discretisation scheme [21]. In this section we hope to provide a discussion about the theoretical properties of algorithms we propose in this thesis. We discuss why or why not various algorithms are resolution complete, under what conditions we can obtain a theoretically complete version of the algorithm, and the computational expense associated with this.

The first two planning algorithms in this thesis – SPAMP from Chapter 3 and **M4M** from Chapter 4 – deliberately ignore or prune away states that may lead to solutions. SPAMP never considers finding a solution that includes an action that changes the configuration of movable objects until within a δ-sphere of the goal configuration. Even though SPAMP will search over all solution trajectories that satisfy this property and return one as the solution in finite time (as long as all interaction constraints are satisfied), by construction it does not search over *all* solution trajectories for the problem which include those that do not satisfy the property. Thus, SPAMP is *incomplete* for the MAMO problems it tries to solve. Similarly, even though **M4M** relaxes the assumption made by SPAMP and allows a solution trajectory to change the configuration of movable objects at any point, it is greedy with respect to valid rearrangement actions and therefore prunes away part of the search space with every valid rearrangement action found. With every valid rearrangement action found by **M4M**, the algorithm commits to finding a solution trajectory that necessarily includes (begins with) the partial path up to the resultant state of that action. A simple hypothetical example can be constructed to illustrate the incompleteness of **M4M**. Consider a problem where a robot *must* first rearrange an object B before an object A in order to retrieve the object-of-interest from the workspace. If **M4M** finds *any* action that rearranges A before it finds one to rearrange B, it will fail to solve the problem.

Since the I−**M4M** algorithm from Chapter 6 includes improvements to E−**M4M** from Chapter 5 without fundamentally changing the algorithm, we will present a theoretical analysis for E−**M4M** in this section, with the same results being applicable for I−**M4M**. E−**M4M** further generalises **M4M** by systematically constructing a discrete graph over the MAMO search space $\mathcal{X} = \mathcal{X}_{\mathcal{R}} \times \mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$ whose edges correspond to one of a discrete set of rearrangement actions made available to the robot. However, the rearrangement actions used by E−**M4M** are directly informed by the negative goal region (NGR) it computes at the outset of planning (Section 5.4.1), and this region is never re-computed. If in case E−**M4M** cannot find a solution with the first NGR it computes, it will fail to find any solution altogether. Although this failure mode will only be detected after all permutations of all rearrangement actions for a given NGR are evaluated by E−**M4M**, it can be averted by then allowing E−**M4M** to compute a different NGR from a different robot trajectory. In this way, provided the MAPF solver used by E−**M4M** and the non-prehensile push planner are themselves complete, we can obtain a theoretically resolution complete version of E−**M4M** by allowing it to search over all possible robot trajectories in $\mathcal{X}_{\mathcal{R}}$ that avoid collisions with immovable obstacles and each generate a different NGR.

The MAPF solver used by E-**M4M**, Conflict-Based Search (CBS) [134], is provably resolution complete. The non-prehensile push planner used by E-**M4M**, described in Section 5.4.3, queries a complete motion planner [3] to find a trajectory to the start pose for a push. It then samples waypoints to compute the pushing action with inverse kinematics. A deterministic version of this planner, without any sampling, is complete with respect to the discrete graph it searches for pushing trajectories[1] since the two pieces (a motion planner to find a trajectory to the start pose and an inverse kinematics solver) are individually complete. In this thesis we included stochastic sampling for waypoints to generate a variety of different pushes which in turn cause a variety of robot-object interactions. The stochastic push planner can be shown to be *probabilistically* complete if the number of those samples goes to infinity in the limit of infinite time. To summarise, if we allow E-**M4M** to search over all robot trajectories and consequently all NGRs, the version of the algorithm that uses a deterministic push planner can be shown to be resolution complete and the version of the algorithm that uses stochastic sampling within the push planner can be shown to be probabilistically complete. Finally, the only different between E-**M4M** and I-**M4M** is a prehensile or pick-and-place rearrangement planner (Section 6.3.1). This queries a complete motion planner [3] twice (once for the trajectory to the grasp pose, and again for the trajectory to the placement pose) and inverse kinematics twice (once for the grasping action, and again for the placement action). Thus, since all pieces are resolution complete, the prehensile or pick-and-place rearrangement planner is resolution complete.

7.1.2 A Note for Practitioners

We present several algorithms in this thesis for solving MAMO problems of varying complexity. Figure 7.1 shows a graph comparing the contributions in this thesis along two axes – the complexity of the workspace we can solve MAMO tasks in ("Workspace Difficulty") and the size of the search space an algorithm tries to find a solution in ("Search Space"). The SPAMP algorithm from Chapter 3 was designed

[1]Note that even though this push planner does not search over *all* possible ways to push an object to a particular pose, it is *resolution* complete with respect to the graph it constructs.

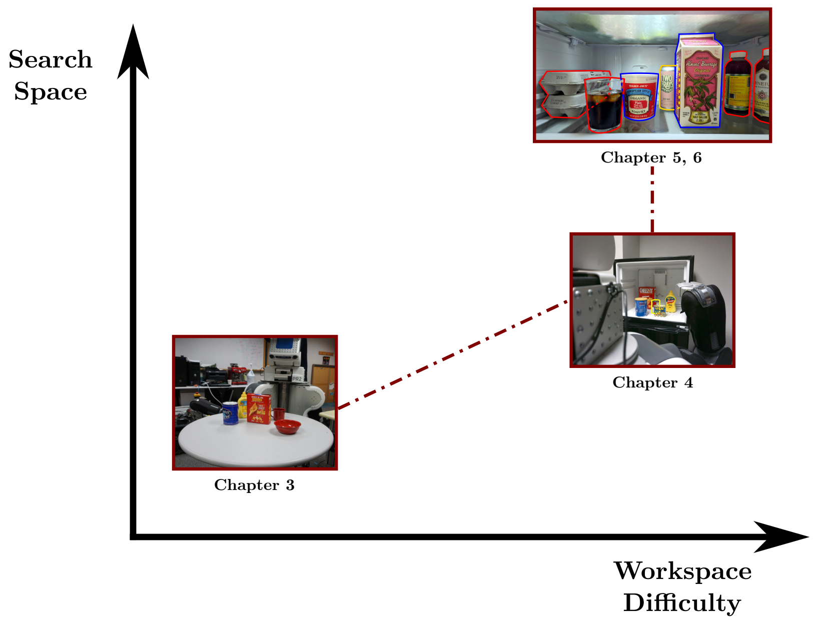


Figure 7.1: A comparison of the work in this thesis based on the difficulty of the workspace they solve problems in and the size of the space they find solutions in.

for "open" workspaces with considerable object-free volume so that we can resort to collision-free motion planning purely in the configuration space of the robot arm $\mathcal{X}_\mathcal{R}$ (without accounting for configurations of movable objects in $\mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$) until we reach a state close to the desired goal/grasp pose. This assumption is easily violated in more cluttered and constrained workspace. Even if it is possible to reach close to the goal without collisions, heavy clutter in a constrained workspace can cause grasping actions to fail, and even if those succeed, we can still violate interaction constraints while trying to retrieve the object-of-interest. **M4M** from Chapter 4 is able to relax this assumption and solve MAMO problems in heavily cluttered and constrained workspaces while keeping track of movable object configurations in $\mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$. However, since **M4M** is greedy with respect to valid rearrangement actions that it finds, it can fail to find solutions in the full MAMO search space $\mathcal{X} = \mathcal{X}_\mathcal{R} \times \mathcal{X}_{O_1} \times \cdots \times \mathcal{X}_{O_n}$. Chapters 5 and 6 present the E-**M4M** and I-**M4M** algorithms that systematically search for solutions in $\mathcal{X}$ by building a graph over different rearrangements of the scene (MAMO states in $\mathcal{X}$) and connecting them via edges that correspond to rearrangement actions. E-**M4M** and I-**M4M** only differ in terms of the edges they add to the graph – the latter uses prehensile or pick-and-place rearrangement actions in addition to the non-prehensile pushes used by the former.

7.1.3 Learning Predictive Models of Push Action Feasibility

While developing the I-**M4M** algorithm, we explored the idea of incorporating a non-uniform cost function within the MAPF solver to bias solutions towards paths that were likely to lead to valid pushing actions. Our hypothesis was that part of the reason E-**M4M** spends the majority of its time inside the push planner (Section 5.6.2) is because it tries to compute pushing trajectories informed by "bad" MAPF solutions that lead to (i) bad initial poses for the push (unreachable due to clutter, outside the reachable workspace, near singular configurations of the robot arm etc.), (ii) joint limit violations and collisions with immovable obstacles while querying the inverse kinematics (IK) solver for a pushing action, and/or (iii) near immediate interaction constraint violations when simulated. If we are able to bias the MAPF solver towards solution paths that are aware of the geometry and physics properties (mass and coefficient of friction) of movable objects, we would be able to alleviate the poor performance of E-**M4M** within the push planner by allowing it to compute pushes less susceptible to the three failure cases we described earlier.

To this end, we collected data from over $100,000$ simulations of our robot arm pushing a single randomly generated object (randomised geometry, mass, and coefficient of friction). Given a randomly sampled desired displacement of the object, we collected data about (i) whether a trajectory to the push start pose exists, (ii) whether the IK calls for the pushing action succeeds, and (iii) whether the push violates interaction constraints when simulated.

