Subdimensional Expansion: A Framework for Computationally Tractable Multirobot Path Planning

Glenn Wagner

CMU-RI-TR-15-33

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

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

December 2015

Thesis Committee:
Howie Choset
Manuela Veloso
Maxim Likhachev
Sven Koenig
Vijay Kumar

Keywords: multirobot systems, path planning, planning with uncertainty

Abstract

Planning optimal paths for large numbers of robots is computationally expensive. In this thesis, we present a new framework for multirobot path planning called subdimensional expansion, which initially plans for each robot individually, and then coordinates motion among the robots as needed. More specifically, subdimensional expansion initially creates a one-dimensional search space embedded in the joint configuration space of the multirobot system. When the search space is found to be blocked during planning by a robot-robot collision, the dimensionality of the search space is locally increased to ensure that an alternative path can be found. As a result, robots are only coordinated when necessary, which reduces the computational cost of finding a path. Subdimensional expansion is a flexible framework that can be used with multiple planning algorithms. For discrete planning problems, subdimensional expansion can be combined with A* to produce the M* algorithm, a complete and optimal multirobot path planning problem. When the configuration space of individual robots is too large to be explored effectively with A*, subdimensional expansion can be combined with probabilistic planning algorithms to produce sRRT and sPRM.

M* is then extended to solve variants of the multirobot path planning algorithm. We present the Constraint Manifold Subsearch (CMS) algorithm to solve problems where robots must dynamically form and dissolve teams with other robots to perform cooperative tasks. Uncertainty M* (UM*) is a variant of M* that handles systems with probabilistic dynamics. Finally, we apply M* to multirobot sequential composition. Results are validated with extensive simulations and experiments on multiple physical robots.

Acknowledgments

This thesis would not have been possible without the support and guidance of many people. First thanks must go to my advisor, Howie Choset, for his advice and support throughout a rather prolonged process. His support was unwavering throughout the years, even during a year and a half slog through a set of ideas that sounded good at first, but proved a dead end.

The members of my committee, Manuela Veloso, Maxim Likhachev, Sven Koenig, and Vijay Kumar, provided invaluable advice and guidance. They offered valuable pointers on research directions, and caught important implicit assumptions that I made on several occasions.

The members of the biorobotics lab were critical to this work. Special thanks are due to Peter Karkus, with whom I collaborated on rCMS. He wrote the rCMS code, and extended a conference paper to form the first draft of the CMS chapter in this thesis. Jae Kim and Konrad Urban wrote the code for the base CMS implementation. I would also like to thank Matt Tesch, Ross Hatton, Dave Rollinson, Chahui Gong, Tony Dear, and Matt Travers for their guidance in the ways of being a grad student, service as a sounding board for ideas, and assistance in editing numerous papers.

I would also like to thank my friends Alex Roper, D W Rowlands [146], Andrea Dubin [56], Kristen Kozak, and Cathy Douglas, as well as the CMU KGB for keeping me, if not entirely sane, at least functional.

Finally, I would like to thank my parents, who provided the resources and education necessary for me to even reach CMU, and always had a sympathetic ear when I ran into problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multirobot systems offer flexibility, sensor coverage, and redundancy, which makes them attractive for tasks such as surveillance, search and rescue, and warehouse automation. Exploiting the benefits of multirobot systems requires addressing a multitude of issues including task assignment, communication, synchronization of world models, and the coordination of large numbers of robots, in addition to all the challenges that face single robot systems.

One of the fundamental problems for multirobot systems is finding safe, collision free paths that take robots to configurations at which they can perform tasks, which is termed the *Multirobot Path Planning* (MPP) problem. There is a fundamental trade-off between path quality and the computational cost of finding solutions. Finding *optimal* solutions (*i.e.* minimal cost paths) is known to be NP-complete[1] [143, 205], while finding solutions to more complex formulations of the MPP problem that allow robots of differing sizes is PSPACE-hard[2] [78]. Conversely, feasible paths of unbounded length can be found in polynomial time [98, 199]. Thus high-quality

[1] NP-complete is a complexity class that represents the hardest problems in NP, such as the traveling salesman problem and Boolean satisfiability. No known polynomial time algorithms exist.

[2] PSPACE-hard problems require polynomial space to solve, and are believed to be harder than NP-complete problems

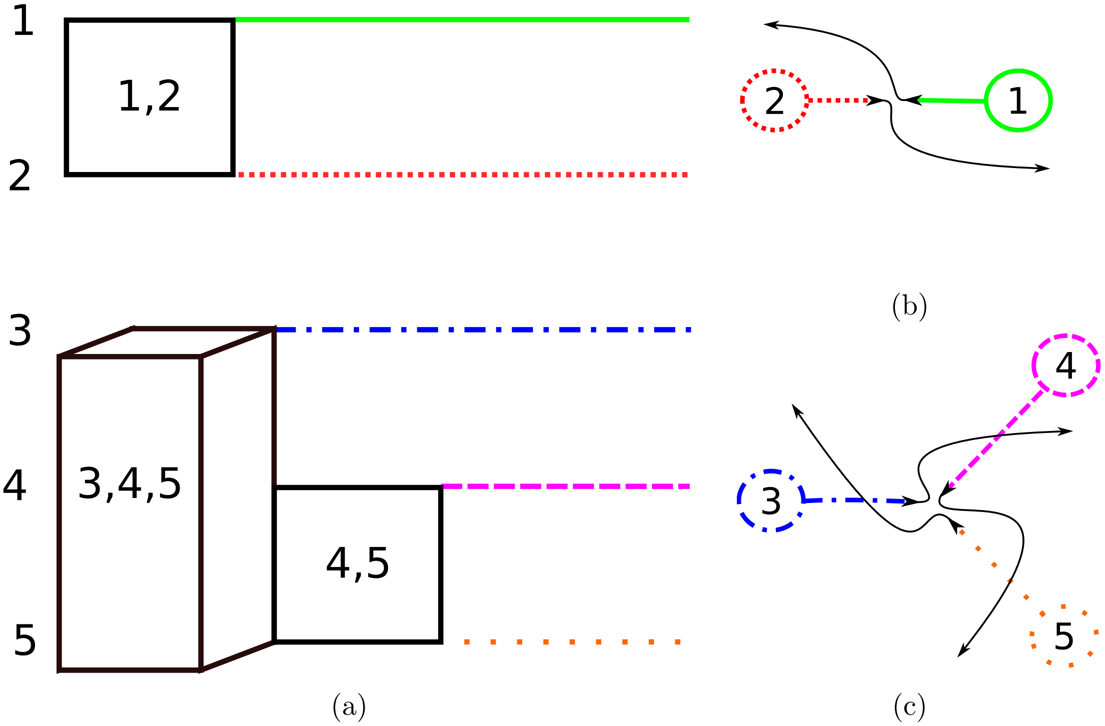


Figure 1.1: An abstract visualization of a variable dimensionality search space constructed by *subdimensional expansion*, for a system of five robots. **(a)** Initially each robot is constrained to its individually optimal path, represented by a single line, but when the individually optimal paths for robots 1 and 2 are found to conflict **(b)**, the local dimensionality of the search space must be increased, as represented by a 2D square. When the individually optimal paths of three robots are found to conflict **(c)**, the local dimensionality of the search space must be increased further, represented by the 3D cube, to include all local paths of the three robots. If robot 3 clears robots 4 and 5 before they resolve their mutual interaction, then the dimensionality of the search decreases so that planning is coupled only for robots 4 and 5.

paths are hard to find, while low quality paths can be found rapidly.

In this thesis, we introduce a new approach that can find high quality paths quickly called subdimensional expansion. Subdimensional expansion is not a specific algorithm, but rather a method for manipulating the search spaces of existing search algorithms to decrease the computational cost of solving MPP problems. Subdimensional expansion starts by finding a path for each robot in its individual configuration space without regard for robot-robot interactions or collisions (*i.e.* as if the robot were the only robot). Combining the individual paths of each robot defines a one-dimensional search space for the full multirobot system embedded in the *joint configuration space*. When robots are found to collide in the multirobot search space, subdimensional expansion locally grows the dimensionality of the search space to allow an alternative path for the robots involved in the collision to be found with coupled planning, while planning for the uninvolved robots remains decoupled (Figure 1.1). Although the search space may grow to cover the entire joint configuration space in the worst case, leading to exponential time complexity, for many problems subdimensional expansion can construct a low dimensional search space that allows for efficient computation of a high quality path.

1.0.1 Contributions

This thesis makes contributions to optimal and ϵ-suboptimal MPP on graphs, MPP for systems where individual robots have many degrees of freedom (DOFs), cooperative path planning where robots must dynamically form teams to execute cooperative tasks, planning with uncertainty, and combined planning and control for multirobot systems (Table 1.1). All these contributions are based on subdimensional expansion, a new framework for MPP that combines computationally efficient planning with the flexibility to be readily applied to many variants of the MPP problem. Applying sub-

Contribution	Explanation	Chapter
Optimal and ϵ-*suboptimal* MPP on graphs	M* and its variants provide state of the art performance for MPP on graphs, including directed graphs	3
MPP with robots with many DOFs	sRRT and sPRM allow for efficient path planning for many robots where each robot has many DOFs	4
Planning for many robots with cooperative paths	CMS allows for planning paths for many robots that must dynamically form teams to perform cooperative tasks	5
Planning with uncertainty	UM* can efficiently find paths for systems with uncertain dynamics where interactions are possible anywhere in the workspace	6
Combined planning and control	M* can be combined with the sequential composition framework to generate plans for multirobot systems that consist of a sequence of controllers, robustness against environmental perturbations and modeling errors	7

Table 1.1: Contributions

dimensional expansion to graph search results in the M* algorithm and its variants. M* provides state of the art performance for finding optimal and ϵ-suboptimal paths. Subdimensional expansion can be combined with probabilistic planning algorithms, resulting in the sRRT and sPRM algorithms, to find paths for multirobot systems where each robot has many degrees of freedom, outperforming existing probabilistic approaches. CMS is an implementation of subdimensional expansion that can find paths for multirobot systems where the robots must temporarily form teams to perform cooperative tasks, a problem that has received relatively little attention. UM*, a variant of M*, can efficiently solve problems where the dynamics of the robots are uncertain. Unlike the work of Melo and Veloso [119], UM* can solve problems where robots can interact anywhere in the workspace, instead of only in well defined interaction regions, but does so at the cost of computing only a single trajectory, rather than a full policy that could respond to sensor measurements received during execution [119]. Finally, we show that M* can be combined with the sequential composition framework to combine planning and control of multirobot systems. This is enabled by the ability of M* to compute plans on directed graphs, and produces paths that are robust to environmental perturbations.

The thesis is organized as follows: We begin by describing the MPP problem and the prior work. Chapter 2 describes subdimensional expansion. Chapter 3 describes M*, an implementation of subdimensional expansion where the configuration space of each robot is represented as a graph. Chapter 4 describes implementations of subdimensional expansion based on probabilistic planning algorithms, which are suitable for systems where each robot has many degrees of freedom. Chapter 5 describes an adaptation of M* to handle problems where robots must dynamically form and dissolve teams to perform cooperative tasks. Robots will not perfectly execute plans, so the last two chapters focus on generating plans that are robust to errors in plan execution, via explicitly accounting for uncertainty at planning time (Chapter 6) and integrating path planning and control via the sequential composition framework (Chapter 7).

1.1 Problem Definition

We start by formally defining the MPP problem. Consider a system of n robots r^i indexed by the set $I = \{1, \ldots, n\}$. Each robot has a free configuration space Q^i_{free}. The joint configuration space that represents the state of the entire system is given by the direct product of the single robot free configuration spaces $Q = \prod_{i \in I} Q^i_{\text{free}}$. By this definition, the joint configuration space may contain robot-robot collisions, but no obstacle-robot collisions. Let the Π denote the space of continuous paths $\pi : [0, 1] \rightarrow Q$ in the joint configuration space. The MPP problem is to find an optimal, collision-free path $\pi_* \in \Pi$, from an initial configuration of the system q_s to a goal configuration q_f, that minimizes a cost functional $g : \Pi \rightarrow \mathbb{R}^+$. To define which states result in robot-robot collisions, we introduce a collision function $\Psi : Q \rightarrow \mathcal{P}(I)$ which returns the set of robots in collision at a given joint configuration, where $\mathcal{P}(I)$ is the power set of I containing all subsets of I. The MPP problem can be expressed as

$$\begin{gathered} \pi_* = \text{argmin } g(\pi) \\ \text{s.t.} \\ \pi_*(0) = q_s \\ \pi_*(1) = q_f \\ \forall t \in [0, 1] \; \Psi(\pi(t)) = \emptyset. \end{gathered} \tag{1.1}$$

There are several important variants of the MPP problem. The *permutation invariant multirobot path planning* problem deals with homogeneous robots where a robot must reach each goal position, but any robot can be assigned to any goal

[183, 204]. The k-color MPP problem is a generalization where there are k classes of robots that are interchangeable within, but not between, classes [164]. In the vehicle routing problem, there are a set of goal positions that must be visited by a robot, and each robot can visit some, all, or none of the goal positions [180]. The *Cooperative Path Planning* (CPP) problem is a variant where multiple robots must temporarily form tightly coordinated teams to perform cooperative tasks. The *Multirobot Path Planning with Uncertainty* (MPPU) problem addresses systems in which the dynamics of the robots are uncertain.

Equation 1.1 looks like the formulation of the single robot path planning problem for a robot with the configuration space Q, which raises the question of how the MPP problem differs from the single robot path planning problem. The first difference is qualitative; the joint configuration space for multirobot systems can exceed 2000 dimensions [86, 192], while single robots typically don't have more than 20 degrees of freedom [136, 172]. The second difference lies in the direct product structure of the joint configuration space, which means that not only can the joint configuration space be factored into the product of single robot configuration spaces, but the actions of the system as a whole can be factored into the actions of individual robots. As a result, a planner can meaningfully reason about the motion of individual robots, whereas the motion of individual joints in a robot arm cannot be generally considered independently of one another.

1.2 Prior Work

MPP algorithms can be characterized by how and to what extent they exploit the direct product structure of the joint configuration space to accelerate planning. In general, the more heavily a MPP depends on the direct product structure, the faster it will find solutions, at the cost of returning more expensive paths and possibly failing to find a valid path. We proceed to discuss the existing literature on MPP grouped by how the algorithms exploit the structure of the joint configuration space.

1.2.1 Reactive Planning

One possible approach to solving the MPP problem is to plan paths for each robot seperately, then run a reactive controller on each robot during execution to avoid collisions and deadlocks[3]. One approach is to simply have the robots stop if they believe a collision is imminent [85], which poses an obvious danger of deadlocks. A less deadlock-prone approach is to command robots to follow a circular path and rotate around one another [38, 75]. A variety of controllers have been inspired by biological swarms [36, 44, 61, 73, 111, 115, 145, 193], using a combination of short-ranged repulsive forces, mid-ranged alignment forces, and long-ranged attractive forces. Swarm-inspired controllers are generally designed to move large numbers of robots to a single destination as a coherent flock.

In the aforementioned approaches, a robot does not consider the fact that the robots with which it interacts are also trying to avoid collisions. Therefore, a robot may work harder than necessary to avoid collisions. van den Berg et al. [187] introduced reciprocal velocity obstacles that split the responsibility for avoiding a collision between the interacting robots, but makes the assumption that all robots execute the same controller [163, 190]. Trautman and Krause [181] used a Gaussian process to learn how other agents/pedestrians react to the presence of a robot, allowing cooperative collision avoidance between inhomogeneous robots.

[3] A deadlock occurs when one or more robots become permanently unable to move under the chosen control scheme. A live lock occurs when robots continue to move but permanently fail to make progress, typically by entering a cycle.

1.2.2 Workspace Decomposition

Another approach is to decompose the workspace into a number of regions. Robots in different regions are known not to interact with one another, and simple rules can be established to govern how robots can move from one region to another. One method is to break the world into a series of corridors, and then prescribe traffic rules for navigating intersections [2, 55, 196]. Švestka and Overmars [178] developed a multi-level hierarchical graph, where a single vertex may represent an entire region of the workspace, while Ryan [148, 149] defines a high-level graph by decomposing the graph that represents the workspace into cliques, stacks, and singletons. Each vertex in the high-level graph can contain a specific number of robots, and has rules for entering and exiting the vertex. A plan for the entire system is first found in the high-level graph; a detailed plan for each robot through the workspace can be extracted from the high-level plan later.

Rather than dividing the workspace into a small number of large regions, the workspace can be split into many small reservation cells. Before a robot is allowed to move into a new cell, it must acquire an exlusive reservation for said cell, either from a centralized authority [55, 200], or via negotiation with nearby robots [142]. As long as each cell is large enough for a robot to come to a complete stop, safety can be guaranteed.

Alternatively, the workspace can be split into regions where coordination between robots is or is not necessary. Interaction regions can be defined as states at which the reward or transition function of a robot depends upon one or more other robot. Varakantham et al. [191] handled such cases by modifying the individual reward function for each robot, increasing the reward for states where synergistic interaction could occur, and decreasing the reward for states with antagonistic interactions. Spaan and Melo [165] learned an individual policy for each robot outside of the interaction states, and a joint policy for robots in interaction states. Furthermore, Spaan and Melo [165] showed that performance could be improved by beginning to coordinate robots at independent states bordering states where rewards depend on the state of multiple robots, *i.e.* it is too late to coordinate robots if they have already crashed. Melo and Veloso [119][120] developed a Q-learning algorithm that builds on the work of Spaan and Melo [165] to learn where coordination is necessary. A "coordinate" action is added to the set of actions available to each robot. When the coordination action is taken, the robot chooses an action based on its current state, and the position of the nearest neighbor robot. Robots will typically learn to take coordinate actions at bottlenecks, where robot-robot interactions are likely. Kok et al. [95][94] presented an approach which performs Q-learning for each robot independently, but stores statistics for the reward of the joint actions that are explored. If these statistics indicate that coordinating actions at a specific location is beneficial, then the algorithm starts learning coordinated actions at that state. This approach has the benefit of being able to handle tasks besides basic path planning, such as capturing targets that required coordinated action by multiple pursers. De Hauwere et al. [48] [49, 50] learned the states relative to a robot that necessitate coordination, *i.e.* a robot may need to coordinate with a robot directly in front of it, but not with a robot far to the rear. The approach of Bnaya et al. [23] computes all paths that a robot would take if no other robots were present. A randomly drawn set of paths are drawn for each robot, and checked for interference. A cost penalty is assessed on moving through states at which robots may interfere with one another. Optimal paths for each robot are then computed subject to the penalty terms, with final collision avoidance provided by a reactive controller.

1.2.3 Rule-Based Path Planning

Rule-based approaches are centralized approaches which use a set of stereotyped behaviors to govern robot-robot interactions during planning. The plans computed by rule-based approaches specify the motion of the entire system and are guaranteed to be collision free. Push and Swap [116, 100], Push and Rotate [51], the Tree-Based Agent Swapping Strategy algorithm [89], and the work of Auletta et al. [7] utilize behaviors that exchange the positions of two robots without disturbing any other robot. Surynek [174, 175] developed algorithms based on the theory of bi-connected graphs that use behaviors similar to Push and Rotate. Warehousing approaches shift robots into configurations which will not interfere with the motion of other robots [40, 133, 197], then plan for a small number of robots at a time.

The rule-based algorithms described in the previous paragraph are guaranteed to find a solution in polynomial time, but the quality of the path is typically low. In particular, the above methods only allow a single robot to move at any given time, which results in very long execution times. Parallel Push and Swap is a variant of Push and Swap which permits simultaneous motion of multiple agents, significantly decreasing path costs [151].

1.2.4 Coupled Planning

Coupled MPP algorithms treat the robots in a multirobot system as components of a single *meta-agent* whose configuration space is the joint configuration space. The MPP problem is then solved by planning a path for the meta-agent using a single robot path planning algorithm. By exploring the joint configuration space, coupled approaches can offer completeness and optimality guarantees, at the expense of high computational cost.

A* [77] could be used to search the joint configuration space, resulting in a simple, coupled planner. However, the exponential growth of the joint configuration space as the number of robots increases quickly renders planning paths with A* computationally infeasible. Iterative Deepening A* (IDA*) reduces the memory consumption of A* by using depth-first search [46, 96]. However, it is only effective when robots are packed densely enough that only a few robots can move at any time. One basic problem with A* based approaches is the presence of many redundant actions which A* will instantiate, but never actually use, such as actions where every robot moves directly away from its goal. *Operator Decomposition* (OD) [167] and *Enhanced Partial Expansion A** (EPEA*) [62, 71] are lazy variants of A* designed for MPP which delay instantiating actions which are heuristically expensive, and thus unlikely to be used as part of the optimal solution. Such lazy evaluation dramatically reduces the effective branching factor of the multirobot system, significantly reducing computational cost of finding a path.

Probabilistic planners were developed to find paths for robot mechanisms with many internal degrees of freedom, for which deterministic planners such as A* were unable to find paths in a reasonable amount of time. The suitability of probabilistic planners for high-dimensional planning has led to the development of coupled algorithms that use probabilistic planners to explore the joint configuration space of multirobot systems [35, 63, 103, 152, 153]. However, the structure and pure size of the joint configuration space of multiple robot systems limits such approaches to relatively small numbers of robots; to the best of our knowledge, the largest problem solved by running a probabilistic planner directly in the joint configuration space of a multirobot system involved 10 robots [63, 34].

An alternate approach to coupled planning is to recast the MPP problem as a *Boolean Satisfiability* (SAT) problem. The SAT problem is to find an assignment of truth values to variables that satisfy a logical formula. The MPP problem can be recast as a SAT problem by creating a set of Boolean variables to track the location of each robot, and adding terms to the Boolean formula to enforce collision avoidance [58, 81, 87, 176, 177]. SAT planners have also been used to find shortcuts to reduce the cost of non-optimal paths computed by rule-based planners [14, 175].

Similarly, the permutation invariant multirobot path planning problem can be reformulated as a network flow problem. Yu and Lavalle [204] showed that such a transformation allows for optimal paths to be found in polynomial time. By composing the workspace into many largely independent cells, the network flow approach can be applied to systems of 1,000,000 robots [86]

1.2.5 Decoupled Planning

Where a coupled planner searches the joint configuration space of a multirobot system, decoupled algorithms explore one or more low dimensional search spaces. Decoupled planners can quickly find paths for systems containing many robots. The drawback of decoupled algorithms is that the search spaces employed by decoupled planners represent only a small portion of the joint configuration space, and thus decoupled algorithms are not guaranteed to find a path for all solvable problems [153]. There are two primary classes of decoupled planners, velocity schedulers and priority planners.

Velocity Scheduling

Velocity scheduling approaches first plan a path for each robot, and then construct a coordination space, which describes the position of each robot along its path. The velocity scheduling approach then seeks a path in the coordination space, *i.e.* a velocity schedule, that moves each robot to its goal without robot-robot collisions [45, 84, 108, 134, 160]. Krishna et al. [99] introduced a decentralized approach where a robot coordinates its velocity schedule with a limited number of neighboring robots. Lavalle and Hutchinson [105] developed a hybrid between velocity scheduling and coupled planning by restricting each robot not to a single path, but rather to a sparse roadmap.

Priority Planning

Priority planners assign each robot a priority, and then plan for each robot in order of decreasing priority, treating higher priority robots as moving obstacles [60]. Priority planning can be readily decentralized, with low priority robots giving way to high priority robots [33, 54, 144, 192]. Saha and Isto [150] attempted to address the issue that low priority robots cannot influence the path of high priority robots by hybridizing priority planners and velocity schedules. They allowed a low priority robot to change the velocity schedule of higher priority robots, but not the path in the workspace that the higher priority robot takes.

A key element to the success of a priority planner is the choice of priority ordering; a high priority robot whose goal is in a bottleneck of the environment can easily prevent a solution from being found. van den Berg and Overmars [186] assigned a priority to robots based on the intial distance the robot is from its goal, with a longer distance to the goal corresponding to a higher priority. Turpin et al. [182, 183] showed that for permutation invariant multirobot path planning there is a priority ordering that is guaranteed to produce a solution. Assign each robot to a goal such that the sum-squared distance traveled is minimized. Then assign priority to robots in decreasing order of distance to the goal. The resulting algorithm can be shown to be complete and run in polynomial time.

Other approaches to priority planning rely on searching the possible priority orderings [12, 159]. One commonly used heuristic to guide search over priority orders is to examine the position of the initial and goal configurations of the robots [30, 117, 188]. If the goal configuration of robot r^1 lies on or blocks the path of r^2, then r^2 is assigned a higher priority than r^1. Conversely, if the initial configuration of r^1 lies on or blocks the path of r^2, then r^1 is assigned a higher priority than r^2. These relations may introduce cycles, which different approaches break in different manners, including random search in priority ordering [117].

1.2.6 Dynamically Coupled Planning

Dynamically coupled planning is an alternative to coupled or decoupled algorithms which grow the search space during planning, so that the search space can initially be very small, then grow only where necessary. In the worst case, the search spaces constructed by dynamically coupled algorithms may cover the entire joint configuration space, but for most problems a substantially smaller search space suffices. Subdimensional expansion is a dynamically coupled planner

Al-Wahedi presented an approach in which paths are found separately for each robot, followed by coupled planning in a window around conflicts, but said approach does not return optimal paths [1]. The work of van den Berg et al. [188] shows how to identify the minimal sets of robots which must execute a cooperative path instead of sequentially executing single robot paths. The dynamic networks of Clark et al. [41] couple online planning during execution for sets of robots capable of mutual communication. The Increasing Cost Search Tree (ICST) [3, 155, 154, 158] limits the cost that can be incurred by an individual robot, then uses pairwise tests to determine for which robots the cost limits must be raised. *Independence Detection* (ID) [167] and *Meta-Agent Conflict Based Search* (MA-CBS) [157] initially attempt to find a path using decoupled planning approaches, but revert to coupled planning for subsets of robots for which the decoupled planner cannot find optimal paths. Standley and Korf [168] introduced a variant of ID that reverts to prioritized planning when the coupled subsets of robots get too big, at the cost of optimality. Calliess and Roberts [32] proposed a method that is very similar to MA-CBS, but based on Mixed-Integer

Programming rather than graph search.

1.2.7 Miscellaneous

There are several interesting approaches that do not cleanly fit into the categories discussed so far. Bhattacharya et al. [22] developed an approach that can find optimal solutions to problems with complex inter-robot constraints, by iteratively replanning for each robot in turn. Initially, the inter-robot constraints are ignored. At the beginning of each iteration, the weight given to inter-robot constraints is increased slightly. The resulting paths can be shown to converge to the optimal path. Ghrist and Koditschek [70] showed that the free joint configuration space of a system of two robots at a Y-intersection had a simple geometry, that of a punctured disk with six triangular fins, which allows for easy planning. Multirobot coordination can be simplified by annotating each point in the workspace with a preferred direction; moving in the preferred direction is cheaper than moving against the preferred direction [82, 130]. Finally, Kloder and Hutchinson [90] developed a clever method of solving the permutation invariant multirobot path planning problem without ever explicitly assigning robots to goal locations by representing the position of the robots and the goal as the complex valued roots of polynomials. A path in polynomial space between two configurations can be computed by interpolating the coefficients of the polynomials that represent the initial and final configurations. However, extracting a workspace trajectory for a given robot requires repeatedly solving a track assignment problem, which makes this approach significantly less practical, and planning in the presence of obstacles is difficult.

Chapter 2

Subdimensional Expansion

In MPP there is an inherent trade-off between path quality and the computational cost of finding a path. However, in many problem instances of interest, the MPP problem naturally decomposes into small subproblems, which permits optimal paths[1] to be found at low computational cost. Specifically, if the interactions between robots are sparse, the MPP problem can be split into two parts: planning paths for individual robots and optimally resolving conflicts between robots.

Subdimensional expansion is a framework for MPP that exploits the aforementioned natural decomposition to find optimal paths at low computational cost. Subdimensional expansion begins by computing an *individual policy* for each robot. The individual policy specifies the *individually optimal path* from each point in the free configuration space of a robot to its goal configuration, neglecting the presence of other robots. The path of the multirobot system induced by each robot obeying its individual policy is termed the *joint policy path*. Robot-robot collisions are likely to be present in the joint policy path.

Subdimensional expansion then uses the individual policies to guide the construc-

[1] An optimal path is a collision-free path which minimizes some cost function.

tion of a search space of variable dimensionality that is embedded in the joint configuration space of the system, in which to coordinate the motion of the multirobot system and resolve any conflicts. Subdimensional expansion makes the *optimistic assumption* that the joint policy path is collision free until there is evidence otherwise, and thus each robot is initially restricted to obeying its individual policy. The resulting search space is one-dimensional, as a point on in the search space is fully determined by how long the robots have executed their individual policies, and planning is fully decoupled, *i.e.* each robot follows an independently computed plan. An *underlying planner*, such as A*, is then employed to find an optimal path in the search space. When the underlying planner encounters a robot-robot collision, the involved robots are permitted to diverge from their individual policies, locally increasing the dimensionality of the search space. In the region of increased dimensionality, planning is conducted as a search over the joint actions of the robots involved in the collision, *i.e.* coupled planning for those robots.

Two constructs, the *backpropagation set* and the *collision set*, are employed to ensure that the search space is only expanded where and as much as necessary. Subdimensional expansion only expands the search space when the underlying planner finds a collision, but the optimal resolution of the collision may require the involved robots to diverge from their individual policies long before the collision would take place. This requires expanding the search space along all paths that the underlying planner has explored that lead to the collision. The backpropagation set of a point q in the search space is used to propagate information about a collision back along all paths that lead to q, and consists of the set of all points for which the underlying planner has considered q as a possible successor. If the underlying planner is A*, then when a vertex is expanded it is added to the backpropagation set of each of its *out-neighbors*, whereas if RRT is employed as the underlying planner the backpropagation set of a configuration contains its parent in the search tree.

Subdimensional expansion uses the collision set to aggregate information about collisions and to determine the local dimensionality of the search space. The collision set C of a given point q in the search space is the set of robots involved in a collision either at q or at successor of q on a path that has been explored by the underlying planner. If a configuration has not been visited by the underlying planner, its collision set is empty. The collision set is computed using the backpropagation set. If the collision set C_k of a configuration q_k changes, including the first time the underlying planner visits a configuration at which a robot-robot collision occurs, then the robots in C_k are added to the collision set of each point in the backpropagation set of q_k. In addition, if a new configuration q_l is added to the backpropagation set of q_k, then the robots in C_k are added to C_l. Note that the above rules imply that the collision set is a function of the current state of search, and the collision set of any given point in the search space will only grow as the search progresses.

Robots in the collision set are known to collide with other robots if restricted to their individually optimal paths, but there is no evidence that robots outside the collision set will collide while obeying their individual policies. Therefore to ensure that a collision-free path can be found, the search space must include any possible joint action for the robots in the collision set, while the robots not in the collision set obey their individual policies. The result is a local increase in the dimensionality of the search space, but the search space will likely still be of lower dimensionality than the joint configuration space in which the search space is embedded. Because the search space is embedded in the joint configuration space, each point in the search space fully defines the configuration of the system, regardless of the local dimensionality of the search space. A locally low dimensional search space just restricts which paths of the system will be explored by the underlying planner.

Although the search space constructed by subdimensional expansion is embedded in a high-dimensional space and is thus hard to visualize, the geometry of the search

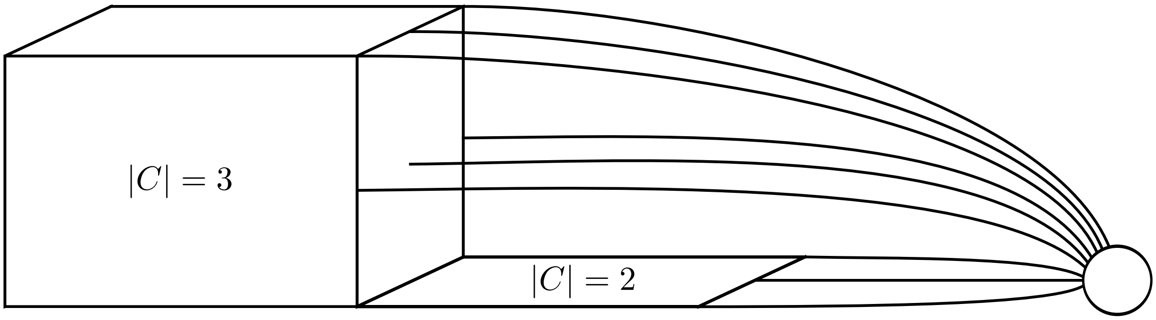


Figure 2.1: Geometric visualization of the search space as embedded in the joint configuration space. The circle represents the goal configuration. The cube represents a region of the search space in which the collision set contains three robots, while the square denotes a region where the collision set contains two robots. The lines denote the joint paths for the multirobot system induced by the individual policies, which connect configurations on the periphery of the higher-dimensional regions of the search space to the goal.

space can still be succinctly described, and provides an alternate way of understanding subdimensional expansion. The search space will have the appearance of a set of elongated "tubes" of decreasing dimensionality embedded in the joint configuration space, extending from the initial configuration towards the goal (Figure 2.1). Each tube grows around an explored path or set of paths that lead to a robot-robot collision, and thus the interior of each tube consists of states with non-empty collision sets. The surface of each tube are covered with one-dimensional "hairs" that extend towards the goal. Each hair is the joint policy path leading from a state on the surface of the tube with an empty collision set to the goal. The search space starts as a single hair, which thickens and branches as robot-robot collisions are found.

To better illustrate the workings of subdimensional expansion, we present an example for MPP on graphs. Planning is done using the M* algorithm, an implementation of subdimensional expansion that uses A* as the underlying planner. M* will be described in detail in section 3, but for the purposes of this example M* can be described as being equivalent to running A* on a small search graph which grows every time a robot-robot collision is found.

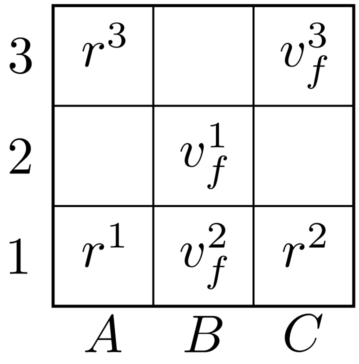
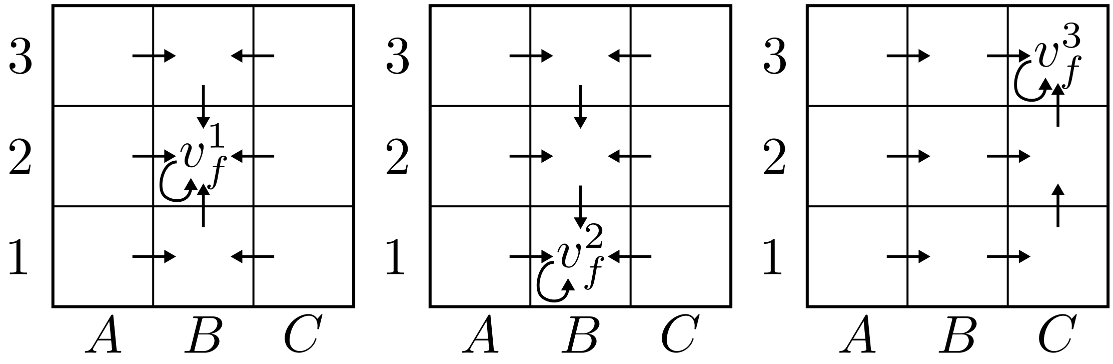


Figure 2.2: Example of the working of subdimensional expansion. Robots r^1, r^2, r^3 start at $A1, C1$, and $A3$ respectively, with goal configurations $B2, B1$ and $C3$.



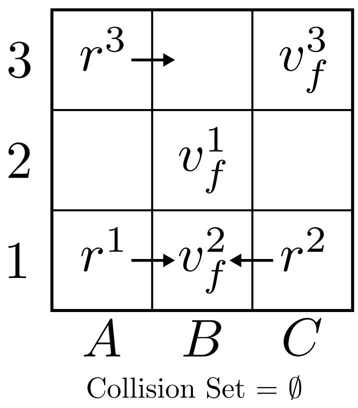
(a) Policy for r^1

(b) Policy for r^2

(c) Policy for r^3

Figure 2.3: Subdimensional expansion starts by computing a individual policy for each robot. The optimal action for a robot at each configuration is indicated by arrows. The loop at the goal state indicates that the robot should seek to remain at its goal.

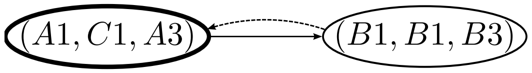
Neighbors of Expanded State
$(B1, B1, B3)$



Post Expansion Open List

Coordinate	f-value	Collision set
$(A1, C1, A3)$	5	$\{1, 2\}$

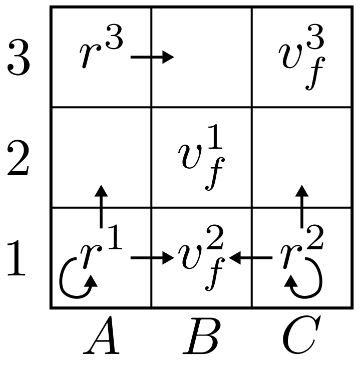
(a) Configuration at Step 1



(b) Search tree after expansion.

Figure 2.4: **(a)** Example of the workings of subdimensional expansion. The robots start at $(A1, C1, A3)$ and have the goal $(B2, A2, C3)$. The grid on the left shows the configuration that is expanded by M* in step one. The arrows show the actions that M* considers for each robot. The tables on the right enumerate the resulting neighboring configurations, and the state of the open list after the expansion and collision set update are completed.

(b) In the search tree solid arrows point from a vertex to its successor states while dashed lines point from a vertex to the elements of its backpropagation set. The vertex expanded in step one is bolded.



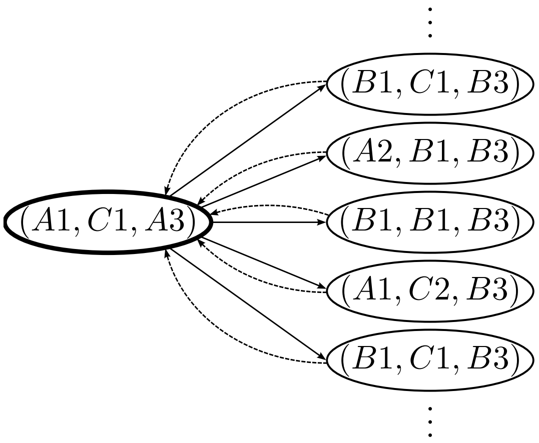
Collision Set = $\{1, 2\}$

Neighbors of Expanded State
$(B1, B1, B3)$,$(A2, B1, B3)$,$(A1, B1, B3)$
$(B1, C2, B3)$,$(A2, C2, B3)$,$(A1, C2, B3)$
$(B1, C1, B3)$,$(A2, C1, B3)$,$(A1, C1, B3)$

Post Expansion Open List

Coordinate	f-value	Collision set
$(A2, B1, B3)$	5	$\emptyset$
$(A1, B1, B3)$	6	$\emptyset$
$(B1, C1, B3)$	6	$\emptyset$
$(A2, C1, B3)$	6	$\emptyset$
$(B1, C2, B3)$	7	$\emptyset$
$(A2, C2, B3)$	7	$\emptyset$
$(A1, C1, B3)$	7	$\emptyset$
$(A1, C2, B3)$	8	$\emptyset$

(a) Configuration at Step 2

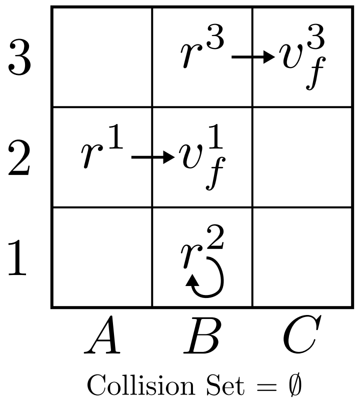


(b) Search tree after expansion.

Figure 2.5: **(a)** Example of the workings of subdimensional expansion. The robots start at $(A1, C1, A3)$ and have the goal $(B2, A2, C3)$. The grid on the left shows the configuration that is expanded by M* in step two. The arrows show the actions that M* considers for each robot. The tables on the right enumerate the resulting neighboring configurations, and the state of the open list after the expansion and collision set update are completed.

(b) In the search tree solid arrows point from a vertex to its successor states while dashed lines point from a vertex to the elements of its backpropagation set. The vertex expanded in step two is bolded.

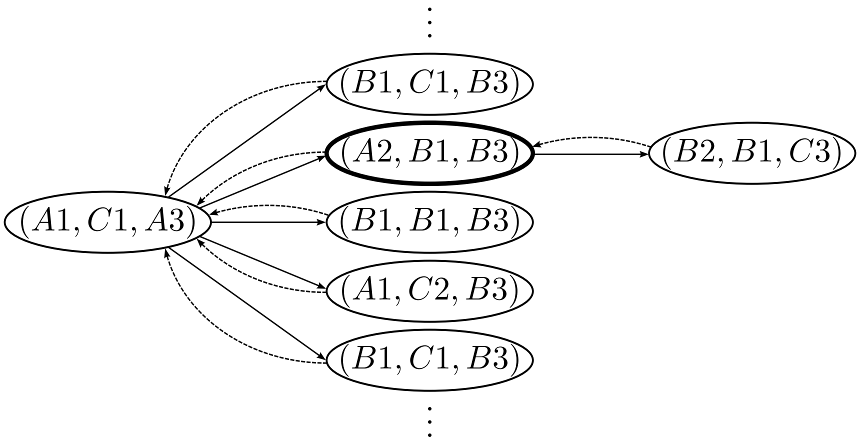
Neighbors of Expanded State
$(A2, B1, C3)$



Post Expansion Open List

Coordinate	f-value	Collision set
$(B2, B1, C3)$	5	$\emptyset$
$(A1, B1, B3)$	6	$\emptyset$
$(B1, C1, B3)$	6	$\emptyset$
$(A2, C1, B3)$	6	$\emptyset$
$(B1, C2, B3)$	7	$\emptyset$
$(A2, C2, B3)$	7	$\emptyset$
$(A1, C1, B3)$	7	$\emptyset$
$(A1, C2, B3)$	8	$\emptyset$

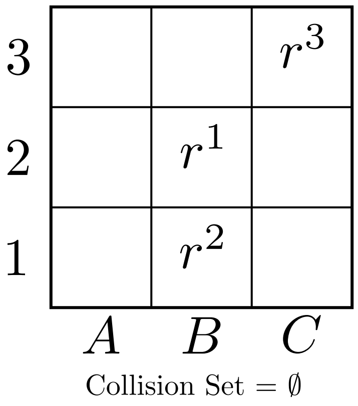
(a) Configuration at Step 3



(b) Search tree after expansion.

Figure 2.6: **(a)** Example of the workings of subdimensional expansion. The robots start at $(A1, C1, A3)$ and have the goal $(B2, A2, C3)$. The grid on the left shows the configuration that is expanded by M* in step three. The arrows show the actions that M* considers for each robot. The tables on the right enumerate the resulting neighboring configurations, and the state of the open list after the expansion and collision set update are completed.

(b) In the search tree solid arrows point from a vertex to its successor states while dashed lines point from a vertex to the elements of its backpropagation set. The vertex expanded in step three bolded.

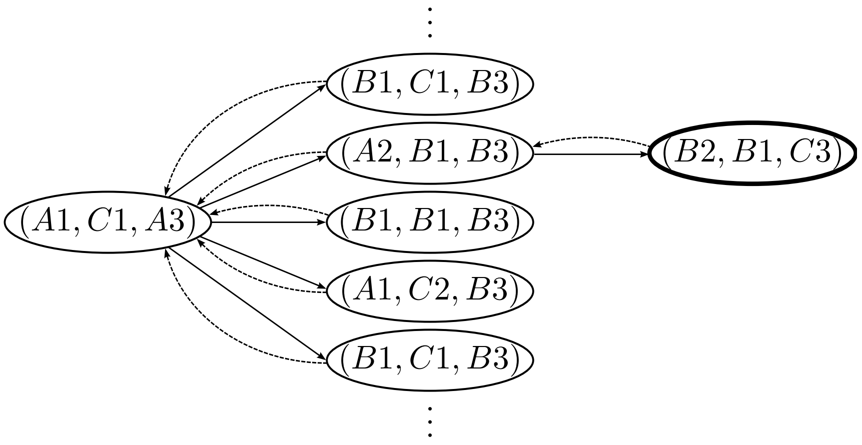


Neighbors of Expanded State

Post Expansion Open List

Coordinate	f-value	Collision set
$(A1, B1, B3)$	6	$\emptyset$
$(B1, C1, B3)$	6	$\emptyset$
$(A2, C1, B3)$	6	$\emptyset$
$(B1, C2, B3)$	7	$\emptyset$
$(A2, C2, B3)$	7	$\emptyset$
$(A1, C1, B3)$	7	$\emptyset$
$(A1, C2, B3)$	8	$\emptyset$

(a) Configuration at Step 4



(b) Search tree after expansion.

Figure 2.7: **(a)** Example of the workings of subdimensional expansion. The robots start at $(A1, C1, A3)$ and have the goal $(B2, A2, C3)$. The grid on the left shows the configuration that is expanded by M* in step four. The arrows show the actions that M* considers for each robot. The tables on the right enumerate the resulting neighboring configurations, and the state of the open list after the expansion and collision set update are completed.

(b) In the search tree solid arrows point from a vertex to its successor states while dashed lines point from a vertex to the elements of its backpropagation set. The vertex expanded in step four is bolded.

Consider a system of three robots, r^1, r^2, and r^3, which move on a graph representing a four connected grid. The X coordinates of the graph are labeled with letters, while the Y coordinates are labeled with numbers. Robot r^1 starts at the initial configuration $v_s^1 = A1$ and has the goal $v_f^1 = B2$. Robots r^2 and r^3 have initial configurations $v_s^2 = C1$ and $v_s^3 = A3$, and goal configurations $v_f^2 = B1$ and $v_f^3 = C3$ respectively (Figure 2.2). The initial configuration of the multirobot system is denoted $(A1, C1, A3)$, while the goal configuration is $(B2, B1, C3)$ The robots incur a cost of 1 for any action, including remaining in place, but the robots can wait at their goal for zero cost.

Subdimensional expansion begins by computing an individual policy for each robot (Figure 2.3). The choice of policies is not unique. For instance, an alternate policy for r^1 would be to move up from $A1$ rather than right. Choice of individual policies is discussed in section 3.5.4.

Once the individual policies are computed, search for the multirobot system can commence. M* maintains an open list of candidate vertices which are explored in order of f-value, the sum of the cost to reach a vertex and a heuristic cost-to-go. When search begins the open list only contains the initial configuration, with an empty collision set (Figure 2.4). An empty collision set means that every robot obeys its individual policy. Therefore, when the initial configuration is expanded, there is only one neighbor, $(B1, B1, B3)$ (Figure 2.4a). At $(B1, B1, B3)$ robots r^1 and r^2 are in collision, which triggers a collision set update. The initial configuration is in the backpropagation set of $(B1, B1, B3)$, (Figure 2.4b), so r^1 and r^2 are added to the collision set of the initial configuration, which implicitly modifies the search graph. To allow the modified search graph to be explored, the initial configuration is added back to the open list (section 3.2).

In the second iteration of M*, the initial configuration is once more taken from the open list, and expanded (Figure 2.5). This time, r^1 and r^2 are in the collision set of the initial configuration, so only r^3 is restricted to its individual policy. As a result, the initial configuration now has nine neighbors (Figure 2.5a), including $(B1, B1, B3)$. The backpropagation set of each neighbor contains only the initial configuration, as the only paths that have been explored lead from the initial configuration to one of its neighbors (Figure 2.5b). The initial configuration has an empty backpropagation set, because no paths have been explored that lead to the initial configuration, and thus collisions at one of the neighbors cannot be propagated to the collision set of a different neighbor. The only robot-robot collision occurs at $(B1, B1, B3)$, and the involved robots have already been added to the collision set of the initial configuration, the only state in the backpropagation set of $(B1, B1, B3)$. Therefore, no further modification of the collision sets is required. The collision-free neighbors are then added to the open list and sorted by f-value.

In the third iteration, the most promising vertex is $(A2, B1, B3)$ (Figure 2.6). $(A2, B1, B3)$ was never previously expanded, and thus has an empty collision set, and therefore a single neighbor $(B2, B1, C3)$, the goal configuration. The goal configuration is collision free, and thus is added to open list. Note that in the counterfactual case that the neighbor of $(A2, B1, B3)$ had contained a robot-robot collision, the involved robots would be added to the collision sets of both $(A2, B1, B3)$ and $(A1, C1, A3)$.

In the fourth iteration, the goal configuration has the lowest f-value of any vertex in the open list, and is thus expanded (Figure 2.7), which indicates that the optimal path has been found.

Chapter 3

M*

A key detail of subdimensional expansion is how the search space is constructed, an issue which depends upon the planner used by a given implementation. We will now describe how subdimensional expansion can be implemented for planning paths for multirobot systems moving in spaces represented by graphs. A* [77] is an attractive planner when the configuration space of each robot can be represented by a graph. A* is optimal, meaning it finds optimal paths, and *complete*, meaning that it will take finite time to either find a path or determine that no path exists. In this section, we present the M* algorithm, a complete and optimal implementation of subdimensional expansion which uses A* as the underlying planner.

3.1 Problem Definition

Consider a system of n robots r^i indexed by the set $I = \{1, \ldots, n\}$. Let the free configuration space of r^i be represented by the directed *configuration graph* $G^i = \{V^i, E^i\}$. V^i is the set of vertices in G^i, each of which represents a configuration of r^i. E^i is the set of directed edges, each of which represents an action that transitions r^i from one configuration to another. Each edge is associated with a positive cost.

Symbol	Meaning
r^i	ith robot
G^i	Configuration graph representing configuration space of r^i
v^i	Vertex in G^i representing a configuration of r^i
v^i_s	Initial configuration of r^i
v^i_f	Goal configuration of r^i
G	Joint configuration graph representing joint configuration space of system
v	Vertex in G representing a configuration of the multirobot system
v_s	Initial configuration of multirobot system
v_f	Goal configuration of multirobot system
v^i_k	Configuration of r^i at joint configuration specified by v_k
$\pi(v_k, v_\ell)$	Path for the multirobot system connecting v_k to v_ℓ
$g(\pi(.))$	Cost of specified path
$\Psi(v_k)$	Set of robots that collide at v_k

Table 3.1: Symbol definitions for multirobot path planning on graphs

Each robot has an initial configuration $v^i_s \in V^i$ and a goal configuration $v^i_f \in V^i$.

The joint configuration space which describes the state of the entire multirobot system is represented by the *joint configuration graph*, which is the direct product of the individual robot configuration graphs $G = G^1 \times \cdots \times G^n$, with vertex set V and edge set E. Recall that the direct product of two graphs, $G^i \times G^j$, has the vertex set $V^i \times V^j$. Two vertices (v^i_k, v^j_k) and (v^i_ℓ, v^j_ℓ) in $V^i \times V^j$ are connected by an edge in the product graph if the edge $e^i_{k\ell}$ connecting v^i_k to v^i_ℓ is present in E^i and the edge $e^j_{k\ell}$ connecting v^j_k to v^j_ℓ is present in E^j. Note that G may contain vertices at which robots collide.

Let Π^i denote the set of all valid paths in G^i, where a valid path consists of a sequence of vertices such that each vertex in the sequence is an out-neighbor of its predecessor in G^i. $\Pi = \Pi^1 \times \cdots \times \Pi^n$ denotes the set of all paths in G. Let $\pi^i(v_k, v_\ell)$ denote a path in G^i from v_k to v_ℓ. The cost of a single robot path $g^i : \Pi^i \to \mathbb{R}^+$ is the sum of the costs of the edges traversed in the path. The cost $g : \Pi \to \mathbb{R}^+$ of a path $\pi(v_k, v_\ell)$ in G is the sum of the costs of the corresponding single robot paths $\pi^i(v_k^i, v_\ell^i)$, where v_k^i is the position of r^i at the joint configuration v_k,

$$g(\pi(v_k, v_\ell)) = \sum_{i \in I} g^i(\pi^i(v_k^i, v_\ell^i)). \tag{3.1}$$

The task of M* is to find an optimal, collision-free path from the joint initial configuration $v_s = v_s^1 \times \cdots \times v_s^n$ to the joint goal configuration $v_f = v_f^1 \times \cdots \times v_f^n$, denoted $\pi_*(v_s, v_f)$. To determine where robots collide with one another, we define a *collision function* $\Psi : V \to \mathcal{P}(I)$ which returns the set of robots in collision at a given vertex, with $\mathcal{P}(I)$ denoting the power set of I which contains all subsets of I. What constitutes a collision depends on the problem being solved, and may represent a physical collision, a contention for a shared resource, or some other conflict. Note that $\Psi(v_k)$ describes the robots which are locally in collision at v_k, whereas C_k collects all collisions occurring at a successor of v_k on some path explored by the underlying A* planner, thus $\Psi(v_k) \subseteq C_k$. For the purpose of description, only collisions at vertices will be considered, as collisions taking place during the traversal of edges can be modeled by inserting additional vertices into the joint configuration graph.

The notation in this thesis can get complex, due to the number of different objects that the text must describe, and the number of different spaces in which said objects may lie. To make the notation more comprehensible, a standard format is employed. The symbol x_z^y refers to an object of type x, where z is a label for the specific object instance, and $y \subset I$ is robot or set of robots which are described by x. For instance, v_k^i refers to a vertex k describing the configuration of robot r^i. The symbols x_k^i and x_k^j refer to the components of x_k describing robots i and j respectively. The symbols i, j, k and ℓ are reserved for short term indexing, and are reused throughout the thesis

Symbol	Meaning
ϕ^i	Individual policy for r^i
$\pi^i_\phi(v^i, v^i_f)$	Path for r^i induced by its individual policy from v^i to v^i_f
$\pi_\phi(v, v_f)$	Path the for multirobot system induced by each robot obeying its individual policy from v to v_f
$\pi_*(v_k, v_\ell)$	Minimal cost, collision-free path connecting v_k to v_ℓ
C_k	Collision set at v_k
V_k^{nbh}	Limited neighbors of v_k
$h(v_k)$	Heuristic cost-to-go from v_k to v_f

Table 3.2: Symbol definitions for M*

in different contexts. The definitions of symbols used in the problem definition are summarized in Table 3.1.

3.2 Algorithmic Description

M* is broadly similar to A* [77] in implementation. The primary difference is that M* restricts the set of possible successors of a vertex based on the collision set. Only robots in the collision set are allowed to consider any possible action; all other robots must obey their individual policies (Figures 2.4-2.7). A more detailed description follows.

M* is most easily described as a set of modifications to A*. Recall that A* maintains an *open list* of vertices v_k to explore. Each vertex represents one point in the joint configuration space of the multirobot system, specifying the configuration of every robot. These are sorted by *f-value*, which is the sum of a *g-value* and a *heuristic cost.* The g-value is the cost of the cheapest path to v_k found thus far, and is therefore an upper bound on $g(\pi_*(v_s, v_k))$. The heuristic cost, $h(v_k)$, is a lower bound on the cost of the optimal path from v_k to the goal, *i.e.* $h(v_k) \leq g(\pi_*(v_k, v_f))$. At each iteration, the vertex v_k with the smallest f-value in the open list is *expanded.* Each neighbor v_ℓ of v_k is added to the open list if the path reaching v_ℓ via v_k is cheaper than the current g-value of v_ℓ. The process continues until the goal vertex v_f is expanded, which indicates that an optimal path to the goal has been found for the multirobot system.

Prior to planning for the multirobot system, M* computes the individual policies $\phi^i : V^i \to V^i$ for each robot, where $\phi^i(v^i)$ is the successor of v^i along the minimal cost path to v^i_f for robot r^i, ignoring robot-robot interactions. ϕ^i can be efficiently computed by Reverse Resumable A* [159]. The path induced by ϕ^i from v^i is denoted $\pi^i_\phi(v^i, v^i_f)$. The *joint policy* $\phi : V \to V$ moves each individual robot along its individual policy, with the joint policy path induced by ϕ from v denoted $\pi_\phi(v, v_f)$. Computing the individual policies permits the efficient computation of the highly informative Sum of Individual Costs (SIC) heuristic, which is commonly employed for multirobot path planning [62, 96, 167]. The SIC heuristic evaluated at v_k is the sum of the costs of the individually optimal paths of all robots

$$h(v_k) = g(\pi_\phi(v_k, v_f)) \leq g(\pi_*(v_k, v_f)). \tag{3.2}$$

The primary difference in implementation between M* and A* lies in the expansion step: while A* considers all neighbors of a vertex v_k for addition to the open list, M* only considers a subset of the neighbors of v_k, denoted the *limited neighbors*. The limited neighbors V^{nbh}_k are the set of neighbors of v_k which can be reached from v_k when each robot not in the collision set C_k of v_k moves according to its individual policy. A robot in the collision set of v_k is allowed to move to any neighboring state in the robots configuration graph G^i. More formally, the limited neighbors V^{nbh}_k are the set of neighbors v_ℓ of v_k such that the i'th component of v_ℓ satisfies one of two properties: i) if $i \in C_k$ then v^i_ℓ is an out-neighbor of v^i_k, or ii) if $i \notin C_k$ then v^i_ℓ is the individually optimal successor of v^i_k according to ϕ^i. If there is a robot-robot collision

Algorithm 1 Pseudocode for collision set backpropagation

```
Require: $v_k$, $C_\ell$, open
  {$v_k$- vertex in the backpropagation set of $v_\ell$}
  {$C_\ell$- the collision set of $v_\ell$}
  {open- the open list for M*}
  if $C_\ell \nsubseteq C_k$ then
    $C_k \leftarrow C_k \bigcup C_\ell$
    if $\neg(v_k \in$ open$)$ then
      open.insert($v_k$) {If the collision set changed, $v_k$ must be re-expanded}
    for $v_m \in v_k$.back_set do
      backprop($v_m, C_k$,open)
```

at v_k then $V_k^{\text{nbh}} = \emptyset$ to prevent paths from passing through collisions.

$$V_k^{\text{nbh}} = \left\{ v_\ell \,\middle|\, \begin{cases} e_{k\ell}^i \in E^i, & i \in C_k \\ v_\ell^i = \phi^i(v_k^i), & i \notin C_k \end{cases} \right\} \tag{3.3}$$

The collision sets of each vertex must be updated whenever M* finds a new path to a robot-robot collision. To this end, M* maintains a backpropagation set for each vertex v_k, which is the set of all vertices v_ℓ that were expanded while v_k was an element of V_ℓ^{nbh}. The backpropagation set is thus the set of neighbors of v_k through which the planner has explored a path to v_k. M* propagates information about a collision at v_k by adding the robots in $\Psi(v_k)$ to the collision set of each vertex v_ℓ in the backpropagation set of v_k. The robots in C_ℓ are then added to the collision set of each vertex in the backpropagation set of v_ℓ, with the process repeating recursively until a vertex v_m is reached with $\Psi(v_k) \subseteq C_m$. Because V_ℓ^{nbh} is dependent on C_ℓ, changing C_ℓ adds new paths through v_ℓ to the search space. To allow these new paths to be explored, v_ℓ is added to the open list (Algorithm 1). Pseudocode for M* is provided in Algorithm 2.

Algorithm 2 Pseudocode for M*

```
{Define default values for vertices}
for all $v_k \in V$ do
    $v_k$.cost ← MAXCOST
    $v_k$.back_set ← $\emptyset$
    $C_k \leftarrow \emptyset$
{Initialize search}
$v_s$.cost ← 0
open ← $\{v_s\}$
while open.empty() == False do
    $v_k$ ← open.pop() {Get cheapest vertex}
    if $v_k = v_f$ then
        {A solution has been found}
        return back_track($v_k$) {Reconstruct the optimal path by following the back pointers}
    for $v_\ell \in V_k^{\text{nbh}}$ do
        $v_\ell$.back_set.append($v_k$) {Add $v_k$ to the back propagation list}
        $C_\ell \leftarrow C_\ell \bigcup \Psi(v_\ell)$
        {Update collision sets, and add vertices whose collision set changed back to open}
        backprop($v_k$, $C_\ell$,open)
        if $\Psi(v_\ell) = \emptyset$ and $v_k$.cost+$f(e_{k\ell})$ < $v_\ell$.cost then
            {$v_k$ is the cheapest route to $v_\ell$}
            $v_\ell$.cost ← $v_k$.cost+$f(e_{k\ell})$
            $v_\ell$.back_ptr ← $v_k$ {Track the best path to $v_\ell$}
            open.insert($v_\ell$)
return No path exists
```

3.3 Completeness and Cost Optimality

In this section, M* will be shown to be both complete and optimal. The description of M* given in 3.2 is well suited to implementation, but provides only a local description of the operation of M*, which is not optimal for proving global properties. In the following subsection, a global description of M* is provided which is more suited to proving properties of the M* algorithm, with a focus on the search space that is constructed by M*. M* will be shown to be equivalent to alternating between running A* on a search graph, and expanding the search graph based on collisions found by A*. As a result, demonstrating that the construction of the search graph takes finite

Symbol	Name	Meaning	A* equivalent
G	Joint configuration graph	Joint configuration space	
G^{sch}	Search graph	Current search graph	Graph that is being searched
G^{exp}	Explored graph	Explicitly constructed by M*	Vertices in the open list
G^{nbh}	Neighbor graph	G^{exp} plus limited neighbors	Vertices in the open list plus their out-neighbors
G^{ϕ}	Policy graph	Individually optimal paths starting from $G^{\text{nbh}} \setminus G^{\text{exp}}$	

Table 3.3: Search graph symbols

time and that the search graph will eventually contain the optimal path, if extant, is sufficient to prove that M* is complete and optimal.

3.3.1 Alternative Graph-Centric Description

M* differs from A* solely in the use of the limited neighbors when expanding a vertex and the presence of the *backprop* function (Algorithm 1). The backprop function does nothing unless a new path to a collision is found. Therefore, between discoveries of new paths to collisions, M* behaves exactly like A* running on a *search graph* G^{sch} which is a subgraph of the joint configuration graph G that represents the joint configuration space.

The search graph G^{sch} consists of three subgraphs: the *explored graph* G^{exp}, the *neighbor graph* G^{nbh}, and the *policy graph* G^{ϕ} (Table 3.3). G^{exp} is the portion of G which has been searched by M*, G^{nbh} represents the limited neighbors of the vertices in G^{exp}, and G^{ϕ} consists of the paths induced by ϕ that connect vertices in G^{nbh} to v_f. Only G^{exp} is explicitly constructed, with G^{nbh} and G^{ϕ} being implicitly defined by

G^{exp} and the collision sets of the vertices in G^{exp}.

We now describe the explored graph G^{exp}, neighbor graph G^{nbh}, and policy graph G^{ϕ} in greater detail. The vertex set of G^{exp} consists of all vertices which have been added to the open list. When a vertex $v_k \in G^{\text{exp}}$ is expanded, its limited neighbors V_k^{nbh} are added to the open list, and thus to the vertex set of G^{exp}. The edges connecting v_k to each of its limited neighbors are added to the edge set of G^{exp}.

The collision set of a vertex is a function of the paths that have been explored by the underlying planner. G^{exp} contains all such paths, and therefore encodes all the information required to compute the collision set of any vertex v_k.

$$C_k = \begin{cases} \Psi(v_k) \bigcup\limits_{v_\ell \in V_k} \Psi(v_\ell) & v_k \in G^{\text{exp}} \\ \emptyset & v_k \notin G^{\text{exp}} \end{cases} \tag{3.4}$$

where $V_k = \{v_\ell \mid \exists \pi(v_k, v_\ell) \subseteq G^{\text{exp}}\}$ is the set of vertices to which there exists a path from v_k in G^{exp}. If $v_k \notin G^{\text{exp}}$, then M* has never visited v_k, and thus v_k has not been explicitly constructed and thus $\Psi(v_k)$ has not yet been computed. In accordance to the optimistic assumption, v_k is assumed to be collision-free, and C_k is initialized as the empty set. Therefore, a path in G^{sch} may contain a vertex v_k in $G^{\text{sch}} \setminus G^{\text{exp}}$ at which robots collide. However, v_k must be added to the open list, and thus to G^{exp}, before any such path could be returned. At that point, $\Psi(v_k)$ would be computed, leading to the out-neighbors of v_k being removed from G^{sch}, as per the definition of the limited neighbors.

The neighbor graph G_k^{nbh} is the subgraph of the joint configuration graph G^{sch} that represents the limited neighbors of $v_k \in G^{\text{exp}}$. G_k^{nbh} contains v_k, V_k^{nbh}, and the edges leading from v_k to the vertices in V_k^{nbh}. Let $G^{\text{nbh}} = \bigcup_{v_k \in G^{\text{exp}}} G_k^{\text{nbh}}$, and therefore $G^{\text{exp}} \subset G^{\text{nbh}}$.

Because $C_k = \emptyset$ for all v_k which are not in the explored graph G^{exp}, search from

$v_k \in G^{\mathrm{nbh}} \setminus G^{\mathrm{exp}}$ will proceed along $\pi_\phi(v_k, v_f)$ until either v_f or a vertex in G^{exp}[1] is reached. The resulting path segment is denoted $\pi_\phi(v_k)$, and is represented as a subgraph G^ϕ_k, whose vertex set is the set of vertices in $\pi_\phi(v_k)$, and whose edge set contains each edge connecting a vertex in $\pi_\phi(v_k)$ to its successor. Let the policy graph be defined as $G^\phi = \bigcup_{v_k \in G^{\mathrm{nbh}} \setminus G^{\mathrm{exp}}} G^\phi_k$.

G^{sch} can now be defined as the union of G^{exp}, the subgraph explored by M*; G^{nbh}, the limited neighbors of vertices in G^{exp},;and G^ϕ, the individually optimal paths connecting vertices in $G^{\mathrm{nbh}} \setminus G^{\mathrm{exp}}$ to v_f. By the definitions of G^{exp}, G^{nbh} and G^ϕ, vertices and edges shift from G^ϕ to G^{nbh}, and from G^{nbh} to G^{exp} as search progresses. However, G^{sch} as a whole only changes when the collision set of a vertex in G^{sch} changes. See Figure 3.1 for an illustration of how the subgraphs change over time.

3.3.2 Proof of Optimality and Completeness

As demonstrated in the previous section, M* can be treated as alternating between exploring the search graph G^{sch} with A* and modifying G^{sch} based on the partial search results. Because A* is complete and optimal [77], M* is complete and optimal if G^{sch} will contain $\pi_*(v_s, v_f)$ and no cheaper path after a finite number of modifications or, if $\pi_*(v_s, v_f)$ does not exist, G^{sch} will be modified at most a finite number of times.

We proceed by showing that if no solution exists, M* will terminate in finite time without returning a path. We then show that M* will eventually find the optimal path if one of two conditions always hold: G^{sch} contains the optimal path, or G^{sch} contains an unexplored path containing a robot-robot collision which costs no more than the optimal path. We complete the proof by showing that at least one of the two conditions always holds.

[1] If $\pi_\phi(v_k, v_f)$ encounters a vertex in the explored graph G^{exp}, then there may be some $v_\ell \in \pi_\phi(v_k, v_f)$ such that $\Psi(v_\ell) \neq \emptyset$, with $v_\ell \in G^{\mathrm{exp}}$. In such a case, $\pi_\phi(v_k, v_f)$ is not wholly within G^{sch}. For this reason, only the portion of $\pi_\phi(v_k, v_f)$ prior to reaching a vertex in G^{exp} is considered.

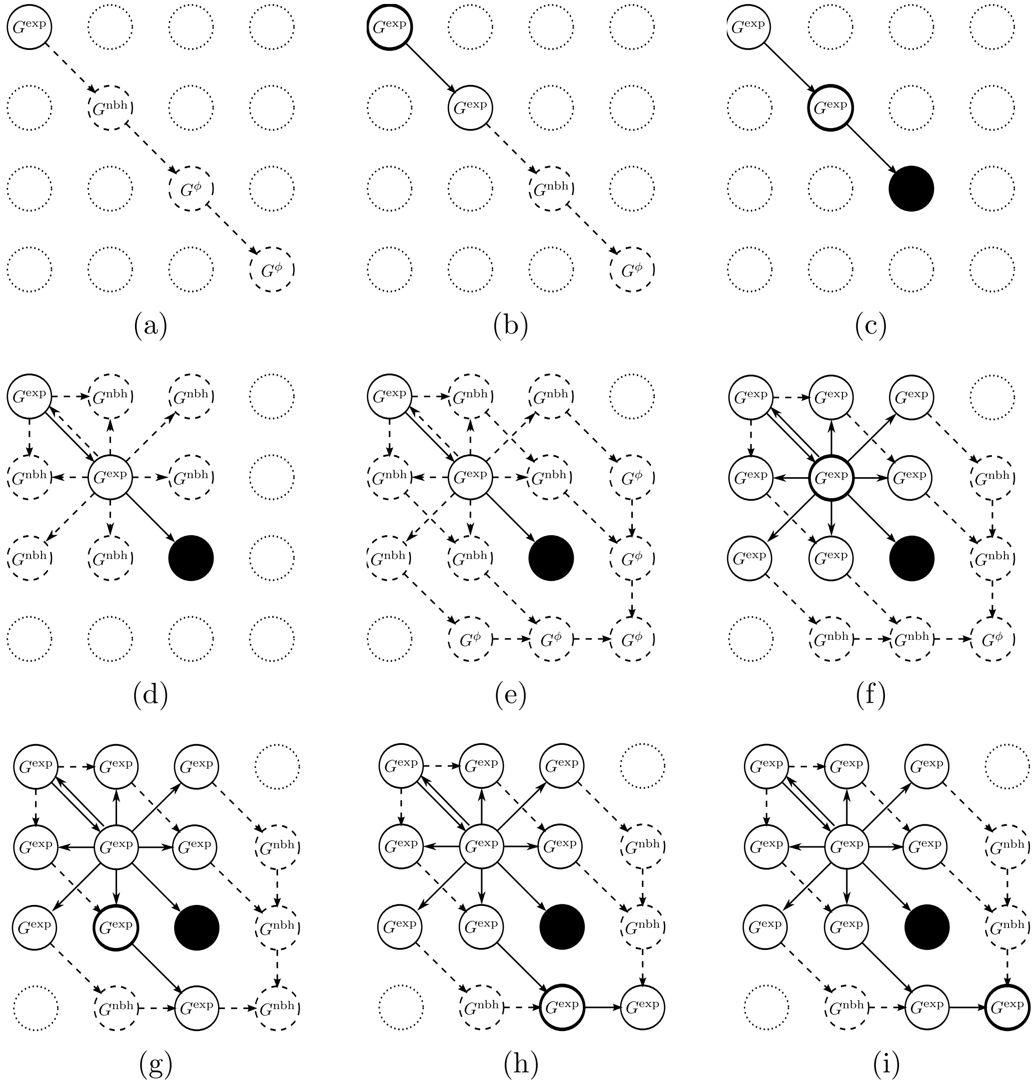


Figure 3.1: The above figures depict how the explored graph G^{exp} and the search graph G^{sch} evolve in the configuration space. Vertices are represented as circles, with arrows representing directed edges. G^{exp} is depicted by solid lines, while $G^{\text{sch}} \setminus G^{\text{exp}}$ is depicted by dashed lines. $G \setminus G^{\text{sch}}$ is represented by dotted lines, with edges suppressed for clarity. A vertex is given a bold outline when it is expanded, while filled circles represent vertices with known robot-robot collisions. v_s is in the upper left, while v_f is in the bottom right. In **(a)**, **(b)**, and **(c)**, the most promising vertex in the open list is expanded, until a collision is found. G^{nbh} is updated to reflect the new collision sets in **(d)**. The policy graph G^{ϕ} is updated in **(e)**. In **(f)** a vertex is re-expanded, having been added back to the open list when its collision set was changed. **(g)**, **(h)**, and **(i)** see the most promising vertices in the open list expanded, until v_f is expanded, indicating that a path has been found.

Lemma 1. *If no solution exists, M* will terminate in finite time without returning a path.*

Proof. Assume no solution exists. As part of M*, A* is run on the search graph G^{sch}. A* will explore all of G^{sch} in finite time and conclude that no solution exists, except if the A* search is interrupted by a modification of G^{sch}. G^{sch} is only modified when the collision set of at least one vertex in G^{sch} is changed. Each modification adds one or more robots to the collision set, and thus each collision set can be modified at most $n-1$ times; the first modification must add at least two robots. Therefore, G^{sch} can be modified at most $(n-1)*|V|$ times. Thus if no solution exists, M* will always terminate in finite time.

We now show that M* will never return an invalid path containing a robot-robot collision. A vertex v_k has out-neighbors only if it is collision free, unless v_k is not in the explored graph G^{exp}. Before M* will return a path passing through v_k, v_k must be added to the open list, and thus to G^{exp}. When v_k which is not collision free is added to the open list, G^{sch} is modified to remove all out-neighbors of v_k, which removes any path passing through v_k from G^{sch}. Therefore, M* will never return a path passing through a state at which robots collide. Thus, if no solution exists, M* will terminate in finite time without returning a path □. □

Next, assume that an optimal collision-free path from v_s to v_f exists, *i.e.* the joint configuration graph G contains an optimal path $\pi_*(v_s, v_f)$.

Lemma 2. *If an optimal path exists, M* will find the optimal path in finite time if one of two cases always hold*

Case 1: *The search graph G^{sch} contains an optimal path, $\pi_*(v_s, v_f)$*

Case 2: *The search graph G^{sch} contains a path $\pi(v_s, v_c)$ such that $g\left(\pi\left(v_s, v_c\right)\right) + h(v_c) \leq g(\pi_*(v_s, v_f))$, and $\exists\, v_b \in \pi(v_s, v_c)$ such that $\Psi(v_c) \not\subseteq C_b$*

Case 2 implies the existence of a path which has not been explored by M* that leads to a robot-robot collision at v_c, and which costs no more than $\pi_*(v_s, v_f)$. If the path had been explored, v_b and v_c would have been added to the open list and thus to the explored graph G^{exp}. In this case, C_b would include all robots involved in the collision at v_c, *i.e.* the robots in $\Psi(v_c)$.

To prove lemma 2, we proceed by showing that if case 1 holds, the optimal path will be found unless a cheaper path containing a collision exists in the search graph G^{sch}, *i.e.*, case 2 holds (Lemma 3). We then show that M* will never explore a suboptimal path to the goal as long as case 2 holds (Lemma 4), and that case 2 will not hold after finite time (Lemma 5). We conclude by proving that either case 1 or case 2 will always hold, demonstrating that the optimal path will be found (Lemma 7).

Lemma 3. *If the search graph G^{sch} contains an optimal path (*i.e.* case 1 holds), M* will find the optimal path, unless case 2 also holds.*

Proof. If case 1 holds, running A* on G^{sch} will find $\pi_*(v_s, v_f)$ in finite time, unless there exists a cheaper path $\pi_{\text{cheaper}}(v_s, v_f) \subseteq G^{\text{sch}}$, which we now show would satisfies the conditions for case 2 to hold. Because $\pi_*(v_s, v_f)$ is a minimal cost collision-free path, $\pi_{\text{cheaper}}(v_s, v_f)$ must contain a robot-robot collision. Therefore a vertex $v_k \in \pi_{\text{cheaper}}(v_s, v_f)$ must exist such that $\Psi(v_k) \neq \emptyset$, and by (Equation 3.2) $g(\pi_{\text{cheaper}}(v_s, v_k)) + h(v_k) < g(\pi_*(v_s, v_f))$. The existence of a path through v_k implies that $v_k \notin G^{\text{exp}}$, as a vertex containing robot-robot collisions has its outneighbors removed when added to the explored graph G^{exp}. Therefore, $C_k = \emptyset$ by (Equation 3.4). Since $\Psi(v_k) \not\subseteq C_k$, v_k fulfills the roles of both v_b and v_c in the definition of case 2. As a result, if case 1 holds, M* will find $\pi_*(v_s, v_f)$, unless case 2 also holds[2] □. □

[2] We note that if the equality $g(\pi(v_s, v_c)) + h(v_c) \leq g(\pi_*(v_s, v_f))$ holds for case 2, then M* may find the optimal path while both case 1 and case 2 hold. We gloss over this point in the main text, as it ultimately does not change the logic of the proof.

Lemma 4. *If the search graph G^{sch} contains an unexplored path cheaper than $g(\pi_*(v_s, v_f))$ (*i.e. *case 2 holds), M* will not return a suboptimal path.*

Proof. If case 2 holds, then $\pi(v_s, v_c)$ will be explored by A* and added to the explored graph G^{exp} before A* finds any path to v_f that costs more than $g(\pi_*(v_s, v_f))$ [77]. Adding $\pi(v_s, v_c)$ to G^{exp} will modify C_b. G^{sch} will then be modified to reflect the new limited neighbors of v_b and A* will be restarted. Therefore, M* will never return a suboptimal path as long as case 2 holds □. □

Lemma 5. *The search graph G^{sch} will cease to contain any unexplored path cheaper $g(\pi_*(v_s, v_f))$ (*i.e. *case 2 will cease to hold) after finite time.*

Proof. For case 2 to hold, there must be at least one vertex v_b such that C_b is a strict subset of I. G^{sch} can be modified at most $(n-1) * |V|$ times before all collision sets are equal to I. Therefore, after a finite number of modifications of G^{sch} case 2 cannot hold. A* will fully explore any finite graph in finite time, implying that the time between any two successive modifications of G^{sch} is finite. Therefore, case 2 will not hold after finite time □. □

With these auxillary results in hand, the proof of lemma 2 is as follows. If case 1 holds, then M* will find the optimal path in finite time, unless case 2 also holds (Lemma 3). While case 2 holds, M* will not return a suboptimal path (Lemma 4), and case 2 cannot hold after finite time (Lemma 5). Therefore, after finite time, only case 1 will hold, implying that M* will find the optimal path in finite time.

To complete the proof of the completeness and optimality of M*, we must show that case 1 or case 2 will always hold. To do so, we first need an auxiliary result (Lemma 6) showing that the optimal path for some subset of robots costs no more than the joint path taken by those robots in the optimal, joint path for the entire set of robots. The auxiliary result is used to demonstrate that an optimal path can be found by combining optimal paths for disjoint subsets of robots.

Let $\pi'_\Omega(v_k, v_f)$ be the path constructed by combining the optimal path for a subset $\Omega \subset I$ of robots with the individually optimal paths for the robots in $I \setminus \Omega$.

Lemma 6. *If the joint configuration graph contains an optimal path $\pi_*(v_k, v_f)$, then $\forall \Omega \subset I$, $g(\pi'_\Omega(v_k, v_f)) \leq g(\pi_*(v_k, v_f)$. Furthermore, if $\Omega_1 \subset \Omega_2$, then $g(\pi'_{\Omega_1}(v_k, v_f)) \leq g(\pi'_{\Omega_2}(v_k, v_f))$.*

Proof. If $\pi_*(v_k, v_f)$ from an arbitrary v_k to v_f exists in G, then for any subset of robots Ω there exists an optimal path $\pi_*^\Omega(v_k^\Omega, v_f^\Omega)$ which costs no more than the path taken by those robots in $\pi_*(v_k, v_f)$. Let $\overline{\Omega} = I \setminus \Omega$ be the complement of Ω and $\pi_\phi^{\overline{\Omega}}(v_k^{\overline{\Omega}}, v_f^{\overline{\Omega}})$ be the path for the robots in $\overline{\Omega}$ induced by each robot obeying its individual policy. $\pi_\phi^{\overline{\Omega}}(v_k^{\overline{\Omega}}, v_f^{\overline{\Omega}})$ costs no more than the paths taken by the robots in $\overline{\Omega}$ in $\pi_*(v_k, v_f)$ by the construction of the individual policies. A path for all robots in I, $\pi'_\Omega(v_k, v_f)$, is then constructed by having each robot in Ω follow its path in $\pi_*^\Omega(v_k^\Omega, v_f^\Omega)$, while each robot in $\overline{\Omega}$ follows its path in $\pi_\phi^{\overline{\Omega}}(v_k^{\overline{\Omega}}, v_f^{\overline{\Omega}})$. Since the individual path for each robot in $\pi'_\Omega(v_k, v_f)$ costs no more than the path for the same robot in $\pi_*(v_k, v_f)$, $g(\pi'_\Omega(v_k, v_f)) \leq g(\pi_*(v_k, v_f))$. By the same logic, if $\Omega_1 \subseteq \Omega_2$, then $g(\pi'_{\Omega_1}(v_k, v_f)) \leq g(\pi'_{\Omega_2}(v_k, v_f))$ □. □

Lemma 7. *The search graph G^{sch} will always contain an optimal path (*i.e.* case 1 will hold) or an unexplored path which costs no more than the optimal path (*i.e.* case 2 will hold) at all points in the execution of M*.*

Proof. We proceed by showing that the limited neighbors of each vertex in G^{sch} are sufficient to construct either the optimal path, or some unexplored, no more expensive path. Consider the vertex $v_k \in G^{\text{sch}}$ with collision set C_k. The successor of v_k in $\pi'_{C_k}(v_k, v_f)$, v_ℓ, is a limited neighbor of v_k by the definition of the limited neighbors

(Equation 3.2). Since $C_\ell \subseteq C_k$ by (Equation 3.4), Lemma 6 implies

$$\begin{aligned} g(\pi'_{C_k}(v_k, v_\ell)) + g(\pi'_{C_\ell}(v_\ell, v_f)) \leq \\ g(\pi'_{C_k}(v_k, v_f)) \leq g(\pi_*(v_k, v_f)) \end{aligned} \tag{3.5}$$

We apply the above bound vertex by vertex from the initial vertex to show that a path $\pi''(v_s, v_f) \in G^{\text{sch}}$ can be constructed which satisfies either case 1 or case 2. The successor of the m'th vertex v_m in $\pi''(v_s, v_f)$ is the successor of v_m in $\pi'_{C_m}(v_m, v_f)$. Applying (Equation 3.5) gives the bound $g(\pi''(v_s, v_f)) \leq g(\pi'_{C_s}(v_s, v_f)) \leq g(\pi_*(v_s, v_f))$. If $\pi''(v_s, v_f) = \pi_*(v_s, v_f)$ then case 1 is satisfied. Otherwise, there is a vertex $v_c \in \pi''(v_s, v_f)$ such that $\Psi(v_c) \neq \emptyset$. Let v_b be the predecessor of v_c, which implies that v_c lies in $\pi'_{C_b}(v_b, v_f)$. Then $\Psi(v_c) \nsubseteq C_b$, because by construction the robots in C_b do not collide with one another in $\pi'_{C_b}(v_b, v_f)$. By (Equation 3.2), $g(\pi''(v_s, v_c)) + h(v_c) \leq g(\pi''(v_s, v_f)) \leq g(\pi_*(v_s, v_f))$, which implies case 2 is satisfied.

There is an edge case which must be considered if case 1 does not hold. If $\pi''(v_s, v_f)$ contains a vertex $v_k \notin G^{\text{exp}}$ with a successor $v_\ell \in G^{\text{exp}}$, C_ℓ may not be a subset of C_k, because no path exists from v_k to v_ℓ in the explored graph G^{exp}, so the bound given by (Equation 3.5) does not apply. However, in this case the path induced by ϕ from v_ℓ must terminate at some vertex v_c with $\Psi(v_c) \neq \emptyset$. We construct a new path by following $\pi''(v_s, v_f)$ to v_ℓ, and then following $\pi_\phi(v_\ell, v_f)$ to v_c. The sum of the cost of this path and $h(v_c)$ must be less than $g(\pi_*(v_s, v_f))$, and $\Psi(v_c) \nsubseteq C_k$, so case 2 still holds □. □

Theorem 1. *M* is complete and optimal.*

Proof. If the joint configuration graph G does not contain an optimal path, then M* will terminate in finite time without returning an invalid path (Lemma 1). If G does contain an optimal path, then the search graph must always contain either the optimal path, or an unexplored path which costs no more than the optimal path (Lemma 7), which implies that then M* will find the optimal path in finite time (Lemma 2). M* will thus find the optimal path in finite time, if one exists, or terminate in finite time if no path exists. Therefore, M* is complete and optimal □. □

3.4 Performance Analysis

Consider M* running on a worst case problem where every robot interacts with every other robot. Over time, the collision sets will grow until each collision set contains every robot, at which point M* will reduce to A*. The question is then how much additional overhead M* imposes in the most difficult problem instances compared to A*. M* may expand each vertex up to n times; once when the collision set is empty, and once when the collision set contains $2, \ldots, n$ robots, where n is the total number of robots. The computational cost of expanding a vertex with a given collision set C is proportional to the number of limited neighbors $b^{|C|}$, where b is the number of outneighbors of each vertex in the individual configuration graphs. Normalized to the cost of a single A* expansion, b^n, the total cost of all M* expansions of a given vertex is

$$\sum_{i=0, i\neq 1}^{n} \left(\frac{1}{b}\right)^{i} \leq \sum_{i=0,}^{n} \left(\frac{1}{b}\right)^{i} = \frac{1-\left(\frac{1}{b}\right)^{n+1}}{1-\frac{1}{b}} \leq \frac{b}{b-1} \tag{3.6}$$

using rules for the sum of finite and infinite geometric series. Therefore, repeated M* expansions of a given vertex do at most a constant factor more work than a single A* expansion of the same vertex.

Updating the collision set of a vertex takes time linear in the number robots, and the collision set of each vertex may be updated at most $(n-1)$ times, and thus total complexity of maintaining the collision sets may be $\mathcal{O}(n^2|V|)$, where $|V|$ is the total number of vertices in the joint configuration graph. $|V|$ is exponential in the number of robots. In practice the cost of maintaining the collision set is not significant.

3.5 Variants of M*

Several variants of M* with improved performance have been developed. *Recursive M** (rM*) breaks the collision set into independent subsets of robots that can be planned for separately, reducing the maximum dimensionality of the search space. Inflated M* uses an inflated heuristic function to reduce planning time, but returns a path costing up to a specified factor more than the optimal path. ODM* and EPEM* replace A* with *Operator Decomposition* (OD) [167] and *Enhanced Partial Expansion A** (EPEA*) [62], variants of A* tuned for multirobot path planning. Recursive versions of ODM* and EPEM* can be created, resulting in ODrM* and EPEM*, as well as their inflated variants. Finally, the performance of M* is sensitive to choice of individual policies. The Meta-Agent Conflict Based Search framework [157] can be employed to optimize the individual policies using rapid, decoupled planning for individual robots, before applying ODrM* or EPErM* to sets of robots requiring coupled planning.

3.5.1 Recursive M*

The M* algorithm described in 3.2 performs coupled planning for all robots in the collision set, even when the collision set consists of spatially separated subsets of robots. rM* finds an optimal, collision-free path for each such subset via a recursive call to rM*. Such paths constrain the motion for each subset of robots in the same fashion that the individual policies constrain the motion of individual robots. By separating the planning for independent subsets of robots, the worst case computational cost of rM* is exponential in the size of the largest set of mutually colliding robots, rather than in the total number of robots found to collide with other robots.

Implementing recursive M* requires few modifications to basic M*. The collision set for v_k in rM* becomes a collection of the largest disjoint sets that can be formed from the collisions reachable from v_k in G^{exp}. For example, if collisions involving the sets of robots $\{1,2\}$, $\{2,3\}$, and $\{4,5\}$ can be reached from v_k, then $C_k = \{\{1,2,3\},\{4,5\}\}$, instead of $\{1,2,3,4,5\}$ as would be the case in basic M*. If r^i is not in any element of C_k then it obeys its individual policy ϕ^i, as in M*. Otherwise, r^i follows the optimal path for the subset of robots in C_k to which it belongs, as computed by a recursive call to rM*. The exception is if $C_k = \{I\}$, in which case $\hat{V}_k$ is computed as usual for M*, using I as the collision set. This functions as the base case of the recursive calls to rM*.

Recursive M* retains the optimality and completeness properties of M*. Each disjoint set of colliding robots can be thought of as a single, high-dimensional meta-agent. The recursive calls to rM* then serve to compute the individual policy for each meta-agent. With these concepts in place, the proofs in section 3.3.2 apply to rM*.

3.5.2 Inflated M*

One problem with the basic M* implementation is that every time a new robot is involved in a collision, it is added to the collision set of v_s. Unless $g(\pi_*(v_s, v_f)) = g(\pi^{\phi}(v_s, v_f))$, v_s must then be re-expanded at a computational cost that is exponential in the size of C_s. Inflating the heuristic by multiplying the heuristic by some $\epsilon > 1$ is known to significantly decrease the time A* requires to find a solution in many cases [139, 25, 97, 132, 69]. Furthermore, the resultant path will cost no more than $\epsilon \cdot g(\pi_*(v_s, v_f))$ [47]. The logic in Section 3.3.2 can be extended to show that M* has the same sub-optimality bound when used with an inflated heuristic.

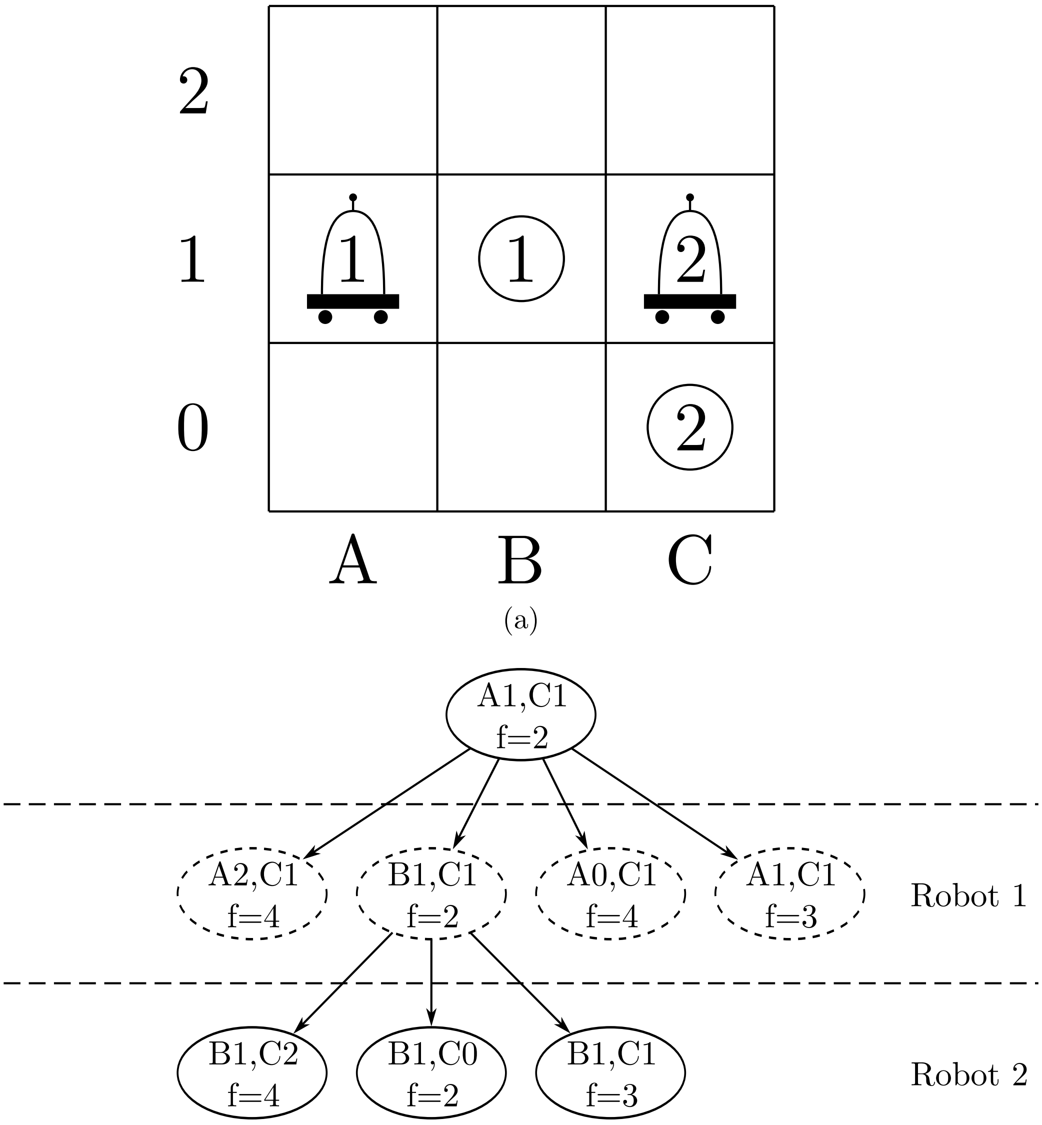
An inflated heuristic benefits M* in two fashions. First of all, an inflated heuristic biases the search towards the leaves of the search tree close to the goal, where a solution is more likely to be found quickly, which is the source of benefit in inflated A*. In addition, the vertices near the leaves of the search tree will generally have smaller collision sets. Therefore, an inflated heuristic will bias search to occur in a region of the search space of low dimensionality.

3.5.3 Replacements for A*

A* is fundamentally limited for multirobot path planning because the number of out-neighbors of a single vertex increases exponentially with the number of robots. A* adds all out-neighbors of a vertex to the open list, even if many will never be expanded. As a result, A* will run out of memory when dealing with systems containing even moderate numbers of robots. OD [167] and EPEA* [62] are variants of A* which delay instantiating expensive neighbors, thus reducing the effective branching factor of the graph. Replacing A* in M* with OD and EPEA* results in the ODM* and EPEM* algorithms, respectively.

ODM*

In ODM*, A* is replaced as the underlying planner by Operator Decomposition, a variant of A* developed explicitly for multirobot path planning. OD mitigates the problem of growth in the number of out-neighbors by procedurally generating the out-neighbors so that low cost neighbors are generated first, and high-cost neighbors may never be instantiated. OD generates two types of search vertices; *standard* and *intermediate*. A standard vertex represents the configuration of all robots in the system. When a standard vertex is expanded, OD generates intermediate vertices which specify all possible actions for the first robot. The cost and heuristic cost-to-go of the intermediate vertices are updated to reflect the new position of the first robot; then the intermediate vertices are added to the open list. When an intermediate vertex is expanded, additional intermediate vertices specifying the action of the next robot are generated. Standard vertices are generated once actions are assigned for the



(a)

(b)

Figure 3.2: Operator Decomposition is used to solve a simple, 2 robot path planning problem **(a)**, where the robots move from vertices $A1$ and $C1$ to the goals $B1$ and $C0$. Initially, the search tree contains a single, standard vertex $\{A1, C1\}$ **(b)**. When $\{A1, C1\}$ is expanded, four intermediate vertices, denoted by dashed lines, are generated to represent the possible actions of the first robot. The intermediate vertex with the lowest f-value is selected for expansion. Three vertices are created, representing the actions of robot 2 which do not collide with the new position of robot 1. Since the new position of all robots has been specified, these are standard vertices. The goal vertex $\{B1, C0\}$ has the lowest remaining f-value, and is expanded next, indicating that a path has been found. [155, adapted]

last robot. This procedure results in the creation of standard vertices which represent heuristically promising actions, such as each robot moving directly towards its goal, before instantiating any less promising vertices. Typically fewer total vertices are created, reducing the computational cost of finding a path.

Figure 3.2 illustrates the vertex expansion of operator decomposition for a problem involving two robots. When coupled with an admissible heuristic, operator decomposition is complete and optimal with respect to path cost. Thus, ODM* is also guaranteed to find optimal paths.

EPEM*

In EPEM*, A* is replaced as the underlying planner by Enhanced Partial Expansion A*, a variant of A* that has been applied to single- and multi-agent planning [62]. EPEA* seeks to eliminate the generation of *excess vertices*, which have a f-value larger than the cost of the optimal path and thus will never be expanded.

EPEA* sorts the open list based on the sum of the f-value of a vertex and an *offset*, $\Delta f(v)$, which is initially set to zero. When EPEA* expands a vertex v_k, it employs a domain specific *Operator Selection Function* (OSF) to instantiate only those neighbors of v_k whose f-value is equal to $f(v_k) + \Delta f(v_k)$. $\Delta f(v_k)$ is then incremented, and v_k is added back to the open list. As a result, no excess vertices will ever be generated.

For multirobot path planning, EPEA* uses an OSF which generates neighbors of a vertex v_k in a two step process: allocating costs to specific robots and generating neighbors. The offset of v_k can be interpreted as an excess cost compared to the heuristically optimal neighbor of v_k. In the first step of expansion, EPEA* allocates individual robots a specific amount of excess to incur. All neighbors of v_k that match the allocation of excess cost are then generated, and added to the open list. This is more efficient than a direct search over all possible neighbors. Felner *et. al.* [62] report that EPEA* outperforms A* and OD when solving dense multirobot path planning problems.

3.5.4 Policy Optimization

The performance of M* is very sensitive to the choice of individual policies when many optimal paths exist for each robot. One choice of individual policies may result in few collisions, while another choice may result in a large number of robots colliding at a single bottleneck, preventing a solution from being found in reasonable time. Therefore, it may be desirable to optimize the choice of individual policies prior to starting M* search.

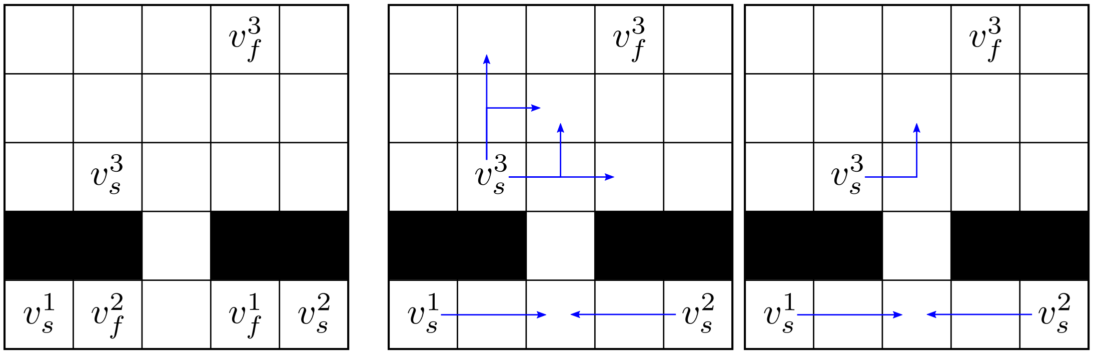
Meta-Agent Conflict Based Search (MA-CBS) [157] is a planning framework introduced by Sharon *et. al.* based on their *Conflict Based Search* (CBS) planning algorithm [156], and generalizes the earlier *Independence Detection* (ID) algorithm by Standley [167]. Conflict-Based Search explores a space of constraints on individual robots, rather than the joint configuration space of the system. Each search vertex contains a set of constraints and the optimal path for each robot subject to the constraints. The constraints prohibit individual robots from occupying a specific position at a specific time that would lead to interference with another robot.

At each step, the search vertex with the smallest total path cost is checked for collisions between the constrained paths of the individual robots. If no collisions are detected, then the optimal solution has been found. If a collision is found between two robots at position q and time t, the search tree branches. Two new vertices are created, each with an added constraint prohibiting one of the involved robots from occupying q at time t. New paths are then computed for each of the involved robots that obey the newly expanded set of constraints. When planning for an individual robot, conflicts with paths of other robots are used for tie breaking: *i.e.* paths which do not conflict with the paths of other robots are preferred, but no additional cost will be incurred to avoid such conflicts.

While the search space for constrained planning is of constant dimensionality, the set of possible constraints grows exponentially. As a result, CBS performs poorly when there are many alternate paths which require a large number of constraints to cover. In such cases, it is more efficient to use coupled search to find a path for the effected robots. MA-CBS [157] is an extension of CBS in which robots are permanently merged into a meta-agent when the number of mutual constraints generated exceeds a merge threshold B. Within a meta-agent, planning is conducted using a coupled planning algorithm respecting the constraints placed on the meta-agent. Internal constraints upon the constituent robots are removed when they are merged into a meta-agent, although the new meta-agent inherits constraints that resulted from collisions with agents not included in the meta-agent. MA-CBS with a given merge threshold B is denoted as MA-CBS(B). Typically, smaller values of B work better in more open environments with many alternate paths, resorting to coupled search earlier, while larger values of B work better in more constrained environments. MA-CBS(0) is equivalent to ID [157].

Using ODrM* as the coupled planner for MA-CBS results in the MA-CBS+ODrM* algorithm. The individual policies computed for ODrM* respect the constraints imposed on the meta-agent, and attempt to minimize conflicts with robots not in the meta-agent. In this fashion, the individual policies are optimized to minimize robot-robot conflicts.

ODrM* and MA-CBS complement each other well. MA-CBS can minimize the total number of collisions via rapid, decoupled search, and is effective in narrow bottlenecks which pose a problem for ODrM*, while ODrM* is more suited to open regions than other coupled planners, as ODrM* will reject alternate, low cost paths which cannot resolve collisions.



(a) Configuration (b) EPEA* Search Tree (c) M* Search Tree

Figure 3.3: Illustrative example of the computation benefit of M* compared with A*, OD, or EPEA*. **(a)** Robots start at v_s^1, v_s^2, and v_s^3 and move to v_f^1, v_f^2 and v_f^3 respectively. **(b)** EPEA* must construct a search tree containing multiple alternate paths before it can consider moving r^1 into the alcove. **(c)** M* does not need to consider alternate paths for r^3 before M* can consider moving r^1 into the alcove.

3.6 Comparison of M* and Similar Algorithms

M*, EPEA*, OD, ID and MA-CBS all exploit the same natural decomposition of the multirobot path planning problem by exploring paths that minimize the costs incurred by individual robots before considering more expensive paths. As a result, there are a number of similarities in these algorithms. This section will describe how M* differs from the other algorithms, and where M* can show a performance improvement.

EPEA* and OD are both approaches that intelligently search the joint configuration space. While EPEA* and OD can delay instantiating unpromising vertices, they cannot identify and exclude unnecessary portions of the joint configuration space. By tracking which robots collide where, M* can construct a search space that excludes unnecessary regions of the joint configuration space. Consider a 3 robot example, where r^1 and r^2 must swap positions in a narrow corridor, while r^3 is alone in an open room (Figure 3.3a). Clearly, r^2 needs to wait for r^1 to enter the alcove, or vice versa. However, such a path would have a greater f-value than the initial state.

Therefore, before OD or EPEA* could consider such a path, they must first examine all optimal alternate paths for r^3, even though none of those paths could possibly resolve the conflict (Figure 3.3b). In the case of M*, r^3 is not involved in any collision, and thus will remain restricted to its individually optimal path (Figure 3.3c). M* can therefore proceed immediately to considering alternate paths for the robots involved in the collision, rather than waisting time on alternate paths for r^3.

MA-CBS, ID and rM* share a common purpose: splitting the multirobot system into independent subsets of robots. The approach rM* takes to splitting the system is less sophisticated than that employed by ID and MA-CBS. When rM* detects a collision between two robots, it immediately merges them to form a meta-agent, instead of checking whether choosing a different individual policy of one of the robots could avoid the collision, as MA-CBS or ID would do. However, rM* has much more fine-grained control over the merging of robots. Once rM* resolves a collision between the agents composing a meta-agent, it splits the meta-agent back into individual robots, whereas once MA-CBS or ID generates a meta-agent, it remains merged. The local merging of rM* will typically not reduce the peak dimensionality of the search space, as v_s accumulates all collisions and must be re-expanded if $g(\pi_*(v_s, v_f)) > f(v_s)$. However, it will reduce the number of vertices at which the search space will have maximal dimensionality. Furthermore, the fine-grained nature of rM* allows it to be used within the MA-CBS or ID frameworks as the coupled planner, thus gaining the benefit of both the more sophisticated policy optimization performed by MA-CBS and ID, and the local merging of agents that rM* provides.