Figures 7.2 and 7.3 show two different results from the model we learn to predict the feasibility of pushing actions [109]. Each corresponding sub-figure in Figures 7.2 and 7.3 shows the result for the same randomly generated object (visualised as a blue outline). We are showing a 2D, top-down view of the workspace shelf on which these objects are placed (rotated 90°) – the "front" of the shelf facing the robot is the left edge of each sub-figure, the "back" of the shelf is the right edge, the left wall of the edge is the top edge, and the right wall of the shelf is the bottom edge. Each point within the workspace shelf is coloured according to the output of the learned model.

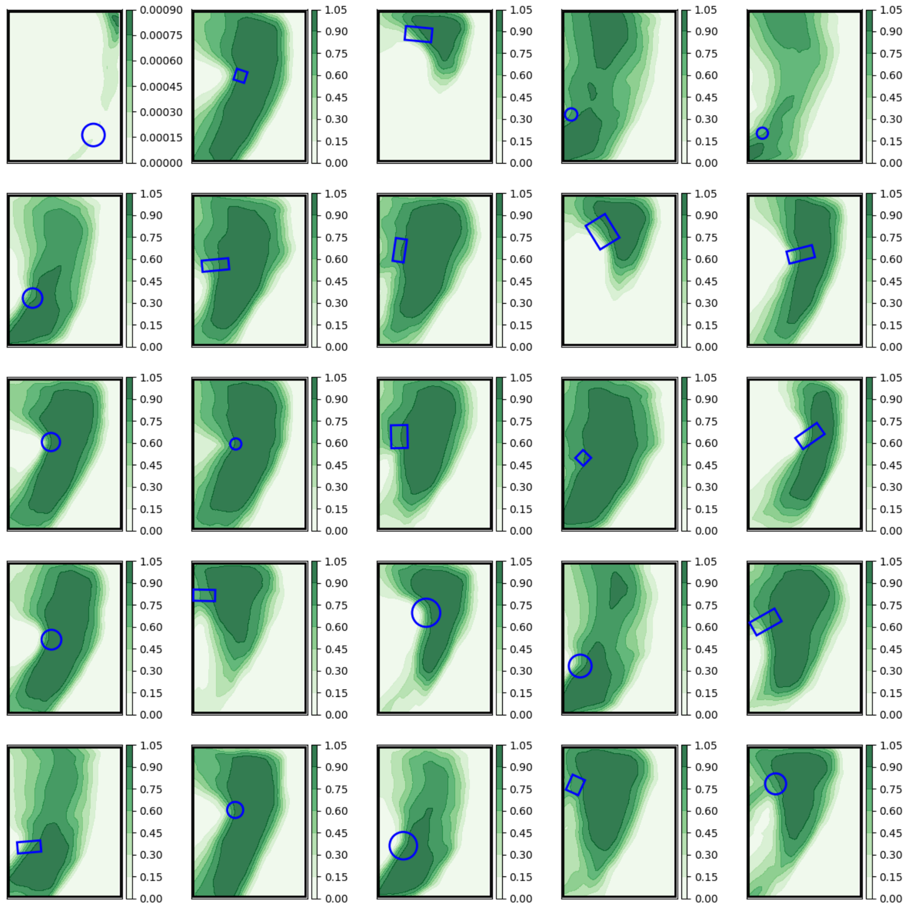


Figure 7.2: Predicted probability of the existence of an inverse kinematics (IK) solution between the current pose of an object (blue outline) and a desired location for it (any point on the workspace shelf, 2D, top-down view shown).

Specifically, the value/colour of a point on the shelf corresponds to the output of the model assuming that we would like to translate the object (no rotation) from its current pose (blue outline) to that point with a pushing action.

Figure 7.2 shows the probability of IK finding a solution along the straight line from the current object location to the desired location. It shows that it is easier for the robot to push objects towards the back-left and front-right of the shelf which is to be expected since we are using the right arm of the PR2 robot. It also shows that it is difficult for the robot to move any object towards itself since that would require reaching a start pose behind the object, something usually kinematically infeasible. Figure 7.3 shows the probability that, when pushed, the object ends at a location

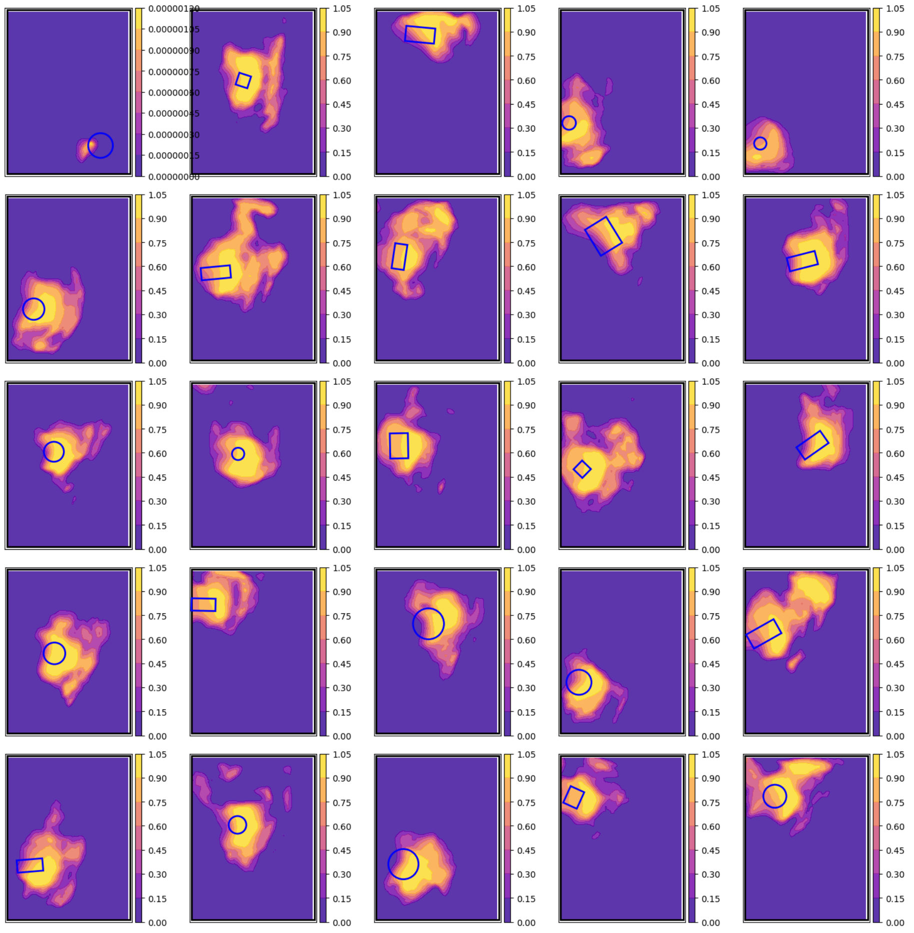


Figure 7.3: Predicted probability of the object (blue outline) achieving a pose within 5 cm of a desired location on the workspace shelf (2D, top-down view shown).

within 5 cm of the desired location on the shelf. The model predicts that in order to achieve a 5 cm tolerance around the desired pose, we should not push objects too far from their current pose and we should try to push cuboids along their longer dimension. The result in Figure 7.3 is explicitly taking into account the output of the model from Figure 7.2 – they are in fact the same neural network models with multiple output heads that predict the probability of IK success and probability of reaching a pose within 5 cm of the desired pose.

We can incorporate the predictions shown in Figure 7.3 as a non-uniform cost function for the MAPF solver in order to bias the solution towards paths that lead to "successful" pushes – those for which we can find an IK solution and have the object reach close to the desired pose. In practice using this non-uniform cost function within the MAPF solver yielded no overall computational benefit in terms of the total time required to solve MAMO planning problems. The only significant difference we observed was an increase in the time spent solving MAPF problems by 10× as more object-object conflicts are encountered by the MAPF solver (median solve time increases from 0.04 s, 0.2 s, and 0.3 s for `Easy`, `Medium`, and `Hard` problems respectively to 0.7 s, 1.6 s, and 3.7 s). In terms of time spent planning pushes, these numbers are 0.5 s, 4.9 s, and 6.8 s without the learned cost function and 0.4 s, 5.6 s, and 7.0 s with the non-uniform cost function for the MAPF solver. In terms of time spent simulating pushes, these numbers are 4.0 s, 16.4 s, and 21.2 s without the learned cost function and 4.4 s, 15.5 s, and 21.8 s with the non-uniform cost function for the MAPF solver. There is a small amount of overhead in querying the learned model (once per MAPF problem solved), but overall it offers no computational benefit regardless of whether it is used as the cost function for the MAPF problem or a heuristic within it. We believe that the reason why we obtain no computational benefit from these models is because they fail to account for the multi-body interactions that make our MAMO problems difficult to solve. Notably, these models have no information whatsoever about any other objects in the scene. Thus they are unable to account for robot-object and object-object interactions that lead to constraint violations such as the forearm or upper arm of the robot hitting an immovable obstacle, or a movable object pushing a different movable object into an immovable obstacle. It is exactly these kinds of multi-body interactions that make it difficult to find valid rearrangement actions for the MAMO problems we solve in this thesis, and by extension make it difficult to find a solution to them. Extending the literature on predicting action feasibility for such MAMO problems or learning heuristics for them is important and difficult. At the same time, if we are able to account for the combinatorics of MAMO problems that grow with the number of objects in the scene, the variety of interactions they can lead to, and the ways in which they can affect the validity of robot actions, we can hope to learn models that benefit our planning algorithms. Advances in the use of *graph neural networks* [61, 89] for manipulation problems is a step towards modeling such complex interactions. These models are capable of capturing the effect of robot actions on inter-object relations, for scenes with varying numbers of objects. It should be possible for us to collect a vast amount of data across MAMO problems to try and predict the feasibility of robot actions for a new MAMO scene, or even to generate a good candidate set of actions to rearrange the current scene.

7.2 Future Work

7.2.1 Learning Heuristics for MAMO

We showed in Section 6.3.2 that the performance of E-**M4M** suffered from an inaccurate heuristic function which caused it to re-expand states from which it was difficult to make progress instead of continuing to grow the graph from states it had reached via valid rearrangement actions. The quality of solutions found by E-**M4M** and I-**M4M** is heavily dependent on the states that they explore during planning. This is governed in part by the heuristic or prioritisation function they use, and in part by the trajectories these algorithms (and **M4M**) generate to refine high-level rearrangements suggested by the MAPF solution. In order to learn a good heuristic function for MAMO planners, we must take into account the configuration of all objects in the scene along with their physical properties. Existing work on learning appropriate high-level abstractions for such problems [74] or predicting the value of states encountered during planning [20] are effective in small state spaces. For MAMO problems however, the dimensionality of the state space changes with the number of objects in the scene which makes it difficult to learn good heuristics. In addition to the number and types of objects in the scene, the heuristic value of a MAMO state can be affected by the physics properties of movable objects (dimensions, masses, coefficients of friction), their proximity to immovable obstacles, and the ability and necessity of the robot to manipulate them.

It is possible to learn predictors of action feasibility and success [109, 145] and to predict the effect of robot actions on the state of objects in the world [139]. Once again these approaches were developed in small state spaces with a single object in the scene. But they offer a promising direction for research on learning similar models for MAMO problems that are capable of predicting the feasibility and probability of success of rearrangement actions given the current scene, and the configuration of objects in the resultant scene. This information can be aggregated into a value function that can be used as heuristic guidance for MAMO planning algorithms to preferentially grow the graph using likely successful actions to like 'valid' (and valuable) states.

7.2.2 Interleaving Planning and Execution

This thesis assumes that prior to planning, we are able to accurately localise all objects in the workspace. It is difficult to satisfy this assumption in the real-world where occlusions make it difficult to even detect the existence of, let alone accurately localise, some objects. It is important for our algorithms to be capable of reasoning about occlusions amongst clutter and be careful when potentially manipulating objects towards unobservable areas of the workspace. We would like to avoid inadvertently violating interaction constraints associated with unseen objects. This can be done by interleaving planning and execution where we always try and manipulate objects into known, visible areas in an attempt to get more sensory information about the scene and update our model of the workspace.

Work on mechanical search [29] and occlusion-aware object retrieval [11, 37, 104] has addressed this issue in the past. For the **M4M** family of algorithms presented in this thesis we can impose restrictions within the MAPF solver on where visible movable objects can move. Unseen areas of the workspace can be optimistically considered occupied by immovable obstacles so that we only try to generate rearrangement actions that manipulate objects into visible areas. To interleave planning and execution, we will also need to specify intermediate goal states that determine when we can terminate planning with a partial path to be executed. This can be done in an 'anytime' fashion [87] where we return the best sequence of rearrangement actions found for the currently visible workspace given a planning time budget. After execution of this partial sequence of rearrangements, we can update our workspace model with new sensed information and continue planning.

7.2.3 Model-Based Object-Centric Rearrangement Actions

The degree of difficulty in solving the MAMO problems we consider in this thesis is dictated by our ability to find valid rearrangement actions (with respect to interaction constraints). Our non-prehensile push planner from Section 4.4.2 in particular geometrically computes a pushing action for each object without taking into account any information about the contact made with the object, forces applied on the object during pushing, or the physical properties of the object (other than its geometry). Incorporating more informed, object-centric rearrangement actions into the planning algorithms developed in this thesis will improve the versatility of the planners and also the likelihood of these actions being valid. We can leverage work on planning rearrangement actions by explicitly modeling the contacts between the robot and object [19, 38] or by using learned policies [168] or by allowing robots to use additional tools [154] to manipulate objects to desired configurations. It is important to keep in mind that the computational performance of the planning algorithms developed in this thesis is limited by how computationally cheap it is to generate and evaluate actions that make up the action space made available to them. When considering the addition of new actions to the action space or replacing existing ones, we must evaluate the trade-off between constructing a more capable and versatile planner, and increasing the computational burden on it to find solutions.

7.3 Conclusion

This thesis adds to the body of work on "Manipulation Planning Among Movable Obstacles" or MAMO by developing algorithms capable of solving challenging, long-horizon tasks in 3D workspaces where objects tilt, topple, lean etc. The work in this thesis highlights the importance of developing algorithms for realistic scenarios robots may be expected to solve and giving them the tools to be able to do so. The algorithms we present in this thesis – SPAMP (Chapter 3), **M4M** (Chapter 4), E–**M4M** (Chapter 5), and I–**M4M** (Chapter 6) – use a physics-based simulator in-the-loop during planning to account for complex interaction constraints to ensure desirable properties in the solution found. We are able to allow objects to tilt and slide (within acceptable limits), lean on each other, and move other objects in turn. Our use of a novel multi-agent pathfinding (MAPF) abstraction to MAMO highlights the importance of decoupling the search over rearrangements of objects in the scene from evaluating actions to achieve those rearrangements. The MAPF abstraction is only one way to generate candidate rearrangement actions for MAMO problems. Regardless, it is capable of easily capturing interdependence between object rearrangements in case the robot is required to rearrange one before being able to successfully rearrange another. Our work leaves open the door for incorporating techniques that have been developed for prehensile, non-prehensile and dexterous robot manipulation under uncertainty in the real-world to long-horizon tasks in 3D workspaces like the MAMO problems we consider with the complex multi-body interactions involved therein.

Bibliography

[1] Aditya Agarwal, Yupeng Han, and Maxim Likhachev. Perch 2.0 : Fast and accurate gpu-based perception via search for object pose estimation. In *IROS*, 2020. 3.5.2, 4.5.2, 5.6.3, 6.4.2

[2] Wisdom C. Agboh and Mehmet R. Dogar. Robust physics-based manipulation by interleaving open and closed-loop execution, 2021. 2.1.4, 6.3.4

[3] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic A*. *Int. J. Robotics Res.*, 35(1-3): 224–243, 2016. doi: 10.1177/0278364915594029. 1.2, 3.4.1, 3.4.2, 4.4, 7.1.1

[4] Y. Aiyama, M. Inaba, and H. Inoue. Pivoting: A new method of graspless manipulation of object by robot fingers. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)*, volume 1, pages 136–143 vol.1, 1993. doi: 10.1109/IROS.1993.583091. 2.1.2

[5] Rachid Alami, Thierry Simeon, and Jean-Paul Laumond. A geometrical approach to planning manipulation tasks. The case of discrete placements and grasps. In Hirofumi Miura, editor, *The fifth international symposium on Robotics research* , pages 453–463. MIT Press, 1990. URL https://hal.archives-ouvertes.fr/hal-01309950. 2.1.1, 4.2, 5.1, 6.1

[6] Fahiem Bacchus and Qiang Yang. The downward refinement property. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, pages 286–293. Morgan Kaufmann, 1991. URL http://ijcai.org/Proceedings/91-1/Papers/045.pdf. 4.2.2

[7] Sangjae Bae, Dhruv Saxena, Alireza Nakhaei, Chiho Choi, Kikuo Fujimura, and Scott J. Moura. Cooperation-aware lane change maneuver in dense traffic based on model predictive control with recurrent neural network. In *2020 American Control Conference, ACC 2020, Denver, CO, USA, July 1-3, 2020*, pages 1209–1216. IEEE, 2020. doi: 10.23919/ACC45564.2020.9147837. URL https://doi.org/10.23919/ACC45564.2020.9147837. 1.2

[8] Jennifer L. Barry, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. A hierarchi-

cal approach to manipulation with diverse actions. In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, pages 1799–1806. IEEE, 2013. doi: 10.1109/ICRA.2013.6630814. URL https://doi.org/10.1109/ICRA.2013.6630814. 6.1, 6.2

[9] Dhruv Batra, Angel X Chang, Sonia Chernova, Andrew J Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, et al. Rearrangement: A challenge for embodied ai. *arXiv preprint arXiv:2011.01975*, 2020. 2.1.3

[10] Servet B. Bayraktar, Andreas Orthey, Zachary K. Kingston, Marc Toussaint, and Lydia E. Kavraki. Solving rearrangement puzzles using path defragmentation in factored state spaces. *IEEE Robotics Autom. Lett.*, 8(8):4529–4536, 2023. doi: 10.1109/LRA.2023.3282788. 6.2

[11] Wissam Bejjani, Wisdom C. Agboh, Mehmet Remzi Dogar, and Matteo Leonetti. Occlusion-aware search for object retrieval in clutter. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021*, pages 4678–4685. IEEE, 2021. doi: 10.1109/IROS51168.2021.9636230. URL https://doi.org/10.1109/IROS51168.2021.9636230. 7.2.2