3.7 M* Results

To validate the performance of M* on systems of up to 200 robots, we turn to simulation. All simulations were implemented in python and run on a computer with

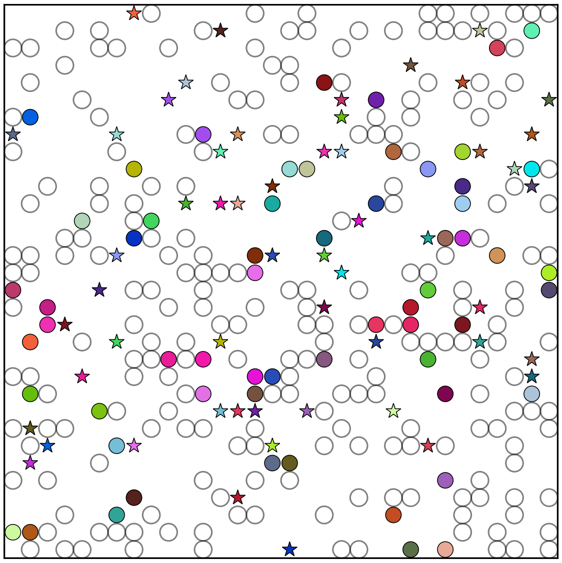
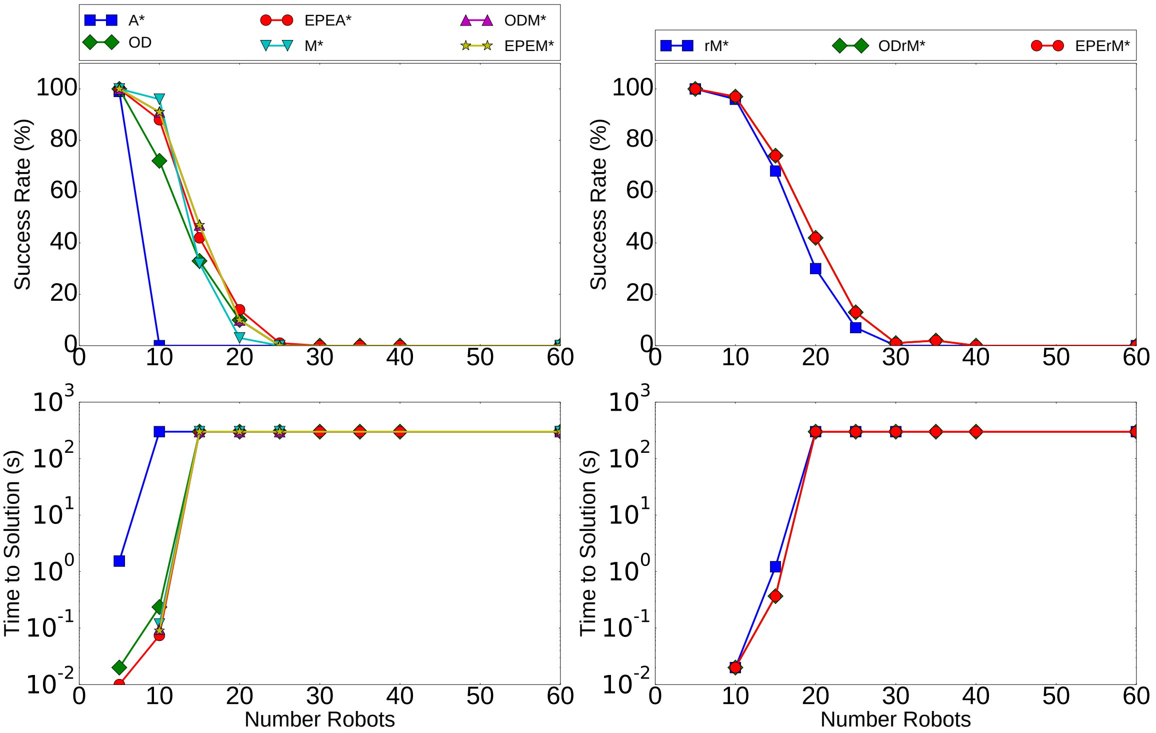


Figure 3.4: A typical four-connected grid world with 32x32 cells for a test run including 40 robots. Colored circles represent the initial positions of the robots, while goal positions are marked with colored stars. Unfilled circles represent the obstacles.

an Intel Core i5-2500 processor clocked at 3.30 GHz (Turbo Boost disabled) with 8 GB of RAM. The test environment was a 32x32, four-connected grid of cells, with a 20% probability of a given cell being marked as an obstacle, as in [167] (Figure 3.4). Unique initial and goal positions for each robot were chosen randomly within the same connected component of the workspace. Any action by an individual robot, including waiting, incurred a cost of one, although a robot could wait at its assigned goal with zero cost. The existence of a wait action implies the presence of a self-loop for each vertex $v_k \in G^i$, so that v_k is its own out-neighbor.

Each trial was given 5 minutes to find a solution. 100 random environments were tested for a given number of robots. We present the percentage of trials that were successful within 5 minutes as well as the median time required to find solutions. Run time is plotted on a logarithmic scale.



(a) M* Variants
Max 60 Robots

(b) rM* Variants
Max 60 Robots

Figure 3.5: Results for A*, OD, EPEA*, M*, ODM*, and EPEM* **(a)** and rM*, ODrM*, and EPErM* **(b)**. The plots on top illustrate the percentage of trials in which a solution was found within 5 minutes, in a 32x32 four-connected grid world. The bottom graphs the median time to solution.

3.7.1 M*, Operator Decomposition and rM*

We start by comparing A*, OD, EPEA*, M*, EPEM*, ODM*, rM*, ODrM* and EPErM* (Figure 3.5). The plateauing of the median time to solution plots is the result of at least 50% of trials reaching the 5 minute time limit. Python's CPU time function has a resolution of one millisecond, resulting in solutions that take less than one millisecond being reported as taking zero time, which cannot be represented on a logarithmic plot.

As expected, A* demonstrated the worst performance, being unable to find solutions for problems of 10 or more robots. A* was limited by the exponential growth in the number of neighbors of a given vertex as the number of robots increases. OD,

Algorithm	Largest Collision Set	Largest Independent Subset of the Collision Set
M*	9	
ODM*	15	
EPEM*	15	
rM*	16	9
ODrM*	25	16
EPErM*	25	16

Table 3.4: Number of robots in the largest collision set encountered in a problem solved by M*, ODM*, EPEM*, rM*, ODrM*, and EPErM* for systems of up to 40 robots in a 32x32 grid world. For rM*, ODrM*, and EPErM* the size of the largest independent subset of the collision set for which coupled planning was successfully performed is also reported.

EPEA*, M*, ODM* and EPEM* all show roughly similar performance. M* solved the most problems with 15 robots, but decayed in performance rapidly until it underperformed all other algorithms at 20 robots. OD generally underperformed EPEA*, M*, ODM*, and EPEM*, while EPEA* unexpectedly showed the best performance for problems involving 20 robots.

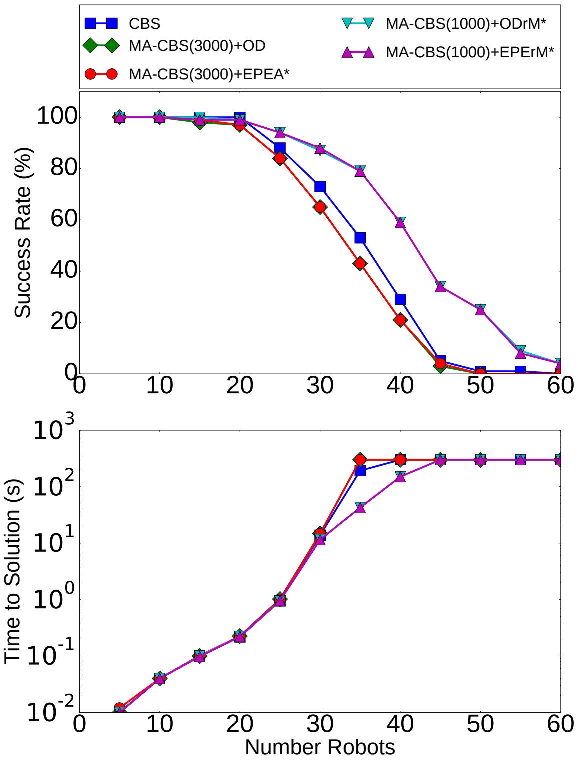
The recursive variants of M* showed noticeable improvement over the non-recursive approaches, and solved twice as many problems involving 20 robots as EPEA* (Figure 3.5b). Recall that rM* uses A* as the underlying planning algorithm, so that rM* typically expands more vertices than ODrM* or EPErM*. Thus, we expected ODrM* to solve more instances within the given time limit. ODrM* and EPErM* solved twice as many problems involving 25 robots as basic rM*. The near identical performance of ODrM* and EPErM* can be accounted for by the similarity in performance of OD and EPEA*.

The degree to which M* and its variants can solve problems which involve dense interactions between many robots can be measured by the maximum size of the collision set of v_s encountered during a successful trial (Table 3.4). Recall that the collision set of v_s accumulates all robots found to collide with another robot at any point in the search. However, if $g(\pi_*(v_s, v_f) = g(\pi_\phi(v_s, v_f)$ then v_s may not be expanded with its largest collision set, depending on how ties are broken when vertex f-values are compared. ODM*, ODrM*, EPEM*, and EPErM* were able to handle larger collision sets than M* and rM*, which is to be expected because OD and EPEA* could solve problems involving more robots than A*.

The recursive implementations solved problems in which roughly twice as many total robots were involved in collisions as the equivalent non-recursive implementation. This is because the recursive implementations split the collision set into independent subsets of robots, for which coupled planning is performed separately. The largest independent subset of the collision set in the recursive implementations were equivalent in size to the largest collision sets for which the non-recursive implementations found solutions. Thus, while recursive implementations could solve problems involving more total robots, the number of robots which could interact in a single region of the workspace, and thus require coupled planning, was determined by the underlying planner.

3.7.2 Policy Optimization

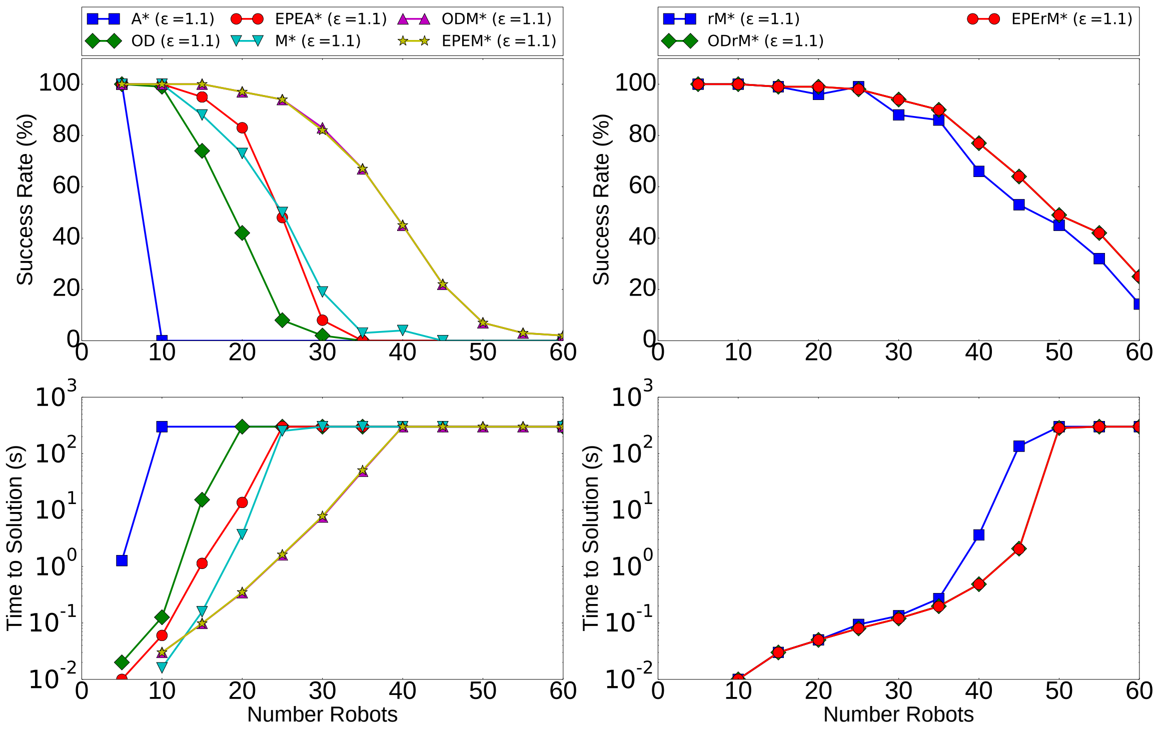
We now present simulation results using the MA-CBS planning framework, and demonstrate that integrating ODrM* or EPErM* provides state of the art results for optimal multirobot path planning. MA-CBS is parametrized by a merge threshold which must be tuned to a specific problem's characteristics. MA-CBS+OD, MA-CBS+EPEA*, MA-CBS+ODrM*, and MA-CBS+EPErM* were tested with merge thresholds of 3, 10, 30, 100, 300, 1000 and 3000. MA-CBS+ODrM* and MA-CBS+EPErM* performed best with a merge threshold of 1000, while MA-CBS+OD and MA-CBS+EPEA* performed best with a merge threshold of 3000.



Policy Optimization

Figure 3.6: Results for CBS, MA-CBS(10)+OD, MA-CBS(20)+ODrM*, and MA-CBS(30)+EPErM*. The plot on top illustrate the percentage of trials in which a solution was found within 5 minutes, in a 32x32 four-connected grid world. The bottom graphs the median time to solution.

The planning results for CBS, equivalent to MA-CBS(∞), MA-CBS(3000)+OD, MA-CBS(3000)+EPEA*, MA-CBS(1000)+ODrM*, and MA-CBS(1000)+EPErM* are given in figure 3.6. CBS outperformed MA-CBS(3000)+OD, MA-CBS(3000)+EPEA*, which is not surprising given that the environment is very cluttered, which is where CBS is known to perform best [157]. The greater planning power of M* allowed MA-CBS(1000)+ODrM* and MA-CBS(1000)+EPErM* to substantially outperform CBS, while the performance of MA-CBS(1000)+ODrM* and MA-CBS(1000)+EPErM* were nearly identical. We note that on 8-connected grids, where there are more alternate paths, the performance benefit of MA-CBS+ODrM* over CBS and MA-CBS+OD becomes even more substantial [64].



(a) M* variants with inflation

(b) rM* variants with inflation

Figure 3.7: Results for A*, OD, M*, ODM*, rM* and ODrM* with a heuristic inflated by 10% **(a)** and extended results to 100 robots for inflated ODM*, inflated rM*, inflated ODrM*, and MA-CBS(20)+ODrM* without an inflated heuristic **(b)**. The plots on top illustrate the percentage of trials in which a solution was found within 5 minutes, in a 32x32 four-connected grid world. The bottom graphs the median time to solution.

3.7.3 Inflated Heuristics

We tested A*, M*, EPEA* and variants of M* with a heuristic inflated by a factor of 1.1 (Figure 3.7a). All algorithms were thus guaranteed to find a path costing no more than 10% more than that of the optimal solution. Inflated A* was still unable to find solutions for systems of 10 or more robots, as each vertex has ten million neighbors. While the success rate for inflated OD, inflated EPEA* and inflated M* all improved, M* benefited substantially more from an inflated heuristic than OD or EPEA* did. Basic inflated M* was held back by inefficient neighbor generation for larger collision sets, and thus performed on par with inflated EPEA*, but inflated

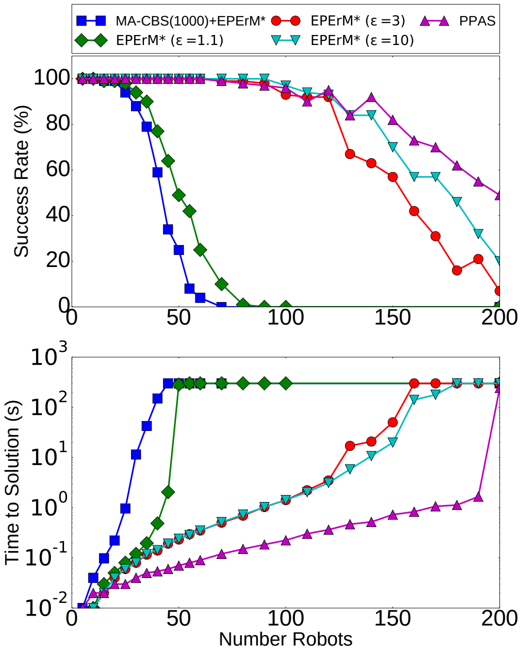
ODM* and inflated EPEM* solved problems involving roughly twice as many robots. The inflated heuristic concentrates search on the leaves of the search graph nearest to the goal, providing a benefit to EPEA*, OD and M*. However, such leaves will also have smaller collision sets, reducing the dimensionality of the search space for M*, and accounting for the greater reduction in computation time for inflated ODM* and inflated EPEM* compared to inflated OD or inflated EPEA*.

Inflated rM*, ODrM* and EPErM* show further improvements in performance, as expected (Figure 3.7b). Inflated ODrM* and EPErM* were able to find solutions more quickly, in more cases, and with a simpler implementation than MA-CBS(1000)+ODrM*, reflecting the computational benefits of relaxing the requirement to find optimal cost paths, even if only slightly.

3.7.4 Comparison to Rule-Based Approaches

M* and inflated M* can find optimal or ϵ-suboptimal paths to problems involving many robots, but in the worst case the computational complexity of M* is still exponential in the number of robots. This raises the question of what benefits M* conveys in practice in comparison to polynomial-time, rule based approaches which do not provide bounds on path cost. To this end, we compared variants of M* against a C++ implementation of Parallel Push and Swap (PPAS) graciously provided by Sajid et al. [151]. The PPAS code was not optimized for performance or run time

Four variants of M* are used as points of comparison, MA-CBS(1000)+EPErM*, which produces optimal paths, and inflated EPErM* with inflation factors of 1.1, 3, and 10. The performance of inflated ODrM* was essentially the same as EPErM*, so results for ODrM* are omitted. The failures of PPAS were the result of the implementation tested crashing. While PPAS has only been shown to be complete on trees, the observed failures are most likely the result of bugs in the provided code.

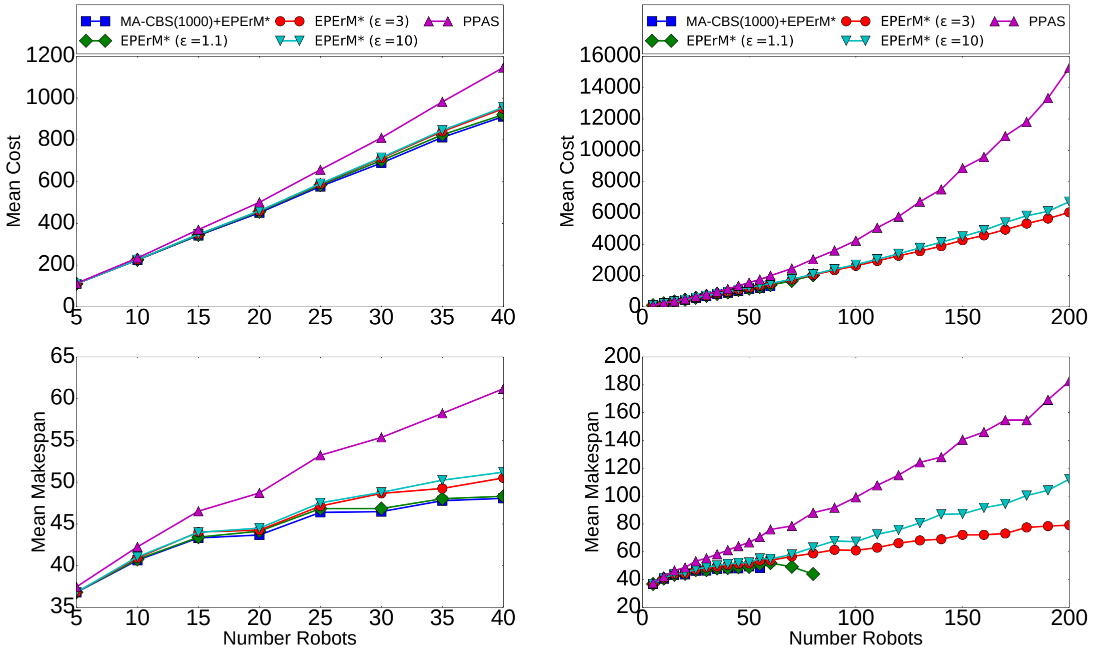


Comparison with Rule-Based Approches

Figure 3.8: Results for MA-CBS(1000)+EPErM*, inflated EPErM* with inflation factors of 1.1, 3, and 10, and Parallel Push and Swap (PPAS). The plot on top illustrate the percentage of trials in which a solution was found within 5 minutes, in a 32x32 four-connected grid world. The bottom graphs the median time to solution. The failures of PPAS were due to the implementation being tested crashing.

All successful runs of PPAS terminated in under 6 seconds.

The mean path cost and mean makespan (time until all robots reach their goals) of paths found by PPAS and M* variants are shown in figure 3.9. PPAS consistently found paths of substantially greater cost than those found by EPErM* variants, demonstrating the benefits of approaches which bound path cost. Note that the cost bounds on inflated EPErM* are loose; while EPErM* ($\epsilon = 10$) could potentially find paths that cost ten times the minimal cost, it generally finds substantially cheaper paths. The results are slightly distorted by the fact that the mean cost and makespan are only computed for trials for which a given algorithm was able to find a solution. As a result, the mean makespan for EPErM* ($\epsilon = 1.1$) appears to decline for instances involving more than 50 robots (Figure 3.9b), but this is an artifact of EPErM* ($\epsilon =$



(a) Path cost comparison (≤40 robots) (b) Path cost comparison (≤200 robots)

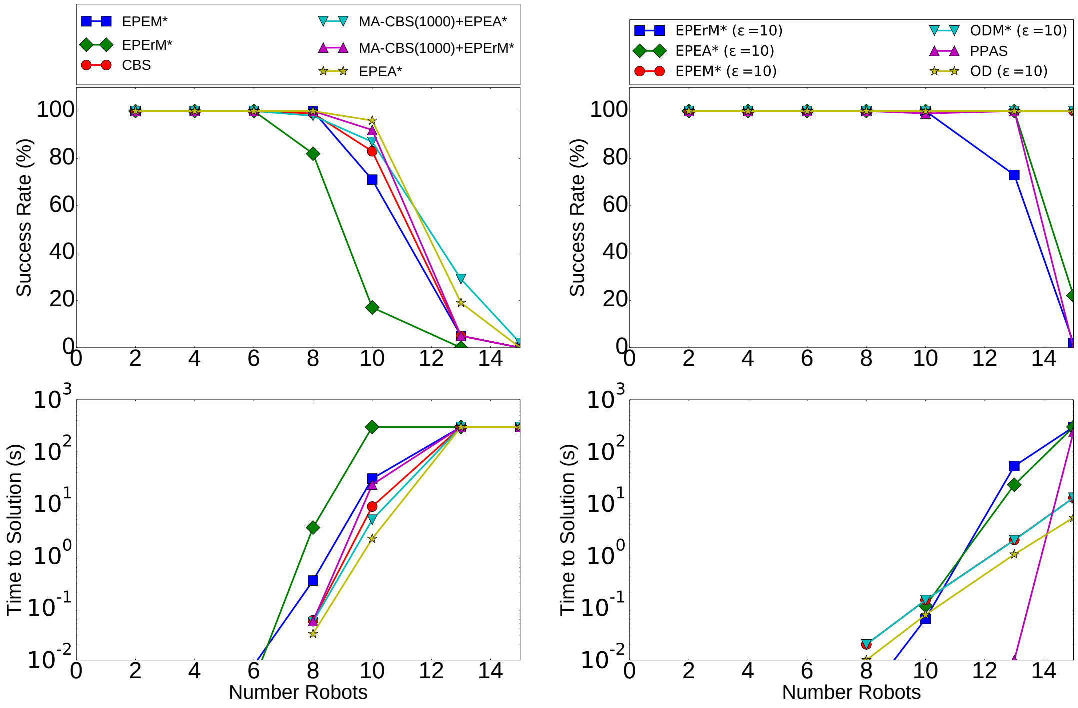
Figure 3.9: Path costs for MA-CBS(1000)+EPErM*, inflated EPErM* with inflation factors of 1.1, 3, and 10, and Parallel Push and Swap (PPAS). The plot on top shows the mean cost of the paths successfully found by the algorithms. The bottom plots show the mean makespan (time until all robots reach their goal). **(a)** Results for trials of up to 40 robots. **(b)** Results for trials of up to 200 robots.

1.1) only solving the easier instances of those problems. However, the success rates of PPAS and EPErM* ($\epsilon = 10$) are similar enough, especially up to 150 robots, for the cost comparisons to be valid.

3.7.5 Fully Coupled Tests

In the previously discussed simulations, the environment was comparatively open, allowing a substantial degree of decoupling between robots. To examine the performance of M* in fully coupled environments, a series of tests were run in a 4x4 gird world with up to 15 robots, equivalent to the 15 puzzle.

Six optimal approaches were tested, EPEM*, EPErM*, CBS, MA-



(a) Optimal Coupled Results

(b) Sub/Non-Optimal Coupled Results

Figure 3.10: Results for fully coupled tests on a 4-connected, 4x4 grid world, with **(a)** optimal and **(b)** suboptimal and non-optimal algorithms. The plot on top illustrate the percentage of trials in which a solution was found within 5 minutes, in a 32x32 four-connected grid world. The bottom graphs the median time to solution.

CBS(1000)+EPErM*, EPEA*, and MA-CBS(1000)+EPEA*[3] (Figure 3.10a). There is a general trend that the more aggressively an algorithm exploits decoupling between robots, the worse its performance. EPErM* is out performed by EPEM*, and EPEA* outperforms both EPEM* and CBS. MA-CBS(1000)+EPEA* does outperform EPEA* for 13 robots, which we interpret as MA-CBS slightly simplifying some problems before falling back on EPEA*.

Five ϵ-suboptimal methods were tested; inflated EPEA*, inflated OD, inflated EPEM*, inflated EPErM*, all with an inflation factor of $\epsilon = 10$. The ϵ-suboptimal methods were tested against PPAS, a non-optimal, rule based method. PPAS can find solutions much faster than any of the ϵ-suboptimal methods, but fails on all of

[3] The results for MA-CBS for fully coupled problems are insensitive to the merge threshold chosen for MA-CBS

the 15 robot problems, because PPAS makes the assumption that there are always at least two free vertices. The failures of PPAS at 10 robots were due to bugs in the implementation that was tested. Inflated EPErM* performed the worst of any of the ϵ-suboptimal methods, due to the overhead of computing paths for disjoint subsets of robots that were later invalidated due to collisions with other robots. Inflated OD outperforms inflated EPEA*, which may be surprising given the performance of those algorithms with a lower inflation factor of $\epsilon = 1.1$ (Figure 3.7a). However EPEA* generates all neighboring vertices of a given f-value at once, while OD iteratively generates the neighboring vertices. High inflation factors bias search towards the goal, causing OD to behave in a more depth-first manner, effectively generating a single neighbor for a given state at a time. Goldenberg et al. [71] described but did not implement optimal-generation variants of EPEA* which may mitigate the reduced performance of EPEA* with large inflation factors. Inflated ODM* and EPEM* are roughly a constant factor slower than inflated OD, but have similar success rates. Note that even with a high inflation factor OD substantially underperforms inflated EPErM* in less cluttered environments; in the 32x32 grid environment inflated OD with $\epsilon = 10$ performs roughly on par with EPErM* $\epsilon = 1.1$.

3.7.6 Critical Densities

The median time to solution plots for M* have a character shape: the time to solution grows gradually as the number of robots increases, before hitting an inflection point and rising sharply. The success rate typically drops sharply at or just before the inflection point. This is particularly evident for inflated rM* (Figure 3.8). As the dimensionality of the search space grows, the cost of exploring the search space grows exponentially. Therefore, there should be a relatively sharp transition between collision sets that define a search space small enough for M* to explore, and collision

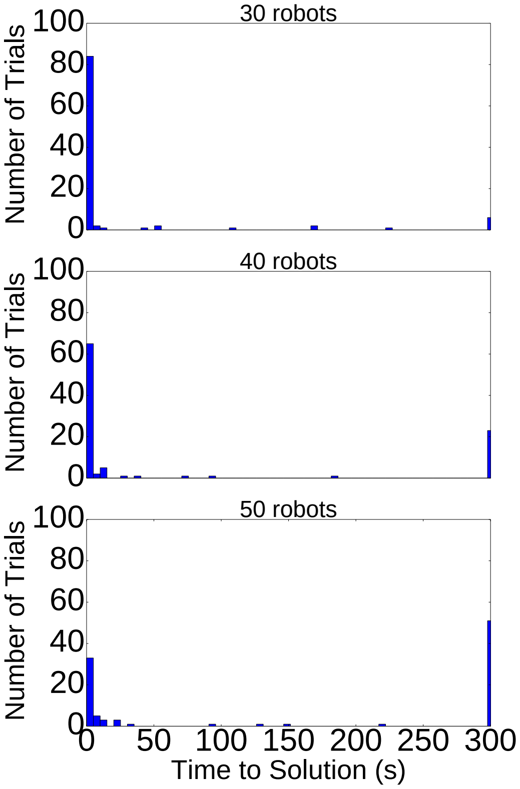


Figure 3.11: Histogram of the time to solution for inflated ODrM* with $\epsilon = 1.1$. The distribution is very peaked, indicating that either inflated ODrM* is generally able to solve a problem rapidly, or it is not able to solve the problem at all in reasonable time.

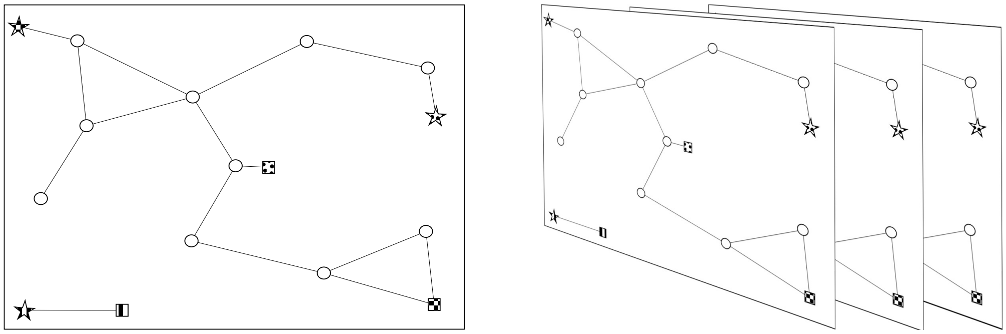
sets that induce too large of a search space. The inflection point is most likely a result of the robots achieving a critical density at which the typical problem requires at least one vertex with a collision set in the critical range to be expanded. Once this point is reached, increasing planning time is unlikely to yield substantially more solutions. The histogram of the time to solution for inflated ODrM* supports the theory that there is a critical size of the collision set. If inflated ODrM* can find a solution it will do so quickly, and otherwise it will time out without finding a solution (Figure 3.11).

Chapter 4

Subdimensional Expansion and Probabilistic Path Planning

M* demonstrates excellent scaling with the number of agents in a multirobot system. However, if a multirobot system is composed of individual robots for which A* cannot quickly find paths, than M* is not appropriate for finding paths for the multirobot system. Probabilistic planners such as *Probabilistic Roadmaps* (PRM) [88] and *Rapidly-Exploring Random Trees* (RRT) [106] were developed to find paths for robots with high-dimensional configuration spaces [24] and complex constraints, which stymied conventional planners. For example, PRMs have demonstrated the ability to find paths for the folding of proteins with more than 200 degrees of freedom [6].

Probabilistic approaches have been applied to smaller multirobot systems by directly planning in the joint configuration space [35, 63, 152, 153]. However, multirobot systems are more challenging for probabilistic planners than a single robot with an equivalent number of degrees of freedom because a candidate path segment must be collision-free for all robots in the system. When the workspace is cluttered, the prob-



(a) Individual PRM

(b) Joint PRM

Figure 4.1: We show a PRM constructed in a single robot configuration space that connects the initial configuration (box) to the goal configuration (star) for three homogeneous robots, indicated by pattern. Circles are randomly generated samples used to construct the PRM **(a)**. We construct a PRM in the joint configuration space of the three robot system by taking the tensor product of three copies of single robot PRM **(b)**.

ability that a given path segment is collision-free declines rapidly as the number of robots in the system increases. For this reason, combining probabilistic planners with multirobot path planning approaches, such as decoupled planning, can substantially improve performance [41, 150].

In this chapter we apply subdimensional expansion to PRMs and RRTs, resulting in the *subdimensional Probabilistic Roadmap* (sPRM) and *subdimensional Rapidly-Exploring Random Tree* (sRRT) algorithms [195], respectively. sPRM and sRRT combine the capability of probabilistic planners to find paths for robots with many degrees of freedom with the ability of subdimensional expansion to construct a low-dimensional search space for multirobot systems, permitting paths to be found for large teams of complex robots.

4.1 sPRM: Subdimensional Expansion with PRMs

A PRM is a graph which provides a sparse description of the configuration which can be constructed in a relatively small amount of time considering the dimensionality of the configuration space. A PRM is constructed by drawing random samples, called *milestones*, from the collision-free configuration space of the system. The milestones comprise the vertices of the PRM. A local planner is employed to find simple paths between pairs of nearby milestones. If the path between a pair of such milestones is collision-free, then an edge is added to the PRM connecting the two milestones. A number of approaches have been developed to ensure that sufficient samples are drawn from confined regions of the configuration space, such as narrow passages, to ensure that the PRM will be connected [26, 79, 198, 5, 52]. Once construction of the PRM is completed, a path connecting two milestones in the PRM can be found using a graph search algorithm, such as A* [88].

Path planning with PRMs thus poses two problems; how to construct a PRM which covers the configuration space of the system of interest, and how to find a path in the PRM once its construction is completed. Constructing a PRM in the joint configuration space of a multirobot system is difficult, as two milestones cannot be connected if a single robot collides with an obstacle on the path generated by the local planner, so the probability that two milestones can be connected declines rapidly as the number of robots increases. Švestka and Overmars [178] showed that a PRM covering the joint configuration space, termed the *joint PRM* in this thesis, of a multirobot system can be computed efficiently by taking the Cartesian product of PRMs constructed in the configuration space of each robot (Figure 4.1). Unfortunately, the number of vertices in the joint PRM, grows exponentially with the number of robots, which makes finding a path in the joint PRM difficult. Švestka and Overmars [178] employed a hierarchical subgraph decomposition of the joint PRM to find a path, which only permitted a single robot to move at any time, and only considered systems of up to 5 robots. In sPRM, a joint PRM is constructed according to the method of Švestka and Overmars [178], then M* is used to efficiently find a path for the multirobot system in the joint PRM.

sPRM follows the lead of Švestka and Overmars [178] by constructing an individual PRM G^i in the configuration space Q^i of each robot r^i. The initial and goal configurations of the robot, q_s^i and q_f^i are used as milestones in addition to random samples (Figure 4.1a). Since the PRM is constructed for a single robot, collision checking only considers robot-obstacle collisions. Homogeneous robots can share copies of a single PRM, in which case the initial and goal configurations of all of the homogeneous robots are used as milestones. Construction of the individual PRM is completed once each initial configuration lies in the same connected component as its associated goal configuration.

Just as A* is used to find a path in a single robot PRM, M* (Section 3) is used to find a collision-free path for the multirobot system. The joint PRM G which covers the joint configuration space is defined as the direct product[1] of the individual PRMs (Figure 4.1b), but is not explicitly constructed. Instead, M* incrementally constructs a low-dimensional search graph G^{sch} during the course of searching for a path, as described in section 3.3.1. Edges are only checked for robot-robot collisions when explored by M*, providing some of the benefits of lazy collision checking described by Bohlin and Kavraki [24]. Furthermore, the joint PRM is free of robot-obstacle collisions by construction, so the collision checker only has to check for robot-robot collisions. In this manner, planning to avoid robot-obstacle collisions is decoupled from planning to avoid robot-robot collisions.

Švestka and Overmars [178] proved that if a collision-free path for the multirobot system exists, the probability that the joint PRM contains a collision-free path goes to 1 as the number of milestones used to construct the individual PRMs goes to infinity. M* is complete 3.3.2, and thus will find a collision-free path in finite time, if such a path is contained in the joint PRM. Therefore, we can conclude that the probability

[1] Here we diverge from Švestka and Overmars [178] who formed the joint PRM using the Cartesian product. The Cartesian product allows a single robot to move at a time, while the direct product allows multiple robots to move simultaneously

that sPRM will find a collision-free path goes to 1 as the number milestones used in the individual robot PRMs goes to infinity, a property known as probabilistic completeness.

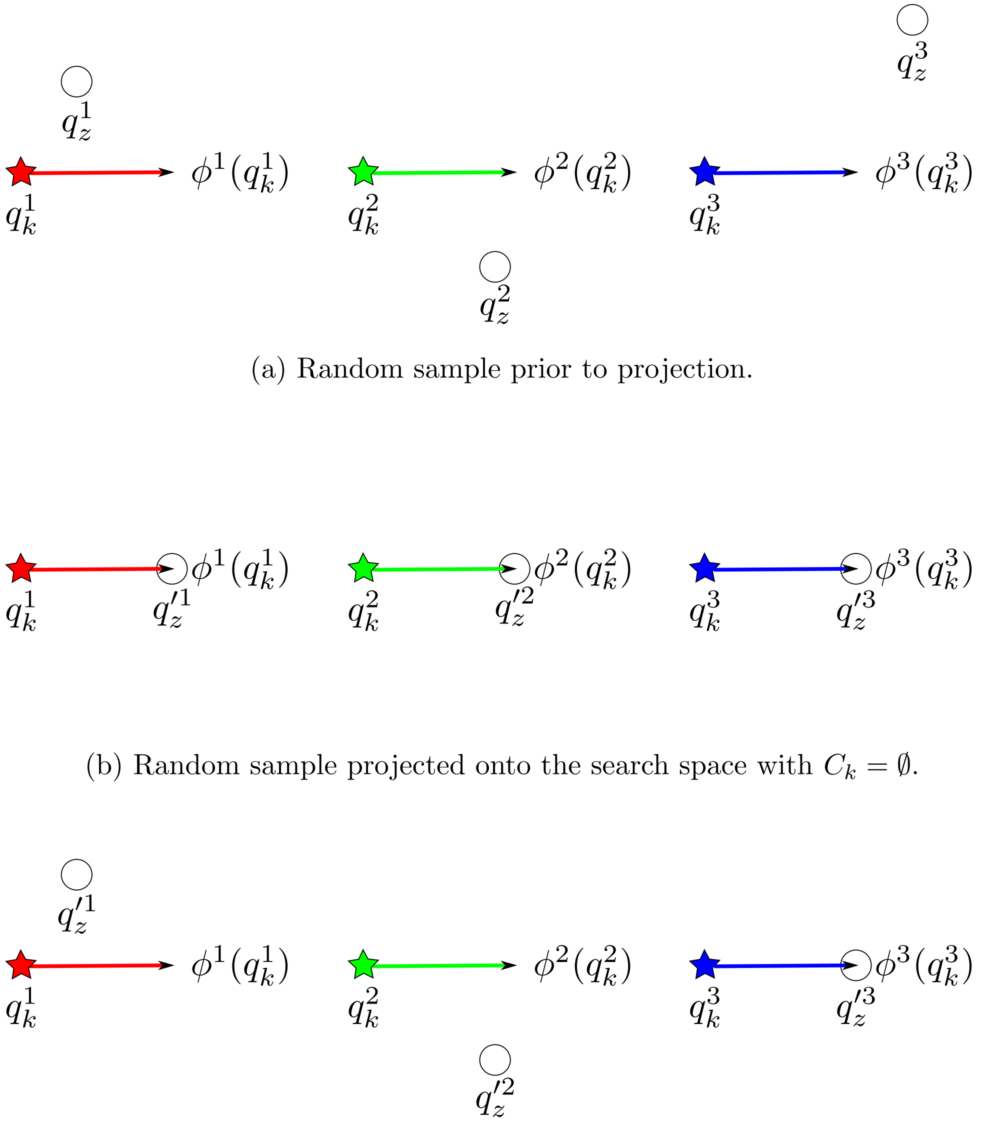
4.2 sRRT: Subdimensional Expansion with RRTs

RRTs were introduced by LaValle and Kuffner as an efficient single-query planner for high dimensional systems [106]. RRTs construct a search tree rooted at the initial configuration of the system. The tree is extended in a three step process. First, a random sample is drawn from the configuration space. A local planner then attempts to extend a path from the nearest vertex in the search tree towards the random sample. Finally, the local path is tested for collisions. If no collisions are found, the final configuration of the local path, which may not be the same as the random sample depending on the local planner, is added to the search tree. Because the search graph is a tree, a path from the initial configuration to the goal can be easily retrieved once a configuration sufficiently close to the goal is added to the tree.

sRRT uses the subdimensional expansion framework to guide the construction of an RRT for multirobot systems. sRRT starts by computing the individual policy for each robot r^i by growing an RRT tree $\mathcal{T}^i$, denoted a *policy tree*, from the goal configuration q^i_f of r^i in Q^i. Since the policy tree is grown backwards from the goal configuration, the parent p^i_k of q^i_k in $\mathcal{T}^i$ is the next step on the path defined by ϕ^i from q^i_k to the goal.

$$\phi^i(q^i_k) = p^i_k. \tag{4.1}$$

If $q^i_k \notin \mathcal{T}^i$, $\mathcal{T}^i$ is first grown until the policy tree is connected to q^i_k. In practice, it is best to connect q^i_k to $\mathcal{T}^i$ using bidirectional search: attempting to make a connection between $\mathcal{T}^i$ and a tree grown from q^i_k as in RRT-Connect [101].



(a) Random sample prior to projection.

(b) Random sample projected onto the search space with $C_k = \emptyset$.

(c) Random sample projected onto the search space with $C_k = \{1, 2\}$.

Figure 4.2: **(a)** A random sample q_s is projected onto the search space in the Voronoi region of the vertex q_k. A three robot configuration space is visualized by showing the coordinates of each robot side by side. q_k^i gives the location of the ith robot in q_k. The arrow points along the individual policy from q_k to the next configuration $\phi(q_k)$. The circles show the random sample q_s before any projection. **(b)** $C_k = \emptyset$, so each robot is restricted to its individual policy, resulting in the projected sample q_s'. **(c)** $C_k = \{r^1, r^2\}$. Therefore, only robot r^3 is restricted to its individual policy, resulting in the projected sample q_s''

Path planning for the full system is performed by growing a tree $\mathcal{T}_f$ forwards in the joint configuration space, from a root vertex at the initial system configuration q_s. The expansion of $\mathcal{T}_f$ is restricted to the search space Q^{sch} determined by subdimensional expansion. The search space is constructed by identifying each vertex $q_k \in \mathcal{T}_f$ with a local search space $Q^{\text{sch}}_k = \{q \mid q^i = \phi^i(q^i_k) \forall i \notin C_k\}$, which is the set of configurations which can be reached from q_k when the robots not in C_k obey their individual policy. The choice of search space causes each robot to flow along its policy tree towards the goal, while allowing robots involved in robot-robot collisions to depart from the policy tree to find a path around the collision.

To grow $\mathcal{T}_f$, random samples must be drawn from Q^{sch}. sRRT generates samples q'_z in Q^{sch} by first drawing a random sample q_z from the joint configuration space, then projecting q_z onto Q^{sch}. Let $q_k \in \mathcal{T}_f$ be the nearest vertex to q_z. q_z can be projected onto Q^{sch}_k by replacing the coordinates for each robot not in C_k with the coordinate for that robot's next step along its individual policy from q_k

$$q'^i_z = \begin{cases} q^i_z & r^i \in C_k \\ \phi^i(q^i_k) & r^i \notin C_k \end{cases} \tag{4.2}$$

where q'^i_z is the coordinate for the i'th robot in q'_z (Figure 4.2). q_z can be thought of as determining how each robot in the collision set deviates from its individual policy.

Once a sample is generated, the local planner finds a path from q_k towards q'_z, which is then checked for robot-robot and robot-obstacle collisions. If r^i is found to be involved in a robot-robot collision, r^i is added to C_k, then recursively added to the collision set of the parent vertex of q_k, in a manner analogous to the backprop function in M* (Algorithm 1). As in M*, growing the collision sets increases the dimensionality of the search space, to ensure that a collision-free path can be found.

We note that sRRT bears a resemblance to bidirectional algorithms such as RRT-

Connect [101]. RRT-Connect attempts to connect a forward tree grown from the initial configuration to a back tree grown from the goal configuration of a system. $\mathcal{T}_f$ in sRRT resembles the forward tree of RRT-Connect, while the policy trees as a group resembles the back tree. However, the back tree in RRT-Connect and the policy trees in sRRT serve different purposes. The back tree in RRT-Connect is grown in the configuration space of the full system, and is intended to assist finding paths in environments where planning from the goal is easier than planning from the initial configuration. The policy trees are grown in the configuration spaces of individual robots, and are intended to decouple planning to avoid robot-obstacle collisions from planning to avoid robot-robot collisions. The policy trees can quickly find a path for each robot to its goal configuration which avoids robot-obstacle collisions, because the dimensionality of the individual robot configuration space is much smaller than the dimensionality of the joint configuration space of the multirobot system. The policy trees guide the construction of $\mathcal{T}_f$ towards the goal configuration, so that only robot-robot collisions must be resolved. The projection defined by (Equation 4.2) ensures that $\mathcal{T}_f$ only explores new paths for robots that would otherwise interact, minimizing the dimensionality of the search space, and thus the computational cost of finding a path.

4.3 Simulation Results: sPRM and sRRT

sRRT and sPRM were tested in simulation on teams of kinematic three link robots. Each robot has a five-dimensional configuration space $(x, y, \theta, \alpha_1, \alpha_2)$, with $(x, y, \theta) \in$ SE(2) specifying the position and orientation of the central link, and $\alpha_1, \alpha_2 \in [-\pi/2, \pi/2]$ specifying the angle of each joint (Figure 4.3). Each link has length 2. The robots are permitted to translate and rotate freely.

10 random test environments were created, each 90x90 and containing 25 square

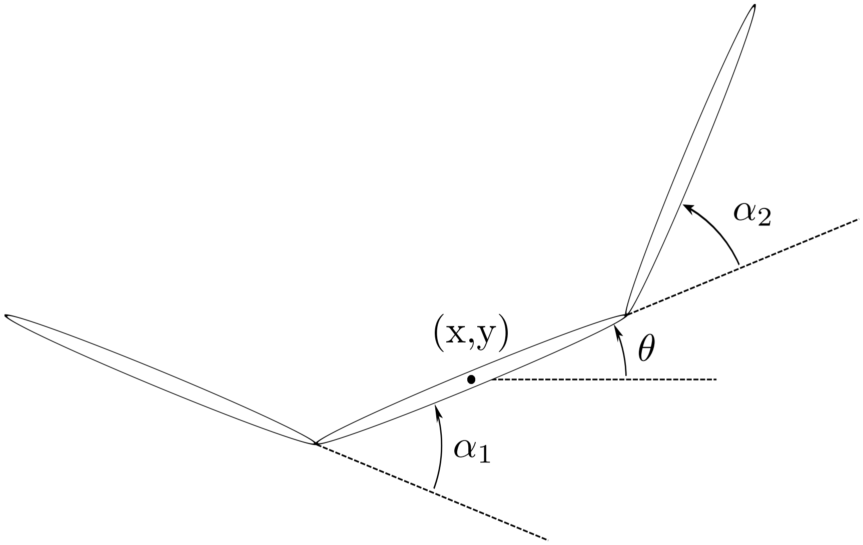


Figure 4.3: Depiction of the three link robot used to test sRRT and sPRM

obstacles with sides of length 2. The initial and goal configurations were chosen randomly, with each test case run 10 times. We present the percentage of trials that were successful within 5 minutes as well as the median time required to find solutions. Run time is plotted on a logarithmic scale. We compare sRRT with RRT-Connect [101] and sPRM using inflated ODrM* with an inflation factor of 10 (Figure 4.4). sPRM had the highest success rate, followed by sRRT and RRT-Connect.

4.4 Conclusions

The greater effectiveness of sPRM compared to sRRT and RRT-Connect for systems consisting of large numbers of robots can be attributed to three factors. In sPRM, robot-robot coordination is conducted on a discrete graph, while in sRRT coordination is performed in a continuous space, which is generally more difficult. Furthermore, the graph used by sPRM is known to be free of robot-obstacle collisions, decoupling coordination from obstacle avoidance, while sRRT must consider both robot-robot and robot-obstacle paths when coordinating robots. sPRM also shares more computation between robots. In the case of homogeneous robots, a PRM grown in the

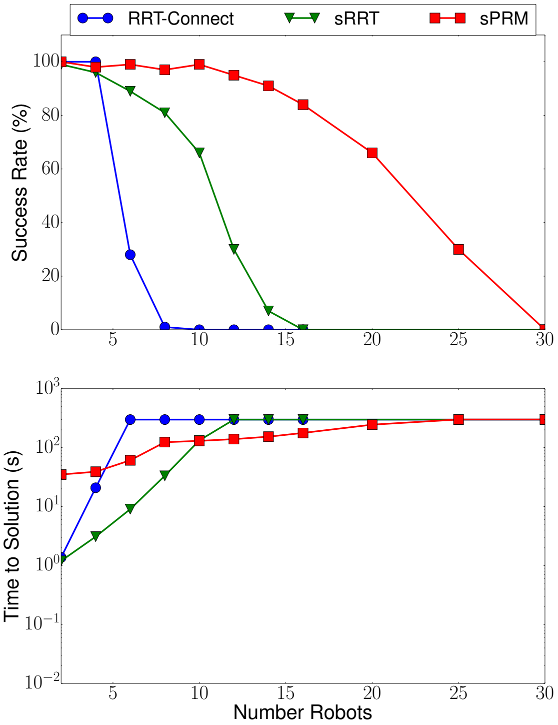


Figure 4.4: Results for RRT-Connect, sRRT and sPRM. The plots on top illustrate the percentage of trials in which a solution was found within 5 minutes. The bottom plot shows median time to solution.

configuration space of one robot can be used to generate individual policies for all robots, while sRRT must grow a separate policy tree for each robot. Also, when sPRM checks whether two robots can traverse a given pair of edges without collision, the result can be cached for use whenever that pair of edges are traversed, regardless of which robots are involved.

For problems involving lower numbers of robots, sPRM takes comparatively more time to find a solution than the RRT based approaches, because of the necessity of constructing a sufficiently dense roadmap prior to calling ODrM*. While sRRT and RRT-Connect continue to construct new paths for individual robots throughout planning, the set of all possible joint paths that will be considered by sPRM is fixed once sPRM stops work on the individual robot roadmaps and calls ODrM*. While sPRM could call ODrM* immediately after the initial configuration of each robot is connected to its goal in the individual robot PRMs, a denser roadmap is desirable. The initial configuration and goal configuration in the joint PRM are not necessarily connected once the initial and goal configurations are connected in the single robot PRMs, as the single robot PRMs do not consider robot-robot collisions. Determining that no solution exists for a multirobot system in the joint PRM is an expensive operation, even using inflated ODrM*. Therefore, it is better to spend the time to construct a denser joint PRM that will be connected, rather than spending a long time determining that a sparse joint PRM is disconnected and in need of extension.

Furthermore a sparse, but connected roadmap, may contain only one of multiple paths through a given field of obstacles. Such bottlenecks result in many robots interacting in a small area, which forces ODrM* to operate in a high-dimensional search space, and significantly increases the cost of finding a path for the multirobot system. A denser roadmap, containing multiple paths through a given region of space, will tend to spread out the individual robots, thus reducing the computational cost of finding a path.

Chapter 5

Constraint Manifold Subsearch

Many interesting problems, such as automated assembly or observation of multiple targets, involve tasks where robots may temporarily come together to form teams. Handling the dynamic formation and dissolution of these robot teams requires solving two problems: scheduling and assigning robots to teams, and coordinating the motions of many robots as independent agents and as parts of varied teams. In this chapter, the assignment of robots to tasks and the order in which tasks must be executed are assumed to be provided *a priori*, and the focus will be on finding high-quality, collision-free paths for large numbers of robots that can dynamically form teams. We term the result the *Cooperative Path Planning* (CPP) problem.

In this chapter, we introduce an algorithm for solving CPP problems called *Constraint Manifold Subsearch* (CMS) [194], based on the M* MPP algorithm (Chapter 3). CMS operates by temporarily merging the agents in a team into a single meta-agent whose configuration space is the *constraint manifold* of the task, *i.e.* the subspace of the team configuration space that satisfies the constraints of the task. CMS is complete and guaranteed to find the optimal solution. Alternatively, CMS can accept ϵ-suboptimality in return for greatly reduced planning time. Following the example of rM*, we show that planning time can be further reduced by splitting the CPP problem into independent subproblems, leading to the *recursive Constraint Manifold Subsearch* (rCMS) algorithm.

5.1 Prior Work

Solving the CPP problem requires solving two qualitatively different problems: coordinating the simultaneous motion of many teams of robots [64, 194, 62, 157, 167] and finding paths for the robots within a team that satisfy the constraints of the task being executed. While there has been a large amount of work in formation control [13, 16, 17, 18, 19, 39, 44, 57, 107, 109, 121, 122, 145, 127, 166, 193, 203] and cooperative manipulation [4, 37, 53, 66, 93, 91, 147, 202], little work has been done in the context of path planning with the dynamic formation and dissolution of teams. We review some of the notable exceptions which have inspired our work here.