[12] Ohad Ben-Shahar and Ehud Rivlin. Practical pushing planning for rearrangement tasks. *IEEE Trans. Robotics Autom.*, 14(4):549–565, 1998. doi: 10.1109/70.704220. 4.2

[13] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman, editors, *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 41–48. ACM, 2009. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380. 4.2.2

[14] Ian M. Bullock and Aaron M. Dollar. Classifying human manipulation behavior. In *2011 IEEE International Conference on Rehabilitation Robotics*, pages 1–6, 2011. doi: 10.1109/ICORR.2011.5975408. 1.1

[15] Samuel R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. Technical report, IEEE Journal of Robotics and Automation, 2004. 2.1.1

[16] Berk Çalli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha S. Srinivasa, Pieter Abbeel, and Aaron M. Dollar. Yale-cmu-berkeley dataset for robotic manipulation research. *Int. J. Robotics Res.*, 36(3):261–268, 2017. doi: 10.1177/0278364917700714. 3.5.1, 4.5.2, 5.6.3, 6.4.2

[17] Ishani Chatterjee, Maxim Likhachev, Ashwin Khadke, and Manuela Veloso. Speeding up search-based motion planning via conservative heuristics. In J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava, editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, pages 674–679. AAAI Press, 2019. URL https://aaai.org/ojs/index.php/ICAPS/article/view/3535. 4.2.2

[18] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan D. Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 8973–8979. IEEE, 2019. doi: 10.1109/ICRA.2019.8793789. URL https://doi.org/10.1109/ICRA.2019.8793789. 1.2.4, 6.1

[19] Xianyi Cheng, Eric Huang, Yifan Hou, and Matthew T. Mason. Contact mode guided motion planning for quasidynamic dexterous manipulation in 3d. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*, pages 2730–2736. IEEE, 2022. doi: 10.1109/ICRA46639.2022.9811872. URL https://doi.org/10.1109/ICRA46639.2022.9811872. 7.2.3

[20] Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Edward Groshev, Christopher Lin, and Pieter Abbeel. Guided search for task and motion plans using learned heuristics. In Danica Kragic, Antonio Bicchi, and Alessandro De Luca, editors, *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, pages 447–454. IEEE, 2016. doi: 10.1109/ICRA.2016.7487165. 4.2, 6.2, 7.2.1

[21] Howie Choset, Kevin M Lynch, Seth Hutchinson, George A Kantor, and Wolfram Burgard. *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005. 7.1.1

[22] Sanjiban Choudhury, Siddhartha S. Srinivasa, and Sebastian A. Scherer. Bayesian active edge evaluation on expensive graphs. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 4890–4897. ijcai.org, 2018. doi: 10.24963/ijcai.2018/679. 5.7

[23] Benjamin J. Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for manipulation with motion primitives. In *IEEE International Conference on Robotics and Automation, ICRA 2010*. IEEE, 2010. 3.4.2

[24] Benjamin J. Cohen, Gokul Subramania, Sachin Chitta, and Maxim Likhachev. Planning for manipulation with adaptive motion primitives. In *IEEE International Conference on Robotics and Automation, ICRA 2011*. IEEE, 2011. 3.1.2,

3.3.3

[25] Benjamin J. Cohen, Mike Phillips, and Maxim Likhachev. Planning single-arm manipulations with n-arm robots. In Levi Lelis and Roni Stern, editors, *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel*, pages 226–227. AAAI Press, 2015. URL `http://www.aaai.org/ocs/index.php/SOCS/SOCS15/paper/view/10876`. 5.7, 6.3.3

[26] Nikolaus Correll, Kostas E. Bekris, Dmitry Berenson, Oliver Brock, Albert J. Causo, Kris Hauser, Kei Okada, Alberto Rodriguez, Joseph M. Romano, and Peter R. Wurman. Analysis and observations from the first amazon picking challenge. *IEEE Trans Autom. Sci. Eng.*, 15(1):172–188, 2018. doi: 10.1109/TASE.2016.2600527. URL `https://doi.org/10.1109/TASE.2016.2600527`. 4

[27] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019. 1.1, 3.5, 4.4, 6.4.1

[28] Joseph C. Culberson and Jonathan Schaeffer. Pattern databases. *Comput. Intell.*, 14(3):318–334, 1998. doi: 10.1111/0824-7935.00065. URL `https://doi.org/10.1111/0824-7935.00065`. 4.2.2

[29] Michael Danielczuk, Andrey Kurenkov, Ashwin Balakrishna, Matthew Matl, David Wang, Roberto Martín-Martín, Animesh Garg, Silvio Savarese, and Ken Goldberg. Mechanical search: Multi-step retrieval of a target object occluded by clutter. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 1614–1621. IEEE, 2019. doi: 10.1109/ICRA.2019.8794143. URL `https://doi.org/10.1109/ICRA.2019.8794143`. 7.2.2

[30] Michael Danielczuk, Arsalan Mousavian, Clemens Eppner, and Dieter Fox. Object rearrangement using learned implicit collision functions. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6010–6017, 2021. doi: 10.1109/ICRA48506.2021.9561516. 2.1.3

[31] Neil T. Dantam, Zachary K. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki. Incremental task and motion planning: A constraint-based approach. In David Hsu, Nancy M. Amato, Spring Berman, and Sam Ade Jacobs, editors, *Robotics: Science and Systems XII, University of Michigan, Ann Arbor, Michigan, USA, June 18 - June 22, 2016*, 2016. doi: 10.15607/RSS.2016.XII.002. URL `http://www.roboticsproceedings.org/rss12/p02.html`. 6.1, 6.2

[32] Dror Dayan, Kiril Solovey, Marco Pavone, and Dan Halperin. Near-optimal multi-robot motion planning with finite sampling. In *2021 IEEE International*

Conference on Robotics and Automation (ICRA), pages 9190–9196, 2021. doi: 10.1109/ICRA48506.2021.9561009. 4.2.1

[33] Mehmet Dogar. *Physics-Based Manipulation Planning in Cluttered Human Environments*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, July 2013. 2.1.4

[34] Mehmet Remzi Dogar and Siddhartha S. Srinivasa. A framework for push-grasping in clutter. In Hugh F. Durrant-Whyte, Nicholas Roy, and Pieter Abbeel, editors, *Robotics: Science and Systems VII, University of Southern California, Los Angeles, CA, USA, June 27-30, 2011*, 2011. doi: 10.15607/RSS.2011.VII.009. URL `http://www.roboticsproceedings.org/rss07/p09.html`. 2.1.4

[35] Mehmet Remzi Dogar and Siddhartha S. Srinivasa. A planning framework for non-prehensile manipulation under clutter and uncertainty. *Auton. Robots*, 33 (3):217–236, 2012. doi: 10.1007/s10514-012-9306-z. URL `https://doi.org/10.1007/s10514-012-9306-z`. 1.2.4, 2.1.1, 2.1.4, 3.2, 4.2, 4.4, 4.5.1, 4.1, 5.2, 5.4.1, 5.6.1, 5.7, 6.1, 6.2, 6.3.2

[36] Mehmet Remzi Dogar, Kaijen Hsiao, Matei T. Ciocarlie, and Siddhartha S. Srinivasa. Physics-based grasp planning through clutter. In *Robotics: Science and Systems VIII, 2012*, 2012. 3.2

[37] Mehmet Remzi Dogar, Michael C. Koval, Abhijeet Tallavajhula, and Siddhartha S. Srinivasa. Object search by manipulation. *Auton. Robots*, 36(1-2): 153–167, 2014. doi: 10.1007/s10514-013-9372-x. URL `https://doi.org/10.1007/s10514-013-9372-x`. 7.2.2

[38] Neel Doshi, Orion Taylor, and Alberto Rodriguez. Manipulation of unknown objects via contact configuration regulation. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*, pages 2693–2699. IEEE, 2022. doi: 10.1109/ICRA46639.2022.9811713. URL `https://doi.org/10.1109/ICRA46639.2022.9811713`. 7.2.3

[39] Wei Du, Sung-Kyun Kim, Oren Salzman, and Maxim Likhachev. Escaping local minima in search-based planning using soft duplicate detection. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. IEEE, 2019. 3.3.6

[40] Wei Du, Fahad Islam, and Maxim Likhachev. Multi-Resolution A*. In Daniel Harabor and Mauro Vallati, editors, *Proceedings of the Thirteenth International Symposium on Combinatorial Search, SOCS 2020, Online Conference [Vienna, Austria], 26-28 May 2020*, pages 29–37. AAAI Press, 2020. URL `https://aaai.org/ocs/index.php/SOCS/SOCS20/paper/view/18515`. 1.2

[41] Stefan Edelkamp, Stefan Schroedl, and Sven Koenig. *Heuristic Search: Theory and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA,

2010. ISBN 0123725127. 2.2

[42] Michael A. Erdmann and Tomás Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2:477–521, 1987. doi: 10.1007/BF01840371. URL https://doi.org/10.1007/BF01840371. 4.2.1, 4.5.3

[43] Tom Erez and Emanuel Todorov. Trajectory optimization for domains with contacts using inverse dynamics. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pages 4914–4919. IEEE, 2012. doi: 10.1109/IROS.2012.6386181. URL https://doi.org/10.1109/IROS.2012.6386181. 2.1.5