Ayanian and Kumar [10], Desaraju and How [54], and Bhattacharya et al. [22] developed CPP planners that could plan for systems where multiple teams form and persist for significant durations. Ayanian and Kumar [10] solved problems where robots must remain in close proximity to the other robots in its team by searching the prepares graph of controllers in a sequential composition framework [31], and later extended the work to consider dynamic formation and dissolution of teams [8], but this algorithm did not scale well to larger numbers of robots or consider the dependencies between teams introduced by subsequent teams sharing robots. Bhattacharya et al. [22] and Desaraju and How [54] both developed CPP algorithms that operate incrementally by adjusting the path of one robot at a time to better match the constraints imposed by the tasks. Desaraju and How [54] developed DM-RRT, a decentralized algorithm where a single robot was allowed to replan its path to better match the task constraints. DM-RRT scales well with increasing numbers

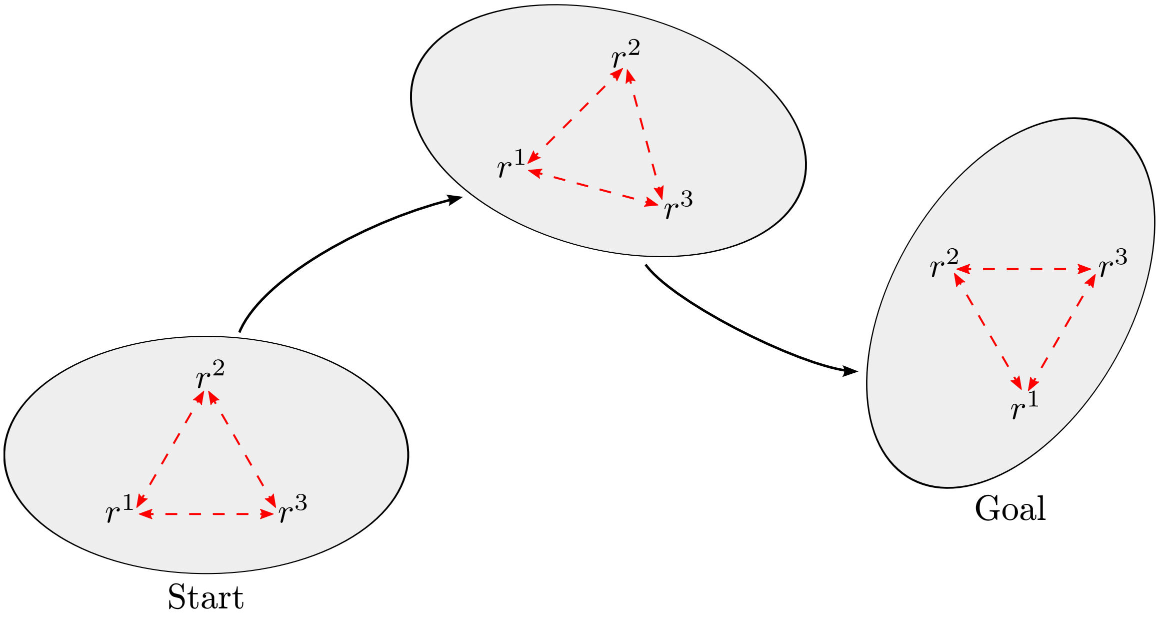


Figure 5.1: A cooperative task τ^j is defined by a set of robots $\mathcal{R}^j$ that will perform the task, an initial joint configuration q_s^j at which the robots can start execution of the task, a joint goal configuration q_f^j at which the task is considered complete, and a set of inter-robot task constraints $\mathfrak{C}^j$ (red, dashed arrows) which must be satisfied during execution.

of robots, but guarantees neither completeness nor optimality. Bhattacharya et al. [22] repeatedly replanned paths for individual robots while gradually increasing the cost of violating task constraints. The resulting approach can be shown to eventually converge to the optimal path, but does not scale well with increasing numbers of robots.

5.2 Problem Definition

The objective of the CPP problem is to find a high quality, collision-free path for a set of n robots r^i, $i \in \{1, 2, \ldots, n\}$, such that the robots complete a set of m cooperative tasks τ^j, $j \in \{1, 2, \ldots, m\}$. Each task τ^j is represented by a tuple $\left(\mathcal{R}^j, q_s^j, q_f^j, \mathfrak{C}^j\right)$ where $\mathcal{R}^j$ is the set of robots assigned to perform the task, q_s^j is the joint configuration of the robots in $\mathcal{R}^j$ at which the task execution can begin, q_f^j is the joint configuration of robots at which the task can be completed, and $\mathfrak{C}^j$ is a set of *task constraint*s that the robots in $\mathcal{R}^j$ must satisfy during task execution (Figure 5.1). We use τ^j to denote both the task and the team of robots that performs the task. To simplify notation, robots are assumed to always be part of a team. If a robot is operating independently of other robots then it is assigned to a singleton team with no constraints.

Each robot r^i has an ordered *task list* that contains the tasks assigned to the robot, such as delivering a large load or driving a rivet, in the order of execution. The robot may have to form a team with a different set of robots to perform each task. The task lists must be consistent, *i.e.* the ordering of the tasks for the entire system cannot induce any cycles.

When a team is at its goal configuration it can take an explicit transition action to complete its task and allow its constituent robots to form new teams for the next tasks in their task lists. If the *forming teams*, $\mathcal{T}^{\text{form}}$, created by the transition action contain robots from multiple teams, termed the *dissolving team* $\mathcal{T}^{\text{diss}}$, then the transition action must be taken simultaneously by all of the dissolving teams.

CMS seeks to minimize the sum of the costs of the paths taken by each team, where the cost of a team's path does not depend on the path of any other team. CMS assumes that a team incurs a non-negative cost for each action. CMS also assumes that τ^i can wait at its goal configuration at zero cost if it is unable to perform the transition action due to one or more of the other associated dissolving teams not being at their goal configuration. All solutions to a given problem will contain the same transition actions, which therefore incur zero cost Finally, CMS assumes that robots incur zero cost for waiting at the goal configuration of their final task.

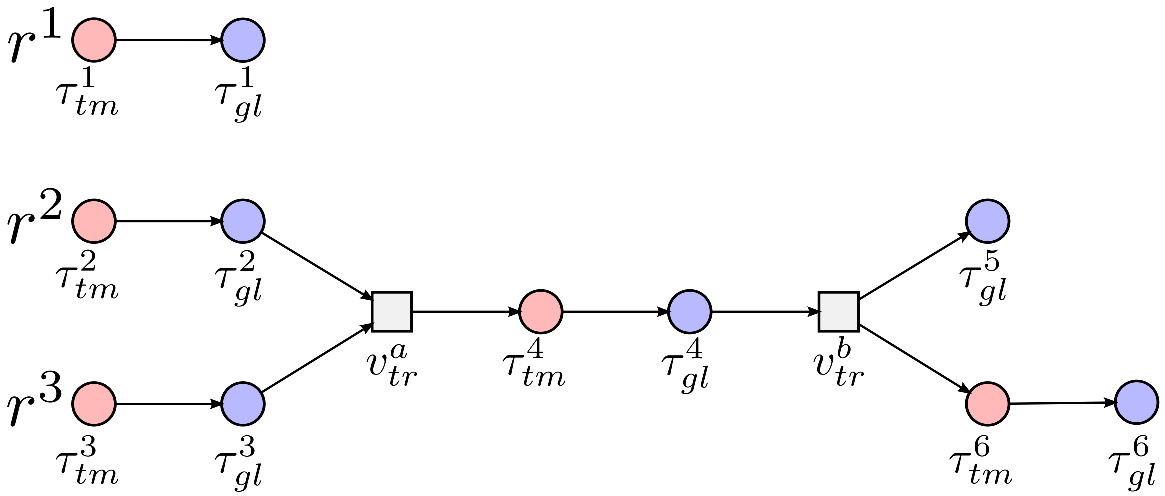


Figure 5.2: Example of a task graph. Robot r^1 is assigned the task list $[\tau^1]$, r^2 is assigned to $[\tau^2, \tau^4, \tau^5]$, and r^3 is assigned to $[\tau^3, \tau^4, \tau^6]$. Note that $q^5_s = q^5_f$, so τ^5 is represented by a single vertex in V^{gl} (blue), and has no corresponding vertex in V^{tm} (red). The transition vertices are the gray squares.

5.2.1 The Task Graph

In the MPP problem a robot r^i only impacts the path taken by r^j if r^j must alter its path to avoid a potential collision with r^i. In the CPP problem, the dependencies between teams are more complex, as a team can only form after an earlier set of teams complete their task, and a team can only complete its task once the other dissolving teams are at their goal configuration. We encode the dependencies between teams in a *task graph*, a common tool in the task scheduling community [74, 80].

The task graph $G^{\text{st}} = \{V^{\text{tm}}, V^{\text{gl}}, V^{\text{tr}}, E^{\text{st}}\}$ is a directed tripartite graph with three types of vertices. Team τ^i is represented by the vertex $\tau^i_{tm} \in V^{\text{tm}}$ before τ^i reaches its goal configuration and by $\tau^i_{gl} \in V^{\text{gl}}$ after reaching its goal configuration for the first time. If $q^i_s = q^i_f$, for instance because τ^i is a single robot that must wait for another team to arrive to form its next team, then the task graph contains τ^i_{gl} but not τ^i_{tm}. The transition vertices $v_{tr} \in V^{\text{tm}}$ represent transition actions. Let $v_{\text{task}} \in V^{\text{tm}} \cup V^{\text{gl}} \cup V^{\text{tr}}$ denote an arbitrary vertex in the task graph.

In the sequel, τ^i may represent τ^i_{tm}, τ^i_{gl}, or the team throughout the execution of its task, depending on context. This notation, while ambiguous, will substantially simplify the later description of CMS by removing the need to constantly refer to whether a given team has reached its goal configuration.

The edges in the task graph are formed as follows. Consider a team τ^i. If $q^i_s = q^i_f$, then it is represented by a single vertex $\tau^i_{gl} \in V^{\text{gl}}$ as the team starts at its goal configuration. Otherwise, the team is represented by two vertices; $\tau^i_{tm} \in V^{\text{tm}}$ and $\tau^i_{gl} \in V^{\text{gl}}$, with an edge connecting τ^i_{tm} to τ^i_{gl}. Assuming τ^i is not the final task for its constituent robots, τ^i_{gl} is connected by an edge to the transition vertex v_{tr} for which $\tau^i \in \mathcal{T}^{\text{diss}}$. Similarly, unless τ^i is the first task (including singleton tasks) for its constituent robot, an edge connects the transition vertex for which $\tau^i \in \mathcal{T}^{\text{form}}$ to τ^i_{tm}, if it exists, or τ^i_{gl}, otherwise.

There is a natural partial ordering on vertices of the task graph wherein a vertex $v^i_{\text{task}} \in G^{\text{st}}$ is *dominated* by $v^j_{\text{task}} \in G^{\text{st}}$, denoted $v^i_{\text{task}} \leq v^j_{\text{task}}$, if there is a path in G^{st} from v^i_{task} to v^j_{task} or $v^i_{\text{task}} = v^j_{\text{task}}$. Two vertices are incomparable if there is no path between them in G^{st}. A team τ^i is dominated by a vertex v^k_{task} if the vertex in V^{tm} or V^{gl} corresponding to τ^i is dominated by v^k_{task}. Finally, a team or transition vertex is dominated by a set of task graph vertices if it is dominated by at least one element of the set. The partial ordering has a fairly simple intuitive meaning. If $\tau^i \leq \tau^j, i \neq j$ then τ^i must complete its task before τ^j can form, while incomparable teams can coexist. If v^i_{task} is a transition vertex, then all teams dominated by v^i_{task} must finish their tasks before the transition corresponding to v^i_{task} can occur.

The path taken by τ^i from its initial configuration to its goal configuration directly depends only upon the paths of the teams with which τ^i potentially collides. However, the teams with which τ^i potentially collides depend upon when τ^i forms and when it completes its task. As a result, the path taken by τ^i is affected by the path taken by τ^j in three cases.

1. $\tau^j \leq \tau^i, i \neq j$
2. CMS finds a potential collision between τ^i and τ^j
3. The successor of τ^i and τ^j in the task graph is the same transition vertex

In case 1, $\tau^j \leq \tau^i$ implies that τ^j must dissolve before τ^i can form. Changing when τ^i forms will effect its position relative to the other teams. As a result, the optimal path may require τ^j to take a short, expensive path so that τ^i can complete its task before a potential collision with a third team could occur. Case 2 is straight forward, as if τ^i and τ^j potentially collide, then τ^i or τ^j must alter their path to avoid the potential collision.

Case 3 implies that τ^i and τ^j must take a common transition action to finish their tasks, which can only be done when all the associated dissolving teams including τ^i and τ^j are at their goal configurations. If τ^i collides with a third robot before τ^i reaches its goal configuration, then τ^i could not have taken a transition action to avoid the collision regardless of the configuration of τ^j, so altering the path of τ^j would have no impact on resolving the collision. However, if τ^i had ever reached its goal configuration before colliding with a third robot, then there is at least one step at which τ^i could have taken a transition action, and thus altering the path of τ^j may be required to resolve the collision at minimal cost.

Consider an environment containing a crevasse that can only be crossed if teams τ^1 and τ^2 combine to form τ^3. Furthermore, τ^2 has two possible paths: a long, cheap path around a sand dune; and a short, expensive path over the sand dune (Figure 5.3). If τ^1 potentially collides with τ^4 before τ^1 reaches its goal configuration (Figure 5.3a), then τ^1 never had a chance to cross the crevasse before the collision no matter what τ^2 did. Therefore the potential collision between τ^1 and τ^4 can be resolved without including τ^2 in the coupled planning. Now alter the problem so that τ^1 is already at its goal configuration, and the potential collision between τ^1 and τ^4 occurs

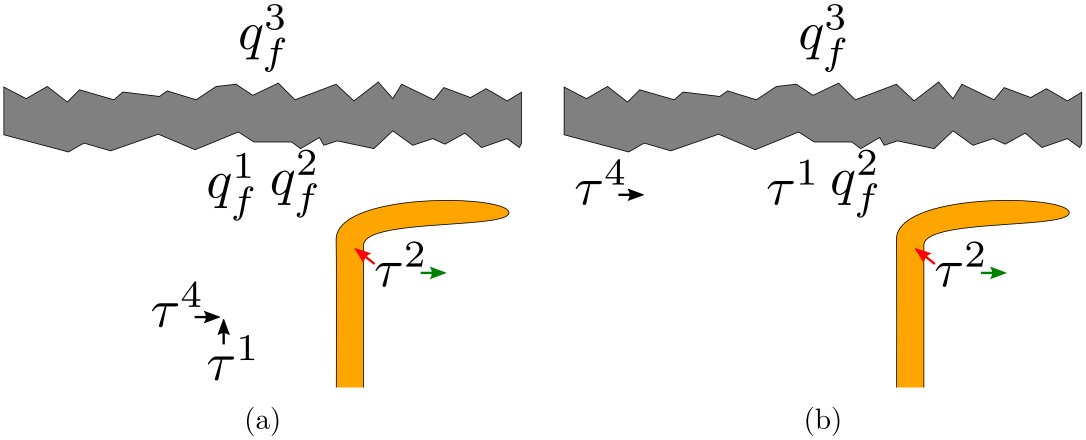


Figure 5.3: Teams τ^1 and τ^2 must combine to form τ^3 to cross a crevasse (gray). τ^2 has two possible paths: a long, cheap path (green arrow) around a sand dune (tan); or a short, expensive path (red arrow) across the dune. **(a)** If τ^4 collides with τ^1 before τ^1 reaches its goal configuration q_f^1, then the collision can be resolved without altering the path of τ^2. **(b)** If τ^4 collides with τ^1 after τ^1 had reached its goal configuration, then the optimal path may require τ^2 to take its shorter, more expensive path so that τ^1 and τ^2 can merge to form τ^3 and cross the crevasse before the potential collision between τ^1 and τ^4.

before τ^2 would reach its goal configuration while following the long, cheap path, but after τ^2 would reach its goal along the short, expensive path. The individual policies will always send τ^2 along the cheaper path. If altering the paths of τ^1 and τ^4 is sufficiently expensive, then the optimal path would be for τ^2 to take the short and expensive path, allowing τ^1 and τ^2 to form τ^3 and cross the crevasse before τ^4 would potentially collide with τ^1. Thus resolving a potential collision between τ^1 and τ^4 requires coupling planning with τ^2, but only if τ^1 had reached its goal configuration before the potential collision.

5.3 Planning on Constraint Manifolds

At its core, CMS takes an existing multirobot path planning algorithm and modifies its search space to quickly find a path that satisfies the task constraints. More specif-

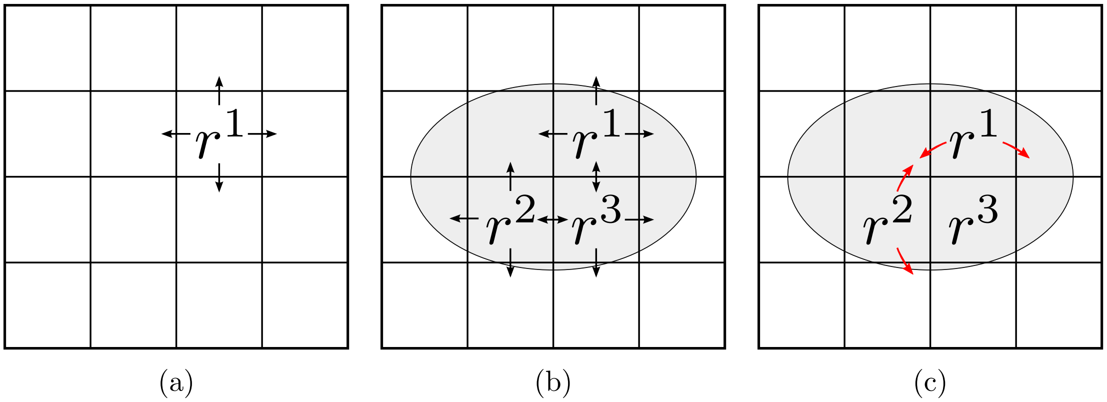


Figure 5.4: Certain cooperative tasks may be incompatible with the joint configuration graph of the robots executing the task. **(a)** A regular grid is an appropriate discretization of the configuration graph of a fully-actuated planar robot. The arrows show the actions permitted by the abstraction: horizontal and vertical translation. **(b)** A team of three robots can translate a common load, represented by the ellipse, with the translation actions afforded by the discretization. **(c)** The red arrows indicate the necessary actions of the robots to rotate the load, but such actions are not afforded by a grid discretization.

ically, CMS plans for each team in the *constraint manifold*[1] of the associated task, the subspace of the joint configuration space of the constituent robots that satisfies the task constraints.

The CPP problem can be naïvely solved by conventional MPP algorithms if the action set is augmented with the actions needed to form or dissolve teams. Violations of the task constraints can then be treated simply as a robot-robot collision. For instance, if a pair of robots is carrying a rigid body and one robot moves too far away from the other, the multirobot path planning algorithm would treat that state as being invalid just as if the first robot had run into the second.

Such an approach would face two serious problems. First, the actions afforded to a team by the joint configuration graph may not be sufficient to complete its assigned task. Figure 5.4 shows a simple example of a joint configuration graph that only allows horizontal and vertical translations, preventing the robot team from performing

[1] Constraint manifold is a term of convenience: CMS can solve problems where the task constraint is satisfied on a subspace that is not a manifold.

a rotation. The second problem is that the constraint manifold of a task is typically of lower dimensionality than the team configuration space in which it is embedded. For instance, the constraint manifold for ten planar robots carrying a rigid body is a 3 dimensional space, rather than a 20 or 30 dimensional space. As a result, most configurations in the team configuration space will violate task constraints. Planning a path for the team would thus require expensive, coupled planning for all of the robots in the team, and would be equivalent to constructing the constraint manifold by exhaustive search.

The constraint manifold for a number of interesting tasks can be exactly computed. For instance, the constraint manifold for a team of planar robots carrying a rigid body is isomorphic to SE(2) and can be parameterized by the position and orientation of the load. Cohen et al. [42] showed that the constraint manifold for dual-arm manipulation by a PR-2 robot is a simple 6 dimensional subspace of the full 14 dimensional configuration space. CMS exploits such exact descriptions of the constraint manifold by restricting robots executing a cooperative task to the constraint manifold.

CMS restricts a team of robots to the constraint manifold of a task by temporarily replacing the robots with a single meta-agent whose configuration space is the associated task's constraint manifold. Let $\Psi_i : \mathcal{M}^i \rightarrow Q^i$ be the embedding that maps the constraint manifold $\mathcal{M}^i$ to the joint configuration space Q^i of the robots in a team τ^i. $\mathcal{M}^i_{\text{start}} = \Psi_i^{-1}(q^i_s)$ is the position of the team in the constraint manifold where the robots start executing task τ^i, and $\mathcal{M}^i_{\text{goal}} = \Psi_i^{-1}(q^i_f)$ is the position of the team in the constraint manifold where the task can be completed.

For the purpose of planning, the constraint manifold associated with each task is discretized. The constraint manifold $\mathcal{M}^i$ of team τ^i is represented by the weighted directed graph $G^i_{\mathcal{M}} = \{V^i_{\mathcal{M}}, E^i_{\mathcal{M}}\}$, termed the *manifold graph*. Each vertex in the vertex set $V^i_{\mathcal{M}}$ represents a configuration on the constraint manifold, while each edge in the edge set $E^i_{\mathcal{M}}$ represents valid transitions between configurations. The weight of an edge represents the cost of the action associated with the edge.

5.4 Constraint Manifold Subsearch

CMS is an extension of M* (Chapter 3) to solve the CPP problem. The following changes to M* are necessary to track task completion and reason about the complex robot-robot interactions in the CPP problem. The joint configuration graph is replaced with the *task augmented joint configuration graph* G^{aug}, that tracks the state of task execution and contains additional edges corresponding to team dissolution/formation events. Secondly, individual policies are computed for each team, rather than for each robot. The heuristic function and the limited neighbors are modified to account for transition actions. Finally, the collision set is replaced with the *conflict set* and the *coupled set* which capture the more complex dependencies between teams of robots.

5.4.1 The task augmented joint configuration graph

The graph explored by CMS must track the progress the system makes in executing tasks, to determine the feasible actions of each robot and the optimal behavior. Therefore, CMS constructs and searches G^{aug} which tracks the active teams and their positions. Each vertex in G^{aug} represents a set of ordered pairs $(\tau^i, v^i_{\mathcal{M}})$ which contains each active team and the vertex in its manifold graph that represents the team's current configuration. Edges correspond to motion of the teams and changes in the content of the active teams.

G^{aug} is the union of one *active graph* $G^{\text{act}}(\mathcal{T}^{\text{act}}_k)$ for every possible set of active teams $\mathcal{T}^{\text{act}}_k$ and a *transition graph* G^{trans} that captures possible transitions between

Search graph	M*	Joint configuration graph that represents the joint configuration space
	CMS	Task augmented joint configuration graph that represents the joint configuration space and the state of task execution
Heuristic	M*	Cost for robots to reach their goals from their current position
	CMS	Cost for the *active teams* to reach their goal configurations and for all teams that dominate the active teams to complete their tasks
Collision set	M*	The robots that collide at v_k or a successor of v_k in the search tree and must consider alternative paths at v_k
	CMS	1. Conflict set: The teams that collide at v_k or a successor of v_k in the search tree 2. Coupled set: The active teams whose path impact the teams in the conflict set, and thus must consider alternate paths at v_k

Table 5.1: Differences between M* and CMS

sets of active teams, *i.e.*

$$G^{\text{aug}} = \bigcup_{\mathcal{T}_k^{\text{act}}} G^{\text{act}}(\mathcal{T}_k^{\text{act}}) \bigcup G^{\text{trans}}. \tag{5.1}$$

An active graph $G^{\text{act}}(\mathcal{T}_k^{\text{act}})$ describes the joint configuration space of a particular set of active teams. $G^{\text{act}}(\mathcal{T}_k^{\text{act}})$ is defined as the direct product of the manifold graphs corresponding to teams in $\mathcal{T}_k^{\text{act}}$,

$$G^{\text{act}}(\mathcal{T}_k^{\text{act}}) = \prod_{\tau^i \in \mathcal{T}_k^{\text{act}}} G^i_{\mathcal{M}}. \tag{5.2}$$

assuming that each vertex $v^i_{\mathcal{M}}$ in the manifold graph $G^i_{\mathcal{M}}$ is first replaced by the ordered pair $(\tau^i, v^i_{\mathcal{M}})$ to match the format of G^{aug}. The cost of an edge in $G^{\text{act}}(\mathcal{T}_i^{\text{act}})$

is the sum of the edge costs in the corresponding manifold graphs. Note that an active graph does not contain any edge which takes a team to its goal configuration for the first time, as such an action would alter the active teams[2]. Such edges are part of the transition graph instead.

The transition graph G^{trans} describes transition events where teams dissolve and new teams form. The vertex set of G^{trans} is the union of vertices in each active graph and the edges in G^{trans} connect different active graph components. An edge is added to the transition graph from a vertex v_k in two cases. In the first case, there is an active team $\tau^i_{tm} \in \mathcal{T}^{\text{act}}_k$ that has never reached its goal configuration and for which there is an edge from v^i_k to its goal configuration. The edge thus represents a transition from τ^i_{tm} to τ^i_{gl}. Such an action incurs the same cost as traversing the corresponding edge in the manifold graph of τ^i_{tm}. The second case is when the dissolving teams of a transition node in the task graph are all at their goal configuration. Those teams can take the transition action to form a new set of teams, resulting in a new set of active teams. A single edge in the transition graph may represent multiple such team transitions, while the teams that do not take the transition action move in their manifold graphs as normal. Combining the active graphs and the transition graph to form task augmented joint configuration graph produces a single graph that captures all possible actions and sets of active teams in a CPP problem.

5.4.2 Algorithmic description of CMS

In this section, we describe CMS as a variant M* that can solve the CPP problem. Recall that M* maintains an open list of vertices sorted by a lower bound on the cost of any path passing through that vertex. In every planning step, M* expands

[2] Recall that we distinguish between a team that has never reached its goal configuration and one that has reached its goal configuration at least once due to changes in dependencies between teams (Section 5.2.1)

the cheapest vertex v_k on the open list and adds its limited neighbors to the open list. The limited neighbors are generated by considering all possible actions for the robots in the collision set C_k of v_k, while the robots not in the collision set take the action specified by their individual policy. The process is repeated until the goal vertex is expanded, at which point an optimal path can be recovered. To solve the CPP problem, CMS searches the task augmented joint configuration graph while altering how the individual policies and the heuristic function are computed, and CMS replaces the collision sets with the conflict set and the coupled set to capture the more complicated dependencies between teams of robots. To minimize duplication this section assumes a good understanding of M*, which is described in detail in chapter 3.

M* computes a single individual policy for each robot. In a similar manner, CMS computes an individual policy ϕ_i for each team τ^i that maps the configuration of τ^i to the action that would move τ^i along an optimal path leading to its goal configuration if there were no other teams. If the team is at its goal configuration the individual policy is to take the team transition action, if the other dissolving teams are at their goal configurations, or to otherwise remain in place.

CMS modifies the sum of cost-to-go heuristic used by M* to account for the cost of sequentially performing multiple cooperative tasks. More specifically, let the cost of executing a task τ^i be the cost of following the individual policy ϕ_i of the team τ^i from the start configuration to the goal configuration of τ^i. Let the cost of finishing τ^i at a vertex v_k be the cost of following ϕ_i from the configuration of τ^i at v_k to the goal configuration of τ^i. The cost-to-go heuristic used by CMS is then the sum of the cost of completing the tasks of the active teams and the cost of executing the individual policies of all tasks that have not been started.

In M* the collision set of a vertex defines the set of robots for which planning must be locally coupled, i.e. the robots that are allowed to explore alternative actions to their individual policies. CMS defines collisions in terms of teams. Furthermore, the active teams at a state preceding the collision may not include the colliding teams, but the preceding teams can still influence the paths of the teams that collide.

To account for dependencies between successive teams, CMS breaks the collision set into two components: the conflict set, which contains the teams that have collided, and the coupled set, which describes the teams that must consider alternatives to the individual policy to avoid the collision. The conflict set C_k of a vertex v_k in the task augmented joint configuration graph is the set of teams that collide at v_k or some successor state in the search tree, keeping in mind that τ^i_{tm} is distinct from τ^i_{gl}. If a new collision is found at v_k then the resulting conflict set is added to all the vertices in the backpropagation set of v_k, in the same fashion that the collision set is backpropagated in M*. Any teams in the conflict set that are dominated by other teams in the conflict set are removed. Note that C_k may contain teams that are not active at v_k.

The coupled set Γ_k is the subset of the active teams $\mathcal{T}^{\mathrm{act}}_k$ at v_k which can influence the path of the colliding teams in C_k. Let $\mathrm{successor}\left(C_k\right) \subseteq V^{\mathrm{gl}} \bigcup V^{\mathrm{tr}}$ denote the set of the successors in the task graph of the teams in C_k. If a team has no successors, then it is treated as its own successor. Γ_k is then defined as

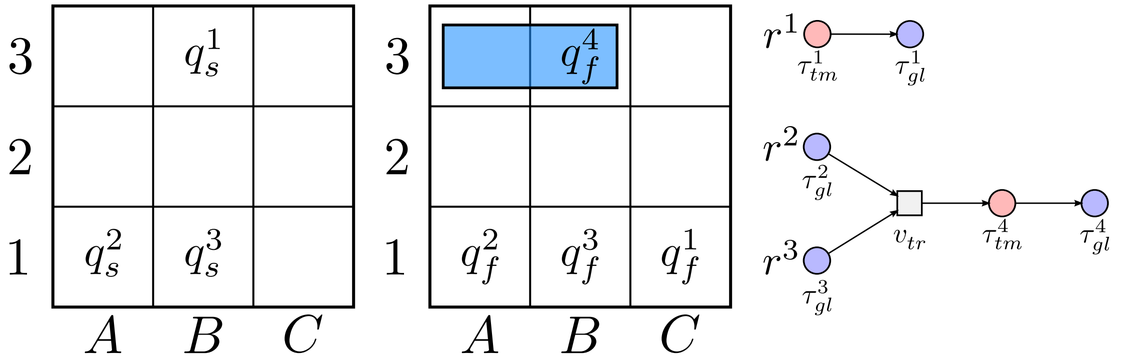
$$\Gamma_k = \left\{\tau^i \in \mathcal{T}^{\mathrm{act}}_k | \exists v_{\mathrm{task}} \in \mathrm{successor}\left(C_k\right) \text{ s.t. } \tau^i \leq v_{\mathrm{task}}\right\}, \tag{5.3}$$

If a team in the conflict set had not reached its goal configuration prior to the collision that added it to the conflict set $\tau^i_{tm} \in V^{\mathrm{tm}}$, the successor will be in V^{gl} and will have the same set of dominated teams as τ^i_{tm}. However, if the team had reached its goal configuration prior to the collision $\tau^i_{gl} \in V^{\mathrm{gl}}$, then its successor will be a transition vertex, which dominates all of the dissolving teams associated with the transition. This reflects the fact that the trajectory of a robot that has reached its goal configuration potentially depends on the trajectories of all the dissolving teams associated with its next transition, and thus indirectly on their predecessors (Section 5.2.1). Vertices in G^{aug} are returned to the open list whenever their coupled set changes, in the same fashion that M* returns previously expanded nodes to the open list when their coupled sets change.

Consider the case of the task graph given in figure 5.2. If teams τ^1 and τ^2 were to collide before τ^1 and τ^2 reached their goal configuration, then the conflict set would be $\{\tau^1_{tm}, \tau^2_{tm}\}$. If CMS were to backtrack to v_ℓ to consider an alternate path, then the coupled set would be the teams dominated by $\{\tau^1_{gl}, \tau^2_{gl}\}$, which is given by $\Gamma = \{\tau^1_{tm}, \tau^2_{tm}\}$. However, if the collision were to occur later, after τ^2 had reached its goal configuration, and thus where the active teams were $\{\tau^1_{tm}, \tau^2_{gl}, \tau^3_{tm}\}$, then the conflict set would be $\{\tau^1_{tm}, \tau^2_{gl}\}$. If CMS then backtracked to v_ℓ, the coupled set would be the teams dominated by $\left\{\tau^1_{gl}, v^a_{tr}\right\}$, which are $\{\tau^1_{tm}, \tau^2_{tm}, \tau^3_{tm}\}$. τ^3_{tm} would be added to the coupled set because τ^2_{gl} might be able to avoid the collision with τ^1 by taking the team formation action earlier, so τ^3_{tm} might need to take a more expensive, but shorter, path.

In M* the limited neighbors of a vertex in the joint configuration graph are determined by two constructs: the individual policies which determine the default action for the robots, and the collision set which determines which robots must consider actions beyond those generated by the individual policies. In a similar manner, the limited neighbors in CMS of a vertex $v_k \in G^{\mathrm{aug}}$ are determined by the individual policies of the active teams and the coupled set. The teams in the coupled set are allowed to take any action permitted by their manifold graph as well as the transition action, if applicable, while the rest of the active teams are restricted to the action generated by their individual policies. Note that if the conflict set and the coupled set are empty there is only one limited neighbor.

Note that a collision between two single robot teams could eventually result in

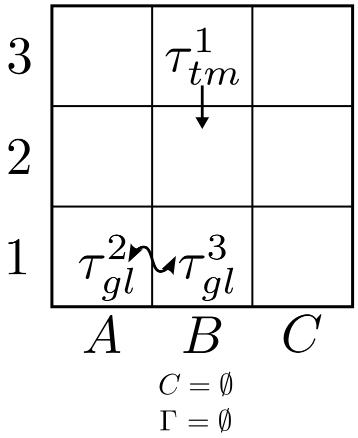


(a) Initial configuration (b) Goal configuration (c) Task graph

Figure 5.5: Initial and goal configurations of the teams in the example of CMS. **(a)** The three robots start as members of singleton teams with the same identifier. The initial configuration of τ^4 overlaps the start and goal configurations of τ^2 and τ^3, and so is omitted for clarity. The goal configuration of τ^2 and τ^3 are the same as the start configuration, so those teams reach their goal configuration immediately. τ^2 and τ^3 combine to form τ^4, which occupies two spaces **(b)**. τ^1 is assigned to move to the bottom right corner, while τ^4 is assigned to move up, until its right hand side occupies $B3$. **(c)** Task graph associated with the CPP problem.

coupled planning for all robots in the system at some predecessor vertex, especially if there are tasks with overlapping set of assigned robots. In such cases finding an optimal solution would be computationally very expensive. Inflating the heuristic function à la inflated M* (Section 3.5.2) biases search towards the final state of the system, which provides a soft limit on how far back in the search CMS will look for an alternate path around collisions, limiting the effective size of the coupled set. However, inflated CMS is ϵ-suboptimal and may return a path that costs up the ϵ times the cost of the optimal path, where ϵ is the inflation factor of the heuristic.

In practice, CMS is based on Operator Decomposition M* (ODM*) (Section 3.5.3), a variant of M* which replaces A* with Operator Decomposition (OD) [167] and which differs from M* only in implementation details.

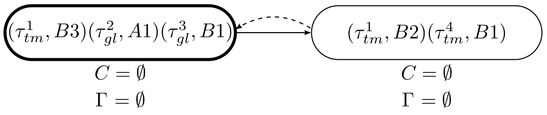


Neighbors of Expanded State
$[(\tau^1_{tm}, B2), (\tau^4_{tm}, B1)]$

Post Expansion Open List

Coordinate	f-value
$[(\tau^1_{tm}, B2), (\tau^4_{tm}, B1)]$	7

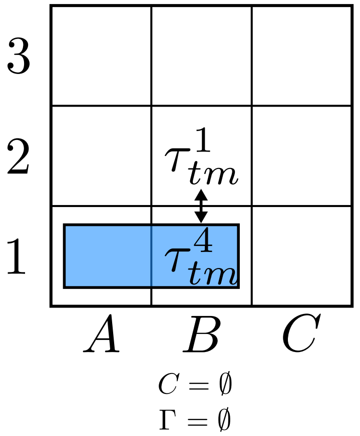
(a) Configuration at Step 1


$(\tau^1_{tm}, B3)(\tau^2_{gl}, A1)(\tau^3_{gl}, B1)$
$C = \emptyset$
$\Gamma = \emptyset$
$(\tau^1_{tm}, B2)(\tau^4_{tm}, B1)$
$C = \emptyset$
$\Gamma = \emptyset$

(b) Search tree after expansion.

Figure 5.6: **(a)** An example of the workings of CMS. The grid on the left shows the configuration that is expanded by CMS in step one, and the conflict and coupled sets of the configuration when it was expanded. The tables on the right enumerate the resulting limited neighbors, and the open list after the expansion is completed.

(b) In the search tree solid arrows point from a vertex to its successor states while dashed lines point from a vertex to the elements of its backpropagation set. The vertex expanded in step one is bolded. The conflict and coupled sets are given for each vertex in the search tree after expansion and conflict set backpropagation are completed.

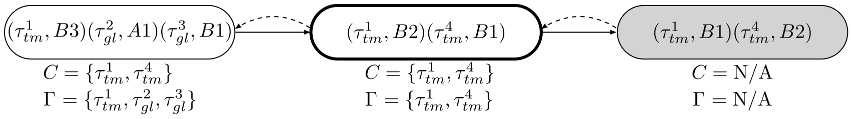


Neighbors of Expanded State
$[(\tau_{tm}^1, B1), (\tau_{tm}^4, B2)]$

Post Expansion Open List

Coordinate	f-value
$[(\tau_{tm}^1, B2), (\tau_{tm}^4, B1)]$	7
$[(\tau_{tm}^1, B3), (\tau_{gl}^2, A1), (\tau_{gl}^3, B1)]$	7

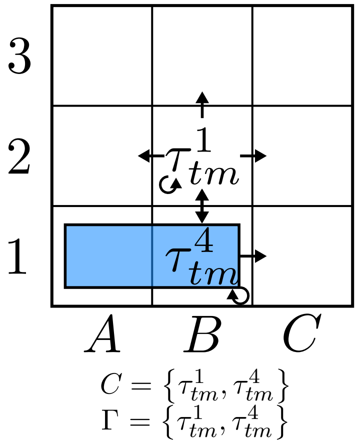
(a) Configuration at Step 2


$(\tau_{tm}^1, B3)(\tau_{gl}^2, A1)(\tau_{gl}^3, B1)$
$C = \{\tau_{tm}^1, \tau_{tm}^4\}$
$\Gamma = \{\tau_{tm}^1, \tau_{gl}^2, \tau_{gl}^3\}$
$(\tau_{tm}^1, B2)(\tau_{tm}^4, B1)$
$C = \{\tau_{tm}^1, \tau_{tm}^4\}$
$\Gamma = \{\tau_{tm}^1, \tau_{tm}^4\}$
$(\tau_{tm}^1, B1)(\tau_{tm}^4, B2)$
$C =$ N/A
$\Gamma =$ N/A

(b) Search tree after expansion.

Figure 5.7: **(a)** An example of the workings of CMS. The grid on the left shows the configuration that is expanded by CMS in step two, and the conflict and coupled sets of the configuration when it was expanded. The tables on the right enumerate the resulting limited neighbors, and the open list after the expansion is completed.

(b) In the search tree solid arrows point from a vertex to its successor states while dashed lines point from a vertex to the elements of its backpropagation set. The vertex expanded in step two is bolded. The conflict and coupled sets are given for each vertex in the search tree after expansion and conflict set backpropagation are completed. Grayed out vertices were never added to the open list due to a collision blocking the edge from their predecessor.


3
2
1
τ_{tm}^{1}
τ_{tm}^{4}
A
B
C
$C = \{\tau_{tm}^{1}, \tau_{tm}^{4}\}$
$\Gamma = \{\tau_{tm}^{1}, \tau_{tm}^{4}\}$

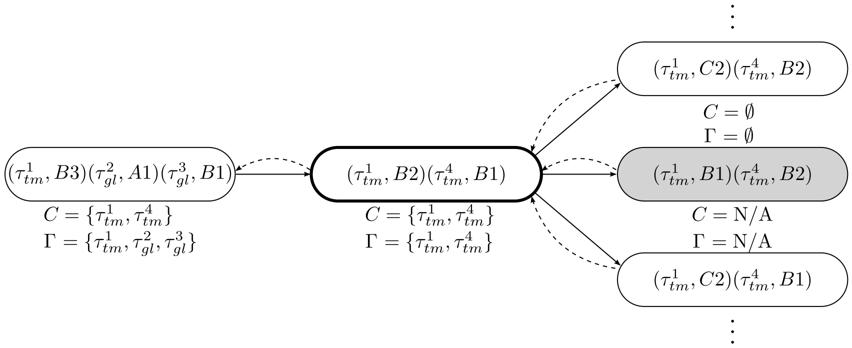
Neighbors of Expanded State

$[(\tau_{tm}^{1}, C2), (\tau_{tm}^{4}, B2)]$, $[(\tau_{tm}^{1}, B1), (\tau_{tm}^{4}, B2)]$,
$[(\tau_{tm}^{1}, B2), (\tau_{tm}^{4}, B2)]$, $[(\tau_{tm}^{1}, B3), (\tau_{tm}^{4}, B2)]$
$[(\tau_{tm}^{1}, A2), (\tau_{tm}^{4}, B2)]$, $[(\tau_{tm}^{1}, C2), (\tau_{tm}^{4}, B1)]$
$[(\tau_{tm}^{1}, C2), (\tau_{tm}^{4}, C1)]$, ...

Post Expansion Open List

Coordinate	f-value
$[(\tau_{tm}^{1}, C2), (\tau_{tm}^{4}, B2)]$	7
$\left[(\tau_{tm}^{1}, B3), (\tau_{gl}^{2}, A1), (\tau_{gl}^{3}, B1)\right]$	7
$[(\tau_{tm}^{1}, B3), (\tau_{tm}^{4}, B2)]$	9
$[(\tau_{tm}^{1}, C2), (\tau_{tm}^{4}, B1)]$	9
$[(\tau_{tm}^{1}, C2), (\tau_{tm}^{4}, C1)]$	11
⋮	⋮

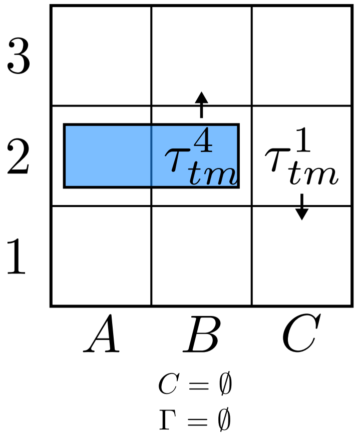
(a) Configuration at Step 3


$(\tau_{tm}^{1}, C2)(\tau_{tm}^{4}, B2)$
$C = \emptyset$
$\Gamma = \emptyset$
$(\tau_{tm}^{1}, B3)(\tau_{gl}^{2}, A1)(\tau_{gl}^{3}, B1)$
$C = \{\tau_{tm}^{1}, \tau_{tm}^{4}\}$
$\Gamma = \{\tau_{tm}^{1}, \tau_{gl}^{2}, \tau_{gl}^{3}\}$
$(\tau_{tm}^{1}, B2)(\tau_{tm}^{4}, B1)$
$C = \{\tau_{tm}^{1}, \tau_{tm}^{4}\}$
$\Gamma = \{\tau_{tm}^{1}, \tau_{tm}^{4}\}$
$(\tau_{tm}^{1}, B1)(\tau_{tm}^{4}, B2)$
$C = \text{N/A}$
$\Gamma = \text{N/A}$
$(\tau_{tm}^{1}, C2)(\tau_{tm}^{4}, B1)$

(b) Search tree after expansion.

Figure 5.8: **(a)** An example of the workings of CMS. The grid on the left shows the configuration that is expanded by CMS in step three, and the conflict and coupled sets of the configuration when it was expanded. The tables on the right enumerate the resulting limited neighbors, and the open list after the expansion is completed.

(b) In the search tree solid arrows point from a vertex to its successor states while dashed lines point from a vertex to the elements of its backpropagation set. The vertex expanded in step three is bolded. The conflict and coupled sets are given for each vertex in the search tree after expansion and conflict set backpropagation are completed. Grayed out vertices were never added to the open list due to a collision blocking the edge from their predecessor.

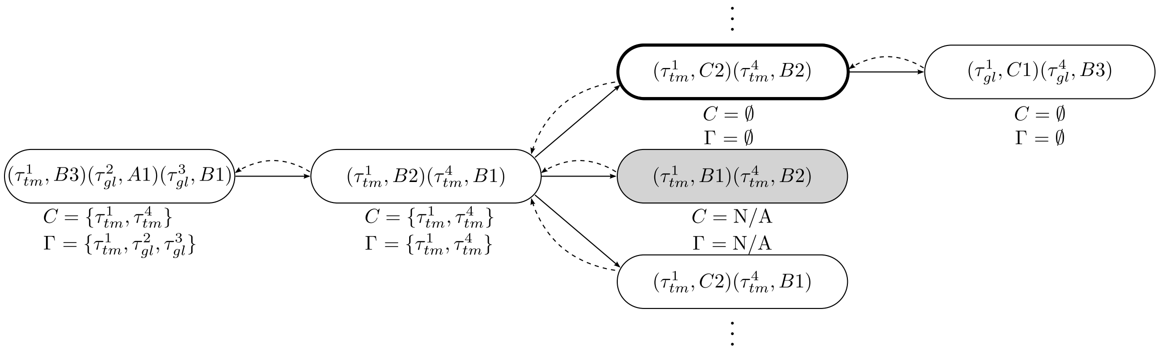


Neighbors of Expanded State
$[(\tau^1_{gl}, C1), (\tau^4_{gl}, B3)]$

Post Expansion Open List

Coordinate	f-value
$[(\tau^1_{gl}, C1), (\tau^4_{gl}, B3)]$	7
$[(\tau^1_{tm}, B3), (\tau^2_{gl}, A1), (\tau^3_{gl}, B1)]$	7
$[(\tau^1_{tm}, B3), (\tau^4_{tm}, B2)]$	9
$[(\tau^1_{tm}, C2), (\tau^4_{tm}, B1)]$	9
$[(\tau^1_{tm}, C2), (\tau^4_{tm}, C1)]$	11
⋮	⋮

(a) Configuration at Step 4


$(\tau^1_{tm}, C2)(\tau^4_{tm}, B2)$
$C = \emptyset$
$\Gamma = \emptyset$
$(\tau^1_{gl}, C1)(\tau^4_{gl}, B3)$
$C = \emptyset$
$\Gamma = \emptyset$
$(\tau^1_{tm}, B3)(\tau^2_{gl}, A1)(\tau^3_{gl}, B1)$
$C = \{\tau^1_{tm}, \tau^4_{tm}\}$
$\Gamma = \{\tau^1_{tm}, \tau^2_{gl}, \tau^3_{gl}\}$
$(\tau^1_{tm}, B2)(\tau^4_{tm}, B1)$
$C = \{\tau^1_{tm}, \tau^4_{tm}\}$
$\Gamma = \{\tau^1_{tm}, \tau^4_{tm}\}$
$(\tau^1_{tm}, B1)(\tau^4_{tm}, B2)$
$C = \text{N/A}$
$\Gamma = \text{N/A}$
$(\tau^1_{tm}, C2)(\tau^4_{tm}, B1)$

(b) Search tree after expansion.

Figure 5.9: **(a)** An example of the workings of CMS. The grid on the left shows the configuration that is expanded by CMS in step four, and the conflict and coupled sets of the configuration when it was expanded. The tables on the right enumerate the resulting limited neighbors, and the open list after the expansion is completed.

(b) In the search tree solid arrows point from a vertex to its successor states while dashed lines point from a vertex to the elements of its backpropagation set. The vertex expanded in step four is bolded. The conflict and coupled sets are given for each vertex in the search tree after expansion and conflict set backpropagation are completed. Grayed out vertices were never added to the open list due to a collision blocking the edge from their predecessor.

5.4.3 Example

We now present a simple example of how CMS operates. Consider a set of three robots r^1, r^2, and r^3. Each robot starts as the sole member of a singleton team, denoted τ^1, τ^2, and τ^3 respectively (Figure 5.5a). Teams τ^2 and τ^3 are assigned to immediately combine to form τ^4, so their initial states are also their goal states (Figure 5.5b). The initial state of the system is therefore $\left[(\tau^1_{tm}, B3), (\tau^2_{gl}, A1), (\tau^3_{gl}, B1)\right]$. τ^1 is assigned to move to the bottom-right corner while τ^4, which occupies two squares because it is composed of two robots, moves vertically. The goal state of the system as a whole is denoted $\left[(\tau^1_{gl}, C1), (\tau^4_{gl}, B3)\right]$. The corresponding task graph is given in Figure 5.5c. When there are multiple choices, the individual policies and tie-breaking between vertices on the open list are chosen to produce an informative but short example.

When CMS expands the initial state $\left[(\tau^1_{tm}, B3), (\tau^2_{gl}, A1), (\tau^3_{gl}, B1)\right]$ the conflict set and the coupled set are empty, because CMS has not found any collisions. Therefore, all teams follow their individual policy (Figure 5.6). τ^2_{gl} and τ^3_{gl} immediately take the transition action to form τ^4_{tm}. τ^1_{tm} moves down, towards its goal. The single limited neighbor defined by the individual policies is $[(\tau^1_{tm}, B2), (\tau^4_{tm}, B1)]$, which is added to the open list as it is free of collisions.

CMS expands $[(\tau^1_{tm}, B2), (\tau^4_{tm}, B1)]$ in the second step (Figure 5.7). The coupled set is still empty, so both teams obey their individual policies; τ^1_{tm} continues to move down while τ^4_{tm} moves up. The resulting limited neighbor is $[(\tau^1_{tm}, B1), (\tau^4_{tm}, B2)]$. The collision checking code indicates that τ^1_{tm} and τ^4_{tm} would collide head-on if they were to attempt to make that move. Therefore, $\{\tau^1_{tm}, \tau^4_{tm}\}$ is added to the conflict set of the preceding states in the search tree (Figure 5.7b), but the conflict set of $[(\tau^1_{tm}, B1), (\tau^4_{tm}, B2)]$ is not set, as the teams never actually reached that state. The coupled set is defined in terms of the conflict set, so when the conflict set changes, so does the coupled set, and thus the limited neighbors of a vertex. Because their coupled sets have changed, $[(\tau_{tm}^1, B2), (\tau_{tm}^4, B1)]$ and $\left[(\tau_{tm}^1, B3), (\tau_{gl}^2, A1), (\tau_{gl}^3, B1)\right]$ are placed back on the open list so that CMS can explore their new limited neighbors. The limited neighbor $[(\tau_{tm}^1, B1), (\tau_{tm}^4, B2)]$ is discarded, as it cannot be reached directly from $[(\tau_{tm}^1, B2), (\tau_{tm}^4, B1)]$.

The third step sees CMS expand $[(\tau_{tm}^1, B2), (\tau_{tm}^4, B1)]$ a second time (Figure 5.8). In this case, all the active teams are in the coupled set, so CMS must generate all possible neighbors. Several of the neighbors feature collisions between τ_{tm}^1 and τ_{tm}^4. The conflict sets of the predecessor states already contain τ_{tm}^1 and τ_{tm}^4 so no backpropagation is necessary, and the neighbors with collisions can be discarded immediately.

In the fourth step, CMS expands $[(\tau_{tm}^1, C2)(\tau_{tm}^4, B2)]$ (Figure 5.9). CMS has not yet explored a path from $[(\tau_{tm}^1, C2)(\tau_{tm}^4, B2)]$, so the conflict set and coupled set are once again empty. The single limited neighbor is $\left[(\tau_{gl}^1, C1), (\tau_{gl}^4, B3)\right]$, which is collision free, and thus added to the open list. $\left[(\tau_{gl}^1, C1), (\tau_{gl}^4, B3)\right]$ is the goal state of the system, and has the lowest f-value of any vertex in the open list. Therefore, CMS will expand the goal state in the fifth step of planning, indicating that CMS has found an optimal, collision free solution to the CPP problem.

Under a different choice of tie breaking for vertices in the open list $\left[(\tau_{tm}^1, B3), (\tau_{gl}^2, A1), (\tau_{gl}^3, B1)\right]$ could have been re-expanded before $[(\tau_{tm}^1, B2), (\tau_{tm}^4, B1)]$. If it had been, the limited neighbors of $\left[(\tau_{tm}^1, B3), (\tau_{gl}^2, A1), (\tau_{gl}^3, B1)\right]$ would have included all possible joint actions for τ_{tm}^1, τ_{gl}^2, and τ_{gl}^3, including joint actions where τ_{gl}^2 and τ_{gl}^3 combine to form τ_{tm}^4, and joint actions where team formation is delayed.

5.5 Recursive Constraint Manifold Subsearch

CMS couples planning for all teams in the coupled set, even when resolving multiple disjoint collisions. *Recursive Constraint Manifold Subsearch* (rCMS), like rM*, resolves collisions by identifying and solving independent *subproblems*. Solving each subproblem requires coupling fewer teams. Since the cost of planning is exponential in the number of coupled teams, rCMS can result in substantial reductions in planning time.

In rCMS, the conflict set is a set of *conflict set elements*, where each conflict set element initially contains a single set of mutually colliding teams. If the coupled sets defined by two conflict set elements are not disjoint, the conflict set elements are merged, and any team dominated by other teams in the conflict set element is removed.

A subproblem is defined by a conflict set element C^i, and is responsible for finding a path that avoids the collision that produced C^i. To this end, let the *resolve set* $\mathcal{T}_{res}\left(C^i\right)$ denote the set of teams upon whom the paths of the teams in C^i depend, i.e. the teams that contain a robot that later forms part of a team in C^i, or teams which a team in C^i depends upon to take the transition action[3].

$$\mathcal{T}_{res}\left(C^i\right) = \left\{\tau^j | \tau^j < \text{successor}\left(C^i_k\right) \vee \tau^j \leq C^i_k\right\}. \tag{5.4}$$

where $v^i_{\text{task}} < v^j_{\text{task}}$ implies $v^i_{\text{task}} \leq v^j_{\text{task}} \wedge v^i_{\text{task}} \neq v^j_{\text{task}}$. The subproblem itself is a copy of the full CPP problem where any team not in $\mathcal{T}_{res}\left(C^i\right)$ is *disabled*. A disabled team cannot collide with other teams, move, or incur cost, which implies that the contribution of a disabled team to the f-value is constant. As a result, the disabled teams have no influence on the paths of the teams in the resolve set. The subproblem is solved once every team is either disabled or is the final team for its constituent robots and is at its goal configuration. The solution is then the paths taken by the robots in the resolve set, with disabled teams ignored.

[3] Recall that each team τ may be represented by two task graph vertices; one before it has reached its goal set $\tau_{tm} \in V^{\text{tm}}$ and the other after it has reached its goal set at least once $\tau_{gl} \in V^{\text{gl}}$. Unless otherwise specified, τ^i may refer to τ^i_{tm} or τ^i_{gl}.

A subproblem defined by a conflict set element C_k^i can be queried from an arbitrary v_k in the task augmented joint configuration graph. Any team in the resolve set will behave normally, while the other teams will be placed at their position in v_k and then disabled. Thus, the solution to the subproblem from v_k depends only upon the teams in the intersection of the active teams and the resolve set, which is precisely the coupled set associated with C_k^i. As a result, the subproblems associated with the conflict set elements at v_k can be solved separately.

Each rCMS planner has an associated resolve set containing the teams that will not be disabled. The resolve set of the top-level rCMS planner contains all of the teams in the task graph. When a rCMS planner expands a vertex $v_k \in G^{\text{aug}}$, it checks how many conflict set elements C_k contains. If C_k contains no conflict set elements or a single conflict set element whose coupled set contains all the active teams that are not disabled, then the limited neighbors are computed as normal for CMS. Otherwise, the planner defines a subproblem for each conflict set element C_k^i, and computes a solution with a rCMS subplanner whose resolve set is $\mathcal{T}_{res}\left(C_k^i\right)$. A single limited neighbor is then generated wherein each team in the coupled set takes the first action in the solution to the appropriate subproblem. Teams in the resolve set that are not in the coupled set of any conflict set element follow their individual policies. All other teams are disabled and cannot take any actions. If any subproblem has no solution, then v_k has no limited neighbors. Otherwise, the resulting limited neighbor is guaranteed to either lie on the optimal path, or a path that contains a collision that would modify at least one conflict set element when the path is explored.

5.6 Completeness and Optimality

The proof that CMS is complete and will return minimal cost paths follows the same basic form as the proof for M* (Section 3.3.2). M* can be treated as alternating between running A* on a search graph and extending the search graph. Lemma 1 shows that if no solution exists, M* will terminate in finite time without returning a solution. Lemma 2 shows that M* will return the optimal path in finite time if the search graph always contains either the optimal path or an unexplored path to a collision that is no more expensive then the optimal path. Lemmas 3-5 prove lemma 2. Lemma 7 proves that the conditions assumed by lemma 2 always hold, supported by an auxiliary result that combining the optimal path for a subset of robots with the joint policy path for the complement produces a path no more expensive then the optimal path (Lemma 6). Taken together, these lemmas prove that M* is complete and will return the optimal path (Theorem 1).

Lemmas 1-5 all hold for CMS with minor alteration; the collision set must be replaced by the coupled set, and robots must be replaced by teams. Lemma 6 is not directly applicable to CMS as the task constraints mean that it is not possible to compute an optimal path for a subset of the robots. We therefore show that the paths generated by combining a solution for a subproblem defined by the conflict set with the joint policy path for the teams not in the subproblem produces a path that costs no more than the optimal path. With this result in hand, a modified version of lemma (Lemma 7) holds, proving that CMS is complete and will return optimal paths.

The following proof relies on the joint policy path costing no more than the optimal path, which is trivial in M*. However, in CMS this holds for the individual policies as defined only if the teams are able to wait at their goal configuration at zero cost. We therefore provide the following lemma.

Lemma 8. *If the task augmented joint configuration graph contains a solution, then the joint policy path $\pi_\phi(v_k, v_f)$ costs no more than the optimal path, $g(\pi_\phi(v_k, v_f)) \leq g(\pi_*(v_k, v_f))$.*

Proof. By construction, the individual policy for each team generates the cheapest path for a team from any configuration in its manifold graph to its goal configuration. Therefore, the joint policy path could only cost more than the optimal path if there were coordination costs incurred due to the timing of each team completing its task. However, waiting at the goal configuration is assumed to incur a team zero cost until all teams required to form the next set of teams are in position (Section 5.2). Therefore, the joint policy path can incur no coordination costs, and thus costs no more than the optimal path. □

Consider a subproblem defined for a conflict set, rather than the conflict set elements used by rCMS (Section 5.5). Such a subproblem may couple planning for teams involved in completely independent collisions, but is otherwise similar to those used in rCMS. We now show that a solution for a subproblem can be used to construct a path that costs no more than the optimal path and contains no collisions between the teams in the resolve set of the subproblem. For a given conflict set C_k, let $\pi'_{C_k}(v_k, v_f)$ be the path from v_k in the task augmented joint configuration graph to v_f constructed by taking the solution for the subproblem induced by C_k for the teams in the subproblem, and having all other teams follow their individual policies. Also, C_k is said to be dominated by C_ℓ, $C_k \leq C_\ell$ if every element of C_k is dominated by an element of C_ℓ.

Lemma 9. *If the task augmented joint configuration graph contains an optimal solution $\pi_*(v_k, v_f)$ from some vertex v_k in the task augmented joint configuration graph, then for any conflict set C_k a path $\pi'_{C_k}(v_k, v_f)$ can be constructed such that $g\left(\pi'_{C_k}(v_k, v_f)\right) \leq g\left(\pi_*(v_k, v_f)\right)$. Furthermore, if $C_k \leq C_\ell$, then $g\left(\pi'_{C_k}(v_k, v_f)\right) \leq g\left(\pi'_{C_\ell}(v_k, v_f)\right)$.*

Proof. If the full problem has an optimal solution, then a solution for the subproblem can be constructed by extracting the paths taken by the teams in the resolve set.

Therefore, the optimal solution for a subproblem costs no more than the path taken by the teams in the resolve set in the optimal solution for the full problem. The individual policies induce minimal cost paths for all teams not in the subproblem. Because the cost of a path is the sum of the costs of the paths of the individual teams, and no coordination costs can be incurred (Lemma 8), $g\left(\pi'_{C_k}\left(v_k, v_f\right)\right) \leq g\left(\pi_*\left(v_k, v_f\right)\right)$.

If every element of $C_k \leq C_\ell$, then the teams in the subproblem associated with C_k are a subset of the teams in the subproblem associated with C_ℓ. Therefore by the logic in the previous paragraph, $g\left(\pi'_{C_k}\left(v_k, v_f\right)\right) \leq g\left(\pi'_{C_\ell}\left(v_k, v_f\right)\right)$ □

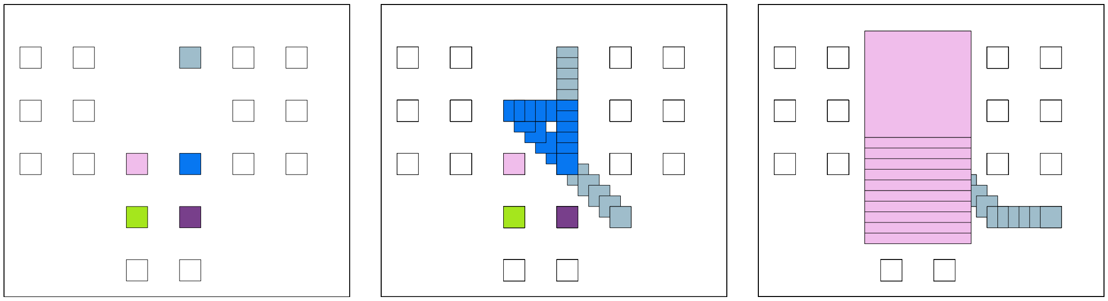
Lemma 10. *The search graph G^{sch} will always contain an optimal path (*i.e. *case 1 of lemma 2 will hold) or an unexplored path which costs no more than the optimal path (*i.e. *case 2 of lemma 2 will hold) at all points in the execution of CMS.*

Proof. By construction, if v_ℓ is the successor of v_k on $\pi'_{C_k}\left(v_k, v_f\right)$, then $C_\ell \leq C_k$ (Section 5.4). With the result of lemma 9, the proof then follows by analogy to the proof of lemma 7. □

Theorem 2. *CMS is complete and optimal.*

Proof. If the task augmented joint configuration graph G does not contain an optimal path, then CMS will terminate in finite time without returning an invalid path (Lemma 1 with slight modification). If G does contain an optimal path, then the search graph must always contain either the optimal path, or an unexplored path which costs no more than the optimal path (Lemma 10), which implies that then CMS will find the optimal path in finite time (Lemma 2 with slight modification). CMS will thus find the optimal path in finite time, if one exists, or terminate in finite time if no path exists. Therefore, CMS is complete and optimal. □

Theorem 3. *rCMS is complete and optimal.*



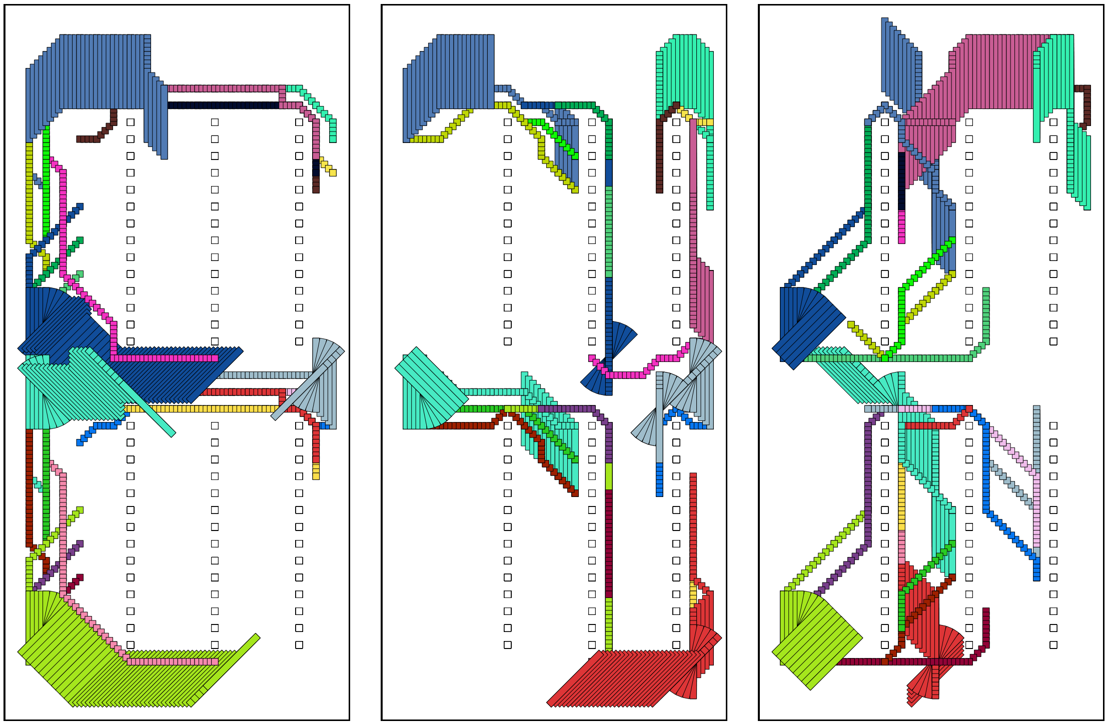
(a) Initial (b) Intermediate (c) Final

Figure 5.10: Small white squares are obstacles, while colored squares represent individual robots, and large squares represent teams. **(a)** The four robots that start at the bottom must transport a large square object to the top of the corridor. **(b)** However, the robots must delay starting the transport task to allow the single robot that starts at other end of the corridor to pass. **(c)** Once the single robot is clear, the transport task can be completed.

Proof. Lemma 9 holds with trivial modification when the path is constructed from the solutions of multiple subproblem, each defined by one of the conflict set elements of v_k. Lemma 10 then holds, given the observation that if v_ℓ is a successor of v_k in the search tree, then each conflict set element in C_ℓ is dominated by exactly one conflict set element in C_k. □

5.7 Results

We validate the performance of CMS and rCMS in simulation. The cooperative tasks take the form of moving large, rigid, rectangular loads. The load must be prevented from contacting any object aside from the robots carrying the load. The constraint manifolds corresponding to the tasks are diffeomorphic to SE(2). When a robot is in a singleton team it moves on an 8-connected grid where waiting as well as vertical and horizontal movement costs 1, while diagonal movements cost $\sqrt{2}$. A robot may wait for zero cost at its final destination or at the start configuration of its next task if the other robots assigned to the task are not yet in position. The manifold graphs



(a) $t \in [0, 20]$ (b) $t \in [20, 40]$ (c) $t \in [40, 60]$

Figure 5.11: Small white squares are obstacles, while colored squares represent individual robots and rectangles represent teams. Eight teams of three robots each must pick up rectangular loads from depots on the periphery, and deliver them to drop points between the rows of obstacles. Each team must make two such deliveries. Path segments are shown for three time windows, where the entire path is 121 units in duration. Please consult the uploaded video for a better depiction of the path.

for teams of multiple robots carrying a load are similar 8-connected grids where the cost movement or waiting is multiplied by the number of robots in the team. Teams can also take an action to rotate by ±45° at cost equal to the number of constituent robots. Actions to form or dissolve teams incur zero cost. To simplify the problem, the loads are assumed to be removed from the workspace after being delivered by a team.

We present the results of two specific simulation runs and a set of randomized trials. The first simulation demonstrates that CMS handles problems where task execution must be delayed. The second simulation is a larger, more realistic problem inspired by warehouse automation. All simulations were implemented in Python and run on a Intel Core i7 processor clocked at 3.30 GHz.

In the first simulation, four robots start at one end of a corridor opposite a single robot (Figure 5.10a). The four robots must carry a large, square object to the top of the corridor. However, while the team is carrying the load it cannot move out of the way of the single robot, which must reach the bottom of the environment. Therefore, the robots must delay executing the task until the single robot has cleared the corridor (Figure 5.10b). Furthermore, one of the robots that will carry the load must move out of the way of the single robot. Once the single robot has cleared the corridor, the transport task can be successfully executed (Figure 5.10c). CMS required 0.5 seconds to compute the optimal solution to this problem, while rCMS took 0.8 seconds. Given that there are no independent collisions, it is not surprising that rCMS takes slightly longer, as rCMS has more overhead.

The second simulation consists of eight teams of three robots each, that must pick up long rectangular loads from depots on the periphery of the workspace, and deliver them to positions between rows of obstacles (Figure 5.11). The robots may move independently when not carrying a load. Each team must deliver two loads before returning to their initial positions. Neither CMS nor rCMS can solve the problem optimally in under 5 minutes. Thus to compute a solution in reasonable time, an inflation factor of 1.2 was used. CMS took 20 seconds to compute the solution while rCMS took 6 seconds. In this problem, there are multiple independent sets of interacting teams, leading to rCMS outperforming CMS.

To investigate how CMS and rCMS scale with the number of robots and tasks, we generated randomized 80x80 grid worlds. Approximately 20% obstacle coverage was generated by randomly placing 320 2x2 obstacles with overlaps permitted. As in the simulated warehouse, tasks consisted of three robots carrying a long rectangular load between randomly chosen positions, with a given robot always teaming with the

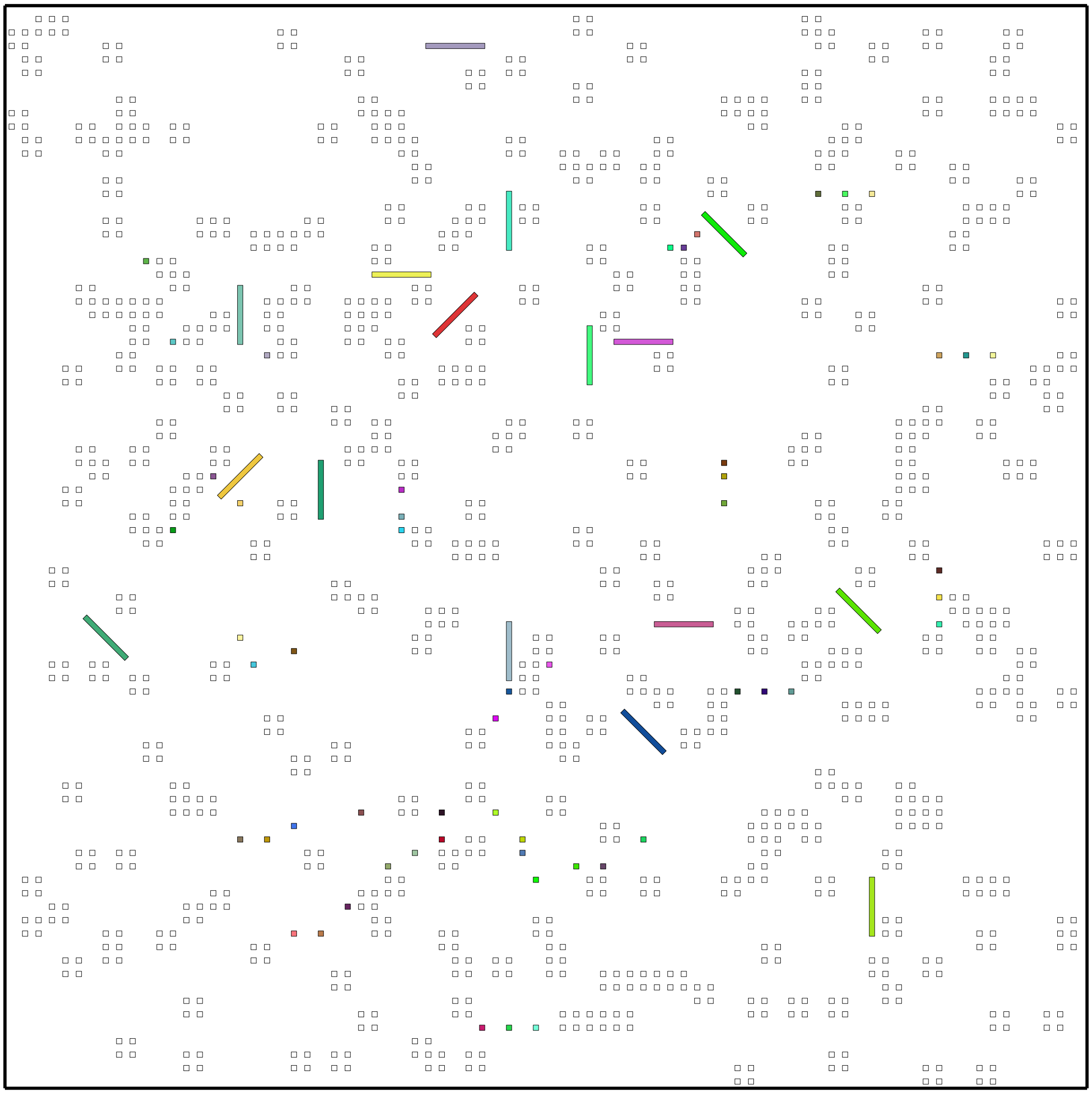


Figure 5.12: Typical random environment with 20% obstacle coverage. The empty squares are obstacles. The colored squares are individual robots, and the long rectangles teams of three robots carrying a heavy load. The configuration of teams is taken from halfway through a path found by inflated $rCMS$ with $\epsilon = 3$ for a problem involving 102 robots in 34 teams.

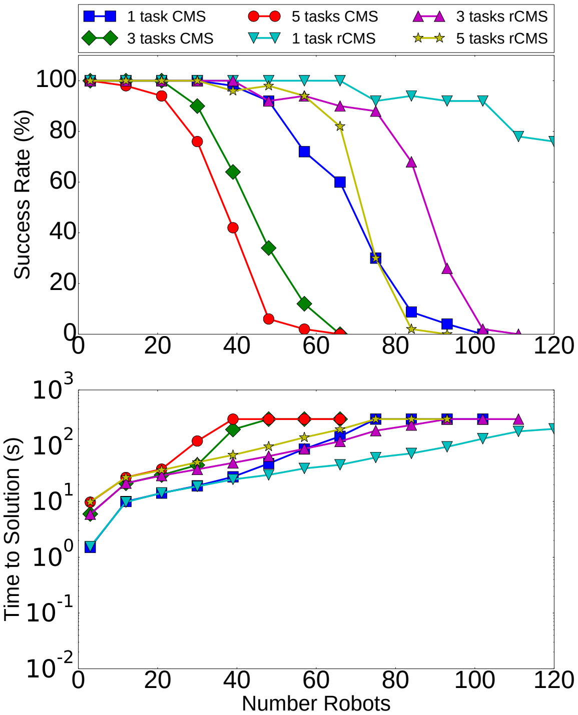


Figure 5.13: Comparison of CMS and rCMS performance on randomly generated 80x80 worlds. All trials used an inflation factor of 3. The top plot shows the percentage of trials for which a solution was found within 20 minutes, while the bottom plot gives the median time until a solution was found.

same robots. All tasks are feasible in isolation, and robots can always move from the end position of one task to the start configuration of the next tasks in the absence of other teams. When a robot was not actively carrying a load it was free to move independently. Randomized trials were generated for between 1 and 40 teams, with 50 such trials generated for each number of teams (Figure 5.12). CMS and rCMS were both run with an inflation factor of 3. Each trial was given 5 minutes to find a solution before the trial was marked as a failure (Figure 5.13). rCMS dramatically outperformed CMS with the success rate of rCMS on problems involving 3 to 5 tasks per robot equivalent to the performance of CMS with only a single task per robot (Figure 5.13).

We then tested the impact of varying the inflation factor of rCMS (Figure 5.14), with each robot assigned 3 tasks. As expected, rCMS was unable to solve anything

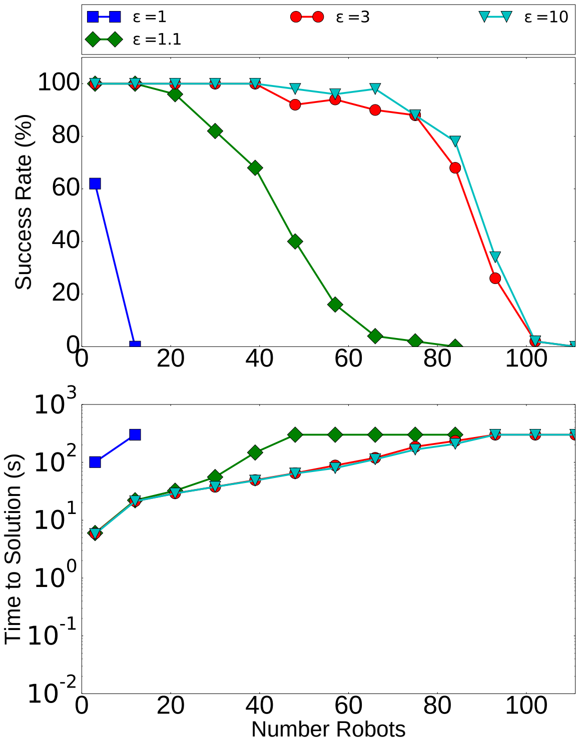


Figure 5.14: Performance of rCMS with varying inflation factors. Each team was assigned 3 tasks, in worlds with 20% obstacle coverage. The top plot shows the percentage of trials for which a solution was found within 5 minutes, while the bottom plot gives the median time until a solution was found.

but the simplest problems optimally ($\epsilon = 1$). When $\epsilon = 1$ almost any collision will force re-expansion of the root vertex of the search tree, where the coupled set has maximal size. Setting $\epsilon = 1.1$ sets a very soft limit to backtracking; to prevent incurring one extra unit of cost, rCMS would be willing to backtrack approximately $10/n$ steps in the search tree, where n is the number of robots that have not reached their final goal. When the inflation factor is increased to $\epsilon = 3$, rCMS will come close to greedily minimizing the heuristic value, primarily backtracking to resolve dead-ends rather than reduce path cost. rCMS would have to incur approximately $3 * n$ extra cost before it would be willing to backtrack one step. As a result, there is a substantial improvement in success rate, and rCMS is able to solve some problems involving 102 robots in 34 teams, where as with $\epsilon = 1.1$ rCMS was only able to

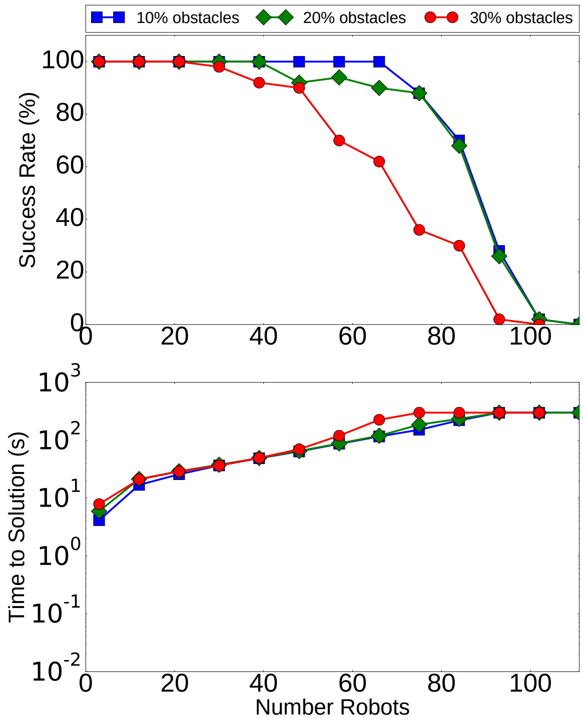
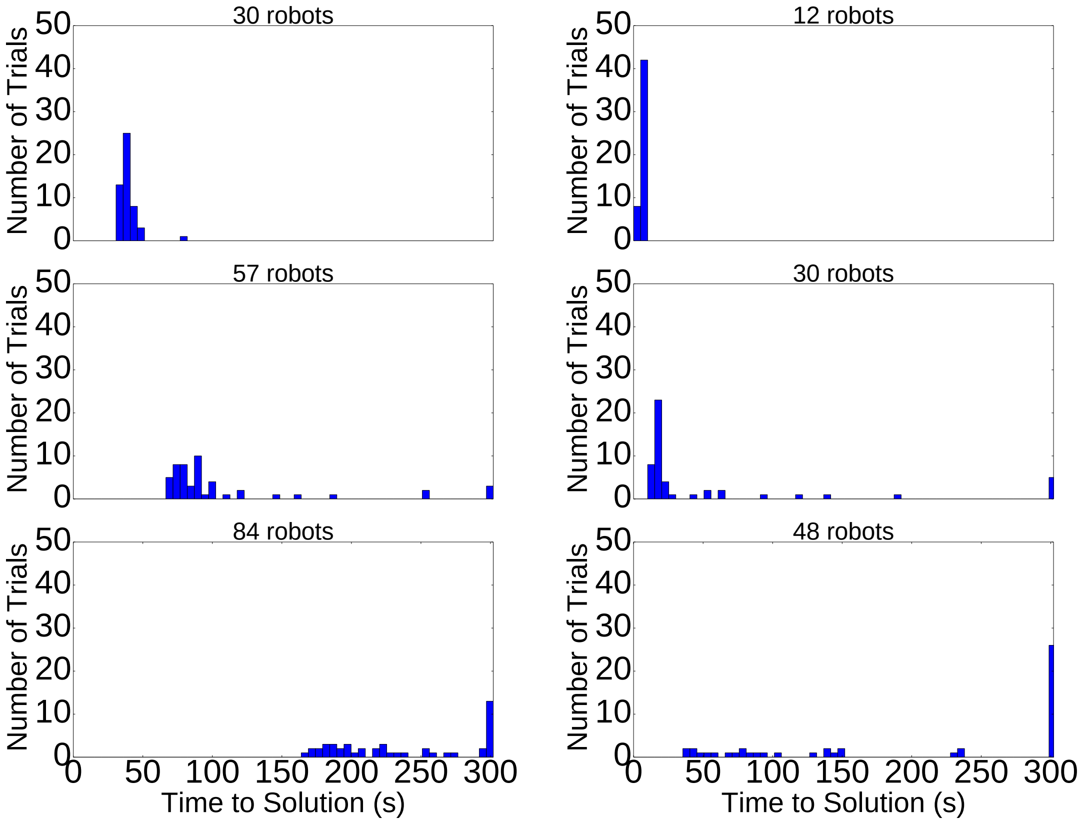


Figure 5.15: Performance of rCMS with varying obstacle densities in randomly generated 80x80 worlds. Inflated rCMS was run with $\epsilon = 3$ and three tasks per robot. The top plot shows the percentage of trials for which a solution was found within 5 minutes, while the bottom plot gives the median time until a solution was found.

solve systems with 75 robots in 25 teams. Increasing ϵ to 10 results in little change in performance, as rCMS is already operating in a close to greedy manner. These results track closely the observed behavior of rM* (Figure 3.8).

rCMS was then tested in environments with 10%, 20%, and 30% obstacle coverage with $\epsilon = 3$ and 3 tasks per robot(Figure 5.15)[4]. There was less variation in run time than anticipated, as M* has previously been observed to suffer when confronted with narrow bottlenecks through which bidirectional traffic flows. However, the difficulty increase from 10% to 20% obstacle coverage is noticeably smaller than the difficulty increase from 20% to 30% obstacle coverage, which may be due to the formation of bottlenecks.

[4] Attempts to generate random worlds with 40% obstacle density failed as the probability of randomly chosen task start and goal locations falling within the same connected component was too low.



(a) 80x80 world

(b) 40x40 world

Figure 5.16: Histogram of time to solution for rCMS with $\epsilon = 3$ and 20% obstacle coverage and three tasks per robot in a 80x80 world **(a)** and in a 40x40 world **(b)**.

In M*, the median time to solution typically grows gradually as the number of robots increases, before hitting an inflection point and growing rapidly (Section 3.7). We interpreted this behavior as the system hitting a critical density of robots that forced the dimensionality of the search space high enough that M* no longer could find solutions. Only the weakest versions of CMS (CMS and rCMS with $\epsilon = 1$ and $\epsilon = 1.1$) showed inflection points in the tested environments. This suggests that the environments were too large for the more powerful planners to hit critical density before timing out, and suggests that increasing the time limit may permit a significant number of additional problems to be solved. The 30% obstacle density case shows weak evidence of an inflection point (Figure 5.15), and being the test point with the highest robot density, supports this interpretation. Furthermore, the histogram of time to solution for rCMS with $\epsilon = 3$ (Figure 5.16a) shows that the distribution of times overlaps with the time limit when the success rate has significantly dropped, whereas in inflated ODrM* there is always a clear separation between the time to solution for feasible problems and unfeasible problems (Figure 3.11). This further suggests that rCMS with larger inflation values encounters problems that simply take longer to solve, rather than problems where the majority exceed a critical density. When rCMS was run on a 40x40 world with a quarter the area of the standard 80x80 world there was a stronger separation between successful and unsuccessful runs, suggesting that in smaller worlds rCMS is more likely to hit a critical density.

5.8 Conclusion and Future Work

We presented the Constraint Manifold Subsearch, a new algorithm for solving the cooperative path planning problem. CMS can compute optimal or ϵ-suboptimal paths for systems with large numbers of robots that can perform cooperative tasks. We also presented rCMS, a variant of CMS that decouples planning for disjoint sets of interacting teams, and show that rCMS significantly outperforms CMS.

There are a number of avenues for future work. CMS has been implemented for only a single cooperative task: multiple mobile robots carrying a single object. In the future, we will pursue techniques for automatically constructing the constraint manifold [21, 161, 42, 171] to apply CMS to more types of tasks. Of special interest is the task of cooperatively carrying an object through an environment that is sufficiently cluttered that robots must disconnect and reconnect to the object clear obstacles [112], potentially resting the load on the ground temporarily.

To simplify the problem, we assumed that actions of teams do not impact the environment, even when the teams move large objects. We believe that this assumption can be removed by treating environmental features that may be manipulated by robots as dummy teams that contain no robots. CMS will then naturally account for dependencies between the teams that manipulate the environmental features and the teams that may be impeded by the environmental features.

Finally, CMS currently assumes that each team can wait at its goal configuration for zero cost, which is necessary to properly decouple planning between teams. We intend to generalize CMS to allow robots to incur cost at goal configurations. We believe that a proper analysis of lower bounds on arrival times of different teams at their goal configurations can reduce coupling, especially when combined with inflation and other manipulation of the heuristic function.

Chapter 6

Planning With Uncertainty

Robots will not precisely follow even the best of plans due to imperfect control and actuation, as well as unknown environmental features and noise. There is thus a need for plans which can be executed even in the face of significant uncertainty. We term the problem of finding paths for multiple robots in the face of uncertain dynamics the *Multirobot Path Planning with Uncertainty* (MPPU) problem. Models for uncertain systems fall into two broad categories: non-deterministic and probabilistic. The dynamics of a non-deterministic system describe the set of states the system could occupy at a given time, but give no indication of the likelihood of any given state. As such, non-deterministic models are useful for systems whose dynamics can be bounded, but for which a detailed model is not available or not reliable [59, 104, 114]. However, non-deterministic models force a conservative approach, as they cannot differentiate between a very low probability collision, which could be safely neglected, and a high probability collision which must be avoided. Probabilistic models maintain a belief state for the system which is a probability distribution over the system's configuration space. Probabilistic models require a more detailed dynamics model, but admit constraints that can accept a low probability of failure, which may be necessary to achieve a high-quality path, or any path at all. Although probabilistic models are a subset of non-deterministic models, algorithms designed to solve probabilistic systems can be used to solve non-deterministic problems[1], while algorithms intended for non-deterministic problems may fail on probabilistic problems. Therefore, our primary focus in this chapter will be on probabilistic systems.

The most general approach to planning with uncertainty is to cast the problem as a *Partially Observable Markov Decision Process* (POMDP) [83, 102, 140]. Solving the POMDP returns a policy that provides the optimal action for any state-observation pair. While POMDPs have been successfully applied to the MPPU problem [184, 123], solving the full POMDPs is prohibitively expensive for all but the smallest problems [129].

MPPU algorithms seek to approximate the full POMDP formulation while minimizing computational cost and maximizing the quality of the resulting plan. Three basic approaches to the MPPU problem are dynamic planning, interaction regions, and belief space planning.

Dynamic replanning approaches assume that uncertainty is insignificant over short periods of time, and the uncertainty in the results of any single action have only local consequences. Under these conditions uncertainty can be handled by taking sensor measurements to observe the result of taking a small number of actions, then replanning without explicitly accounting for uncertainty based on the observed state of the system. Several early approaches for single robots in dynamic environments computed a velocity that would take the robot towards the goal while maximizing the time the robot could maintain that velocity without colliding with a obstacle [67, 162]. D* and similar algorithms based on RRTs [27, 63, 92, 110, 124, 169] were developed to allow for rapid, global replanning for single robots in an unknown or partially

[1] To adapt a probabilistic algorithm to a non-deterministic system, assign a uniform probability distribution to the feasible states of the non-deterministic system and set thresholds to not accept any probability of failure

known world. Bruce and Veloso [28] applied a replanning approach to multirobot systems, combining separate global path planning with one-step look-ahead velocity selection. Safety was guaranteed by requiring that each robot be able to come to a safe stop by breaking with maximal effort immediately after finishing its next planned action. Dynamic replanning approaches will fail when the uncertainty associated with a single action is significant, or can lead to two qualitatively different results with lasting consequences.

An alternate approach specific to multirobot systems is to identify a limited set of interaction regions where robots may interact, and thus robots only require coordination when in the interaction region. In reservation based approaches, the planner reserves the interaction regions for specific robots, and requires other robots to wait safely outside the interaction region until the robot that has the region reserved has passed through [55, 65, 128, 196, 201]. Reservation based approaches generally do not consider uncertainty in the path of single robots, and may lead to robots waiting for prolonged periods of time if a robot with a reservation for an interaction region is delayed.

Melo and Veloso [119] exploited interaction regions to produce a simplified version of the full POMDP formulation, called Decentralized Sparse-Interaction Markov Decision Process (DEC-SIMDP). They constructed a separate POMDP for each robot r^i. The focal robot r^i could observe its full state, but could only observe the state of another robot when both r^i and the other robot were in the same interaction region. All robots aside from r^i were assumed to obey a fixed policy generated by solving either single agent MDP or a multiagent MDP for all robots aside from r^i. As a result, the policy of r^i only directly depended upon the position of the other robots when in the interaction regions, substantially the computational cost of computing a policy.

Solving a DEC-SIMDP produces a policy and as a result can handle bimodal distributions, such as when the robot may be in one of two different passages requiring different paths. The policy will also base its commanded action upon sensor measurements received during execution, and as a result will not cause a robot to wait outside an interaction region for a robot that was significantly delayed. However, like reservation based systems DEC-SIMDPs are only applicable to environments containing a low number of small, well defined interaction regions. This is most likely to be the case for structured environments like road networks, hallways, or warehouses.

A final approach is to plan paths in the belief space of the system. The belief space of a system is the space of probability distributions over the configuration space of the system. The belief state of a single robot can be represented as a Gaussian. An extended Kalman filter is often used to propagate the Gaussian belief state along a given path and account for expected measurements. Planning can be done using A* or RRTs [72, 141]. Extensions account for non-maximal likelihood measurements [189], the delay between receiving an updated position measurement and the robot correcting for error [29], and the truncation of the belief state due to possible collisions with obstacles [131, 173]. Other approaches allow for non-Gaussian belief spaces [118, 138]. van den Berg et al. [189] used a priority planner (Section 1.2.5) to perform belief space planning for a multirobot system, but did not account for the impact of interactions between robots on the belief states of the robots. Belief space path planning is more efficient than solving the full POMDP because belief space planning computes a single nominal trajectory in belief space, rather than a full control policy. As a result the paths produced by belief space planning will not react to sensor measurements received during execution, unlike the policy generated by solving a POMDP or DEC-SIMDP. Thus if a belief space plan calls for r^1 to wait for r^2 to pass through a bottleneck, r^1 will wait for r^2 even if r^2 was delayed and r^1 could safely navigate the bottleneck. However, belief space planning does not require well defined interaction regions, and thus is applicable to cluttered environments where interaction regions are either ill-defined or too numerous for reservation or DEC-SIMDP based approaches.

In this chapter, we analyze the MPPU problem and show that it only approximately has the direct product structure relied upon by most MPP algorithms. We then present *Uncertainty M** (UM*), an extension of M* to handle the MPPU problem. UM* uses subdimensional expansion to efficiently explore the *joint belief space* of a multirobot system. Because the MPPU problem lacks a direct product structure, UM* is not complete or optimal. However, unlike decoupled algorithms (Section 1.2.5) which may fail to find solutions for realistic problems, UM* will only fail to find solutions in contrived cases. We introduce a non-Gaussian belief space representation that is appropriate for MPPU. UM* is then compared in simulation to several alternate approaches to MPPU and is shown to work well when complete plans are required.

6.1 Multirobot Path Planning with Uncertainty

The MPPU problem seeks to find paths for systems of multiple robots whose location and dynamics are uncertain. The system is described by joint belief states $b : Q \to \mathbb{R}^{\geq 0}$ in the joint belief space $\mathcal{B}$ of probability distributions over the joint configuration space of the system. The objective of the MPPU problem is to find an optimal path $\pi_* \left(b_s, b_f\right)$ for a system of n robots $r^i, i \in I = \{1, \dots, n\}$ from an initial joint belief state[2] b_s to some final belief b_f whose probability density is sufficiently concentrated in the goal region. The cumulative probability that each robot collides with other robots along the path leading to b_f must be below a threshold value, *i.e.* $P^i_{\text{cumulative}} \left(b_f\right) \leq \delta_{\text{col}}\ \forall i \in I$. To simplify the problem, we assume that if two robots collide both are removed from the workspace, so each robot can collide only once. Otherwise a

[2] If the robots start perfectly localized, then b_s is a delta function

Symbol	Meaning
col^i	An auxiliary state added to the configuration space of robot r^i to represent that robot having collided with another robot or obstacle
$Q^i = Q^i_{\text{nat}} \cup \{col^i\}$	The configuration space of a robot r^i is formed by taking the normal configuration space of the robot Q^i_{nat} and adding a configuration to represent r^i having collided
$\mathcal{B} = \left\{b : Q \to \mathbb{R}^{\geq 0}\right\}$	Joint belief space of the multirobot system
$b \in \mathcal{B}$	Joint belief of the full multirobot system
$\mathcal{B}^i = \left\{b^i : Q^i \to \mathbb{R}^{\geq 0}\right\}$	Belief space for robot r^i
$b^i \in \mathcal{B}^i$	Belief for robot r^i. If b is a joint belief, then b^i denotes the marginal probability of the state r^i in b
$P^i_{\text{cumulative}}(b) = b^i(col^i)$	Cumulative probability of r^i colliding with another robot along the path leading to b
δ_{col}	maximum acceptable cumulative collision probability
$\Psi : Q \to \mathcal{P}(I)$	Collision function that returns the set of robots that collide at a given joint configuration

Table 6.1: Uncertainty M* notation

detailed model of the collision dynamics would be required. A path is optimal if it minimizes a cost function $g\left(\pi\left(b_s, b_f\right)\right) = \sum_i g^i\left(\pi^i\left(b^i_s, b^i_f\right)\right)$. The cost function is chosen so that the cost of a robot performing a given action is independent of the probability of the robot having collided prior to taking said action.

A probability distribution over the configuration space of a robot r^i does not capture the probability that r^i has collided with another robot and is ill defined if the robot may have been removed from the workspace. Therefore for the purpose of the MPPU problem the configuration space Q^i of r^i is augmented by an additional state, col^i, which represents r^i having collided with another robot. The system as a whole has the joint configuration space $Q = \prod_i Q^i$. The joint belief space of the system is then the space of probability distributions

$$\mathcal{B} = \left\{ b : Q \to \mathbb{R}^{\geq 0} | \int_q b\left(q\right) \mathrm{d}q = 1 \right\}. \tag{6.1}$$

The belief state of a single robot $b^i : Q^i \to \mathbb{R}^{\geq 0}$ is the marginal distribution of the state of r^i in a given joint belief b. The cumulative probability of r^i having collided with another robot along any path leading to b is then $P^i_{\text{cumulative}}\left(b\right) = b^i\left(col^i\right)$.

Each robot r^i can take a set of actions $\mathcal{A}^i$. The local belief dynamics $\mathcal{D}^i : Q^i \times \mathcal{A}^i \to \mathcal{B}^i$ gives the belief state of r^i after it takes an action while perfectly localized at some position, subject to

$$\int_{q^i_\ell \in Q^i} \mathcal{D}^i\left(q^i_k, a^i\right)(q^i_\ell)\mathrm{d}q^i_\ell = 1 \tag{6.2}$$

$$\mathcal{D}^i\left(col^i, a^i\right)\left(q^i_\ell\right) = \begin{cases} 1 & q^i_\ell = col^i \\ 0 & q^i_\ell \in Q^i \setminus \left\{col^i\right\} \end{cases}. \tag{6.3}$$

The belief dynamics $\mathrm{Dyn}^i : \mathcal{B}^i \times \mathcal{A}^i \to \mathcal{B}^i$ of r^i that describe the evolution of a belief over time, neglecting other robots, are given by

$$\mathrm{Dyn}^i\left(b^i, a^i\right) = \int_{q^i \in Q^i} b^i\left(q^i\right)\mathcal{D}^i\left(q^i, a^i\right)\mathrm{d}q^i \tag{6.4}$$

If there were no collisions then the joint belief dynamics for the system as a whole would be

$$\mathrm{Dyn}_{\text{nocol}} : \mathcal{B} \times \mathcal{A} \to \mathcal{B} \tag{6.5}$$

$$\mathrm{Dyn}_{\text{nocol}}\left(b_k, a_k\right)\left(q_\ell\right) \mapsto \int_q b_k(q) \prod_i \mathcal{D}^i\left(q^i, a^i_k\right)\left(q^i_\ell\right) \tag{6.6}$$

The collision free joint belief dynamics must be corrected to account for robot-robot collisions. Let $\Psi : Q \to \mathcal{P}(I)$ map from a position in the joint configuration space to the robots that would collide at that point[3]. Now let $\phi : Q \to Q$ map a configuration q_k to a new configuration similar to q_k where every robot r^i that collides at q_k is moved to col^i

$$\phi(q_\ell) \mapsto \prod_i \begin{cases} q_\ell^i & r^i \notin \Psi(q_\ell) \\ col^i & r^i \in \Psi(q_\ell) \end{cases} \tag{6.7}$$

Next define a kernel that maps the probability mass at a configuration q to $\phi(q)$

$$K : Q \times Q \to \mathcal{B} \tag{6.8}$$

$$K(q_k, q_\ell) \mapsto \delta(q_k - \phi(q_\ell)) \tag{6.9}$$

where δ is the Dirac delta function and K maps a configuration q_ℓ to a belief that is a delta function centered at $\phi(q_\ell)$. The joint belief dynamics for the system as a whole can now be written

$$\mathrm{Dyn} : \mathcal{B} \times \mathcal{A} \to \mathcal{B} \tag{6.10}$$

$$\mathrm{Dyn}(b_k, a_k)(q_\ell) \mapsto \int_q K(q_\ell, q)\, \mathrm{Dyn}_{\mathrm{nocol}}(b_k, a_k)(q)\, \mathrm{d}q \tag{6.11}$$

Joint belief dynamics of the form of 6.10 have several important properties. The belief dynamics of a given robot r^i are independent from the belief dynamics of any robot with which it does not collide. The joint belief dynamics are not the Cartesian product of the individual robot belief dynamics. Requiring that the belief dynamics depend on local dynamics (Equation 6.5) ensures that the behavior of the robot depends only upon its state, and not upon its belief, which is non-physical. Finally, the belief dynamics are conservative, so the preimage of any open set of a belief must contain at least as much probability mass as the open set.

[3] $\mathcal{P}(I)$ denotes the power set of I, *i.e.* the set of all subsets of I

Note that according to the above definitions, a robot could potentially be said to collide with itself. Collisions between the robot and the environment which may occur if a robot has imperfect localization can be represented as a self-collision.

6.1.1 Structure of the MPPU problem

The MPP problem as defined in this thesis has a direct product structure where both the joint configuration space and the joint dynamics are the Cartesian product[4] of the single robot configuration spaces and dynamics respectively. In M*, this assumption was embodied by the fact that the joint configuration graph is the direct product of the configuration graphs of the individual robots (Section 3.1), with collisions effectively setting the cost of some vertices to be infinite (impassable). The direct product structure means that if a robot can reach some individual configuration as part of a team, it can reach that configuration by itself or as part of any subteam. As a result, if no solution exists for a MPP problem involving a set of robots Ω, then no solution exists for a problem involving a superset of the robots $\Omega' \supset \Omega$. This permits MPP algorithms to resolve collisions by altering the paths of only those robots directly involved in the collision.

The MPPU problem lacks the direct product structure of the MPP problem, as collision checking couples the dynamics of the individual robots. Consider a single robot r^1 in an obstacle-free environment. Since no robot-robot collisions are possible, r^1 will never reach a belief where $b^1\left(col^1\right) > 0$. However, if a second robot r^2 is added, then collisions can occur, and r^1 may reach a belief with $b^1\left(col^1\right) > 0$. Therefore, the reachable configuration space of a robot or system of robots may actually expand when additional robots are added.

The above concern may seem esoteric, but it is possible to construct a system

[4] If $f : A \to X$ and $g : B \to Y$, then $f \times g : A \times B \to X \times Y$ and $(f \times g)(a, b) \mapsto (f(a), g(b))$

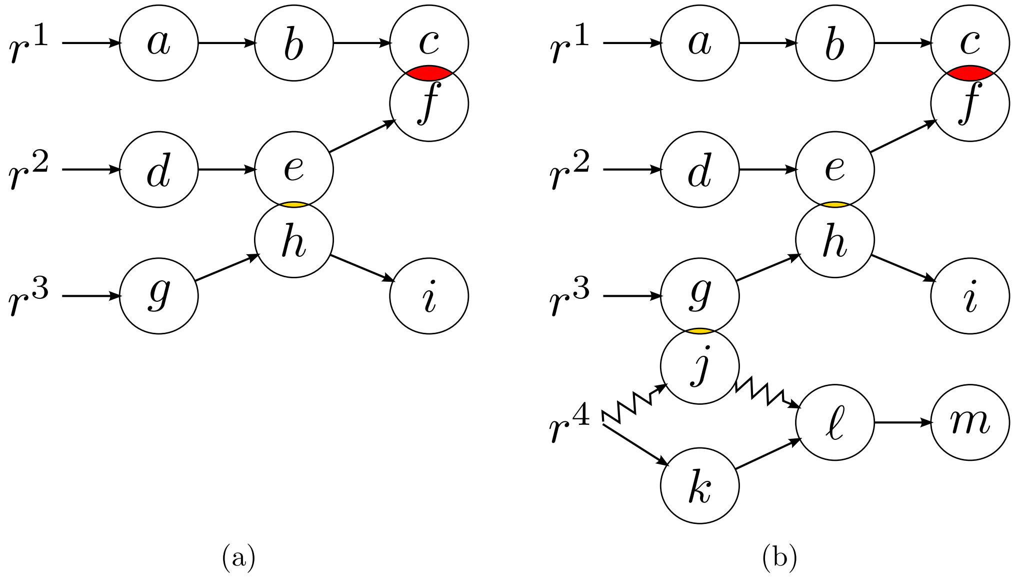


Figure 6.1: **(a)** A three robot problem with a collision threshold of $\delta_{\text{col}} = 0.59$. Each robot has only a single path. There is a 20% chance of a collision if states e and h are occupied simultaneously, and a 50% chance of collision if states c and f are occupied simultaneously. There is no solution, as $P^2_{\text{cumulative}}\left(b_f\right) = 0.6$. In **(b)** a fourth robot is added with two paths: an expensive path passing through j and a cheap path passing through k. There is a 20% chance of collision if states g and j are occupied simultaneously. The only solution is when r^4 chooses the more expensive path, at which point $P^2_{\text{cumulative}}\left(b_f\right) = 0.58$, just below the threshold value.

where a robot must go out of its way to collide with a second robot for a solution to exist. Consider a system of three robots with a collision threshold of $\delta_{\text{col}} = 0.59$, *i.e.* there can be at most a 59% chance of a given robot colliding with other robots (Figure 6.1a). Each robot has only one path it can follow, *i.e.* the action set for each robot at every configuration contains a single element. There is a 20% chance of a collision between r^2 and r^3 when they simultaneously occupy states e and h. If r^2 does not collide with r^3, then there is a 50% chance of a collision between r^2 and r^1 while occupying states c and f. As a result, r^2 has the highest cumulative probability of collision, at $P^2_{\text{cumulative}}\left(b_f\right) = 0.6$, which is above the collision threshold of 0.59. Therefore the problem as stated has no solution.

Now consider adding a fourth robot r^4 to the problem (Figure 6.1b). r^4 has two possible paths: an expensive path passing through state j and a cheap path passing through state k. If r^4 takes the cheaper path, it will not collide with any other robot, and thus r^2 will violate the constraint on cumulative collision probability. However, if r^4 takes the more expensive path, there is a 20% probability at state j that it will collide with r^3. Because r^3 might collide with r^4, there is now only a 16% chance that r^3 will collide with r^2 at state h. Propagating the beliefs forward, the cumulative probability of collision of r^2 at the end of the path will now be 0.58, which is below the threshold. Therefore to a resolve an interaction between robots r^1, r^2 and r^3, it is necessary for r^4 to choose an expensive path containing a possible collision over a cheaper, collision-free path. This is fundamentally a consequence of the joint configuration space not being a direct product.

As a result, most complete MPP algorithms will either be inapplicable to the MPPU problem, or lose completeness guarantees. EPEA* and OD are exceptions as they perform exhaustive search of the entire joint configuration space, but will be very inefficient. That said, completeness will only be lost in very unusual cases, where a robot must choose to collide with another robot, to reduce the probability that the second robot will collide with a third robot. Such situations are unlikely to appear in realistic problems, unlike priority planners cannot reposition robots in a dead-end corridor, a realistic problem.