[44] Caelan Reed Garrett. *Sampling-Based Robot Task and Motion Planning in the Real World*. PhD thesis, Massachusetts Institute of Technology, USA, 2021. 6.2

[45] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Backward-forward search for manipulation planning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, pages 6366–6373. IEEE, 2015. doi: 10.1109/IROS.2015.7354287. URL https://doi.org/10.1109/IROS.2015.7354287. 6.2, 6.3.2

[46] Caelan Reed Garrett, Toms Lozano-Prez, and Leslie Pack Kaelbling. FFRob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1):104–136, 2018. doi: 10.1177/0278364917739114. 6.2

[47] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Toms Lozano-Prez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4 (1):265–293, 2021. doi: 10.1146/annurev-control-091420-084139. URL https://doi.org/10.1146/annurev-control-091420-084139. 6.1, 6.2

[48] Ken Goldberg. Completeness in robot motion planning. In *Workshop on Algorithmic Foundations of Robotics*, pages 419–429. Citeseer, 1994. 7.1.1

[49] Nika Haghtalab, Simon Mackenzie, Ariel D. Procaccia, Oren Salzman, and Siddhartha S. Srinivasa. The provable virtue of laziness in motion planning. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 6161–6165. ijcai.org, 2019. doi: 10.24963/ijcai.2019/855. 6.3.3

[50] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136. URL https://doi.org/10.1109/TSSC.1968.300136. 1.2

[51] Patrik Haslum. Reducing accidental complexity in planning problems. In

Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1898–1903, 2007. URL `http://ijcai.org/Proceedings/07/Papers/306.pdf`. 4.2.2

[52] Kris K. Hauser. The minimum constraint removal problem with three robotics applications. *Int. J. Robotics Res.*, 33(1):5–17, 2014. doi: 10.1177/0278364913507795. URL `https://doi.org/10.1177/0278364913507795`. 5.1

[53] Kris K. Hauser and Jean-Claude Latombe. Multi-modal motion planning in non-expansive spaces. *Int. J. Robotics Res.*, 29(7):897–915, 2010. doi: 10.1177/0278364909352098. 6.1

[54] Joshua A. Haustein, Jennifer King, Siddhartha S. Srinivasa, and Tamim Asfour. Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3075–3082, 2015. doi: 10.1109/ICRA.2015.7139621. 2.1.3

[55] Juan David Hernández, Mark Moll, and Lydia E Kavraki. Lazy evaluation of goal specifications guided by motion planning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 944–950. IEEE, 2019. 3.3.6

[56] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.*, 14:253–302, 2001. doi: 10.1613/jair.855. URL `https://doi.org/10.1613/jair.855`. 4.2.2

[57] François Robert Hogan and Alberto Rodriguez. Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics. In Ken Goldberg, Pieter Abbeel, Kostas E. Bekris, and Lauren Miller, editors, *Algorithmic Foundations of Robotics XII, Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics, WAFR 2016, San Francisco, California, USA, December 18-20, 2016*, volume 13 of *Springer Proceedings in Advanced Robotics*, pages 800–815. Springer, 2016. doi: 10.1007/978-3-030-43089-4_51. URL `https://doi.org/10.1007/978-3-030-43089-4_51`. 2.1.5

[58] Robert C. Holte, T. Mkadmi, Robert M. Zimmer, and Alan J. MacDonald. Speeding up problem solving by abstraction: A graph oriented approach. *Artif. Intell.*, 85(1-2):321–361, 1996. doi: 10.1016/0004-3702(95)00111-5. URL `https://doi.org/10.1016/0004-3702(95)00111-5`. 4.2.2

[59] Baichuan Huang, Shuai D. Han, Jingjin Yu, and Abdeslam Boularias. Visual foresight trees for object retrieval from clutter with nonprehensile rearrangement. *IEEE Robotics Autom. Lett.*, 7(1):231–238, 2022. doi: 10.1109/LRA.2021.3123373. 4.2, 5.2, 6.1, 6.2

[60] Eric Huang, Zhenzhong Jia, and Matthew T. Mason. Large-scale multi-object

rearrangement. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 211–218, 2019. doi: 10.1109/ICRA.2019.8793946. 5.2, 6.1, 6.2

[61] Yixuan Huang, Adam Conkey, and Tucker Hermans. Planning for multi-object manipulation with graph neural network relational classifiers. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 1822–1829. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10161204. URL `https://doi.org/10.1109/ICRA48891.2023.10161204`. 7.1.3

[62] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In Federico Morán, Alvaro Moreno, Juan Julián Merelo Guervós, and Pablo Chacón, editors, *Advances in Artificial Life, Third European Conference on Artificial Life, Granada, Spain, June 4-6, 1995, Proceedings*, volume 929 of *Lecture Notes in Computer Science*, pages 704–720. Springer, 1995. doi: 10.1007/3-540-59496-5_337. URL `https://doi.org/10.1007/3-540-59496-5_337`. 1.2.4, 6.1

[63] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In Philippe Besnard and Steve Hanks, editors, *UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, Montreal, Quebec, Canada, August 18-20, 1995*, pages 338–345. Morgan Kaufmann, 1995. 5.5.3

[64] Aaron M Johnson, Jennifer E King, and Siddhartha Srinivasa. Convergent planning. *IEEE Robotics and Automation Letters*, 1(2):1044–1051, 2016. 2.1.4, 6.3.4

[65] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, pages 1470–1477. IEEE, 2011. doi: 10.1109/ICRA.2011.5980391. URL `https://doi.org/10.1109/ICRA.2011.5980391`. 1.2.3, 4.2, 4.4, 6.1, 6.2

[66] Marc D. Killpack, Ariel Kapusta, and Charles C. Kemp. Model predictive control for fast reaching in clutter. *Auton. Robots*, 2016. 3.2

[67] Beomjoon Kim, Luke Shimanuki, Leslie Pack Kaelbling, and Toms Lozano-Prez. Representation, learning, and planning algorithms for geometric task and motion planning. *The International Journal of Robotics Research*, 41(2):210–231, 2022. doi: 10.1177/02783649211038280. 4.2, 6.2

[68] Jennifer E. King. *Robust Rearrangement Planning Using Nonprehensile Interaction*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, December 2016. 2.1.4, 3.2, 5.2, 6.1, 6.2

[69] Jennifer E. King, Joshua A. Haustein, Siddhartha S. Srinivasa, and Tamim

Asfour. Nonprehensile whole arm rearrangement planning on physics manifolds. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2508–2515, 2015. doi: 10.1109/ICRA.2015.7139535. 2.1.3, 6.1, 6.2

[70] Jennifer E. King, Marco Cognetti, and Siddhartha S. Srinivasa. Rearrangement planning using object-centric and robot-centric action spaces. In *2016 IEEE International Conference on Robotics and Automation, ICRA*. IEEE, 2016. 2.1.1, 3.2, 4.2

[71] Jennifer E. King, Vinitha Ranganeni, and Siddhartha S. Srinivasa. Unobservable monte carlo planning for nonprehensile rearrangement tasks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4681–4688, 2017. doi: 10.1109/ICRA.2017.7989544. 2.1.3

[72] Zachary K. Kingston, Mark Moll, and Lydia E. Kavraki. Sampling-based methods for motion planning with constraints. *Annu. Rev. Control. Robotics Auton. Syst.*, 1:159–185, 2018. doi: 10.1146/annurev-control-060117-105226. 6.2

[73] Nikita Kitaev, Igor Mordatch, Sachin Patil, and Pieter Abbeel. Physics-based trajectory optimization for grasping in cluttered environments. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 3102–3109. IEEE, 2015. doi: 10.1109/ICRA.2015.7139625. URL `https://doi.org/10.1109/ICRA.2015.7139625`. 2.1.5

[74] George Dimitri Konidaris, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *J. Artif. Intell. Res.*, 61:215–289, 2018. doi: 10.1613/jair.5575. URL `https://doi.org/10.1613/jair.5575`. 7.2.1

[75] Daniel Kornhauser, Gary L. Miller, and Paul Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *FOCS25*, pages 241–250, Florida, October 1984. IEEE. 4.2

[76] Michael C. Koval, Jennifer E. King, Nancy S. Pollard, and Siddhartha S. Srinivasa. Robust trajectory selection for rearrangement planning as a multi-armed bandit problem. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, pages 2678–2685. IEEE, 2015. doi: 10.1109/IROS.2015.7353743. 2.1.3, 2.1.4

[77] Oliver Kroemer, Scott Niekum, and George Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *J. Mach. Learn. Res.*, 22:30:1–30:82, 2021. URL `http://jmlr.org/papers/v22/19-804.html`. 4.2.2

[78] Athanasios Krontiris and Kostas Bekris. Dealing with difficult instances of

object rearrangement. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015. doi: 10.15607/RSS.2015.XI.045. 2.1.3

[79] Athanasios Krontiris and Kostas E. Bekris. Dealing with difficult instances of object rearrangement. In Lydia E. Kavraki, David Hsu, and Jonas Buchli, editors, *Robotics: Science and Systems XI, Sapienza University of Rome, Rome, Italy, July 13-17, 2015*, 2015. doi: 10.15607/RSS.2015.XI.045. 4.2, 5.2, 6.1, 6.2