The paradoxical requirement that robots deliberately collide with one another to reduce the probability that yet another robot collides is a result of having n seperate constraints, and trading slack in one or more constraint to satisfy another. Now consider an alternate formulation of the MPPU problem subject to a single constraint on the total expected number of collisions

$$\sum_i P^i_{\text{cumulative}}(b) \leq \delta_{\text{col}} \tag{6.12}$$

The joint belief dynamics are locally conservative, which implies that if a given amount of probability mass is to be moved out of a state in collision, an equal or greater amount of probability mass must be moved in an earlier belief. A robot r^i can only change the belief state of r^j by colliding with it or another robot. Thus any attempt by r^i to prevent a collision between two other robots will result in an increase in the expected number of collisions. Therefore, if a solution exists for a team of robots, then there is a solution for any subteam and constraint violations can be resolved by altering the paths of only those robots directly involved in the violation. However, the single constraint means that disjoint sets of colliding robots cannot be considered separately, as all contribute to the same constraint. Intuitively, there is no principled mechanism for partitioning the permitted number of expected collisions between subproblems. This rules out approaches such as rM* and MA-CBS, which are known to dramatically increase planning performance (Section 3.7).

MPPU is fundamentally different from the MPP problem because the belief dynamics for the robots are inherently coupled, removing the direct product structure of the MPP problem. Without the direct product structure, the MPPU problem is harder to decompose into tractable subproblems, which makes designing efficient, provably correct and optimal algorithms for MPPU significantly more difficult.

6.2 Uncertainty M*

In this section we introduce UM*, a variant of M* (Chapter 3) adapted to solve MPPU problems. UM* differs from M* in that UM* searches the *joint belief graph* that describes the joint belief space, and computes collision sets by considering constraint violations.

The joint belief graph $G = \{V, E\}$ is a discretized representation of the joint belief space and the joint belief dynamics. Each vertex $v_k \in V$ represents a belief state

$b_k \in \mathcal{B}$. An edge $e_{k\ell} \in E$ that connects v_k to v_ℓ is associated with an action $a_{k\ell} \in \mathcal{A}$ such that $\mathrm{Dyn}\left(b_k, a_{k\ell}\right) = b_\ell$. The joint belief graph for a specific MPPU problem is implicitly defined by the initial belief b_s and the action set. The joint belief graph as a whole is implicitly defined by repeated application of actions. The belief graphs for single robots are constructed in a similar manner, and used to compute the individual policies.

In M* collisions were treated as occurring at vertices based on the position of the robots in the joint configuration space. However in UM* the belief dynamics are such that it is impossible for two robots to collide at a given belief (Section 6.1); those robots would be moved to the special *col* state instead. Thus it is more natural to describe a collision as part of a transition from one belief to the next, *i.e.* as occurring at an edge. All of the proofs for M* still hold under these conditions (Section 3.1).

The second modification is to the calculation of the collision set. In M* replanning is triggered by the exploration of a vertex where two or more robots were in collision. In UM* replanning is triggered when the probability of a robot colliding is high enough to violate the collision probability constraint. UM* splits the collision set C_k into two components: the *threshold robot*s C_k^{thresh} and the *associated robot*s C_k^{assoc}. The threshold robots are those robots whose cumulative collision probability exceeds the threshold value at v_k or some successor of v_k in the search tree. The associated robots are those robots which have a non-zero probability of colliding with a threshold robot at some edge in the subset of the search tree rooted at v_k. The robots in C^{thresh} are known to violate their collision constraint, so UM* must find an alternate path that reduces the likelihood of collision for the threshold robots. Doing so requires changing the paths of the threshold robots or the paths of the robots with which they collide, *i.e.* the associated robots. The associated robots have slack in their constraints as otherwise they would be threshold robots. As a result alternate paths for the robots that collide only with robots in C_k^{assoc} do not need to be considered. While altering the path of such a robot could alter the belief state of a robot in C_k^{assoc}, and thus indirectly the belief state of a threshold robot, the same holds true for every robot in the system, even if it never collided with another robot (Section 6.1.1). Furthermore, such second-order interactions are much weaker, and thus can be neglected in the interest of efficiency. With $C_k = C_k^{\text{thresh}} \bigcup C_k^{\text{assoc}}$ the limited neighbors of v_k in UM* are computed in the same fashion as in M* (Equation 3.3).

The threshold robots and associated robots are computed in a similar fashion to how M* computes the collision set. If a set of robots exceeds the collision probability threshold at v_k, they are added to the threshold robots C_k^{thresh}. Then C_k^{thresh} and C_k^{assoc} are added to the threshold robots and associated robots, respectively, of each vertex v_ℓ in the backpropagation set of v_k. Furthermore, any robots that have a non-zero probability of colliding with a robot in C_k^{thresh} on the edge $e_{\ell k}$ connecting v_ℓ to v_k are added to C_ℓ^{assoc}. If C_ℓ^{thresh} or C_ℓ^{assoc} were changed, v_ℓ repeats the process (Algorithm 3).

UM* can be implemented using the inflated and recursive variants of M* (Section 3.5). rM* is substantially more efficient than M* (Section 3.7) with minimal drawbacks. Therefore, UM* is based on rM* throughout this chapter.

UM* notionally plans over a joint belief graph which represents the joint belief space. However, unless the joint beliefs have a very simple representation such as a Gaussian, the size of the representation of a single joint belief grows exponentially in the number of robots. To simplify the representation, the belief distribution for each robot is assumed to be independent,

$$b\left(q\right) = \prod_i b^i\left(q^i\right). \tag{6.13}$$

Note that the belief dynamics still properly account for robot-robot collisions. If r^i has a chance of colliding with r^j at some position v_k, then the subsequent belief of r^i

Algorithm 3 Pseudocode for collision set backpropagation in UM*

```
Require: $v_k$, $C_\ell^{\text{thresh}}$, $C_\ell^{\text{assoc}}$, open
  {$v_k$- vertex in the backpropagation set of $v_\ell$}
  {$C_\ell^{\text{thresh}}$- threshold collision set of $v_\ell$}
  {$C_\ell^{\text{assoc}}$- associated collision set of $v_\ell$}
  {open- the open list for M*}
  $C_k^{\text{thresh}} \leftarrow C_k^{\text{thresh}} \cup C_\ell^{\text{thresh}}$
  $C_k^{\text{assoc}} \leftarrow C_k^{\text{assoc}} \cup C_\ell^{\text{assoc}} \cup$ robots that potentially collide with robots in $C_\ell^{\text{thresh}}$ on $e_{k\ell}$
  if $C_k^{\text{thresh}}$ or $C_k^{\text{assoc}}$ changed then
    if $\neg(v_k \in \text{open}) \wedge C_k$ changed then
      open.insert($v_k$) {If the collision set changed, $v_k$ must be re-expanded}
    for $v_m \in v_k$.back_set do
      {Propagate changes to predecessors of $v_k$}
      backprop$\left(v_m, C_k^{\text{thresh}}, C_k^{\text{assoc}}, \text{open}\right)$
```

will have a lower density at v_k^i than if r^j were not potentially there.

6.3 Constrained M*

rM* gains a significant performance advantage over M* (Figure 3.7) by breaking a problem into independent subproblems. When each robot has an individual constraint on the probability that it collides with another robot the constraints on any given subproblem are simply the union of the constraints of the constituent robots[5]. However, a constraint on the total expected number of collisions is a global constraint on the system as a whole; there is no obvious way to determine a separate threshold for each subproblem.

Stentz [170] showed that a single robot path that obeys a single constraint could be generated by treating the constraint as a weighted penalty cost. Specifically, let the constraint be of the form $c(\pi) \leq \delta$, where c is a non-decreasing function. Then let the new cost function $g'(\pi) = g(\pi) + wc(\pi)$ be the sum of the path cost and a weighted

[5] Ignoring that the full MPPU is not the direct product of the problems for individual robots, and thus robots not involved in a given collision can take actions that adjust the probability of collisions between the robots involved in a collision

Algorithm 4 Pseudocode for UM*

```
{Define default values for vertices}
for all $v_k \in V$ do
  $v_k$.cost ← MAXCOST
  $v_k$.back_set ← $\emptyset$
  $C_k^{\text{thresh}} \leftarrow \emptyset$
  $C_k^{\text{assoc}} \leftarrow \emptyset$
{Initialize search}
$v_s$.cost ← 0
{Open list is sorted by f-value}
open ← $\{v_s\}$
while open.empty() == False do
  $v_k$ ← open.pop() {Get cheapest vertex}
  if Succ$(b_k)$ then
    {A solution has been found. Reconstruct the optimal path by following the back pointers}
    return back_track($v_k$)
  for $v_\ell \in V_k^{\text{nbh}}$ do
    {Add $v_k$ to the back propagation list}
    $v_\ell$.back_set.append($v_k$)
    {Compute threshold robots}
    $C_\ell^{\text{thresh}} \leftarrow C_\ell^{\text{thresh}} \bigcup \{i \mid P^i_{\text{cumulative}}(b_\ell) > \delta_{\text{col}}\}$
    {Update collision sets, and add vertices whose collision set changed back to open (Algorithm 3)}
    backprop($v_k$, $C_\ell^{\text{thresh}}$, $C_\ell^{\text{assoc}}$,open)
    {If $v_\ell$ doesn't violate constraints, and $v_k$ is the cheapest path to $v_\ell$, update costs and add to open list}
    if $P^i_{\text{cumulative}}(b_\ell) > \delta_{\text{col}} \forall i \in I$ and $v_k$.cost+$g(e_{kl}) < v_\ell$.cost then
      {$v_k$ is the cheapest route to $v_\ell$}
      $v_\ell$.cost ← $v_k$.cost+$g(e_{kl})$
      {Track the best path to $v_\ell$}
      $v_\ell$.back_ptr ← $v_k$
      open.insert($v_\ell$)
return No path exists
```

penalty term related to violations of the constraint. The optimal, constrained path is found by finding the path that minimizes g' while performing binary search on w. If the path that minimizes g' violates the constraint w is increased, otherwise w is decreased. Stentz [170] proved that this process will converge on the optimal solution for problems with a single constraint and developed algorithms based on A* and D*, called *Constrained A** (CA*) and *Constrained D** (CD*) respectively. CD* is valuable because this process requires repeated global planning, where the environment changes slightly between each iteration. Using D* substantially reduces the planning time compared to A*. Later work led to the K2 algorithm [135] that could find a high quality solution for a single robot with multiple constraints, but K2 is not readily extensible to MPPU.

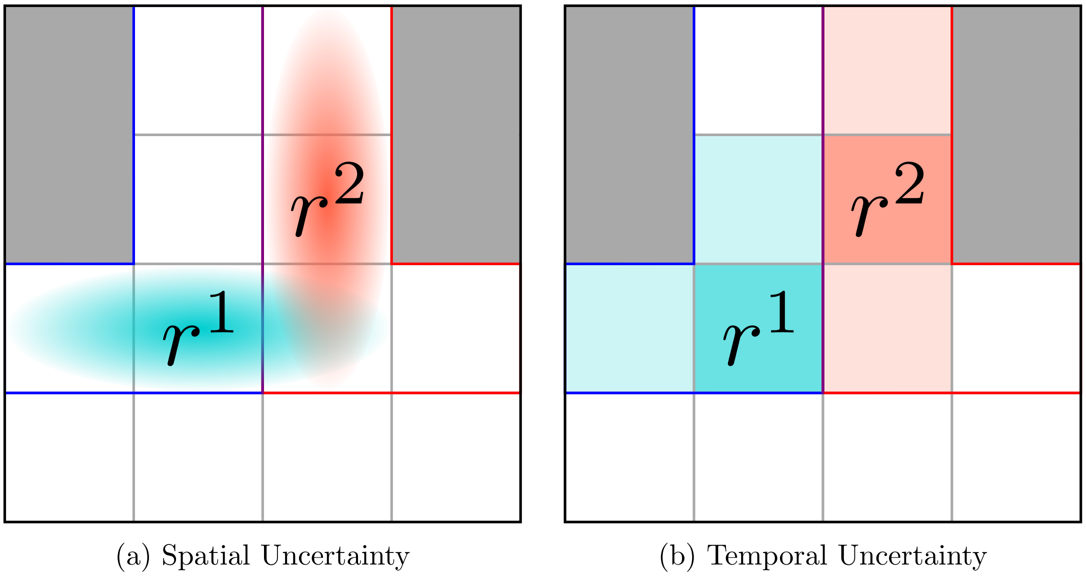
UM* can be readily adapted to use the penalty cost approach, using the sum of the probabilities that individual robots collide with other robots at each step as the penalty term. The resulting algorithm, called *Constrained M** (CM*), can solve problems where the constraint is on the probability that each robot collides with other robots and when the constraint is on the total number of expected collisions. In either case, CM* uses a single weight, so CM* is a purely heuristic method when dealing with multiple single robot constraints. The benefit is that CM* can run the recursive implementation of M* while subject to an expected total number of collisions constraint. The global adjustments to the cost function allow balancing of the minimization of robot-robot collisions with the additional cost incurred to avoid collisions across all subproblems. CM* assumes that the search has converged when two successive valid paths have the same cost, neglecting the penalty term. As with UM*, CM* is assumed to be based on rM* throughout the remainder of the chapter.

Unfortunately there is no D* equivalent for M*. As a result, only the individual policies can be shared between iterations of CM* and everything else must be replanned from scratch.

6.4 Belief Representation for MPPU

In most of the work on single robot belief space planning, the belief state of the system is represented by Gaussian distributions [29, 72, 131, 141, 173, 189]. In these works the primary source of uncertainty is imperfect localization, and the main challenge is obstacle avoidance. We are interested in problems where the individual robots are highly capable, and the primary challenge arises from the presence of many robots. We therefore assume that each robot has perfect localization and can accurately track a trajectory in the workspace, but synchronization between robots is imperfect. For such a system, spatial Gaussian beliefs do not provide a good representation, because they cannot conform to the path actually tracked by the robots. Instead, the belief state of the robots are represented as a distribution over position along the planned trajectory, directly modeling uncertainty due to synchronization errors (Figure 6.2b). Such a representation allows for useful planning when the size of the distribution is large compared to the size of environmental features.

Modeling position as a distribution over position in the planned trajectory poses a problem during planning, when the final trajectory is not yet known. If a robot were allowed to be either ahead or behind schedule the distribution would depend on parts of the path that have not been computed (Figure 6.3a), and the full distribution is needed for proper collision checking. Therefore, we model the nominal position of a robot as being as far along the path as physically possible. All uncertainty can then be modeled as delays, which would place the robot somewhere on the path leading to its nominal position (Figure 6.3b). As a result, the belief distribution of the robot is fully defined throughout planning.



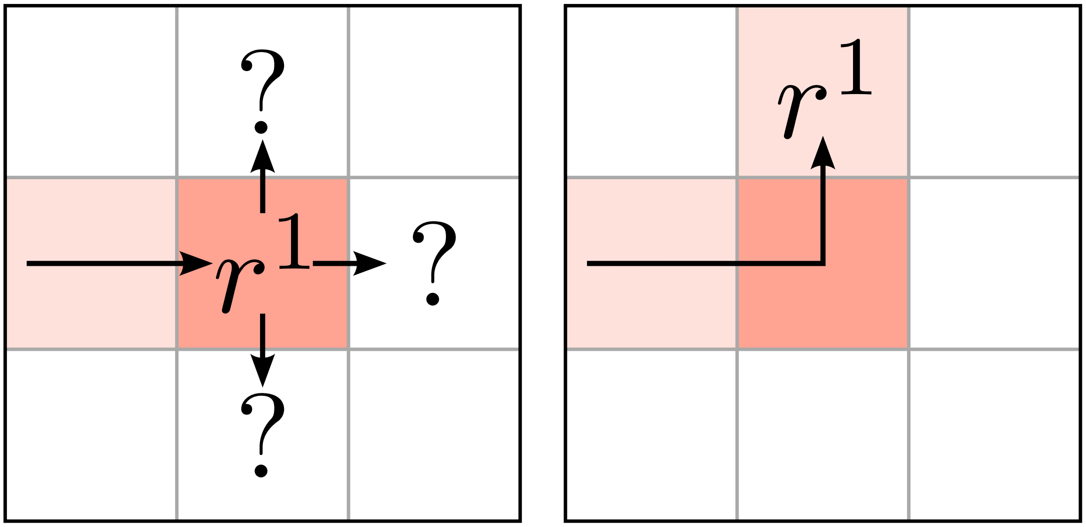
(a) Spatial Uncertainty

(b) Temporal Uncertainty

Figure 6.2: **(a)** Gaussian beliefs over robot positions (red and blue shaded regions) are commonly used when planning for single robots with uncertainty. However, spatial Gaussians do not accurately reflect the belief distribution when uncertainty is dominated by synchronization issues between multiple robots, and robots can accurately track spatial trajectories (corridors outlined in red and blue). As a result, a spatial Gaussian belief may predict collisions that are not possible. **(b)** Expressing uncertainty as a belief over position along the planned path produces beliefs that better represent the synchronization problems of multirobot path planning.

6.4.1 Multirobot Systems with Finite Probability of Delay

Consider a system of n robots r^i indexed by the set $I = \{1, \ldots, n\}$. Each robot moves on a configuration graph $G^i = \{V^i, E^i\}$ (Section 3.1). The set of actions available to a robot when at a vertex v_k^i corresponds to the edges leading to the out-neighbors. A collision function $(\Psi^{ij} : E^i \times E^j \rightarrow \{0, 1\})$ returns one if robots r^i and r^j would collide if they simultaneously traverse a given pair of edges and zero otherwise. At each time step, there is a P_{delay} probability that the robot will delay at its current location rather than taking the planned action. During execution, each robot counts the number of unplanned delay actions it takes, and will subsequently skip an equal number of planned delay actions. Thus planned delay actions serve as an indirect synchronization action. Every action a robot plans to take incurs cost 1



(a) Plan time belief with expected position

(b) Plan time belief with most advanced position

Figure 6.3: For planning with uncertainty, the belief of the robots position is described as a distribution over position along the trajectory (red shading). **(a)** If the nominal position of the robot is the expected position, then the portion of the belief corresponding to moving faster than expected depends upon part of the trajectory which has not yet been computed. **(b)** By recasting the nominal position as the most advanced possible state, and recasting all uncertainty as delay, the belief can be described purely in terms of the portion of the trajectory which has already been planned.

except waiting at the goal configuration of the robot which incurs no cost.

The belief state b_k^i of r^i is represented by two sequences: $\mathrm{pos}_k^i : \{1, \ldots, m\} \rightarrow V^i$ and $\mathrm{prob}_k^i : \{1, \ldots, m\} \rightarrow [0, 1]$. pos_k^i is the sequence of positions that r^i would have passed through if it never was unexpectedly delayed and prob_k^i contains the probabilities that r^i occupies each state in pos_k^i. The position furthest along the path is given by $\mathrm{pos}_k^i(1)$, which r^i occupies with probability $\mathrm{prob}_k^i(1)$. The probability that r^i has collided with another robot is given implicitly by $P_{\mathrm{cumulative}}^i(b_k^i) = 1 - \sum_{j=1}^m \mathrm{prob}_k^i(j)$. Tracking the full joint probability would be computationally impractical, so the belief distributions for each robot are assumed to be independent. Therefore, $b = \prod_i b^i$.

The belief dynamics for r^i are written

$$\mathrm{Dyn}^i\left(\left(\mathrm{pos}_k^i, \mathrm{prob}_k^i\right), e_{xy}^i\right) \mapsto \left(\mathrm{pos}_\ell^i, \mathrm{prob}_\ell^i\right) \tag{6.14}$$

$$\|\mathrm{pos}_k^i\| = m \tag{6.15}$$

$$\mathrm{pos}_\ell^i = \begin{cases} \left\{v_y^i, \mathrm{pos}_k^i\left(1\right), \dots, \mathrm{pos}_k^i\left(m\right)\right\} & x \neq y \\ \mathrm{pos}_k^i & x = y \,(\text{planned delay}) \end{cases} \tag{6.16}$$

where $\left(\mathrm{pos}_k^i, \mathrm{prob}_k^i\right)$ represents the belief state of r^i and e_{xy}^i is associated with a specific action that takes r^i from v_x^i to v_y^i. If the action is not a planned delay $(x \neq y)$,

$$\mathrm{prob}_\ell^i\left(w\right) = \begin{cases} \left(1 - P_{\text{delay}}\right)\mathrm{prob}_k^i\left(1\right) & w = 1 \\ \left(1 - P_{\text{delay}}\right)\mathrm{prob}_k^i\left(w\right) + P_{\text{delay}}\mathrm{prob}_k^i\left(w-1\right) & 1 < w \leq m \\ P_{\text{delay}}\mathrm{prob}_k^i\left(w-1\right) & w = m+1 \end{cases} \tag{6.17}$$

If the action is a planned delay, $(x = y)$,

$$\mathrm{prob}_\ell^i\left(w\right) = \begin{cases} \mathrm{prob}_k^i\left(1\right) + \left(1 - P_{\text{delay}}\right)\mathrm{prob}_k^i\left(w+1\right) & w = 1 \\ P_{\text{delay}}\mathrm{prob}_k^i\left(w\right) + \left(1 - P_{\text{delay}}\right)\mathrm{prob}_k^i\left(w+1\right) & 1 < w < m \\ P_{\text{delay}}\mathrm{prob}_k^i\left(w\right) & w = m \end{cases} \tag{6.18}$$

The joint belief dynamics differs from the product of the individual robot belief dynamics in that the joint belief dynamics includes collision checking, which depends on the probability that robots traverse each edge. Consider the case of r^i taking action a^i from the belief b_k^i, and let $b_\ell^i = \mathrm{Dyn}^i\left(\left(\mathrm{pos}_k^i, \mathrm{prob}_k^i\right), a^i\right)$.[6] Assuming that the action was not a planned delay, the probability of r^i traversing edge e_{xy}^i is given

[6] The action is specified directly instead of in terms of an edge to reduce overlap in notation

by

$$P\left(e^i_{xy}|b^i_k,a^i\right)=\begin{cases}\sum\limits_{\substack{j\\ \mathrm{pos}^i_k(j)=v^i_x\\ \mathrm{pos}^i_\ell(j)=v^i_y}}\left(1-P_{\mathrm{delay}}\right)\mathrm{prob}^i_k\left(j\right) & x\neq y\\ \sum\limits_{\substack{j\\ \mathrm{pos}^i_k(j)=v^i_x}}P_{\mathrm{delay}}\mathrm{prob}^i_k\left(j\right) & x=y\end{cases}\tag{6.19}$$

If the action was a planned delay, then

$$P\left(e^i_{xy}|b^i_k,a^i\right)=\begin{cases}\sum\limits_{\substack{j\\ \mathrm{pos}^i_k(j+1)=v^i_x\\ \mathrm{pos}^i_\ell(j)=v^i_y}}\left(1-P_{\mathrm{delay}}\right)\mathrm{prob}^i_k\left(j\right) & x\neq y\\ \sum\limits_{\substack{j\neq 1\\ \mathrm{pos}^i_k(j)=v^i_x}}P_{\mathrm{delay}}\mathrm{prob}^i_k\left(j\right)+\left(\mathrm{pos}^i_k\left(1\right)=v^i_x\right)\mathrm{prob}^i_k\left(1\right) & x=y\end{cases}\tag{6.20}$$

where $\mathrm{pos}^i_k\left(1\right)=v^i_x$ evaluates to 1 if true and 0 if false.

Now consider the case where the system takes a joint action a starting at b_k which would place the system at b_ℓ if there were no collisions. The probability that r^i would not collide with another robot if it traversed $e^i\in E^i$ is

$$P_{\mathrm{free}}\left(e^i|b_k,a\right)=\prod_{j\neq i}\left(1-\sum_{e^j\in E^j}\Psi^{ij}\left(e^i,e^j\right)P\left(e^j|b^j_k,a^j\right)\right)\tag{6.21}$$

The joint belief dynamics, including collision checking, can now be written

$$\mathrm{Dyn}\left(b_k,a\right)=\mathrm{Dyn}\left(\left(\mathrm{pos}_k,\mathrm{prob}_\ell\right),e_{xy}\right)\mapsto b_\ell=\left(\mathrm{pos}_\ell,\mathrm{prob}_\ell\right)\tag{6.22}$$

$$e_{xy}\equiv a\tag{6.23}$$

$$\|\mathrm{pos}^i_k\|=m\tag{6.24}$$

$$\mathrm{pos}^i_\ell=\begin{cases}\left\{v^i_y,\mathrm{pos}^i_k\left(1\right),\ldots,\mathrm{pos}^i_k\left(m\right)\right\} & x\neq y\\ \mathrm{pos}^i_k & x=y\,(\text{planned delay})\end{cases}\tag{6.25}$$

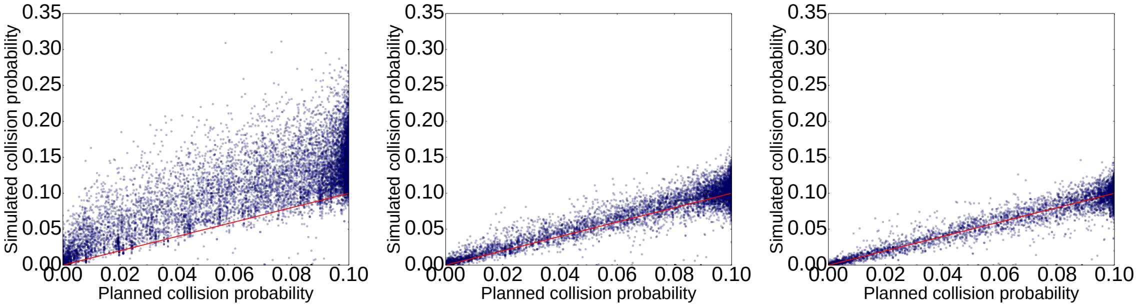
If the action is not a planned delay, $(x \neq y)$,

$$\mathrm{prob}_{\ell}^{i}\left(w\right) = \begin{cases} P_{\mathrm{free}}\left(\mathrm{pos}_{\ell}^{i}\left(2\right), \mathrm{pos}_{\ell}^{i}\left(1\right) | b_k, a\right)\left(1 - P_{\mathrm{delay}}\right)\mathrm{prob}_{k}^{i}\left(1\right) & w = 1 \\ \\ P_{\mathrm{free}}\left(\mathrm{pos}_{k}^{i}\left(w\right), \mathrm{pos}_{k}^{i}\left(w-1\right) | b_k, a\right)\left(1 - P_{\mathrm{delay}}\right)\mathrm{prob}_{k}^{i}\left(w\right) + \\ P_{\mathrm{free}}\left(\mathrm{pos}_{k}^{i}\left(w-1\right), \mathrm{pos}_{k}^{i}\left(w-1\right) | b_k, a\right) P_{\mathrm{delay}}\mathrm{prob}_{k}^{i}\left(w-1\right) & 1 < w \leq m \\ \\ P_{\mathrm{free}}\left(\mathrm{pos}_{k}^{i}\left(w-1\right), \mathrm{pos}_{k}^{i}\left(w-1\right) | b_k, a\right) P_{\mathrm{delay}}\mathrm{prob}_{k}^{i}\left(w-1\right) & w = m+1 \end{cases} \tag{6.26}$$

where $P_{\mathrm{free}}\left(\mathrm{pos}_{k}^{i}\left(w\right), \mathrm{pos}_{k}^{i}\left(w-1\right) | b_k, a\right)$ is the probability that r^i traverses the edge connecting $\mathrm{pos}_{k}^{i}\left(w\right)$ to $\mathrm{pos}_{k}^{i}\left(w-1\right)$ without colliding with another robot. If the action is a planned delay, $(x = y)$,

$$\mathrm{prob}_{\ell}^{i}\left(w\right) = \begin{cases} P_{\mathrm{free}}\left(\mathrm{pos}_{k}^{i}\left(1\right), \mathrm{pos}_{k}^{i}\left(1\right) | b_k, a\right)\mathrm{prob}_{k}^{i}\left(1\right) + \\ P_{\mathrm{free}}\left(\mathrm{pos}_{k}^{i}\left(2\right), \mathrm{pos}_{k}^{i}\left(1\right) | b_k, a\right)\left(1 - P_{\mathrm{delay}}\right)\mathrm{prob}_{k}^{i}\left(2\right) & w = 1 \\ \\ P_{\mathrm{free}}\left(\mathrm{pos}_{k}^{i}\left(w\right), \mathrm{pos}_{k}^{i}\left(w\right) | b_k, a\right) P_{\mathrm{delay}}\mathrm{prob}_{k}^{i}\left(w\right) + \\ P_{\mathrm{free}}\left(\mathrm{pos}_{k}^{i}\left(w+1\right), \mathrm{pos}_{k}^{i}\left(w\right) | b_k, a\right)\left(1 - P_{\mathrm{delay}}\right)\mathrm{prob}_{k}^{i}\left(w+1\right) & 1 < w < m \\ \\ P_{\mathrm{free}}\left(\mathrm{pos}_{k}^{i}\left(w\right), \mathrm{pos}_{k}^{i}\left(w\right) | b_k, a\right) P_{\mathrm{delay}}\mathrm{prob}_{k}^{i}\left(w\right) & w = m \end{cases} \tag{6.27}$$

As defined, the support of the belief of r^i will cover the entire path taken by r^i, even though the probability mass at many states will be infinitesimal. Therefore, states at the front and back of the distribution are removed if the probability mass at those states are less than some threshold value P_{prune}. After states are pruned, the remaining belief is re-normalized to preserve the total probability mass of the belief. P_{prune} must be chosen carefully, as it sets how much probability mass can "leak" out of the belief at each step (Figure 6.4). If $P_{\mathrm{prune}} = 0.01$ then after 30 steps, there may be a 30% chance that a robot will have "leaked" out of the support of the belief, and into the untracked tail. This can lead to UM* significantly underestimating the probability of collision (Figure 6.4). In the validation experiments, $P_{\mathrm{prune}} = 0.001$ led to accurate estimation of the collision probability, and setting $P_{\mathrm{prune}} = 0.0001$ did



(a) Min Probability Density 0.01 (b) Min Probability Density 0.001 (c) Min Probability Density 0.0001

Figure 6.4: Scatter plots showing the accuracy of the plans computed by UM* based on the minimum density at which elements of the belief state will be pruned. Each dot represents a single robot. The x-axis gives the probability of collision computed for that robot by UM*. The y-axis gives the observed probability of collision after running 100 realizations of each plan. If UM* was perfectly accurate all points would lie on the red line. If UM* is too aggressive at pruning the belief state **(a)** UM* will substantially underestimate the probability of collision. Reducing the pruning threshold to 0.001% produces accurate results for the tested environment **(b)**, and further reductions do not leader to improve accuracy **(c)**.

not noticeably improve collision probability prediction[7]. Large belief distributions cause a significant increase in the cost of performing collision checking, which can significantly harm performance (Figure 6.13).

6.5 Results

UM* was tested in simulated environments for the problem described in the previous section (Section 6.4.1). Each trial took place in a 32x32 four-connected grid of cells, with a 20% probability of a given cell being marked as an obstacle (Figure 6.5). Unique initial and goal positions for each robot were chosen randomly within the same connected component of the workspace. Any action by an individual robot, including waiting, incurred a cost of one, although a robot could wait at its assigned

[7] If a robot has a 10% probability of colliding, the standard deviation of the estimated collision probability from 100 trials would be $\sqrt{\frac{p(1-p)}{n}} = 3\%$

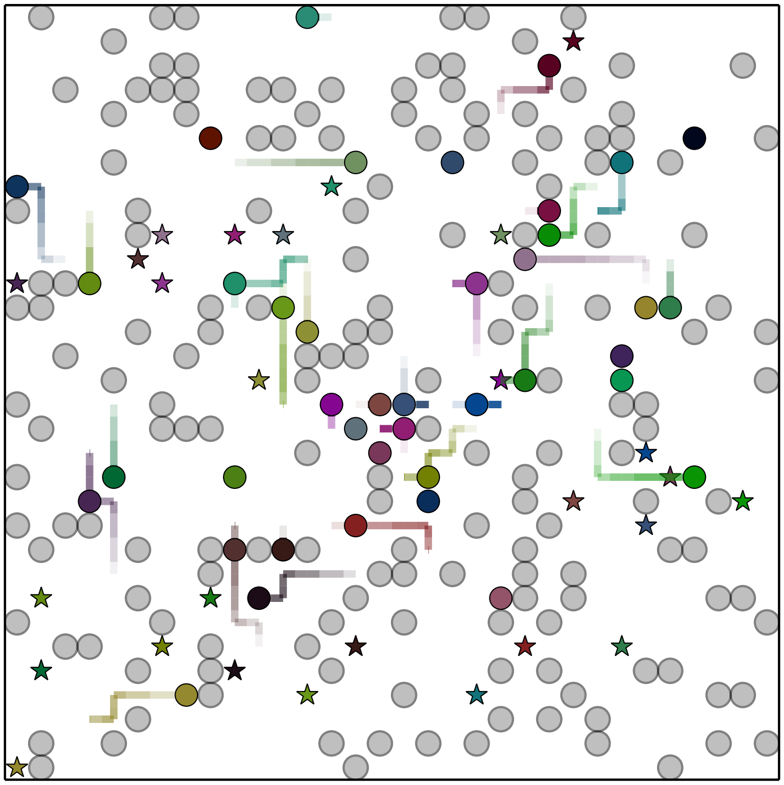


Figure 6.5: Typical step in a 40-robot plan computed by UM*. The gray circles are obstacles. Colored stars denote the goal configuration of robots. The colored bars represent the planning-time belief state as to where the robots would be. The color of more probable states are more saturated. The colored circles denote the actual position of the robots in one realization of the plan.

goal with zero cost. Unless otherwise stated, $P_{\text{delay}} = 0.1$, $\delta_{\text{col}} = 0.1$, and the heuristics used by M* and UM* were inflated by a factor of $\epsilon = 3$.

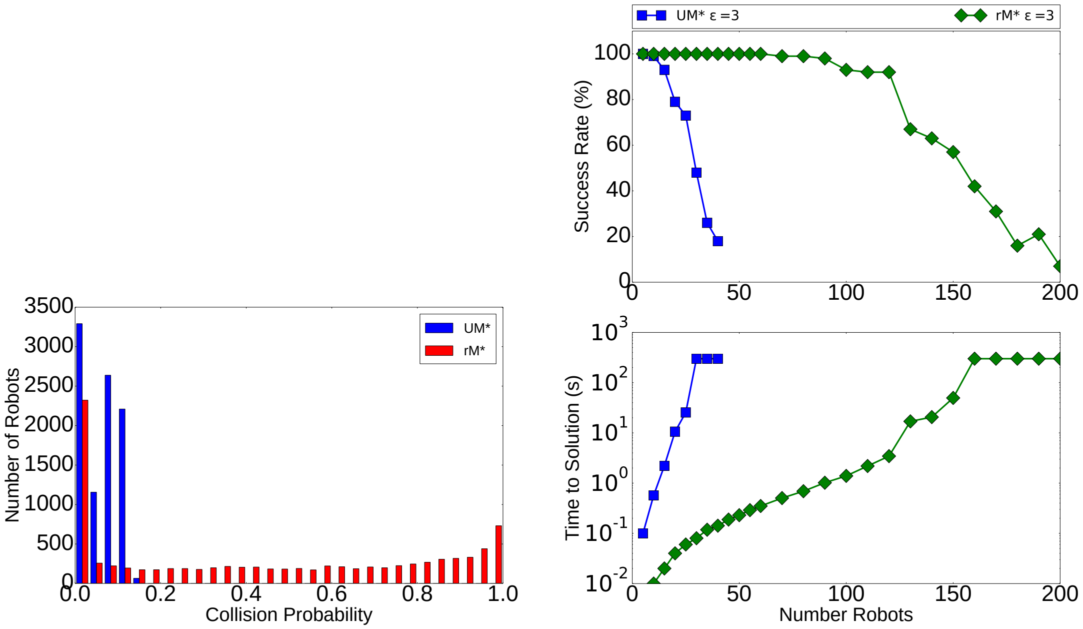
Each trial was given 5 minutes to find a solution. 100 random environments were tested for a given number of robots. We present the percentage of trials that were successful within 5 minutes as well as the median time required to find solutions. Run time is plotted on a logarithmic scale.

6.5.1 Comparison to Alternate Approaches

The first question is whether explicitly planning for uncertainty provides a significant benefit. We therefore compare UM* to three alternative approaches to solving the MPPU problem: running rM* without accounting for uncertainty, running rM* where robots are penalized for passing close to one another, and online replanning during plan execution.

Comparison with rM*

UM* was first compared to rM*, where rM* assumes that robots never take unplanned delay actions[8] (Figure 6.6). As expected, the probability that a robot will collide with other robots is much higher when following a path generated by rM* then when following a path generated by UM*, with a significant number of robots being guaranteed to collide with other robots (Figure 6.6a). However, the improvement in safety comes at a heavy cost. UM* can only solve approximately 20% of problems containing 40 robots, while rM* has a similar success rate for problems containing 180 robots.



(a) Collision probability during execution

(b) Planning performance

Figure 6.6: Comparison between UM* and rM*, where rM* ignores the probabilistic dynamics. Each robot has a 10% chance of delaying rather than taking its planned action. The heuristics of UM* and rM* are inflated by 3. **(a)** Every trial solved by both UM* and rM* was executed 100 times to compute the collision probability of each robot, and are plotted in a histogram on a per-robot basis. The percentage of trials that were solved successfully in under five minutes and the median time to find solutions are plotted in **(b)**.

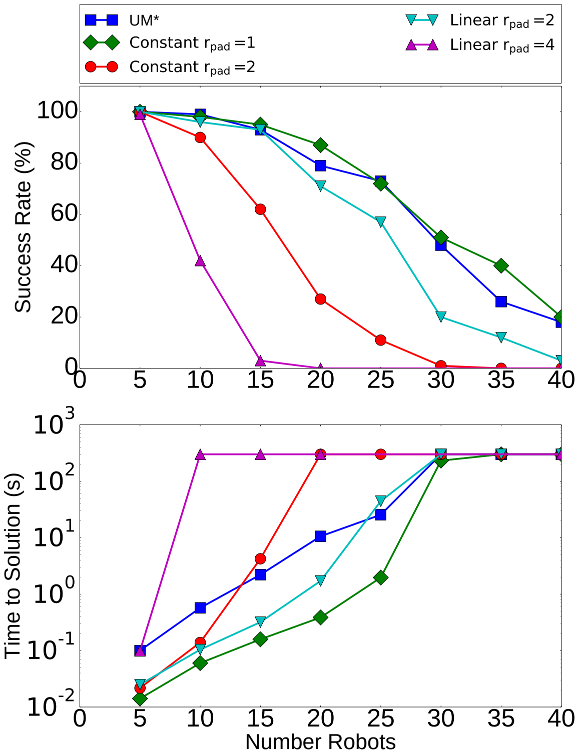
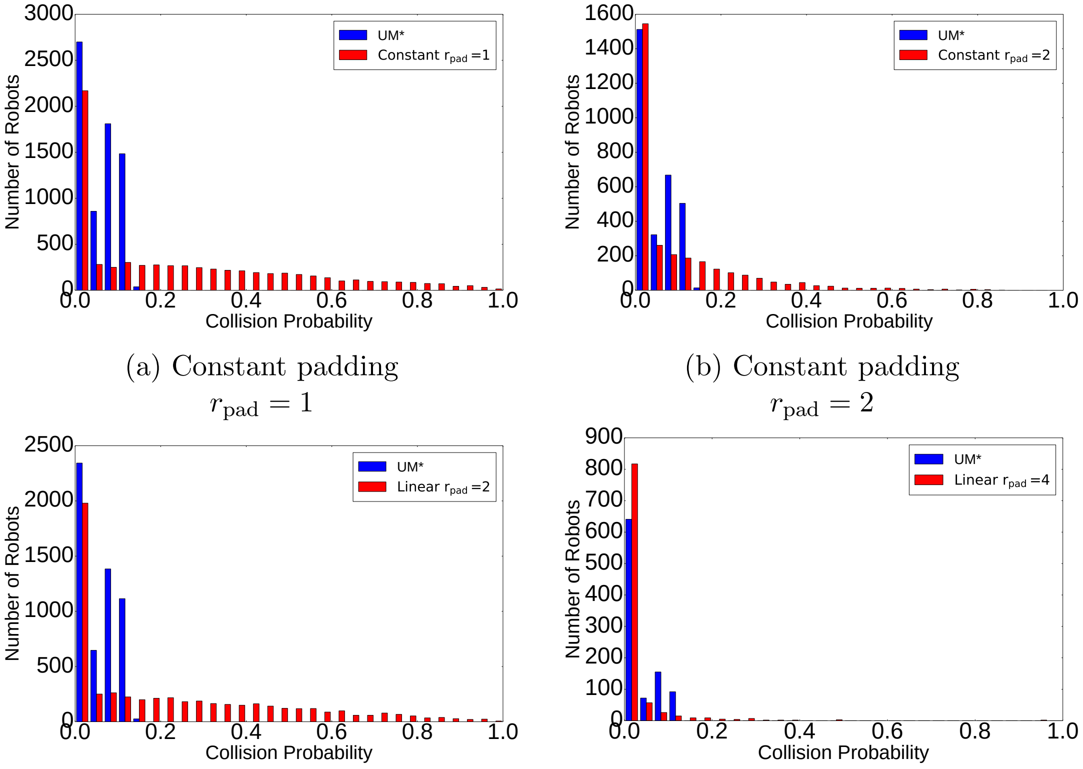


Figure 6.7: Comparison of performance of UM* and padded rM*. All trials were run with $P_{\text{delay}} = 0.1$, $g_{\text{pad}} = 10$, and $\epsilon = 3$. The top figure gives the percentage of trials that were solved successfully in under five minutes and the bottom plot gives the median time to find solutions.



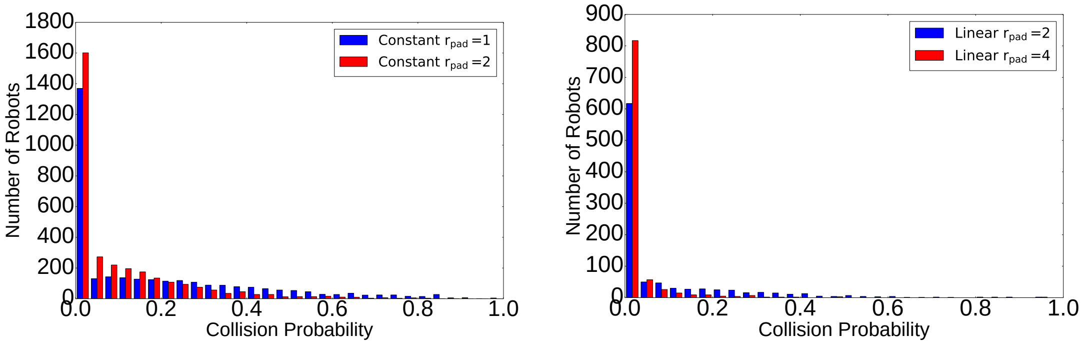
(a) Constant padding $r_{\text{pad}} = 1$

(b) Constant padding $r_{\text{pad}} = 2$

(c) Linear padding $r_{\text{pad}} = 2$

(d) Linear padding $r_{\text{pad}} = 4$

Figure 6.8: Histograms of the probability that individual robots will collide while executing plans. For each subfigure, the trials for which both UM* and a version of padded rM* found a solution were identified, and each solution was executed 100 times to estimate the probability that each individual robot would collide with another robot. The histogram shows how many robots had a given probability of collision. Note that the total number of robots plotted varies depending on how many trials a given version of padded rM* was able to solve



(a) Constant padding

(b) Linear padding

Figure 6.9: Comparison of different values of r_{pad}. For each form of padding, trials that were solved at two different values of r_{pad} were identified, and each solution was executed 100 times to estimate the probability that each individual robot would collide with another robot. The histogram shows how many robots had a given probability of collision. **(b)** Linear padding with $r_{\text{pad}} = 4$ was only able to solve problems where linear padding with $r_{\text{pad}} = 2$ had lower probability of collision (Figure 6.8c). The effect for constant padding **(a)** is not as noticeable

Comparison with padded rM*

One standard way of handling uncertainty in sensing or execution is to inflate the size of the robots to encourage paths with greater clearance. To this end, we tested rM* with two forms of padding. In constant padding, a pair of robots r^i and r^j incur an extra penalty cost g_{pad} if the distance between r^i and r^j is less than or equal to a padding radius r_{pad}. In linear padding, the penalty cost incurred by r^i and r^j varies linearly from g_{pad} if r^j is coincident with r^i to 0 if the distance between r^i and r^j is greater than or equal to r_{pad}. rM* adds any robot that incurs a non-zero penalty cost to the collision set. Several values for the penalty cost were tested, $g_{\text{pad}} \in \{3, 10, 30\}$. A value of $g_{\text{pad}} = 10$ worked best; $g_{\text{pad}} = 3$ failed to prevent robot-robot collisions, and $g_{\text{pad}} = 30$ did not substantially reduce collisions compared to $g_{\text{pad}} = 10$.

When r_{pad} was small ($r_{\text{pad}} = 1$ for constant padding, and $r_{\text{pad}} = 2$ for linear padding) the success rate for padded rM* was on par with UM*, and the time to solution was noticeably lower (Figure 6.7). However, such minor padding resulted

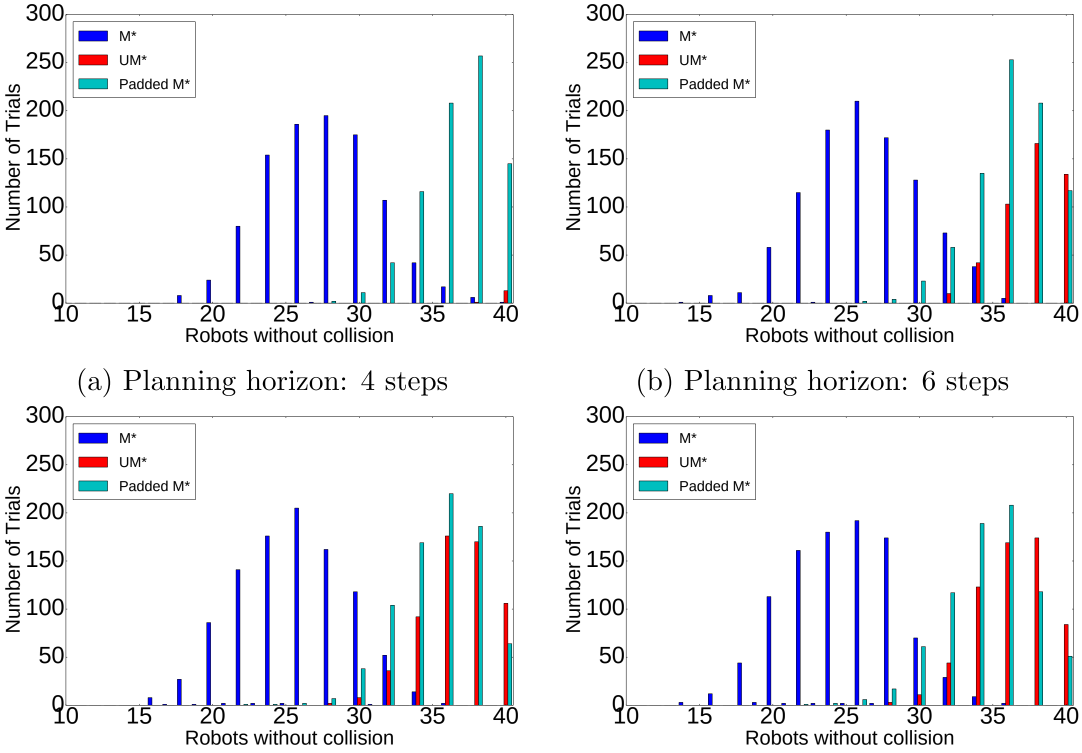
[8] Recall that UM* is implemented on top of rM*

in high likelihood of robot-robot collisions (Figures 6.8a, 6.8c). Doubling r_{pad} substantially reduced the probability of robot-robot collisions (Figures 6.8b, 6.8d), but significantly reduced the number of solved problems. Furthermore, the collision probabilities can be misleading, as padded rM* will fail more on harder problems, where robot-robot collisions are more likely (Figure 6.9). This effect explains most of the reduction in collision probability for linear padding with $r_{\text{pad}} = 4$ (Figure 6.9b), but is not as significant for constant padding with $r_{\text{pad}} = 2$ (Figure 6.9a).

The tested environment contains many features that are only one or two cells wide, but the support of the belief for a single robot can exceed 7 cells in length (Figure 6.5). Thus if robot-robot collisions are to be avoided, the padding region around the robots must be large compared to the features of the environment through which the robots must navigate. Robots would have difficultly passing one another in corridors, and may even interfere with one another while on opposite sides of impassable obstacles. Properly accounting for the true uncertainty in robot dynamics is thus important when dealing with belief states with support large compared to the size of environmental features.

Receding Horizon Planning

Another approach to dealing with uncertainty is receding horizon planning, where planning is run to some depth instead of planning all the way to the goal. In the implementation tested here, the robots execute half of the computed plan before replanning. Thus, if the horizon is 4 steps away, the robots will take two actions, then a new receding horizon path will be computed. UM*, rM*, and padded rM* were tested in receding horizon planning. There are two important details. Each invocation of the planner is independent, so the UM* planner will only constrain the probability of robots colliding within a single limited depth plan, not over the course of the entire path to the goal. rM* also behaves slightly differently in a receding



(a) Planning horizon: 4 steps

(b) Planning horizon: 6 steps

(c) Planning horizon: 8 steps

(d) Planning horizon: 10 steps

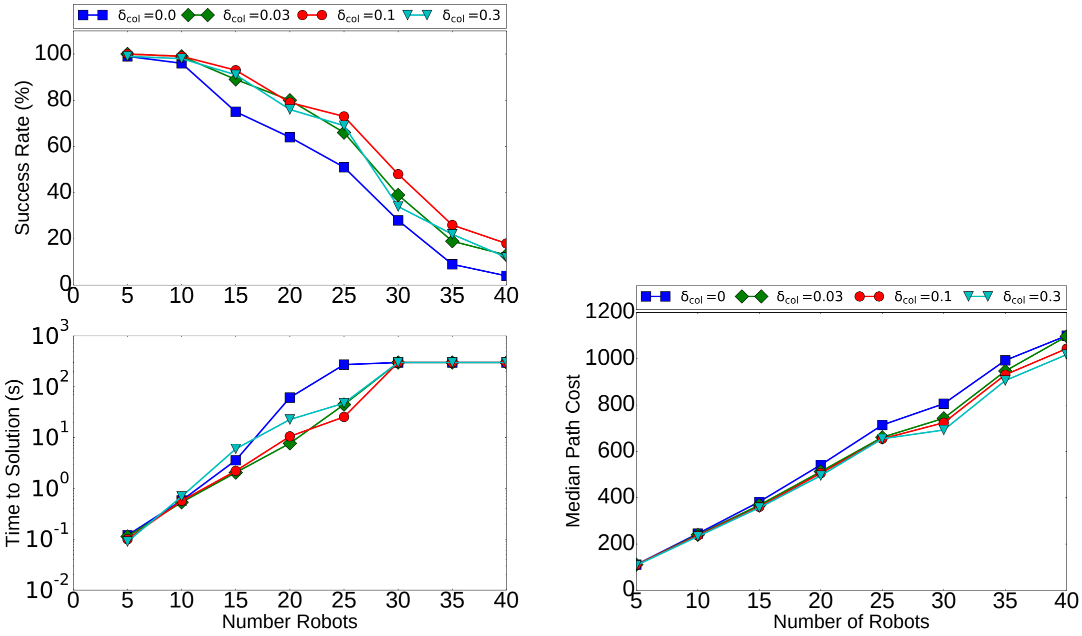
Figure 6.10: Number of robots that reach their goals without collision with receding horizon planning. Planning with receding horizons is an online process. When robots collide, they are removed from the environment. The plots show how many robots successfully reach their goals without colliding when following a receding horizon planner. The robots execute half the planned steps before replanning the path. Unlike most similar histograms in this chapter, the results for all trials are plotted, not just those successfully completed by all approaches.

horizon context, as the subplanners invoked by rM* share the same receding horizon. As a result, the subplanners are not complete which can cause rM* to improperly conclude that no solution exists. In that circumstance, rM* returns the path to the node with the lowest heuristic value that rM* has expanded.

Receding horizon planning is inherently an online approach and the robot dynamics are non-deterministic. Therefore receding horizon planning was run on 100 randomly generated problems containing 40 robots, with each trial run 10 times. Each trial was given 30 seconds total of planning time. If all robots that had not collided with other robots had not reached their goal before the budgeted planning time was exhausted the trial was counted as a failure. UM* was run with the collision threshold set to $\delta_{\text{col}} = 0.075$ to approximately match the safety of paths generated by padded rM*. In all other respects the environments were the same as in other trials.

As would be expected, rM* and padded rM* perform the best with short planning horizons, which maximizes information about where the robots actually are (Figure 6.10), and minimizes the support of the belief states, which permits minimal padding. UM* benefits from a longer planning horizon. If the planning horizon is too short UM* is prone to live-locks, where the robots oscillate back and forth. As expected, UM* and padded rM* produce safer paths than simple rM*, although they are both more prone to timing out. With the collision threshold tuned so that the safety of paths generated by UM* and padded rM* were approximately equal safety, padded rM* was able to solve 800 trials while UM* was only able to solve 600 trials, showing the benefits of padded rM* for comparatively short planning horizons.

UM* has a substantial advantage over more heuristic approaches when the support of the belief of each robot is large, as happens when planning a full solution to a MPPU problem. When the support of the belief of individual robots is comparable in size to typical environmental features, then accurately accounting for the true belief both makes finding solutions easier



(a) UM* performance
varying collision threshold

(b) UM* path cost
varying collision threshold

Figure 6.11: **(a)**Comparison of performance of UM* with differing collision thresholds. All trials were run with $P_{\text{delay}} = 0.1$ and $\epsilon = 3$. The top figure gives the percentage of trials that were solved successfully in under five minutes and the bottom plot gives the median time to find solutions. **(b)** Median path cost for paths found by UM* with varying collision thresholds. Only considers trials which were solved for all values of collision threshold.

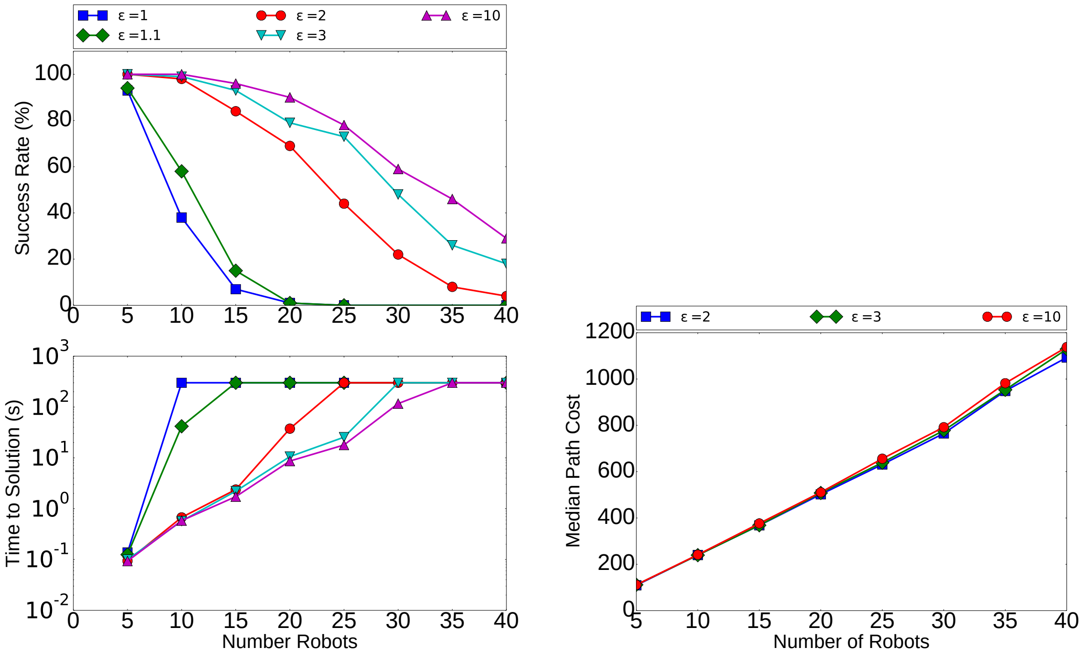
6.5.2 Scaling of UM*

With the benefits of UM* established, we wish to investigate how UM* scales with its two principle tuning parameters: the collision threshold δ_{col} and the heuristic inflation factor ϵ. The collision threshold trades off path safety for path quality, and the heuristic inflation factor trades path quality for planning time. A third parameter, the pruning threshold P_{prune}, trades accuracy in the belief representation for reduced planning time.

UM* was tested with a selection of collision thresholds, $\delta_{\text{col}} \in \{0, 0.03, 0.1, 0.3\}$ with $P_{\text{delay}} = 0.1$ and $\epsilon = 3$ (Figure 6.11). When $\delta_{\text{col}} = 0$, UM* is equivalent to padding the robot to exactly cover the support of its belief. UM* had the lowest success rate when $\delta_{\text{col}} = 0$, indicating that avoiding all overlap requires expensive detours. Although the difference is small, a collision threshold of 0.1 produced the best success rate. The similarity to P_{delay} is likely not coincidental. Assume that two robots start fully localized with r^i occupying the position v_k that r^j would like to occupy. If r^i takes a step away from v_k while r^j steps into v_k, then $b^i(v_k) = 0.1$ and $b^j(v_k) = 0.9$, which leads to a probability of collision of 0.09, just under the threshold. Setting $\delta_{\text{col}} = P_{\text{delay}}$ thus allows robots to pass close to one another, but quickly detects beliefs that will deeply interpenetrate one another. When $\delta_{\text{col}} = 0.3$, UM* could get stuck in deep local minima when robots cross paths, where the beliefs could overlap for multiple planning steps before violating the constraint. Finding a path out of such a local minima would be difficult.

As expected, $\delta_{\text{col}} = 0.3$ produces the lowest path costs while $\delta_{\text{col}} = 0$ produces the highest cost paths. These results match the degree to which the collision threshold constrains the paths. The difference in path costs is approximately 10%. The path costs are only significant between $\delta_{\text{col}} = 0$ and $\delta_{\text{col}} \neq 0$. The results for 30 or more robots must be taken with caution, as only eight 30 robot trials were solved for all collision thresholds, and there was only a single 35 and 40 robot problem that was solved for all collision thresholds.

The other tuning parameter is the inflation factor ϵ (Figure 6.12). UM* performs similarly for $\epsilon = 1$ and $\epsilon = 1.1$, which suggests that preventing constraint violations is more expensive for UM* than for M*, where setting $\epsilon = 1.1$ substantially increased performance (Figure 3.7). Going from $\epsilon = 1.1$ to $\epsilon = 2$ leads to a substantial increase in performance, while further increase of the inflation factor sees rapidly decreasing gains (Figure 6.12a), matching the behavior of rM* (Figure 3.8). The cost bounds imposed by the heuristic inflation factors are very loose. The difference in median path cost between $\epsilon = 2$ and $\epsilon = 10$ is less than 10%, again matching the behavior of rM* (Figure 3.9).



(a) UM* performance varying inflation factor

(b) UM* path cost varying inflation factor

Figure 6.12: **(a)** Comparison of performance of UM* with differing heuristic inflation factors. All trials were run with $P_{\text{delay}} = 0.1$ and $\delta_{\text{col}} = 0.1$. The top figure gives the percentage of trials that were solved successfully in under five minutes and the bottom plot gives the median time to find solutions. **(b)** Median path cost for paths found by UM* with varying heuristic inflation factors. Only considers trials which were solved for all values of the inflation factor.

As expected, lower values of the pruning threshold P_{prune} led to lower success rates and longer planning times (Figure 6.13). A lower value of P_{prune} leads to larger support for the belief state of each robot, complicating collision checking, but improving the accuracy of the belief representation. In these tests, $P_{\text{prune}} = 0.001$ led to good accuracy, with smaller values providing no improvement (Figure 6.4).

6.5.3 CM*

CM* was tested with two different constraints; the probability of collision for each robot being below a threshold and the total expected number of collisions being under a threshold. For the single robot constraint, the threshold was set at $\delta_{\text{col}} = 0.1$. For

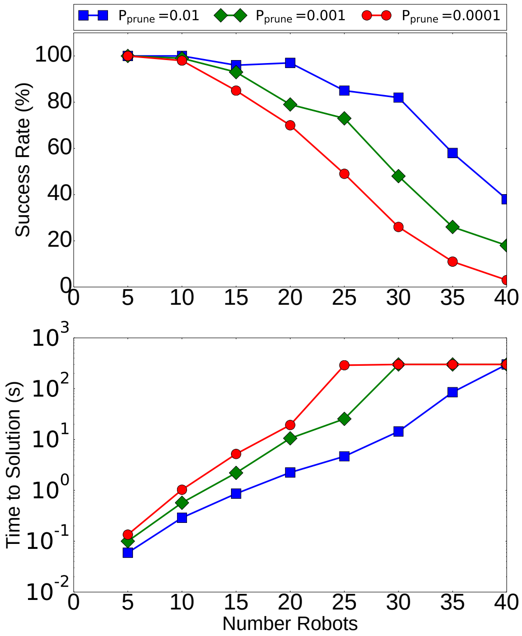
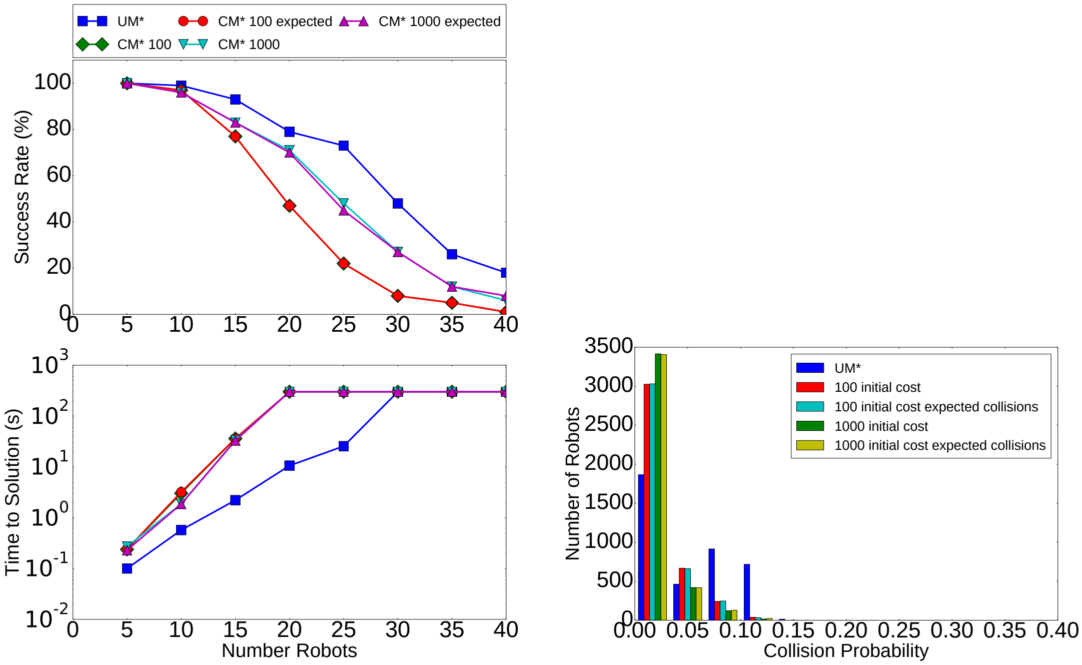


Figure 6.13: Comparison of the performance of UM* with different pruning thresholds P_{prune} for the underlying belief. P_{prune} was tested at 0.01, 0.001, and 0.0001. The top figure gives the percentage of trials that were solved successfully in under five minutes and the bottom plot gives the median time to find solutions.

the expected number of collisions threshold the threshold was set at 0.1 times the number of robots in the problem. UM* subject to the constraint that each robot has a probability of collision of less than $\delta_{\text{col}} = 0.1$ was used as a point of comparison (Figure 6.14), because the recursive implementation of UM* cannot operate under the expected number of collisions constraint, and the recursive implementation is necessary for efficient computation. CM* was tested with initial penalty costs 100 and 1000 times the probability of robot-robot collisions. UM* consistently outperforms CM* (Figure 6.14a). The performance of CMS subject to the single robot constraint and the expected number of collisions constraints were nearly identical. Furthermore, the probability that a robot collides when following a CMS path subject to the individual and expected number of collisions are nearly identical (Figure 6.14b) are nearly identical, and skewed to collision probabilities well below the constraint. This suggests that to find a solution the penalty costs are being set sufficiently high to avoid nearly all collisions.

6.6 Summary and Conclusions

In this chapter, we showed that the MPPU problem lacks the direct product structure upon which most MPP algorithms rely. We then presented UM*, a variant of rM* for the MPPU problem. UM* loses completeness only in cases which require a robot that does not normally collide with any other robot to deliberately collide with another robot. We then gave a non-Gaussian belief space representation that is appropriate for multirobot systems when individual robots can localize themselves well, but have little ability to synchronize their actions with other robots. We then compared UM* with several heuristic approaches to MPPU. UM* works well when a complete plan for the system is required; when continuous, short horizon replanning is feasible UM* is outperformed by a simple padding approach. We then tested UM* against CM*, a variant inspired by CD*, and showed that CM* underperforms UM*.



(a) CM* performance

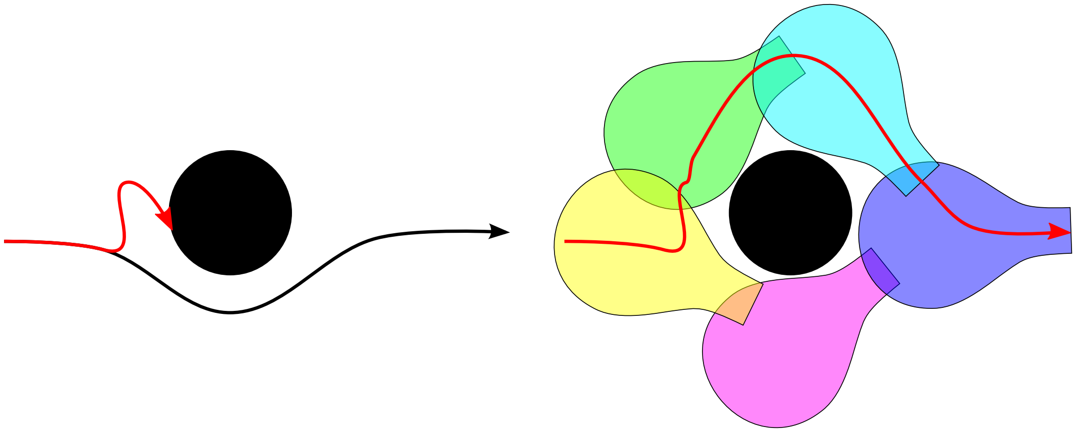
(b) CM* collision probability

Figure 6.14: Comparison of UM* and CM*. UM* and CM* were tested with a single robot collision threshold of 0.1. CM* was also tested with a constraint that the total expected number of collisions be no more than 0.1 times the number of robots. CM* was tested with the initial penalty cost being 100 and 1000 times the probability of collision. The heuristic inflation factor was 3, and $P_{\text{delay}} = 0.1$. **(a)** Comparison of performance of UM* and CM*. The top figure gives the percentage of trials that were solved successfully in under five minutes and the bottom plot gives the median time to find solutions. **(b)** Histogram of robot collision probability. For every trial solved by UM* and all versions of CM* the resulting plans were executed 100 times to estimate true robot-robot collision probabilities. The histogram shows the probability that a robot will collide with other robots, with each robot in each trial being counted separately. Note that the domain of the plot extends only to a 50% probability of collision, instead of the 100% used in most similar plots.

Chapter 7

Multirobot Sequential Composition

Conventional path planning algorithms compute a single trajectory in the configuration space of the system. However, perturbations and process uncertainties can force the system to depart from the planned trajectory. Running a trajectory-tracking controller can provide robustness to small perturbations. However, because the planner only considers a single trajectory, there is no guarantee that the trajectory-tracking controller will avoid collisions while recovering from larger perturbations (Figure 7.1a). *Sequential composition* seeks to produce paths that are robust to perturbations by planning not in the configuration space of the system, but rather over a set of controllers [20, 31, 43, 125, 126, 179]. The resulting plan is a sequence of controllers, chosen such that the goal set of each controller lies in the domain of attraction of the succeeding controller (Figure 7.1b), defining a plan over a "thick" region of the configuration space, instead of a single "thin" trajectory. As long as the system remains within the domains of the planned controllers safety is guaranteed. Furthermore, the system can readily detect when it is subject to a large enough perturbation that the original sequence of controllers can no longer guarantee safety and then compute a new plan.



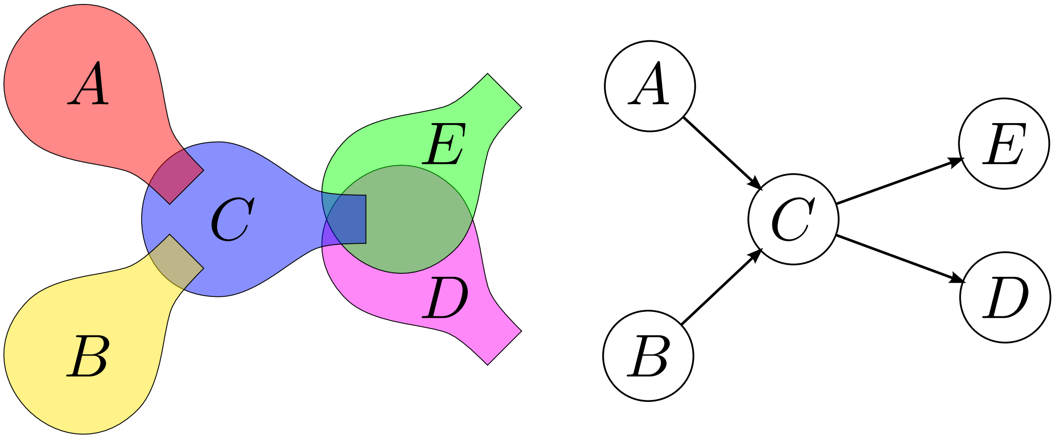
(a) Path planning (b) Sequential composition

Figure 7.1: **(a)** Conventional path planning algorithms compute a plan consisting of a single trajectory in configuration space (black arrow). While a trajectory controller can compensate for small disturbances, large disturbances during execution (red arrow) can cause the tracking controller to run into an obstacle. **(b)** Sequential composition planners [31, 43] compute a plan consisting of a sequence of controllers , chosen so that the goal set of each controller is a subset of the domain of attraction of the subsequent controller. In this example the initial path is yellow, purple, blue. The planned controllers guarantee safety as long as perturbations are not large enough to force the system out of their domains. Perturbations large enough to force the robot out of the domain of attractions of the controllers are easily detected, and can be compensated for by selecting a different sequence of controllers

Computing a sequential composition plan is a multi-step affair. First, a set of controllers with well defined domains must be deployed in the environment. The domain of a controller is an invariant subset[1] of the domain of attraction of the controller that lies entirely within the free configuration space. Controllers should be deployed so that the union of their domains cover as large a fraction of the free configuration space as possible.

The second step is to describe how the controllers can be sequenced. Controller A is said to *prepare* controller B, denoted $A \succ B$, if the goal set of A is a subset of the domain of B. A single controller can prepare multiple controllers. Once a robot reaches the goal set of its current controller, it will be in the domain of all

[1] If a robot starts in the domain of a controller it will remain within the domain of the controller.



(a) Controllers

(b) Prepares graph

Figure 7.2: **(a)** An example of a controller deployment where $A \succ C$, $B \succ C$, $C \succ D$ and $C \succ E$. **(b)** The associated prepares graph.

the prepared controllers, and can choose which to execute next according to a precomputed policy. The *prepares graph* is a directed graph that captures the prepares relationship between controllers (Figure 7.2). Each controller is represented as a vertex in the prepares graph whose out-neighbors are the prepared controllers. Thus a feasible sequence of controllers can be computed by finding a path in the prepares graph.

Prior work on sequential composition has primarily focused on combined control and path planning for single robots, where the prepares graph is sufficiently small that a policy over the entire prepares graph can be computed [20, 31, 43, 125, 126, 179]. The result is a control policy defined over a much larger fraction of the free configuration space than would be possible for a single conventional controller. Ayanian and Kumar [9] applied sequential composition to MPP by constructing controllers directly in the joint configuration space of the system [10, 11], producing a prepares graph which grew exponentially with the number of robots. Ayanian and Kumar [9] used A* to find a path in the prepares graph, which limited the approach to small numbers of robots. While not using the language of sequential composition, Ulusoy et al. [185]

describe an approach for using Linear Temporal Logic (LTL) solvers to coordinate robots executing single robot primitives, explicitly allowing for the primitives to be controllers. They also introduced region automata to handle primitives with different durations. However, the computational cost of the LTL solver limited the approach to a system of 2 robots.

In this chapter, we apply M* to multirobot sequential composition to facilitate robust planning for systems containing large numbers of robots. Our basic approach is to compute a prepares graph for each robot, then take the direct product to define a *joint prepares graph* for the system as a whole. M* can then be used to find a path in the joint prepares graph which assigns a sequence of single robot controllers to each robot. However, the time a robot takes to execute a controller varies with the geometry of the domain of the controller and random perturbations. We introduce the *time augmented joint prepares graph*, an adaptation of region automata [185], which captures differences in average execution time for different controllers. We deal with stochastic variations in execution time using UM* to explicitly reason about uncertainty (Chapter 6).

7.1 M* for multirobot sequential composition

Consider system of n robots r^i, $i \in \{1, \ldots\}$. Robot r^i has a configuration space Q^i and dynamics $\mathcal{F}^i : Q^i \times \mathcal{U}^i \to \mathcal{T}Q^i$, where $\mathcal{U}^i$ is the space of control inputs of r^i and $\mathcal{T}cspace^i$ is the tangent bundle of Q^i that contains the configuration velocities. A controller for r^i is a mapping $\mathcal{C}^i_k : Q^i \to \mathcal{D}^i_k$ where $\mathcal{D}^i_k \subset Q^i_{\text{free}}$ is the domain of the controller in the free configuration space. The flow of the controller $\Phi_{\mathcal{C}^i_k} : Q^i \times \mathbb{R} \to Q^i$ maps the configuration q^i of r^i at time zero to the configuration it would occupy $\Phi_{\mathcal{C}^i_k}(q^i, t)$ at time t under the influence of controller $\mathcal{C}^i_k$.

Each controller $\mathcal{C}^i_k$ has a goal set $\mathcal{G}^i_k \subset \mathcal{D}^i_k$. The flow of valid controller takes every point in its domain to the goal set in finite time

$$\forall q^i \in \mathcal{D}^i_k \; \exists T \geq 0 \; s.t \; \Phi_{\mathcal{C}^i_k}\left(q^i, T\right) \in \mathcal{G}^i \wedge \forall t \in [0, T] \; \Phi_{\mathcal{C}^i_k}\left(q^i, t\right) \in \mathcal{D}^i_k \tag{7.1}$$

Note that by this definition r^i executing $\mathcal{C}^i_k$ may leave $\mathcal{D}^i_k$, but only by passing through $\mathcal{G}^i_k$.

Finally, we say that controller $\mathcal{C}^i_k$ prepares $\mathcal{C}^i_\ell$, $k \neq \ell$, if $\mathcal{G}^i_k \subset \mathcal{D}^i_\ell$, denoted $\mathcal{C}^i_k \succ \mathcal{C}^i_\ell$. $\mathcal{C}^i_k$ prepares itself only if $\mathcal{G}^i_k$ is an invariant set, *i.e.* once r^i reaches $\mathcal{G}^i_k$ it will never leave $\mathcal{G}^i_k$ without executing a different controller. We term the controllers that prepare a given controller $\mathcal{C}^i_k$ the *preparing controller*s of $\mathcal{C}^i_k$. The prepares relations between a set of controllers $\mathfrak{C}^i$ for a robot r^i are described by the prepares graph G^i, where the vertex set of G^i is $\mathfrak{C}^i$. An edge connects $\mathcal{C}^i_k$ to $\mathcal{C}^i_\ell$ if and only if $\mathcal{C}^i_k \succ \mathcal{C}^i_\ell$.

The joint prepares graph of the multirobot system is the direct product of the individual prepares graphs $G = \prod_{i=1}^n G^i$, with vertex set V and edge set E. Each vertex v_k in G is associated with a tuple of controllers $(\mathcal{C}^1_k, \ldots, \mathcal{C}^n_k)$, which assigns r^i to execute $\mathcal{C}^i_k$. Let v_f be associated with a tuple of controllers that stabilize the robots at their ultimate goal states.

Specifying the vertex v_s that represents the initial state of the system requires care. Assume the robots start at an initial joint configuration $q_s \in Q$. Each robot may start in the domain of multiple controllers. Thus, the system could start from one of exponentially many vertices in the joint prepares graph, all of which would have to be added to the open list of M*. To allow M* to efficiently construct the optimal set of initial controllers, we define an initial vertex v_s that is associated with a tuple of dummy controllers $\left(\mathcal{C}^1_{\text{dummy}}, \ldots, \mathcal{C}^n_{\text{dummy}}\right)$. $\mathcal{C}^i_{\text{dummy}}$ prepares every controller $\mathcal{C}^i_k \in \mathfrak{C}^i$ for which $q^i_s \in \mathcal{D}^i_k$. M* can then intelligently select the proper set of controllers to execute first.

M* can then be used to compute a path in the joint prepares graph that describes the sequence of controllers to be executed by each robot.

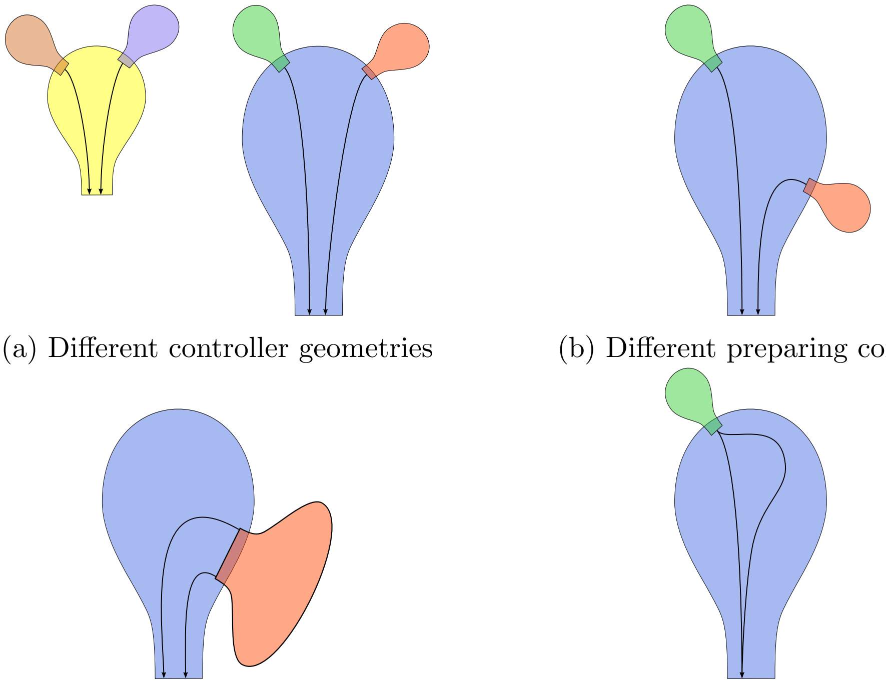
7.2 Synchronization Issues

The construction of the joint prepares graph embodies an assumption that each robot will transition from one controller to its successor at the same time, which would imply that every controller takes the same amount of time to be executed by each robot. However, differences in controller geometry and execution errors ensure that the time required to traverse a controller, termed the *duration* of a controller, not only differs between controllers, but is actually a random variable. There are several reasons for differences in duration

1. Controllers have domains of different shapes and command different velocities, and thus require different amounts of time to traverse (Figure 7.3a).
2. The goal set of different preparing controllers will be at different distances from the goal set of the controller. Therefore, the duration of a controller will depend upon the preparing controller (Figure 7.3b).
3. The time a robot requires to execute a controller from different configurations within the goal set of the same preparing controller will be different (Figure 7.3c).
4. A robot will take different amounts of time to execute the same controller from the same starting point due to environmental perturbations, noise in position estimates, and other stochastic influences (Figure 7.3d).

Reasons 1 and 2 are deterministic in that their contribution to the duration is due solely upon the geometry of the controller and its preparing controllers[2]. In the next section, we specialize region automata [185] for sequential composition, producing

[2] Assuming that the goal set of each controller is small



(a) Different controller geometries

(b) Different preparing controller

(c) Different positions in goal set of preparing controller

(d) Different path due to noise and modeling errors

Figure 7.3: Larger controllers will typically have longer durations than smaller controllers **(a)**. The duration of a single controller will depend on the preparing controller **(b)** and where in the goal set of the preparing controller the robot starts executing the controller **(c)**. Finally, perturbations of the trajectory during execution ensure that the duration will not be exactly the same even if the robot starts twice in exactly the same spot **(d)**

the time augmented joint prepares graph which accounts for geometric variations in duration.

Reasons 3 and 4 are inherently stochastic. Reason 3 is inherent to the fact that sequential composition reasons about controllers, rather than specific robot configurations, while 4 is inherent to non-ideal robots. We address stochastic uncertainty by planning paths for the system using UM* (Chapter 6), which explicitly reasons about uncertainty.

7.3 Time Augmented Prepares Graph

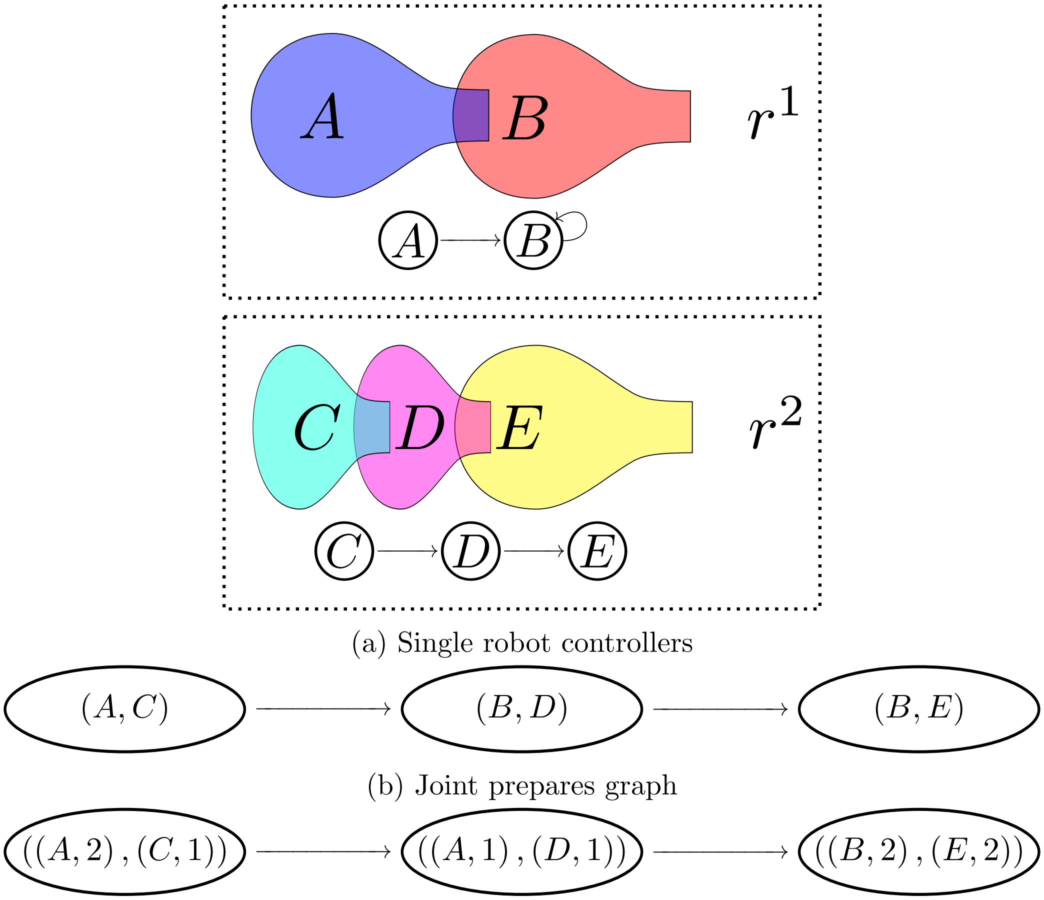
The time augmented joint prepares graph is intended to account for deterministic differences in the duration of controllers (Section 7.2). To do so, the time augmented joint prepares graph tracks how long each robot has spent in each controller, and only assigns new controllers to robots in the order that they complete their assigned controllers. To this end, we define the *nominal duration* $t_{nom} : \mathfrak{C}^i \times \mathfrak{C}^i \to \mathbb{R}^+$ over the set of all deployed controllers $\mathfrak{C}^i$ for r^i, where $t_{nom}\left(\mathcal{C}^i_k, \mathcal{C}^i_\ell\right)$ is the time that r^i is expected to require to execute controller $\mathcal{C}^i_k \in \mathfrak{C}^i$ when prepared by $\mathcal{C}^i_\ell$. The time augmented joint prepares graph is equivalent to the region automata of [185], but specialized for sequential composition.

A vertex v_k in the time augmented joint prepares graph is a set of ordered pairs $(\mathcal{C}^i_k, t^i_k)$, where $\mathcal{C}^i_k \in \mathfrak{C}^i$ is the controller assigned to r^i, and t^i_k is the expected time-to-go before the robot will complete executing $\mathcal{C}^i_k$. The neighbors of v_k assign new controllers to the robots that will finish their current controllers first, and update the time-to-go for the robots that will take longer to finish executing their assigned controllers. More specifically, denote the minimal time-to-go for any robot as $\delta t_k = \min_i t^i_k$. The neighbors of v_k are then

$$\text{neighbors}(v_k) = \left\{ v_\ell \middle| \begin{array}{ll} v_\ell^i = (\mathcal{C}_m^i, t_{nom}(\mathcal{C}_m^i, \mathcal{C}_k^i)) & t_k^i = \delta t_k \wedge \mathcal{C}_k^i \succ \mathcal{C}_m^i \\ v_\ell^i = (\mathcal{C}_k^i, t_k^i - \delta t_k) & t_k^i > \delta t_k \end{array} \right\}. \tag{7.2}$$

An example of the time augmented joint prepares graph is given in figure 7.4. The controllers for robot r^1 have the prepares relation $A \succ B$, and the controllers for r^2 have the relation $C \succ D \succ E$ (Figure 7.4a). Controllers A, B, and E have a nominal duration of 2 units. Controllers C and D have a nominal duration of 1 unit. The resulting joint prepares graph is given by $(A, C) \rightarrow (B, D) \rightarrow (B, E)$ (Figure 7.4b). The transition $(A, C) \rightarrow (B, D)$ is not feasible, because A takes longer to execute than C. The time augmented joint prepares graph avoids such problems. The initial vertex v_1 is given by $((A, 2), (C, 1))$. Robot r^2 will finish executing C in 1 unit of time, while r^1 requires 2 units of time to execute A. Therefore $\delta t_1 = 1$. According to equation 7.2, the neighbor of v_1 is $v_2 = ((A, 1), (D, 1))$. Controller D is expected to be executed by r^2 in 1 unit of time. Controller A requires 2 units of time to be executed, but r^1 has already executed A for 1 unit of time. Therefore, r^1 has a time-to-go of 1 unit. Thus, $\delta t_2 = 1$, with r^1 and r^2 finishing their current controllers at the same time. Therefore, the out-neighbor of v_2 is $v_3 = ((B, 2), (E, 2))$.

From the algorithmic point of view, M* can plan paths in the time augmented joint prepares graph without modification. However, the existing implementations of M* assume that the graph describing the full system is the product of single robot graphs, which is not the case for and the time augmented joint prepares graph or region automata [185]. An alternative to the time augmented joint prepares graph that can be used directly by existing M* implementations is the *Approximate Time Augmented Joint Prepares Graph* (ATAJPG), which represents the execution of controllers using a fixed temporal resolution. The ATAJPG is the direct product of single



(a) Single robot controllers

(b) Joint prepares graph

(c) Time augmented joint prepares graph

Figure 7.4: **(a)** Robots r^1 and r^2 have different controller sets and prepares graphs. The larger controllers A, B, and E have a nominal duration of 2, while the smaller controllers C and D can be executed in 1 time unit. **(b)** The joint prepares graph is the direct product of the single robot prepares graphs, and thus ignores differences in nominal duration of the controllers. The transition $(A, C) \rightarrow (B, D)$ is not realistic, because A takes longer to execute than C. **(c)** The time augmented joint prepares graph is formed by augmenting each controller with a time-to-go.

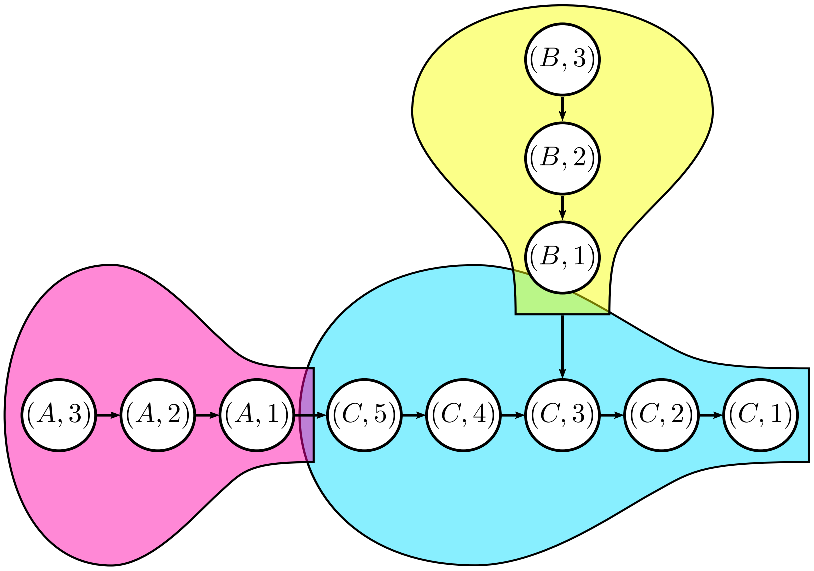
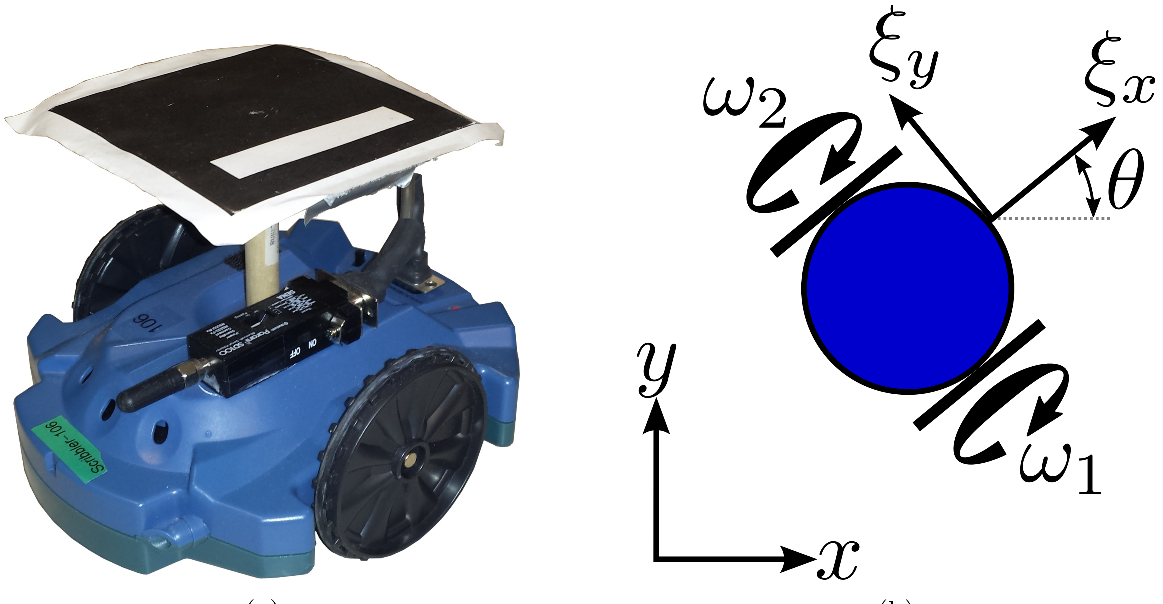


Figure 7.5: Example of the approximate time augmented prepares graph with a time resolution of 1. The nominal duration of controller C is 5 when prepared by controller A and 3 when prepared by controller B.

robot *approximate time augmented prepares graph*s, in which each controller is represented by a number of vertices determined by its largest nominal duration, recalling that the nominal duration of a controller depends upon its preparing controller. If a controller has a maximal nominal duration of 5 seconds and the time resolution is 1 second, then the controller is represented in the approximate time augmented prepares graph by 5 vertices (Figure 7.5). The vertices representing a single controller are arranged in a chain, and annotated with the time to go. The last vertex associated with controller $\mathcal{C}_k^i$ (smallest time to go) is connected to the vertices representing controllers $\mathcal{C}_\ell^i, \mathcal{C}_k^i \succ \mathcal{C}_\ell^i$ with a time-to-go equal to $t_{nom}\left(\mathcal{C}_\ell^i, \mathcal{C}_k^i\right)$.

The advantage of the ATAJPG is that it is a drop-in replacement for the joint prepares graph in M* implementations. The disadvantage is that the ATAJPG will be up to a constant factor larger than the time augmented joint prepares graph, where the constant factor is the maximal number of vertices used to represent a single controller. The extra vertices correspond to situations where no robot is at the last vertex representing its current controller, and thus have only a single successor



(a) (b)

Figure 7.6: Parallax Scribbler robot as used in experiments **(a)**. To simplify control design, the body frame was placed at the front of the robot **(b)**. The scribbler can then be treated as a fully actuated robot in position with heading left uncontrolled.

in the ATAJPG. The extra vertices do not contribute to the branching factor of the graph, so running M* on the ATAJPG should take a constant factor more time then running M* on the time augmented joint prepares graph.

7.4 Implementation

Sequential composition addresses issues of control in the face of uncertain dynamics and environmental noise, which makes experimental validation particularly useful. Logistical constraints limited the number of physical robots that could be run simultaneously and the size of the workspace. Experiments were therefore run in a mixed reality framework, combining physical and simulated robots to increase the number of agents and the size of the workspace.

7.4.1 Scribbler robots

Parallax Scribbler robots provided by Manuela Veloso served as the test bed. The Scribbler robot is a differential drive robot that is nearly circular in shape (Figure 7.6a), with a radius of 7.2 cm. Each Scribbler robot was equipped with a Parani SD100 Bluetooth serial adapter providing a 9600 baud serial connection to an offboard computer. The Scribbler robots have minimal onboard computation and no odometry sensors, requiring offboard localization and control. Each robot was programmed to obey motor speed commands received over the Bluetooth link.

The Scribbler robots showed a significant variation in performance due to their age. However, the four best robots had similar performance, with a maximum speed of 0.3 meters per second (m/s), and a maximal angular velocity of 3.5 radians per second (rad/s). Note that an ideal differential drive robot with the radius and maximal linear velocity of the Scribblers would have a maximal angular velocity of 4.2 rad/s. The Scribblers were observed to have difficulty turning counterclockwise, presumably due to asymmetries in the drive train, which may account for the reduced average angular velocity.

According to the unicycle model, a differential drive robot can directly control its angular $\dot{\xi}_\theta$ and longitudinal $\dot{\xi}_x$ velocities. The main emphasis of this thesis is on planning for multiple robots, so to simplify controller design we follow [10] and place the body frame at the front of the robot, instead of directly between the wheels as in the unicycle model (Figure 7.6b). With the offset body frame the Scribbler can be treated as a fully-actuated planar robot, leaving θ uncontrolled. The kinematics of the robot are then given by

$$\begin{bmatrix} \dot{\xi}_x \\ \dot{\xi}_y \end{bmatrix} = \rho_{wheel} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} \tag{7.3}$$

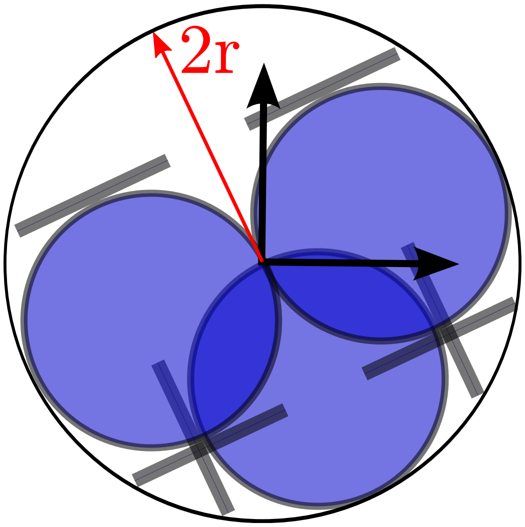


Figure 7.7: Control of a differential drive robot can be simplified by placing the body frame at the front of the robot, and only controlling x and y. However, this means that when the body frame of the robot is at a given position, the robot might occupy any points in a circle that is twice the diameter of the robot.

where ρ_{wheel} is the radius of a wheel and ω_1 and ω_2 are the angular velocities of the left and right wheels, respectively. A velocity field defined in the world frame can be converted into a controller for the robot by

$$\begin{bmatrix}\omega_1\\ \omega_2\end{bmatrix} = \frac{1}{2\rho_{wheel}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\cos(\theta) & \sin(\theta)\\ -\sin(\theta) & \cos(\theta)\end{bmatrix}\begin{bmatrix}\nu_x(x,y)\\ \nu_y(x,y)\end{bmatrix} \tag{7.4}$$

where $\nu_x(x,y)$ (respectively $\nu_y(x,y)$) is the commanded world velocity in the x (respectively y) direction when the body frame of the robot is at (x,y,θ). The downside of this model is that the robot extends up to 2 radii from the origin of the body frame and does not actively control θ, effectively doubling the required clearance to avoid collisions [10] (Figure 7.7).

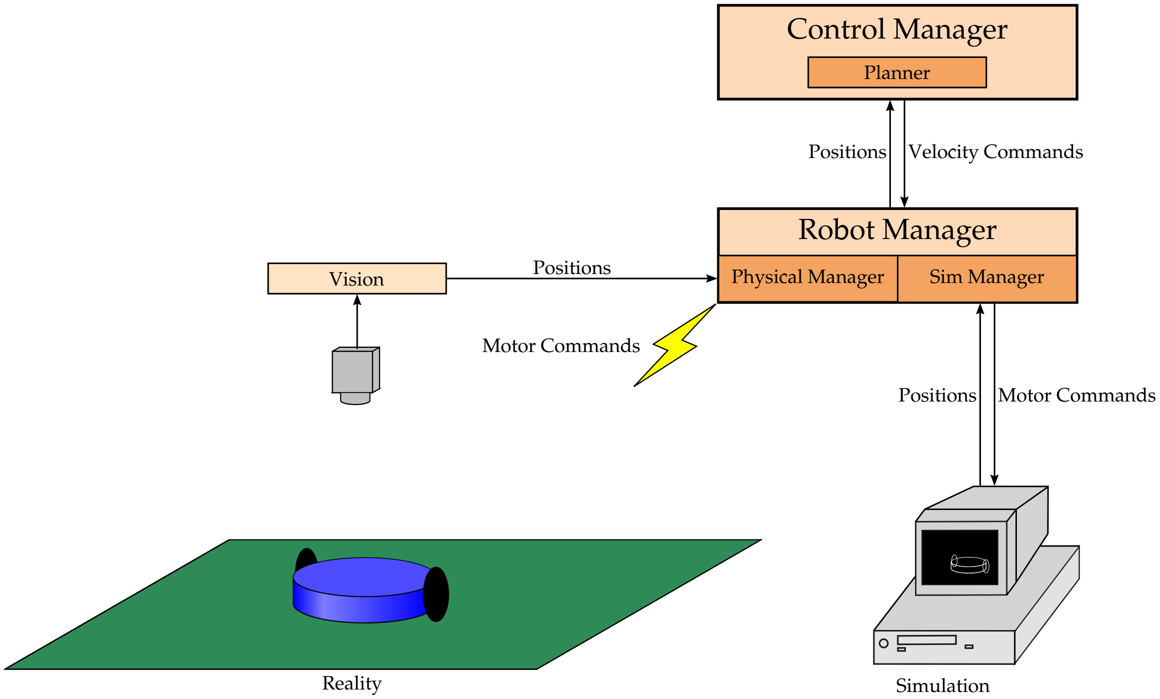


Figure 7.8: Architecture for experiments.

7.4.2 Architecture

The limited sensing and computational resources of the Scribbler robots required the use of a centralized localization and control scheme. Furthermore, limitations in the size of the test environment and the number of available robots led to a mixed-reality approach, utilizing both real and simulated robots. The high level architecture consisted of a robot manager and a control manager (Figure 7.8). The robot manager provided the control manager with estimates of the position of each robot, and converted the world frame velocity commands of the control manager into motor commands. The control manager was responsible for planning, determining the active controller for each robot and computing commands of each robot's active controller.

The robot manager was a wrapper around two sub-components: the physical robot manager and the sim manager, responsible for communicating with the real and simulated robots respectively. The physical robot manager was responsible for establishing serial communications over Bluetooth with each robot, sending motor controls to the robots via Bluetooth, and running computer vision to localize the robots using an overhead camera. An Aruco tag was placed on the top of each robot (Figure 7.6a) to allow an overhead camera to localize the robots using the Aruco computer vision library [68]. In the lab, localization was precise to within 1 mm and 1/100 of a radian, which when combined with the relatively low performance of the Scribblers meant no filtering was necessary. The sim manager interfaced directly with the simulator, reading out the position of the robots and passing in commanded motor speeds. The robot manager hid whether a given robot was physical or simulated from the control manager: the interface for providing position estimates and receiving commanded world frame velocities was the same in either case.

The control manager computed a joint path for the multirobot system using M* or UM* when explicitly considering uncertainty. The joint path was split into separate paths for each individual robot, and executed separately. The control manager supported optional synchronization when running a plan computed by M* by slowing down robots which are ahead of other robots by σ^η, where σ is the *synchronization factor* and η is the number of steps in the plan the robot is ahead of the slowest robot.

7.4.3 Controller Design

Controllers were defined on convex, polygonal domains in the workspace following the work of Habets and Van Schuppen [76]. An affine vector field over a triangular domain is fully defined by the value of the field at the vertices of the triangle. A continuous, piecewise-affine field can be generated over a convex polygon by triangulating the polygon and specifying the value of the field at each vertex of the polygon. For the vector field to define a valid controller for use in sequential composition, the trajectory of any particle that flows along the vector field starting from within the domain must

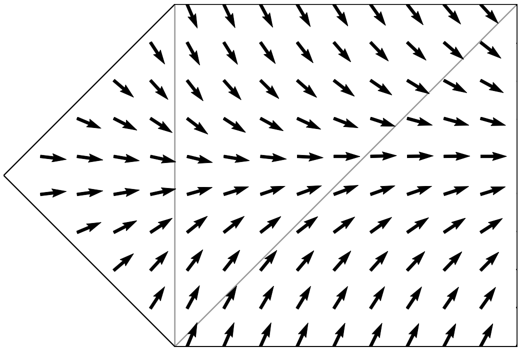


Figure 7.9: Example of controller design. The goal face is the face furthest to the right. The gray lines indicate the internal triangulation of the polygonal domain.

leave the polygon through a specified goal face after finite time, and must not leave the domain of the controller through any other face.

Consider the case of designing a controller whose domain is a convex m-gon, where $\mathbf{v}_k$ denotes the coordinates of the kth vertex. Let $\hat{\mathbf{n}}_k$ denote the outward-pointing unit normal vector for the face connecting vertex $\mathbf{v}_k$ to $\mathbf{v}_{k+1}$, where all arithmetic is mod m. Without loss of generality, assume that the goal face is face 0. The piece-wise affine vector field defined on a triangulation of the polygon is a valid controller if the value of the vector field $\mathbf{u}_k$ at each vertex $\mathbf{v}_k$ satisfies Habets and Van Schuppen [76]

$$\left.\begin{aligned} \hat{\mathbf{n}}_0 \cdot \mathbf{u}_k &> 0 \\ \hat{\mathbf{n}}_{(2k-1)\%m} \cdot \mathbf{u}_k &\leq 0 \end{aligned}\right\} k \in \{0, 1\} \tag{7.5}$$

$$\left.\begin{aligned} \hat{\mathbf{n}}_0 \cdot \mathbf{u}_k &> 0 \\ \hat{\mathbf{n}}_{k-1} \cdot \mathbf{u}_k &\leq 0 \\ \hat{\mathbf{n}}_k \cdot \mathbf{u}_k &\leq 0 \end{aligned}\right\} i \geq 2 \tag{7.6}$$

Roughly, every $\mathbf{u}_k$ must have a component pointing out of the goal face, and an inward component relative to every adjacent non-goal face [76]. We choose $\mathbf{u}_k$ by running a linear program that finds the unit vector that satisfies (Equation 7.5) and maximizes the objective function

$$c\left(\mathbf{u}_i\right) = \begin{cases} \dfrac{\hat{\mathbf{n}}_0 - \hat{\mathbf{n}}_{(2k-1)\%m}}{\left|\hat{\mathbf{n}}_0 - \hat{\mathbf{n}}_{(2k-1)\%m}\right|} \cdot \mathbf{u}_i & i \in \{0, m-1\} \\ \dfrac{\bar{\mathbf{v}} - \mathbf{v}_k}{\left|\bar{\mathbf{v}} - \mathbf{v}_k\right|} \cdot \mathbf{u}_i & i \in \{1, \ldots, m-2\} \end{cases} \tag{7.7}$$

where $\bar{\mathbf{v}}$ denotes the centroid of the polygon. At vertices not adjacent to the exit face, the objective function rewards controllers that move a robot towards the centroid of the domain as quickly as possible. At vertices adjacent to the exit face, the objective function balances the need to move robots near the adjacent non-exit face into the interior of the polygon with the need to move robots out through the exit face. We also choose to rescale the resulting piecewise-affine vector field to have constant magnitude (Figure 7.9).