[80] Athanasios Krontiris, Rahul Shome, Andrew Dobson, Andrew Kimmel, and Kostas E. Bekris. Rearranging similar objects with a manipulator using pebble graphs. In *14th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2014, Madrid, Spain, November 18-20, 2014*, pages 1081–1087. IEEE, 2014. doi: 10.1109/HUMANOIDS.2014.7041499. 4.2, 5.2, 6.1, 6.2

[81] Vince Kurtz and Hai Lin. Contact-implicit trajectory optimization with hydroelastic contact and ilqr. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2022, Kyoto, Japan, October 23-27, 2022*, pages 8829–8834. IEEE, 2022. doi: 10.1109/IROS47612.2022.9981686. URL `https://doi.org/10.1109/IROS47612.2022.9981686`. 2.1.5

[82] Kyo Kutsuzawa, Sho Sakaino, and Toshiaki Tsuji. Sequence-to-sequence model for trajectory planning of nonprehensile manipulation including contact model. *IEEE Robotics and Automation Letters*, 3(4):3606–3613, 2018. doi: 10.1109/LRA.2018.2854958. 2.1.2

[83] Jean-Claude Latombe. *Robot motion planning*, volume 124 of *The Kluwer international series in engineering and computer science*. Kluwer, 1991. ISBN 978-0-7923-9206-4. doi: 10.1007/978-1-4615-4022-9. URL `https://doi.org/10.1007/978-1-4615-4022-9`. 4.2

[84] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. ISBN 9780511546877. doi: 10.1017/CBO9780511546877. URL `http://planning.cs.uiuc.edu/`. 4.2.1

[85] Steven M. LaValle and Jr. James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001. doi: 10.1177/02783640122067453. 5.2, 5.6.1

[86] Jinhwi Lee, Younggil Cho, Changjoo Nam, Jonghyeon Park, and ChangHwan Kim. Efficient obstacle rearrangement for object manipulation tasks in cluttered environments. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 183–189. IEEE, 2019. doi: 10.1109/ICRA.2019.8793616. 2.1.1, 4.2, 5.2, 6.1, 6.2

[87] Maxim Likhachev, Geoffrey J. Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Process-*

ing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada], pages 767–774. MIT Press, 2003. URL `https://proceedings.neurips.cc/paper/2003/hash/ee8fe9093fbbb687bef15a38facc44d2-Abstract.html`. 1.2, 7.2.2

[88] Weiyu Liu, Chris Paxton, Tucker Hermans, and Dieter Fox. Structformer: Learning spatial structure for language-guided semantic rearrangement of novel objects. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*, pages 6322–6329. IEEE, 2022. doi: 10.1109/ICRA46639.2022.9811931. URL `https://doi.org/10.1109/ICRA46639.2022.9811931`. 2.1.6

[89] Xibai Lou, Yang Yang, and Changhyun Choi. Learning object relations with graph neural networks for target-driven grasping in dense clutter. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*, pages 742–748. IEEE, 2022. doi: 10.1109/ICRA46639.2022.9811601. URL `https://doi.org/10.1109/ICRA46639.2022.9811601`. 7.1.3

[90] Kendall Lowrey, Svetoslav Kolev, Jeremy Dao, Aravind Rajeswaran, and Emanuel Todorov. Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system. In *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 35–42, 2018. doi: 10.1109/SIMPAR.2018.8376268. 2.1.2

[91] Kendall Lowrey, Svetoslav Kolev, Jeremy Dao, Aravind Rajeswaran, and Emanuel Todorov. Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system. In Hanna Kurniawati, Evan M. Drumwright, Bruce A. MacDonald, Thierry Fraichard, and Nan Ye, editors, *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots, SIMPAR 2018, Brisbane, Australia, May 16-19, 2018*, pages 35–42. IEEE, 2018. doi: 10.1109/SIMPAR.2018.8376268. URL `https://doi.org/10.1109/SIMPAR.2018.8376268`. 2.1.6

[92] Kevin M. Lynch. Toppling manipulation. In *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, USA, May 10-15, 1999, Proceedings*, pages 2551–2557. IEEE Robotics and Automation Society, 1999. doi: 10.1109/ROBOT.1999.773981. 2.1.2, 2.1.6

[93] Kevin M. Lynch and Matthew T. Mason. Stable pushing: Mechanics, controllability, and planning. *The International Journal of Robotics Research*, 1996. 2.1.2, 3.2, 4.2

[94] Kevin M. Lynch and Matthew T. Mason. Dynamic nonprehensile manipulation: Controllability, planning, and experiments. *The International Journal of Robotics Research*, 18(1):64–92, 1999. doi: 10.1177/027836499901800105. URL

https://doi.org/10.1177/027836499901800105. 2.1.2

[95] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance GPU based physics simulation for robot learning. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. 1.1

[96] Aditya Mandalika, Sanjiban Choudhury, Oren Salzman, and Siddhartha S. Srinivasa. Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles. In J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava, editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019*, pages 745–753. AAAI Press, 2019. URL https://ojs.aaai.org/index.php/ICAPS/article/view/3543. 6.3.3

[97] Matthew T. Mason. Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 5(3):53–71, 1986. doi: 10.1177/027836498600500303. URL https://doi.org/10.1177/027836498600500303. 1.1, 2.1.2, 4.2

[98] Igor Mordatch, Zoran Popovic, and Emanuel Todorov. Contact-invariant optimization for hand manipulation. In Jehee Lee and Paul G. Kry, editors, *Proceedings of the 2012 Eurographics/ACM SIGGRAPH Symposium on Computer Animation, SCA 2012, Lausanne, Switzerland, 2012*, pages 137–144. Eurographics Association, 2012. doi: 10.2312/SCA/SCA12/137-144. URL https://doi.org/10.2312/SCA/SCA12/137-144. 2.1.5

[99] Igor Mordatch, Emanuel Todorov, and Zoran Popovic. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph.*, 31(4): 43:1–43:8, 2012. doi: 10.1145/2185520.2185539. URL https://doi.org/10.1145/2185520.2185539. 2.1.5

[100] João Moura, Theodoros Stouraitis, and Sethu Vijayakumar. Non-prehensile planar manipulation via trajectory optimization with complementarity constraints. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*, pages 970–976. IEEE, 2022. doi: 10.1109/ICRA46639.2022.9811942. URL https://doi.org/10.1109/ICRA46639.2022.9811942. 2.1.5

[101] Muhayyuddin, Mark Moll, Lydia Kavraki, and Jan Rosell. Randomized physics-based motion planning for grasping in cluttered and uncertain environments. *IEEE Robotics and Automation Letters*, 3(2):712–719, 2018. doi: 10.1109/LRA.

2017.2783445. 2.1.4

[102] Shohin Mukherjee, Sandip Aine, and Maxim Likhachev. epa*se: Edge-based parallel a* for slow evaluations. In Lukás Chrpa and Alessandro Saetti, editors, *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria, July 21-23, 2022*, pages 136–144. AAAI Press, 2022. 5.7, 6.3.3

[103] Changjoo Nam, Jinhwi Lee, SangHun Cheong, Brian Y. Cho, and ChangHwan Kim. Fast and resilient manipulation planning for target retrieval in clutter. In *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, pages 3777–3783. IEEE, 2020. doi: 10.1109/ICRA40945.2020.9196652. 4.2, 5.2, 6.1, 6.2

[104] Changjoo Nam, SangHun Cheong, Jinhwi Lee, Dong Hwan Kim, and ChangHwan Kim. Fast and resilient manipulation planning for object retrieval in cluttered and confined environments. *IEEE Trans. Robotics*, 37(5):1539–1552, 2021. doi: 10.1109/TRO.2020.3047472. URL `https://doi.org/10.1109/TRO.2020.3047472`. 7.2.2

[105] Venkatraman Narayanan and Maxim Likhachev. Heuristic search on graphs with existence priors for expensive-to-evaluate edges. In Laura Barbulescu, Jeremy Frank, Mausam, and Stephen F. Smith, editors, *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017*, pages 522–530. AAAI Press, 2017. 5.7

[106] Ramkumar Natarajan, Muhammad Suhail Saleem, Sandip Aine, Maxim Likhachev, and Howie Choset. A-MHA*: Anytime multi-heuristic A*. In Pavel Surynek and William Yeoh, editors, *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, pages 192–193. AAAI Press, 2019. URL `https://aaai.org/ocs/index.php/SOCS/SOCS19/paper/view/18383`. 1.2

[107] Ramkumar Natarajan, Garrison L. H. Johnston, Nabil Simaan, Maxim Likhachev, and Howie Choset. Torque-limited manipulation planning through contact by interleaving graph search and trajectory optimization. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 8148–8154. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10161297. URL `https://doi.org/10.1109/ICRA48891.2023.10161297`. 2.1.5

[108] Jun Ota. Rearrangement planning of multiple movable objects by a mobile robot. *Adv. Robotics*, 23(1-2):1–18, 2009. doi: 10.1163/156855308X392654. 4.2

[109] Robert Paolini, Alberto Rodriguez, Siddhartha S. Srinivasa, and Matthew T.

Mason. A data-driven statistical framework for post-grasp manipulation. *Int. J. Robotics Res.*, 33(4):600–615, 2014. doi: 10.1177/0278364913507756. URL https://doi.org/10.1177/0278364913507756. 7.1.3, 7.2.1

[110] R. Papallas, A. G. Cohn, and M. R. Dogar. Online replanning with human-in-the-loop for non-prehensile manipulation in clutter a trajectory optimization based approach. *IEEE Robotics and Automation Letters*, 2020. 3.2

[111] Sangbeom Park, Yoonbyung Chai, Sunghyun Park, Jeongeun Park, Kyungjae Lee, and Sungjoon Choi. Semi-autonomous teleoperation via learning non-prehensile manipulation skills. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2022. 5.2, 6.1, 6.2

[112] Chris Paxton, Yotam Barnoy, Kapil D. Katyal, Raman Arora, and Gregory D. Hager. Visual robot task planning. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 8832–8838. IEEE, 2019. doi: 10.1109/ICRA.2019.8793736. URL https://doi.org/10.1109/ICRA.2019.8793736. 2.1.6

[113] Judea Pearl. *Heuristics - intelligent search strategies for computer problem solving*. Addison-Wesley series in artificial intelligence. Addison-Wesley, 1984. ISBN 978-0-201-05594-8. 2.2, 4.2.2, 5.7

[114] Michael A. Peshkin. *Planning Robotic Manipulation Strategies for Sliding Objects*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1986. 2.1.1

[115] Lerrel Pinto, Aditya Mandalika, Brian Hou, and Siddhartha Srinivasa. Sample-efficient learning of nonprehensile manipulation policies via physics-based informed state distributions. *arXiv preprint arXiv:1810.10654*, 2018. 2.1.2

[116] Lerrel Pinto, Aditya Mandalika, Brian Hou, and Siddhartha S. Srinivasa. Sample-efficient learning of nonprehensile manipulation policies via physics-based informed state distributions. *CoRR*, abs/1810.10654, 2018. 3.2

[117] Lerrel Pinto, Aditya Mandalika, Brian Hou, and Siddhartha S. Srinivasa. Sample-efficient learning of nonprehensile manipulation policies via physics-based informed state distributions. *CoRR*, abs/1810.10654, 2018. URL http://arxiv.org/abs/1810.10654. 2.1.6

[118] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Trans. Robotics*, 2010. 2.1.4, 3.2

[119] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artif. Intell.*, 1(3): 193–204, 1970. doi: 10.1016/0004-3702(70)90007-X. URL https://doi.org/10.1016/0004-3702(70)90007-X. 1.2

[120] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory

optimization of rigid bodies through contact. *Int. J. Robotics Res.*, 33(1):69–81, 2014. doi: 10.1177/0278364913506757. URL https://doi.org/10.1177/0278364913506757. 2.1.5

[121] Ahmed H Qureshi, Arsalan Mousavian, Chris Paxton, Michael Yip, and Dieter Fox. NeRP: Neural Rearrangement Planning for Unknown Objects. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021. doi: 10.15607/RSS.2021.XVII.072. 2.1.3

[122] Ahmed Hussain Qureshi, Arsalan Mousavian, Chris Paxton, Michael C. Yip, and Dieter Fox. Nerp: Neural rearrangement planning for unknown objects. In Dylan A. Shell, Marc Toussaint, and M. Ani Hsieh, editors, *Robotics: Science and Systems XVII, Virtual Event, July 12-16, 2021*, 2021. doi: 10.15607/RSS.2021.XVII.072. URL https://doi.org/10.15607/RSS.2021.XVII.072. 2.1.6

[123] Fabio Ruggiero, Vincenzo Lippiello, and Bruno Siciliano. Nonprehensile dynamic manipulation: A survey. *IEEE Robotics and Automation Letters*, 3(3):1711–1718, 2018. doi: 10.1109/LRA.2018.2801939. 2.1.2

[124] David Russell, Rafael Papallas, and Mehmet Remzi Dogar. Adaptive approximation of dynamics gradients via interpolation to speed up trajectory optimisation. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 10160–10166. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10161090. URL https://doi.org/10.1109/ICRA48891.2023.10161090. 2.1.5

[125] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artif. Intell.*, 5(2):115–135, 1974. doi: 10.1016/0004-3702(74)90026-5. URL https://doi.org/10.1016/0004-3702(74)90026-5. 4.2.2

[126] Terence D. Sanger. Neural network learning control of robot manipulators using gradually increasing task difficulty. *IEEE Trans. Robotics Autom.*, 10(3):323–333, 1994. doi: 10.1109/70.294207. URL https://doi.org/10.1109/70.294207. 4.2.2

[127] Dhruv Saxena and Maxim Likhachev. Planning for manipulation among movable objects: Deciding which objects go where, in what order, and how. *Proceedings of the International Conference on Automated Planning and Scheduling*, 33(1):668–676, Jul. 2023. doi: 10.1609/icaps.v33i1.27249. URL https://ojs.aaai.org/index.php/ICAPS/article/view/27249. 1.2, 1.2.3, 5, 6, 6.1, 6.5

[128] Dhruv Mauria Saxena and Maxim Likhachev. Planning for complex non-prehensile manipulation among movable objects by interleaving multi-agent pathfinding and physics-based simulation. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 8141–8147. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10161006. URL

https://doi.org/10.1109/ICRA48891.2023.10161006. 1.2, 1.2.2, 4, 5.1, 5.7, 6.1

[129] Dhruv Mauria Saxena, Sangjae Bae, Alireza Nakhaei, Kikuo Fujimura, and Maxim Likhachev. Driving in dense traffic with model-free reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, pages 5385–5392. IEEE, 2020. doi: 10.1109/ICRA40945.2020.9197132. URL https://doi.org/10.1109/ICRA40945.2020.9197132. 1.2

[130] Dhruv Mauria Saxena, Muhammad Suhail Saleem, and Maxim Likhachev. Manipulation planning among movable obstacles using physics-based adaptive motion primitives. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6570–6576, 2021. doi: 10.1109/ICRA48506.2021.9561221. 1.2, 1.2.1, 3, 4.2, 5.2

[131] Dhruv Mauria Saxena, Tushar Kusnur, and Maxim Likhachev. Amra*: Anytime multi-resolution multi-heuristic a*. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3371–3377, 2022. doi: 10.1109/ICRA46639.2022.9812359. 1.2

[132] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Meta-agent conflict-based search for optimal multi-agent path finding. In Daniel Borrajo, Ariel Felner, Richard E. Korf, Maxim Likhachev, Carlos Linares López, Wheeler Ruml, and Nathan R. Sturtevant, editors, *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*. AAAI Press, 2012. URL http://www.aaai.org/ocs/index.php/SOCS/SOCS12/paper/view/5402. 4.2.1

[133] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Meta-agent conflict-based search for optimal multi-agent path finding. In Daniel Borrajo, Ariel Felner, Richard E. Korf, Maxim Likhachev, Carlos Linares López, Wheeler Ruml, and Nathan R. Sturtevant, editors, *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*. AAAI Press, 2012. 4.2.1

[134] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artif. Intell.*, 219:40–66, 2015. doi: 10.1016/j.artint.2014.11.006. URL https://doi.org/10.1016/j.artint.2014.11.006. 4.2.1, 4.3.1, 1, 5.4, 5.4.2, 7.1.1

[135] Rahul Shome and Kostas E. Bekris. Synchronized multi-arm rearrangement guided by mode graphs with capacity constraints. In Steven M. LaValle, Ming Lin, Timo Ojala, Dylan A. Shell, and Jingjin Yu, editors, *Algorithmic Foundations of Robotics XIV, Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics, WAFR 2021, Oulu, Finland, June 21-23,*

2021, volume 17 of *Springer Proceedings in Advanced Robotics*, pages 243–260. Springer, 2021. doi: 10.1007/978-3-030-66723-8_15. 4.2, 5.2, 6.1, 6.2

[136] Rahul Shome, Kiril Solovey, Andrew Dobson, Dan Halperin, and Kostas E Bekris. drrt*: Scalable and informed asymptotically-optimal multi-robot motion planning. *Autonomous Robots*, 44(3):443–467, 2020. 4.2.1

[137] David Silver. Cooperative pathfinding. In R. Michael Young and John E. Laird, editors, *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA*, pages 117–122. AAAI Press, 2005. 4.2.1

[138] Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani. Manipulation planning with probabilistic roadmaps. *Int. J. Robotics Res.*, 23(7-8): 729–746, 2004. doi: 10.1177/0278364904045471. URL https://doi.org/10.1177/0278364904045471. 6.1

[139] Anthony Simeonov, Yilun Du, Beomjoon Kim, Francois Robert Hogan, Joshua B. Tenenbaum, Pulkit Agrawal, and Alberto Rodriguez. A long horizon planning framework for manipulating rigid pointcloud objects. In Jens Kober, Fabio Ramos, and Claire J. Tomlin, editors, *4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA*, volume 155 of *Proceedings of Machine Learning Research*, pages 1582–1601. PMLR, 2020. URL https://proceedings.mlr.press/v155/simeonov21a.html. 7.2.1

[140] Kiril Solovey and Dan Halperin. k-color multi-robot motion planning. In Emilio Frazzoli, Tomas Lozano-Perez, Nicholas Roy, and Daniela Rus, editors, *Algorithmic Foundations of Robotics X*, pages 191–207, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 4.2

[141] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4739–4748. PMLR, 2018. URL http://proceedings.mlr.press/v80/srinivas18b.html. 2.1.6

[142] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In Pavel Surynek and William Yeoh, editors, *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, pages 151–159. AAAI Press, 2019. URL https://aaai.org/ocs/index.php/SOCS/SOCS19/paper/view/18341. 4.2.1

[143] Mike Stilman and James J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *Int. J. Humanoid Robotics*, 2(4): 479–503, 2005. doi: 10.1142/S0219843605000545. 2.1.1, 3.1, 3.2, 4.2

[144] Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour. Manipulation planning among movable obstacles. In *2007 IEEE International Conference on Robotics and Automation, ICRA*. IEEE, 2007. 1.2, 2.1.1, 3.1, 3.2, 4.1, 4.2, 4.5.1, 5.1, 5.1, 5.2, 6.1, 6.1, 6.2, 6.3.2

[145] Freek Stulp, Andreas Fedrizzi, Lorenz Mösenlechner, and Michael Beetz. Learning and reasoning with action-related places for robust mobile manipulation. *J. Artif. Intell. Res.*, 43:1–42, 2012. doi: 10.1613/jair.3451. URL `https://doi.org/10.1613/jair.3451`. 7.2.1

[146] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. `https://ompl.kavrakilab.org`. 3.5.1, 4.5.1, 5.6.1

[147] Ioan Alexandru Sucan and Lydia E. Kavraki. A sampling-based tree planner for systems with complex dynamics. *IEEE Trans. Robotics*, 2012. 2.1.4, 3.2, 3.5.1, 5, 4.2, 4.5.1, 4.1, 5.2, 5.6.1

[148] Muhammad Suhail Saleem and Maxim Likhachev. Planning with selective physics-based simulation for manipulation among movable objects. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6752–6758, 2020. doi: 10.1109/ICRA40945.2020.9197451. 3.2, 3.3.2, 3.5.1, 4.2, 4.5.1, 4.1, 5.2, 5.6.1

[149] Russell H. Taylor, Matthew T. Mason, and Kenneth Y. Goldberg. Sensor-based manipulation planning as a game with nature. In *Proceedings of the 4th International Symposium on Robotics Research*, page 421429, Cambridge, MA, USA, 1988. MIT Press. ISBN 0262022729. 2.1.1

[150] Wil Thomason, Marlin P. Strub, and Jonathan D. Gammell. Task and motion informed trees (tmit*): Almost-surely asymptotically optimal integrated task and motion planning. *IEEE Robotics Autom. Lett.*, 7(4):11370–11377, 2022. doi: 10.1109/LRA.2022.3199676. 6.2

[151] Wil B. Thomason and Ross A. Knepper. A unified sampling-based approach to integrated task and motion planning. In Tamim Asfour, Eiichi Yoshida, Jaeheung Park, Henrik Christensen, and Oussama Khatib, editors, *Robotics Research - The 19th International Symposium ISRR 2019, Hanoi, Vietnam, October 6-10, 2019*, volume 20 of *Springer Proceedings in Advanced Robotics*, pages 773–788. Springer, 2019. doi: 10.1007/978-3-030-95459-8_47. 6.2

[152] Emanuel Todorov. Convex and analytically-invertible dynamics with contacts

and constraints: Theory and implementation in mujoco. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pages 6054–6061. IEEE, 2014. doi: 10.1109/ICRA.2014.6907751. URL https://doi.org/10.1109/ICRA.2014.6907751. 2.1.5

[153] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109. 1.1

[154] Marc Toussaint, Kelsey R. Allen, Kevin A. Smith, and Joshua B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In Hadas Kress-Gazit, Siddhartha S. Srinivasa, Tom Howard, and Nikolay Atanasov, editors, *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, 2018. doi: 10.15607/RSS.2018.XIV.044. URL http://www.roboticsproceedings.org/rss14/p44.html. 7.2.3

[155] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, volume 87 of *Proceedings of Machine Learning Research*, pages 306–316. PMLR, 2018. URL http://proceedings.mlr.press/v87/tremblay18a.html. 1.2.4, 5.7, 6.3.4

[156] Matthew Turpin, Kartik Mohta, Nathan Michael, and Vijay Kumar. Goal assignment and trajectory planning for large teams of interchangeable robots. *Auton. Robots*, 37(4):401–415, 2014. doi: 10.1007/s10514-014-9412-1. URL https://doi.org/10.1007/s10514-014-9412-1. 4.2.1

[157] Nikolaus Vahrenkamp, Martin Do, Tamim Asfour, and Rüdiger Dillmann. Integrated grasp and motion planning. In *2010 IEEE International Conference on Robotics and Automation*, pages 2883–2888. IEEE, 2010. 3.2

[158] Michiel van de Panne and Alexis Lamouret. Guided optimization for balanced locomotion. In Demetri Terzopoulos and Daniel Thalmann, editors, *Computer Animation and Simulation '95*, pages 165–177, Vienna, 1995. Springer Vienna. ISBN 978-3-7091-9435-5. 2.1.5, 4.2.2

[159] Jur P. van den Berg, Mike Stilman, James Kuffner, Ming C. Lin, and Dinesh Manocha. Path planning among movable obstacles: A probabilistically complete approach. In *Eighth International Workshop on the Algorithmic Foundations of Robotics, WAFR 2008*. 2.1.1, 3.2, 4.2, 5.1, 5.2, 6.1, 6.2

[160] William Vega-Brown and Nicholas Roy. Asymptotically optimal planning under piecewise-analytic constraints. In Ken Goldberg, Pieter Abbeel, Kostas E.

Bekris, and Lauren Miller, editors, *Algorithmic Foundations of Robotics XII, Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics, WAFR 2016, San Francisco, California, USA, December 18-20, 2016*, volume 13 of *Springer Proceedings in Advanced Robotics*, pages 528–543. Springer, 2016. doi: 10.1007/978-3-030-43089-4_34. 6.1, 6.2

[161] E Vieira, D Nakhimovich, K Gao, R Wang, J Yu, and K E Bekris. Persistent homology for effective non-prehensile manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2022. 4.2, 5.1, 5.2, 6.1, 6.2

[162] Glenn Wagner and Howie Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011*, pages 3260–3267. IEEE, 2011. doi: 10.1109/IROS.2011.6095022. URL https://doi.org/10.1109/IROS.2011.6095022. 4.2.1

[163] Rui Wang, Kai Gao, Daniel Nakhimovich, Jingjin Yu, and Kostas E. Bekris. Uniform object rearrangement: From complete monotone primitives to efficient non-monotone informed search. In *International Conference on Robotics and Automation (ICRA) 2021*, 2021. 2.1.1, 4.2, 6.1, 6.2

[164] Rui Wang, Kai Gao, Jingjin Yu, and Kostas Bekris. Lazy rearrangement planning in confined spaces. *Proceedings of the International Conference on Automated Planning and Scheduling*, 32(1):385–393, Jun. 2022. doi: 10.1609/icaps.v32i1.19824. URL https://ojs.aaai.org/index.php/ICAPS/article/view/19824. 5.1, 5.2, 6.1, 6.2

[165] Gordon T. Wilfong. Motion planning in the presence of movable obstacles. *Ann. Math. Artif. Intell.*, 1991. 1.2, 2.1.1, 3.1, 3.2, 4.2

[166] J. Zachary Woodruff and Kevin M. Lynch. Planning and control for dynamic, nonprehensile, and hybrid manipulation tasks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4066–4073, 2017. doi: 10.1109/ICRA.2017.7989467. 2.1.2

[167] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In Hadas Kress-Gazit, Siddhartha S. Srinivasa, Tom Howard, and Nikolay Atanasov, editors, *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, 2018. doi: 10.15607/RSS.2018.XIV.019. 1.2.4, 5.7, 6.3.4

[168] Kechun Xu, Hongxiang Yu, Renlang Huang, Dashun Guo, Yue Wang, and Rong Xiong. Efficient object manipulation to an arbitrary goal pose: Learning-based anytime prioritized planning. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27,*

2022, pages 7277–7283. IEEE, 2022. doi: 10.1109/ICRA46639.2022.9811547. URL https://doi.org/10.1109/ICRA46639.2022.9811547. 7.2.3

[169] A. Yamashita, T. Arai, Jun Ota, and H. Asama. Motion planning of multiple mobile robots for cooperative manipulation and transportation. *IEEE Transactions on Robotics and Automation*, 19(2):223–237, 2003. doi: 10.1109/TRA.2003.809592. 2.1.2

[170] Jingjin Yu and Steven M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press, 2013. 4.3.1

[171] Jingjin Yu and Steven M. LaValle. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016. doi: 10.1109/TRO.2016.2593448. 4.2.1

[172] Weihao Yuan, Johannes A. Stork, Danica Kragic, Michael Y. Wang, and Kaiyu Hang. Rearrangement with nonprehensile manipulation using deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 270–277, 2018. doi: 10.1109/ICRA.2018.8462863. 2.1.3

[173] Weihao Yuan, Johannes A. Stork, Danica Kragic, Michael Yu Wang, and Kaiyu Hang. Rearrangement with nonprehensile manipulation using deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 270–277. IEEE, 2018. doi: 10.1109/ICRA.2018.8462863. URL https://doi.org/10.1109/ICRA.2018.8462863. 2.1.6

[174] Wentao Yuan, Chris Paxton, Karthik Desingh, and Dieter Fox. SORNet: Spatial object-centric representations for sequential manipulation. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=mOLu2rODIJF. 2.1.3

[175] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2020. 2.1.3

[176] Stefan Zickler and Manuela M. Veloso. Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*. IFAAMAS, 2009. 2.1.4, 3.2