Goal controllers stabilize a robot at the centroid of the domain using linear attractive fields with magnitude proportional to the distance from the centroid. Goal controllers also provide the robots with the ability to wait in place. Normally a robot r^i will transition from $\mathcal{C}_k^i$ to the next controller in its plan $\mathcal{C}_{k+1}^i$ as soon as r^i enters $\mathcal{D}_{k+1}^i$. However, this would effectively prevent a robot from executing a goal controller multiple times in a row to wait for prolonged period of time, as this would imply that $\mathcal{D}_k^i = \mathcal{D}_{k+1}^i$, so the robot would immediately skip over any repetitions of the goal controller in the plan. Instead, each goal controller is explicitly assigned a nominal duration. When a robot begins executing a goal controller it starts a clock and will not transition to the next controller in its plan until it has spent time at least equal to the nominal duration in its current controller. If the next step in the plan calls for the robot to wait again at the same goal controller, the robot resets its clock and waits for another fixed period of time.

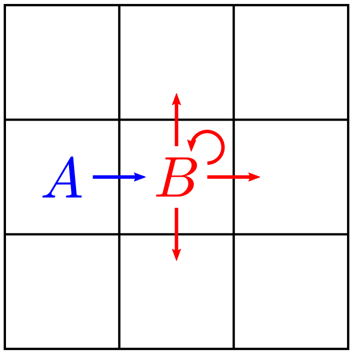


Figure 7.10: Consider a controller defined in square A whose goal face is shared with square B. The controller in A prepares every controller whose domain is B except for the controller in B whose goal face is shared by square A.

7.5 Experiments

We mounted a 720p webcam on the ceiling of the lab, which allowed Scribblers to be localized in a region 1.6x1.2 meters. Controllers were deployed in the visible region by covering the workspace with squares 0.15 meters on a side, slightly more than one Scribbler diameter. Up to five controllers were deployed in each square, four controllers each using a different face of the square as a goal face, and one goal controller which allows the robot to wait in place, which is especially important if the square contains the robot's goal configuration. Controllers were only constructed if they would prepare at least one other controller, *i.e.* the goal face of the controller must be shared with at least one other square. A controller prepares every controller in the square that shares its goal face, except for the controller which would immediately return the robot to its starting square (Figure 7.10). The simulated robots occupied a space twice as large, centered on the visible region. All controllers except goal controllers were set to command a constant speed of 0.12 m/s.

Controllers are deemed to interfere if they share a vertex. This guarantees a clearance of two Scribbler radii between the origins of the body frames of the robots. To guarantee that no collisions occur a clearance of four radii would be required. Unfortunately, the workspace for the physical robots is not large enough to require a

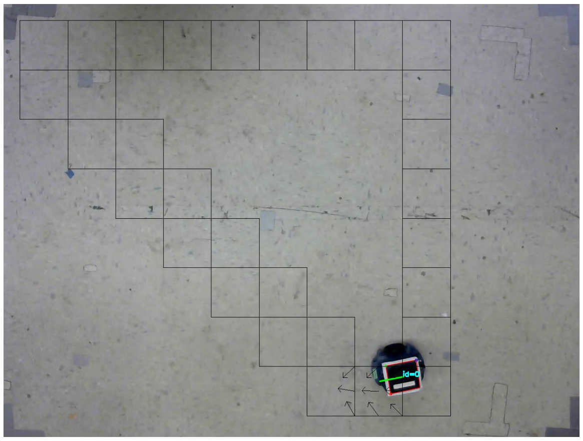


Figure 7.11: Course used to calibrate the robots and simulation. The physical and simulated robots were each driven through 10 laps of the course. Each box is the domain of one or more controller in the calibration path. The arrows represent the vector field of the active controller.

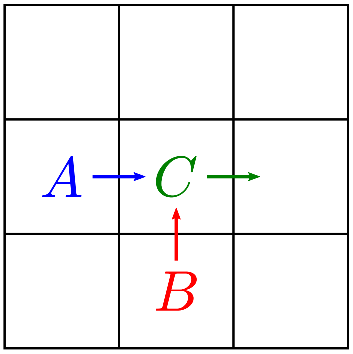
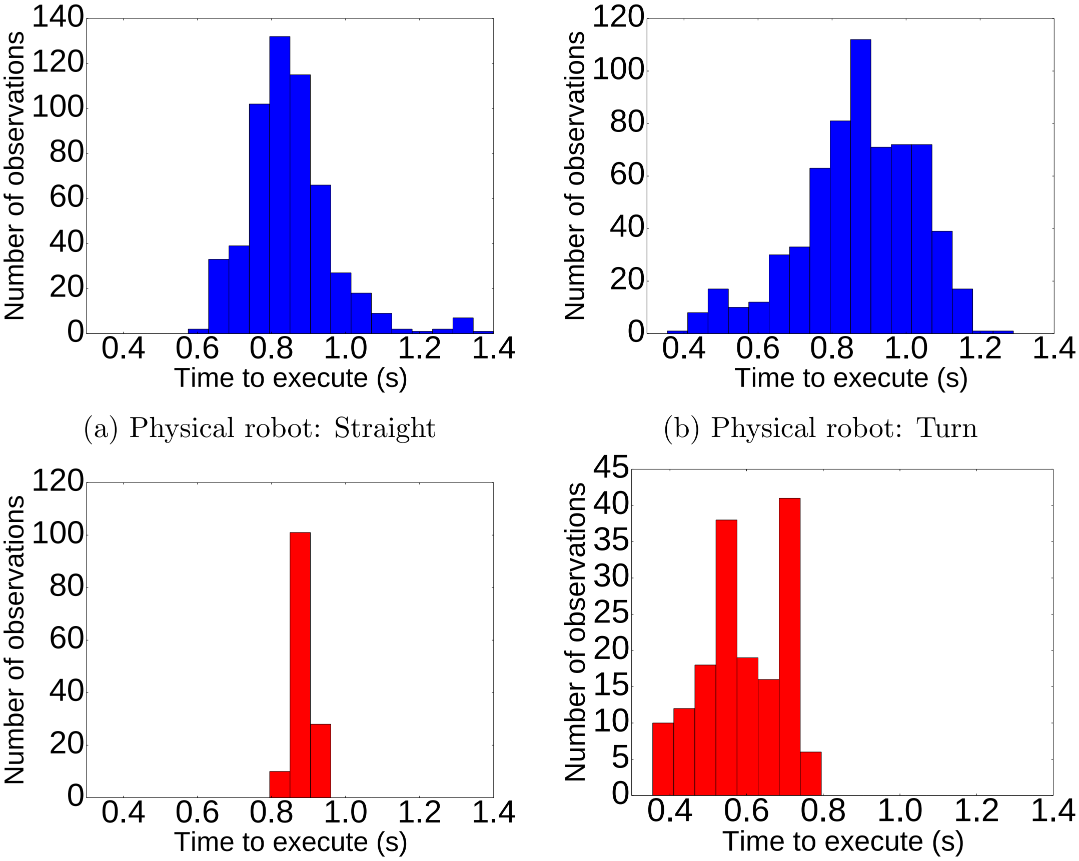


Figure 7.12: Controller C is labeled as a straight controller when prepared by controller A and as a turn controller when prepared by controller B.

four radii clearance. From a practical standpoint, we have never observed a collision between robots that were executing controllers that did not share a vertex.

To calibrate the simulation and estimate the duration of each controller we ran every physical robot and the simulated robot through 10 laps of a calibration course (Figure 7.11). Controllers were divided into two categories depending on the relative geometry of the controller and its preparing controller: straight and turn. A controller is labeled a straight controller if its goal face is parallel to the goal face of its preparing controller, and is otherwise labeled as a turn controller (Figure 7.12). Note that the same controller may be assigned both labels when prepared by different controllers.



(a) Physical robot: Straight

(b) Physical robot: Turn

(c) Simulated robot: Straight

(d) Simulated robot: Turn

Figure 7.13: Histogram of the time required for a physical and simulated robot to execute a turn or a straight controller

The physical robots took 0.85 ± 0.12 seconds to execute a straight controller, and 0.87 ± 0.16 seconds. Covering 0.15 meters in 0.88 seconds suggests that the physical robots were traveling at an average of 0.17 m/s, instead of the commanded 0.12 m/s. We therefore sped up the simulated robots by a factor of 1.4, which led the simulated robots to executing a straight controller in an average of 0.88 ± 0.03 seconds and a turn controller in an average of 0.59 ± 0.10 seconds. The simulated robots were able to execute the turn controllers in less time because they were able to cut the corner to a degree dependent upon their position when entering the controller, explaining the increased variance. The physical robots were able to cut the corner on turn controllers as well, explaining the lower minimal time to execute a turn controller compared to a straight controller. However, the physical robots are much less responsive than the simulated robots, and so tended to overshoot while executing the diagonal section of the path, leading to a very high variance.

M* and ATAJPG

We choose to use a time resolution of 0.3 seconds for the ATAJPG, and set the nominal duration of each controller to the mean duration for robots with the same duration, recalling that the nominal duration of a controller depends upon its preparing controller, and round to the nearest whole number of vertices. The nominal duration for physical robots executing straight and turn controllers and for the simulated robots executing straight controllers was 0.9 seconds, and so these controllers were represented by three vertices. The simulated robot took approximately 0.6 seconds to execute a turn controller, and so those were represented by two vertices.

UM*

To combine sequential composition and planning with uncertainty, we used UM* and the delay model from section 6.4.1. We used a constant delay probability, with specialized delay probabilities for different controllers left as future work. For the delay model from section 6.4.1 to work, the nominal duration must be the minimal time in which a controller can be executed, which for the physical robots is approximately 0.6 seconds for both the straight and turn controllers (Figure 7.13). The delay probability must be chosen so that the resulting belief distribution for controller execution time will best match the observed distribution of controller durations. Assume a controller is represented by n vertices in the joint prepares graph or ATAJPG, where each vertex represents δ_t seconds of controller execution. We wish to find the probability that a robot will take k steps to execute the controller, for a duration of $k\delta_t$, when the robot has a P_{delay} probability of delaying at each step. This is equivalent to the probability that the robot delays $k-n$ times in the first $k-1$ steps, followed by not delaying at the final step, which is given by

$$P\left(k\right) = \begin{pmatrix} k-1 \\ k-n \end{pmatrix} P_{\text{delay}}^{k-n}\left(1-P_{\text{delay}}\right)^{n-1}\left(1-P_{\text{delay}}\right) \tag{7.8}$$

$$= \begin{pmatrix} k-1 \\ k-n \end{pmatrix} P_{\text{delay}}^{k-n}\left(1-P_{\text{delay}}\right)^{n} \tag{7.9}$$

Equation 7.9 is the negative binomial distribution which has a mean value [137]

$$\bar{k} = \frac{n}{1-P_{\text{delay}}} \tag{7.10}$$

Given a fixed nominal duration t_{nom}, $n = \frac{t_{nom}}{\delta_t}$. Therefore, for a fixed mean duration $\bar{t} = \bar{k}\delta_t$

$$\bar{k}\delta_t = \left(\frac{\frac{t_{nom}}{\delta_t}}{1-P_{\text{delay}}}\right)\delta_t \tag{7.11}$$

$$\bar{t} = \frac{t_{nom}}{1-P_{\text{delay}}} \tag{7.12}$$

which is independent of the number of vertices used to represent a controller. Thus given an observed nominal duration and mean duration, the appropriate delay probability P_{delay} is

$$P_{\text{delay}} = \frac{\bar{t}-t_{nom}}{\bar{t}}. \tag{7.13}$$

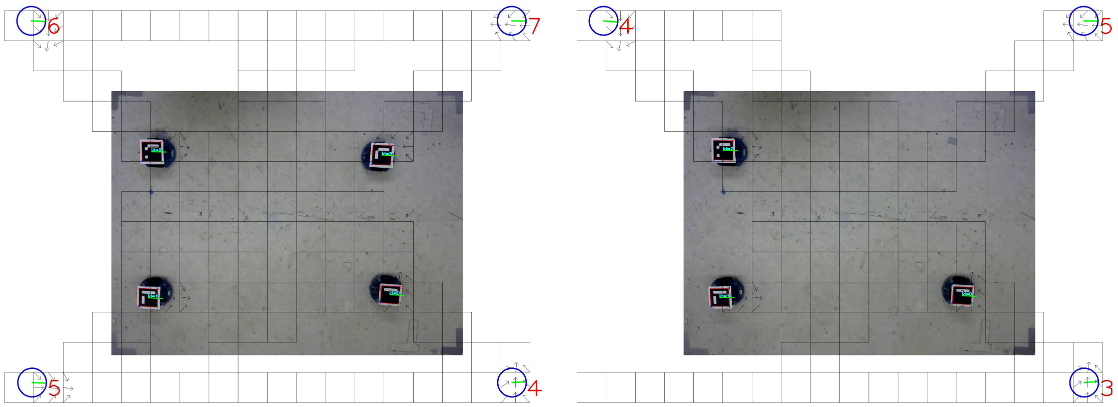
For straight controllers with $t_{nom} = 0.6$ and $\bar{t} = .85$, (Equation 7.13) implies $P_{\text{delay}} = 0.29$, while for turn controllers with $t_{nom} = 0.6$ and $\bar{t} = 0.87$ $P_{\text{delay}} = 0.31$. Therefore we use $P_{\text{delay}} = 0.3$.

UM* and the ATAJPG

Our current UM* implementation does not support variable delay probabilities, so we must use $P_{\text{delay}} = 0.3$ for all controllers. As mentioned earlier, the minimum time for a real robot to execute any controller is approximately 0.6 seconds. A simulated robot takes a minimum of approximately 0.4 seconds, which with $P_{\text{delay}} = 0.3$ implies an mean duration of 0.6 seconds which closely matches the observed value. A simulated robot executes a straight controller in a minimum of 0.8 seconds. However, with a fixed delay probability, this would correspond to a mean duration of 1.1 seconds, which is well above the observed mean. For the purposes of UM* planning on the ATAJPG we treat simulated robots as being able to execute a straight controller in a minimum of 0.6 seconds, like the real robots. The time resolution for the ATAJPG was set to 0.2 seconds per vertex, which meant that a simulated robot executing a turn controller would be represented by 2 vertices, and all other controllers by 3 vertices.

7.5.1 Test Cases

UM* cannot handle as many robots as M* so we tested two problems, one involving six robots and the other eight robots (Figure 7.14). The initial and goal configurations of the robots were defined by two rectangles; the inner rectangle lying entirely within the field of view of the camera, and the outer rectangle surrounding the inner rectangle outside the field of view. In the eight robot case, physical robots were placed at each vertex of the inner rectangle, and simulated robots were placed at the vertices of the outer rectangle. The goal configuration for both types of robots was the vertex directly opposite the robot's starting vertex on the appropriate rectangle. This induced a double 'X' pattern, leading to mutual interaction between all robots. In the six robot case one physical and one simulated robot were omitted.



(a) Eight robot problem

(b) Six robot problem

Figure 7.14: Initial configuration of the eight **(a)** and six **(b)** robot problems. The empty blue circles with a red number are the simulated robots. Robots seek to move to the diametrically opposite position. The squares are the domains of the controllers used in one solution to the problem.

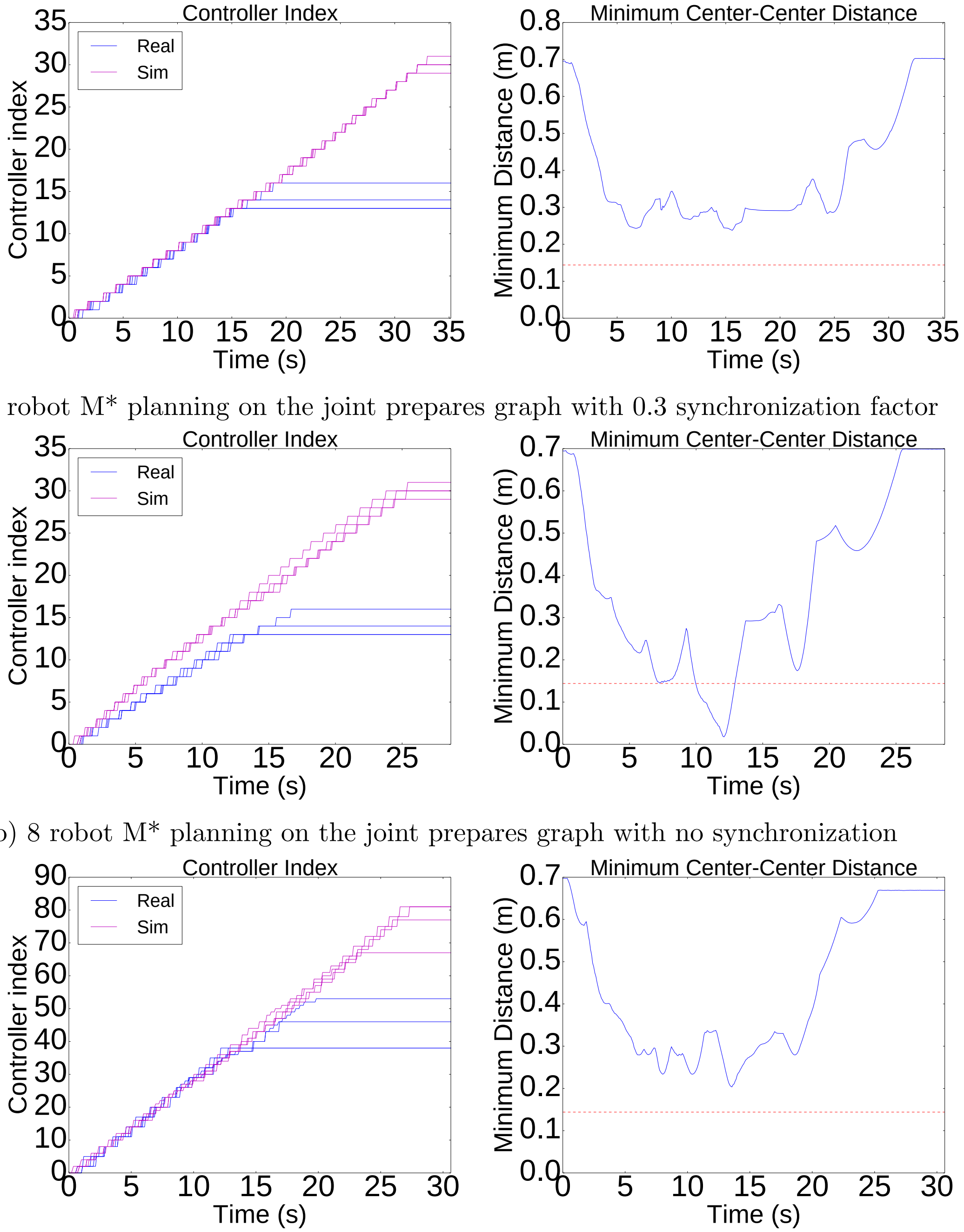
The problems described here are difficult. Each robot effectively occupies a space equal to two squares (as every pair of robots must be separated by an empty square). The camera's field of view is only 11 squares by 8 squares, and all robots seek to pass through this region. In the context of chapter 3 this would be roughly equivalent to solving a problem where 6 or 8 robots must pass through an area that is 5x4 (remember that each robot occupies 2 squares), which is a rather high density.

M*

We first describe the results when using M* as the planner and not explicitly considering uncertainty. We tested three conditions; planning on the joint prepares graph with a synchronization factor of 0.3, planning on the joint prepares graph with no synchronization, and planning on the ATAJPG with no synchronization. We ran each condition three times, however the results for each trial were very similar so we only plot results for the first trial. The plans generated by M* would always be safe if executed exactly as planned by the robots. To track how closely the robots adhere the plan we plot the minimal distance between the centers (not body frames) of the robots and the controller indices of each robot as a function of time. Recall that a sequential composition plan assigns each robot a sequence of controllers. The controller index of a robot is the index of the controller the robot is currently executing within the planned sequence of controllers.

When M* plans of the joint prepares graph and robots are run with a synchronization factor of 0.3 the robots remain almost perfectly synchronized (Figure 7.15a), and as a result stay a safe distance from one another. When the synchronization is removed, the simulated robots execute their plans significantly faster than the physical robots (Figure 7.15b). The resulting synchronization errors leads to one grazing collision between a real and simulated robot at 7 seconds and a serious collision where a simulated robot almost completely overlapped a real robot for several seconds starting 10 seconds into the run. We believe that the difference in speed comes from the simulated robot being able to execute turn controllers more quickly than physical robots, and planning on the joint prepares graph implicitly assumes that the time required to execute each controller is the same. The ATAJPG accounts for deterministic differences in execution time for different controllers, including whether they are executed by real or simulated robots. As a result, even without explicit synchronization the robots are almost as coordinated when executing a plan computed in the ATAJPG (Figure 7.15c) as they were when executing a plan computed on the joint prepares graph with active synchronization (Figure 7.15a). We note that the physical robots showed great consistency; the paths followed in all three replicates of a given trial were nearly identical which likely contributed to the efficacy of planning on the ATAJPG.

Running M* on the six robot test case (Figure 7.14b) produced qualitatively similar results. When plans were generated in the joint configuration graph and no synchronization was applied, the simulated robots moved through their plans more



(a) 8 robot M* planning on the joint prepares graph with 0.3 synchronization factor

(b) 8 robot M* planning on the joint prepares graph with no synchronization

(c) 8 robot M* planning on the ATAJPG with no synchronization

Figure 7.15: The controller index and minimum distance between any two robots for the eight robot trial (Figure 7.14b) with M* planning. The controller index is the index of the controller in a robot's plan that the controller is executing at a given time. If the robots were perfectly synchronized, they would always have the same controller index. If robots are closer together than 2 Scribbler radii (red dashed line) they are in collision.

Algorithm	Robots	Planning Time (s)
M*	8	0.4
M* ATAJPG	8	4.5
M*	6	0.2
M* ATAJPG	6	0.8
UM*	6	74.9

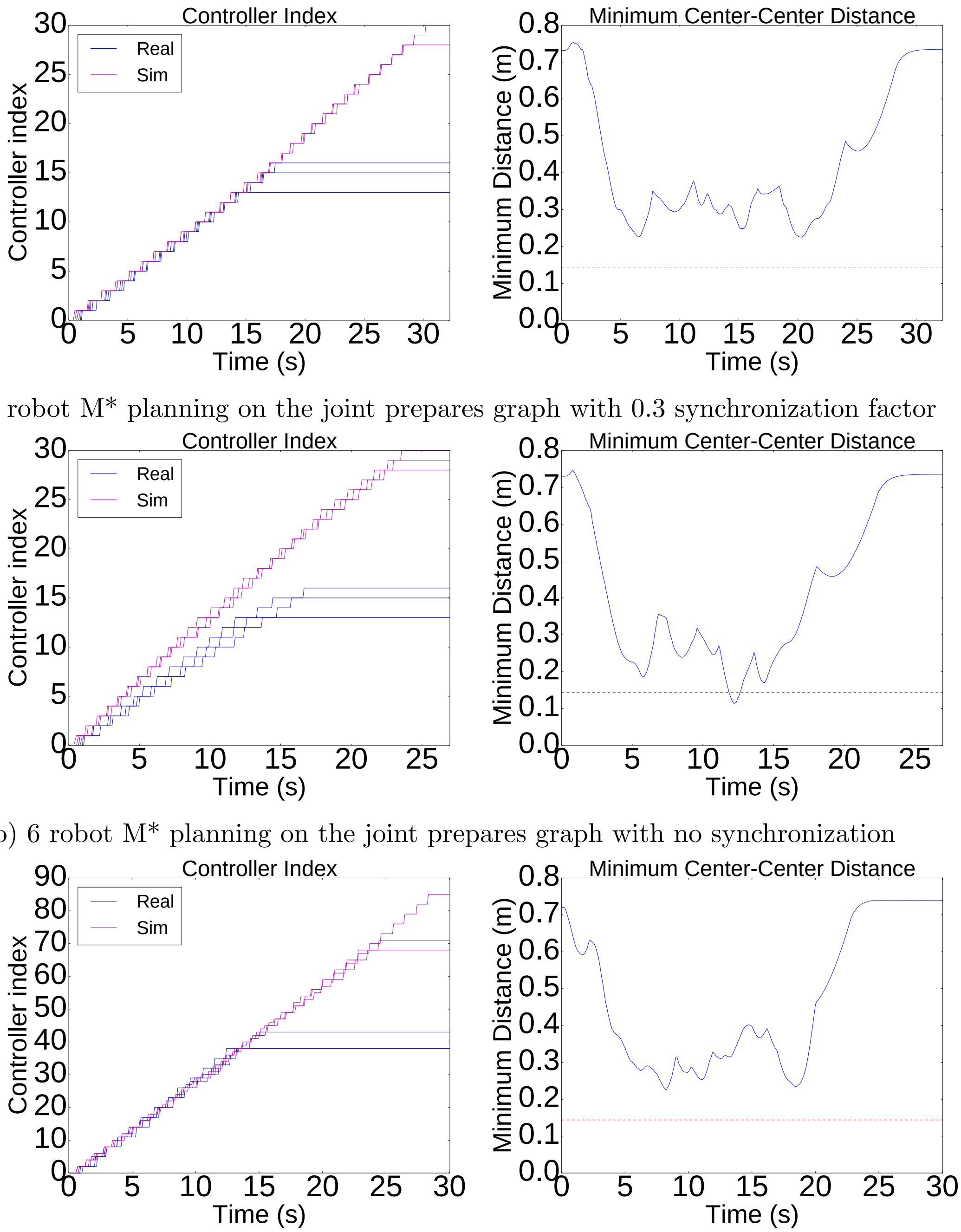
Table 7.1: Time to compute plans for multirobot sequential composition experiments

quickly, leading to a robot-robot collision (Figure 7.16b). When path execution was synchronized (Figure 7.16a) or planning was conducted on the ATAJPG and execution was not synchronized (Figure 7.16c) coordination between the robots was much better and no collisions occurred.

The time required to compute the plan for each problem case is given in table 7.1. The ATAJPG for M* contains approximately three times as many vertices as the joint prepares graph, so we would expect M* to take three times as long to find a path in the ATAJPG as the joint prepares graph. For the six robot problem, planning on the ATAJPG took four times as long as on the joint prepares graph, which is approximately as expected. However, planning on the ATAJPG took M* 10 times longer than planning on the joint prepares graph for the eight robot case. This can be explained by the fact that M* identified and avoided a significant collision while planning on the ATAJPG that was not found when planning on the joint prepares graph, which would have increased the planning time.

UM*

The comparatively high robot density caused significant problems for planning with uncertainty, as UM* is significantly less able to deal with large numbers of robots than M* (Figure 6.6a). Furthermore, the delay probability of 0.3 is significantly higher than the delay probability used in chapter 6. Taken together, this means that the belief distributions will be relatively larger. For UM* to solve the 6 robot

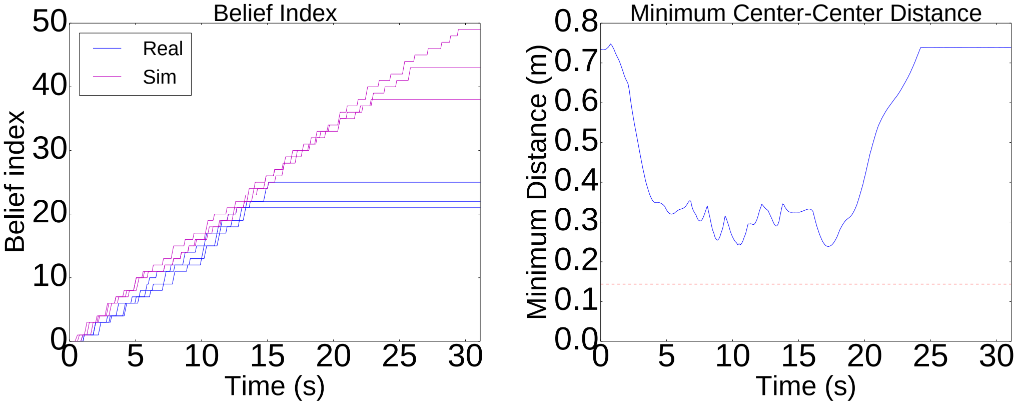


(a) 6 robot M* planning on the joint prepares graph with 0.3 synchronization factor

(b) 6 robot M* planning on the joint prepares graph with no synchronization

(c) 6 robot M* planning on the ATAJPG with no synchronization

Figure 7.16: The controller index and minimum distance between any two robots for the six robot trial (Figure 7.14b) with M* planning. The controller index is the index of the controller in a robot's plan that the controller is executing at a given time. If the robots were perfectly synchronized, they would always have the same controller index. If robots are closer together than 2 Scribbler radii (red dashed line) they are in collision.



(a) 6 robot UM* planning on the joint prepares graph with no synchronization

Figure 7.17: The controller index and minimum distance between any two robots for the six robot trial with UM* planning. If the robots were perfectly synchronized, they would always have the same controller index. If robots are closer together than 2 Scribbler radii (red dashed line) they must be in collision.

problems, the belief distributions had to be pruned aggressively, removing any state whose probability mass was below 0.1, which is known to reduce planning accuracy (Figure 6.4).

UM* plans a sequence of beliefs rather than a sequence of controllers, where each belief is a probability distribution over robots. We therefore plot the index of the current most likely belief for each robot to assess coordination between the robots (Figure 7.17). Although the coordination was not as good as the M* results with synchronization or with planning on the ATAJPG, the path was still safely executed, thanks to UM* explicitly accounting for variation in execution time of controllers. The time required to execute the path was approximately the same as the time required to perform the M* path planned on the ATAJPG, indicating that no unreasonable diversions were generated.

Finding a path in the joint prepares graph took UM* 75 seconds, when M* was able to find a solution in 0.2 seconds, which indicates planning with uncertainty is much more difficult on the chosen test case. UM* failed to find a path in the ATAJPG even when given 20 minutes. Each controller may be represented by up to three vertices in

ATAJPG. Thus for a belief in the ATAJPG to represent the distribution of the robot in the workspace with the same accuracy, its support must cover three times as many vertices in the ATAJPG as in the joint prepares graph (*i.e.* the pruning threshold must be a third of its value for planning in the joint prepares graph). Profiling indicates that computing the impact of collisions on the belief distribution took about half the planning of the planning time. Our collision checking code is quadratic in size of the support of the belief distributions. Given that the ATAJPG is approximately three times the size of the joint prepares graph, and collision checking would take nine times as long, we might expect UM* to take 27 times longer to find a path in the ATAJPG, even if no additional collisions were found. However, UM* could not find a solution in the ATAJPG even when the pruning threshold was set at 0.1, and thus the support of the belief contained the same number of vertices, but covered 1/3 of the area in the workspace as planning in the joint prepares graph. Our best explanation is that UM* search on the ATAJPG found a collision that was very hard to resolve, causing failure of the planner.

7.6 Conclusions

In this chapter, we showed that M* can be used to enable combined planning and control of multiple robots using the sequential composition framework. We adapted region automata for to the multirobot sequential composition framework to address deterministic differences in the duration of controllers, producing the time augmented joint prepares graph. To allow direct reuse of existing implementations of M* we show that the time augmented joint prepares graph can be approximated as the direct product of single robot approximate time augmented prepares graphs, producing the ATAJPG. We then describe how UM* can be applied to multirobot sequential composition to account for stochastic differences in execution time.

We validate our results on experiments involving up to eight agents, of which four were physical robots and four were simulated robots. M* was able to produce plans that could be safely executed without centralized synchronization by planning in the ATAJPG. UM* was also able to compute plans that could be safely executed without centralized synchronization, but only for a simplified 6 robot problem. However, UM* was unable to find plans in the ATAJPG even in the six robot case.

The robots used in these experiments proved very consistent; each test case was run three times and the resulting trajectories were qualitatively the same each time. We believe this contributed significantly to the effectiveness of the M* planning in the ATAJPG in comparison to UM*. We also suspect that as a robot executed multiple controllers, differences in execution time tended to cancel out, and thus we believe we overestimated the effective uncertainty in the position of the robots. If true, this would help explain the performance of UM*.

Chapter 8

Conclusions

The MPP problem deals with truly enormous spaces; realistic problems can be constructed where enumerating the possible actions that a system can take at even one time step is not physically possible[1]. Such large problems can still be solved due to the direct product structure inherent to the MPP problem. The direct product structure arises from the fact that the joint configuration space is the product of single robot configuration spaces, and the action set for the system as a whole is the product of the action sets of the individual robots. Finally, the constraints are the logical conjunction of the constraints on robots or pairs of robots. Taken together, the direct product structure implies that a subset of robots can be identified that are responsible for any given constraint violation, and the constraint can be resolved by considering alternate paths for only the responsible robots. Exploiting the direct product structure is what differentiates a true MPP algorithm from a planner that just happens to be applied to a multirobot system.

MPP variants alter the direct product structure. The action set of the CPP problem deviates from the direct product structure at a small number of configurations,

[1] A 200 robot system on a four-connected grid such as those solved by inflated rM* in figure 3.8 has $5^{200} \approx 10^{139}$ possible actions. The universe contains at most 10^{123} bits of information [113]

where the robots can form or dissolve teams. Even though the number of states with non-product action sets is low, the necessary coupling between robots increases dramatically. The MPPU problem fundamentally lacks the direct problem of the MPP problem, in that the result of a robot taking an action depends on the state and action of all the other robots, to the extent that a robot can reach states as part of a multirobot system it could not reach on its own. However in practice, the structure of the MPPU problem is sufficiently close to a direct product that approaches developed for the MPP can still be applied.

In this thesis, we developed a general framework for solving the MPP problem called subdimensional expansion, which directly and explicitly exploits the product structure of the MPP problem. Subdimensional expansion starts by constructing a search space by planning for each robot individually, and then explores the search space for a path that does not violate any constraints. When it encounters a constraint violation, subdimensional expansion expands the search space by only considering alternate paths for the robots which could actually resolve the constraint violation. Both the construction of the low-dimensional search space and the identification of a small subset of robots responsible for a given constraint violation are only feasible due to the product structure of the MPP problem.

Subdimensional expansion is both flexible and efficient. Because subdimensional expansion functions by defining a search space, it can be paired with the appropriate underlying planner to allow rapid planning for robots with both many and few degrees of freedom, and can alter its criteria for growing the search space to account for the different dependencies between robots in variants of the MPP problem. Although flexibility often costs efficiency, variants of M* are currently state-of-the art for finding optimal and ϵ-suboptimal solutions to MPP on graphs[2].

[2] The more recent work on enhanced CBS (ECBS) [15] can outperform inflated rM* using tight bounds, but inflated rM* performs better with loose bounds. We also anticipate that inflated rM* can be combined with ECBS in the same way ODrM* was combined with MA-CBS.

Appendix A

Notation

The notation in this thesis can get complex, as we deal with many similar objects that describe different robots, or sets of robots. To make the notation more comprehensible, a standard format is employed. Superscripts are used to denote which robots a given object describes, while subscripts are used to denote specific instances. For instance, v_k^i refers to the k'th vertex in the configuration graph of r^i.

Given an object describing the system as a whole, for instance v_k in the joint configuration graph, adding a superscript refers to the state of a specific robot. Thus v_k^i would be the configuration graph vertex that describes the configuration of r^i when the system as a whole is the joint configuration described by v_k.

The symbols i and j are reserved for short term indexing of robots, while k and ℓ are used to index specific instances of vertices in graphs, the collision set associated with a vertex, etc. Then general rule is that subscripts are fixed within a paragraph, but not over longer time scales. Thus if v_k refers to the same vertex if it appears twice in a single paragraph, but this does not imply that v_k would still refer to that same vertex in a later paragraph. Furthermore, if the same subscript appears next to two different types of objects in the same paragraph, the objects are implied to be related. For instance, C_k would be the collision set of v_k.

Glossary

active graph ($G^{\mathrm{act}}()$) The equivalent of the joint configuration graph when a specific set of teams are active in CMS. Associated with a maximal incomparable subset of $V^{\mathrm{tm}} \cup V^{\mathrm{gl}}$. 89–91

active team ($\mathcal{T}^{\mathrm{act}}$) The set of teams that are currently performing tasks. 89–94, 101, 103, 197

approximate time augmented prepares graph A variant of the prepares graph where each controller is represented by a number of vertices corresponding to its maximal nominal duration.. xii, 167, 187

associated robot (C^{assoc}) The robots that potentially collide with a threshold robot at the current vertex or some explored successor. Used in UM*. 129, 130

backpropagation set The backpropagation set of a point q in the search space is the set of all points for which the underlying planner has considered q as a possible successor. 18, 19, 22–27, 34, 93, 130, 131

collision function 31

1) ($\Psi : Q \to \mathcal{P}(I)$) A mapping from a point in the joint configuration space to the set of robots that are involved in a collision at that configuration. 6

2) ($\Psi ij : E^i \times E^j \to \{0, 1\}$) A mapping that returns one if two robots would collide if they simultaneously traverse a specified pair of edges, and zero otherwise. 135

collision set (C) The collision set of a point q is the set of robots involved in a collision at some successor state of q in the search tree of M*. xv, 18–20, 22–27, 32–35, 37–40, 42, 43, 45–48, 57, 58, 60, 61, 65, 66, 89, 90, 92–94, 104, 128–132, 146, 191–195

complete A planning algorithm is complete if it is guaranteed in finite time to either find a solution or prove that no solution exists. 11, 14, 29, 35, 36, 38, 42, 44, 45, 47, 50, 61, 79, 103, 106, 127, 154

configuration graph (G^i) A graph that represents the configuration space of some system. Each vertex represents a state in the configuration space, and edges represent valid transitions between states. 29, 30, 33, 45, 125, 135, 191, 195

conflict set (C) One of two parts of the extension of the collision set for use in CMS. The conflict set of a vertex v_k in the task augmented joint configuration graph is the set of vertices in the task graph that correspond to teams that collided at some successor of v_k in the search tree.. 89, 90, 92–94, 96–102, 104, 105, 193, 194

conflict set element An element of the conflict set in rCMS. Each conflict set element is a set of task graph vertices corresponding to colliding teams. The coupled sets of each conflict set element of a given vertex in the task augmented joint configuration graph must be disjoint (if not, the effected conflict set elements are merged. 102, 103, 105, 107, 194, 196

constraint manifold The submanifold of a configuration space which satisfies a set of constraints. In this work, constraint manifold is a term of convinence, and can be used to refer to subspaces which satisfy a set of constraints, but are not manifolds.. 87, 195

coupled set (Γ) One of two parts of the extension of the collision set for use in CMS. The set of teams that need to perform coupled planning given a particular conflict set. 89, 90, 92–104, 194

disabled A disabled team cannot collide with other teams, and cannot move or incur cost, implying that its contribution to the f-value of a vertex is constant. As a result, a disabled cannot influence the path of any other team. rCMS uses disabled robots to form subproblems without having to form new CPP problems that contain a variable number of robots. 102, 103, 194

dissolving team The dissolving teams of a transition action are the set of teams that combine to form a new set of teams, the forming teams. 82, 83, 85, 92–94, 194

duration Random variable describing the time required for a robot to reach the goal set of a controller $\mathcal{C}$ from the goal set of the preparing controller. 162–164, 176, 178–180, 187

explored graph (G^{exp}) The portion of the search graph which M* has explicitly constructed (Section 3.3.1). 36–42, 44, 195

forming team The forming teams are the teams formed by a given transition action taken by a set of dissolving teams. 82, 194

free configuration space (Q^i_{free}) The subspace of the configuration space of robot r^i which is free of self-collisions or collisions with environment obstacles. 6, 29

individual policy (ϕ^i) Term in M*. A policy which at every point in the configuration space of a robot r^i dictates the best possible action if there were no other robots. 17–21, 26, 27, 32, 33, 43, 46, 47, 51, 52, 54, 71–73, 76, 86, 89, 92–94, 100, 103–106, 129, 195, 196

individually optimal path (π^i_ϕ) The optimal path for robot r^i if no other robots were present. 17, 19, 33, 36, 38, 43, 54, 196

joint belief graph A graph representing the joint belief space of a multirobot system symbol. 128–130

joint belief space ($\mathcal{B} = \left\{b : Q \to \mathbb{R}^{\geq 0} \middle| \int_q b\left(q\right) \mathrm{d}q = 1\right\}$) The space of probability distribution functions defined on the product of the joint configuration space, where an state col^i has been added to the configuration space of each robot r^i to denote that robot having collided. An element of the joint belief space describes the probability of the robots occupying a specify set of positions and having collided with other robots or not. 121, 122, 128, 130, 194

joint configuration graph (G) A graph representing the joint configuration space of a multirobot system. In this work, constructed by taking the tensor product of the configuration graphs of the constituent robots. xi, 30, 31, 36, 37, 40, 43–45, 87, 89, 90, 94, 125, 182, 191, 193, 195–197

joint configuration space (Q) The configuration space representing a multiagent system. Constructed by taking the direct product of the configuration spaces of the constituent agents. 3, 6–8, 11–13, 15, 16, 18–20, 30, 32, 36, 51, 53, 67–70, 73, 74, 90, 121, 122, 125, 127, 129, 159, 189, 193–195

joint policy (ϕ) A policy for a multirobot system where each robot obeys its individual policy. 33

joint policy path (π_ϕ) The path produced when all robots follow their individual policies. In CMS, the path produced when all teams follow their individual policies. 17, 18, 20, 33, 104, 105

joint prepares graph The tensor product of the prepares graphs associated with individual robots of a multirobot system. Each vertex in the joint prepares graph specifies which single-robot controller should be executed by each individual robot. xii, xiii, 160–162, 165–167, 178, 181–187, 197

joint PRM A PRM defined in the joint configuration space of a multirobot system. When formed from the product of single robot PRMs, can be used as the joint configuration graph in M*. 69, 70, 77

limited neighbor (V_k^{nbh}) The neighbors of a vertex v_k in the joint configuration graph which can be reached when the robots not in the collision set of v_k obey their individual policies (Equation 3.3). 32, 33, 36–38, 42, 43, 45, 89, 92, 94, 96–101, 103, 195

manifold graph ($G_{\mathcal{M}}$) A graph representing the constraint manifold of a team of robots. Used in CMS. 88–91, 94, 105, 107

meta-agent A set of robots treated as a single, more complex robot. 11, 47, 52, 54

neighbor graph (G^{nbh}) Construct in M*. Consists of the explored graph, the limited neighbors of the vertices in the explored graph, and the edges connecting the vertices in explored graph to their limited neighbors. 36, 37

nominal duration ($t_{nom} : \mathfrak{C} \times \mathfrak{C} \to \mathbb{R}^+$) $t_{nom}(\mathcal{C}_k^i, \mathcal{C}_\ell^i)$ is the nominal time required for robot r^i to finish executing the controller $\mathcal{C}_k^i$ from the goal set of $\mathcal{C}_\ell^i$, $\mathcal{C}_\ell^i \succ \mathcal{C}_k^i$. 164–167, 174, 178, 179, 193

optimal A planning algorithm is optimal if it is guaranteed to find the minimal cost path. 1, 5, 11, 29, 35, 36, 38, 42, 44, 45, 47, 50, 63, 64, 106

out-neighbor The out-neighbors of a vertex v_k in a directed graph are the vertices v_l such that the directed edge from v_k to v_l exist.. 18, 30, 33, 36, 37, 40, 48, 55, 135, 165

permutation invariant multirobot path planning Multirobot path planning where a robot must reach each goal location, but the assignment of robots to goals is a free parameter. 6, 13, 14, 16

policy graph (G^{ϕ}) The subgraph representing the individually optimal paths starting from $G^{\text{nbh}} \setminus G^{\text{exp}}$. 36–39

policy tree ($\mathcal{T}^{i}$) A RRT grown from the goal state of a robot r^{i} that is used to compute the robot's individual policy in sRRT. 71, 73, 74, 76

prepares ($\succ$) A controller A prepares controller B, written $A \succ B$ if the goal set of A lies within the domain of attraction of B. 158, 159, 161, 165, 167, 176, 196

prepares graph A directed graph used in sequential composition with edges pointing from a controller, represented as a vertex, to the controllers which it prepares. 159–161, 166, 193, 195, 196

preparing controller

Singular The controller executed by a robot before entering the current controller

Plural The set of controllers that prepare the current controller

. 161–163, 167, 176, 178, 194

resolve set ($\mathcal{T}_{res}\left(C^{i}\right)$) The subset of teams that preceed the teams in a conflict set element. These are the robots that need to be explicitly planned for to find a path that prevents the collision(s) that produced the conflict set element. 102, 103, 105, 106

search graph (G^{sch}) In M*, the implicitly defined subgraph of the joint configuration graph searched by the underlying planner (Section 3.3.1). In CMS the search graph is a subgraph of the task augmented joint configuration graph. 36, 38–44, 104, 106, 194, 196

sequential composition A combined control/planning paradigm where a set of controllers are placed in the environment such that each controller converges to the domain of attraction of one or more other controller. Instead of planning a trajectory in the configuration space, planning in sequential composition returns a sequence of controllers to execute, by searching the prepares graph. viii, xii, xv, 4–6, 157–160, 162, 164, 168, 172, 178, 182, 184, 187, 196

subdimensional expansion An approach to generating a low dimensional search space for multirobot path planning. xi, 2, 3, 5, 15, 17–26, 29, 121, 190, 200

ϵ-suboptimal A path is ϵ-suboptimal if it costs no more than ϵ times the cost of the optimal path. A multirobot path planner is ϵ-suboptimal if guaranteed to find ϵ-suboptimal paths. xi, 3–5, 61, 64, 65, 79, 95, 115, 190

subproblem A subset of a CPP problem that can be solved independently of the rest of the problem. Characterised by the subgraph of the task graph dominated by a set of task graph vertices. 102–107, 194

synchronization factor Factor by which the control manager slows robots that are running fast, raised to the power of the number of steps the robot is in front of the slowest robot. 172, 181–183, 185

task augmented joint configuration graph (G^{aug}) An extension of the joint configuration graph the active teams and their positions. Each vertex represents a set of ordered pairs $(\tau^i, v^i_{\mathcal{M}})$. viii, 89–93, 103–106, 193, 194, 196

task constraint A set of constraints placed on the robots performing a given cooperative task that must be satisfied for the task to be completed successfully. 80–82, 86–88, 104

task graph ($G^{\text{st}} = \{V^{\text{tm}}, V^{\text{gl}}, V^{\text{tr}}, E^{\text{st}}\}$) A directed tripartite graph where one subset of vertices correspond to teams that have never reached their goal, another to teams that have reached their goal at least once, and the third to transitions between teams.. xi, 83–85, 91, 93–95, 102, 103, 193, 194, 196

task list The task list for robot r^i is an ordered list of tasks which r^i is required to perform. 82

threshold robot (C^{thresh}) The set of robots whose probability of collision exceeds the permitted threshold at the current vertex or at one of the explored successors. Used in UM*. 129, 130, 132, 193

time augmented joint prepares graph A variant of the joint prepares graph that accounts for different nominal times of execution for single robot controllers (Figure 7.4). xii, 160, 164–168, 187

transition graph A graph used in CMS to allow exploration of team formation/dissolution actions. 89, 91

underlying planner A multirobot path planning algorithm used to explore the search space generated by subdimensional expansion. Typically a coupled planner such as A* or RRT. 18–20, 29, 193, 196

Acronyms

ATAJPG Approximate Time Augmented Joint Prepares Graph. xiii, 165, 167, 168, 178, 180–188

CA* Constrained A*. 133

CBS Conflict Based Search [156] (Section 3.5.4). 51, 52, 59, 63, 64

CD* Constrained D*. 133, 154

CM* Constrained M*. viii, xii, 131, 133, 152–155

CMS Constraint Manifold Subsearch (Chapter 5). viii, xii, xv, 4, 5, 79, 82, 84–101, 103, 104, 106–109, 111, 114–116, 153, 193–197

CPP Cooperative Path Planning. A variant of the MPP problem where robots teams must form cooperative teams to perform tasks. xii, 7, 79–81, 83, 87, 89, 91, 92, 95, 101, 102, 189, 194, 196

EPEA* Enhanced Partial Expansion A*. A variant of A* tailored to the MPP problem [62, 71] (Section 3.5.3). xi, 12, 46, 48, 50, 53, 54, 56–61, 64, 65, 127

ID Independence Detection [167]. 15, 51–54

MA-CBS Meta-Agent Conflict Based Search [157] (Section 3.5.4). 15, 51–54, 58–64, 128

MPP Multirobot Path Planning. xii, 1, 3–8, 11, 12, 17, 20, 79, 83, 87, 121, 125, 127, 128, 154, 159, 189, 190, 199, 200

MPPU Multirobot Path Planning with Uncertainty. viii, xii, 7, 117, 118, 121, 122, 125, 127–129, 131, 133, 134, 142, 149, 154, 190

OD Operator Decomposition. A variant of A* tailored to MPP [167] (Section 3.5.3). xi, 12, 46, 48, 50, 53, 54, 56–61, 64, 65, 127

POMDP Partially Observable Markov Decision Process. 118–120

PRM Probabilistic Roadmap. 67–71, 75, 77, 195, 200

rCMS Recursive Constraint Manifold Subsearch (Chapter 5.5). viii, xii, 80, 101–103, 105–107, 109–115, 194

rM* recursive M*. vii, xii, xv, 46, 47, 54, 56–58, 60, 61, 65, 128, 130, 131, 133, 142–147, 149, 151, 154, 189, 190

RRT Rapidly-Exploring Random Tree. 67, 68, 71, 73–76, 120, 196, 197, 200

SAT Boolean Satisfiability. 12, 13

sPRM subdimensional Probabilistic Roadmap. A probabilistic MPP algorithm that uses M* to find a path in a product graph generated from PRMs. viii, xi, 4, 5, 68–71, 74–76

sRRT subdimensional Rapidly-Exploring Random Trees. An implementation of subdimensional expansion that uses RRTs as the underlying planner. viii, xi, 4, 5, 68, 71, 73–76, 196

UM* Uncertainty M*. viii, xii, xiii, 4, 5, 121, 128–133, 139, 140, 142–147, 149–155, 160, 164, 172, 178, 180, 184, 186–188, 193, 197

Bibliography

[1] Khaled Mohamed Al-Wahedi. *A Hybrid Local-Global Motion Planner for Multi-Agent Coordination*. PhD thesis, Case Western Reserve University, 2000.

[2] R Alami, S Fleury, M Herrb, F Ingrand, and F Robert. Multi-robot cooperation in the MARTHA project. *IEEE Robotics & Automation Magazine*, 5(1):36–47, mar 1998. ISSN 10709932.

[3] Faten Aljalaud and Nathan R Sturtevant. Finding Bounded Suboptimal Multi-Agent Path Planning Solutions Using Increasing Cost Tree Search. In *Sixth International Symposium on Combinatorial Search*, pages 203–204, 2013.

[4] Javier Alonso-Mora, Ross A Knepper, Roland Siegwart, and Daniela Rus. Local Motion Planning for Collaborative Multi-Robot Manipulation of Deformable Objects. In *IEEE International Conference on Robotics and Automation*, pages 5495–5502, Seattle, Washington USA, 2015. ISBN 9781479969227.

[5] Nancy M Amato, O. Burchan Bayazit, Lucia K Dale, Christopher Jone, and Daniel Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *International Workshop on the Algorithmic Foundations of Robotics*, 1998.

[6] Nancy M Amato, Ken a Dill, and Guang Song. Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. *Journal of computational biology*, 10(3-4):239–55, jan 2003. ISSN 1066-5277.

[7] V. Auletta, a. Monti, M. Parente, and P. Persiano. A Linear-Time Algorithm for the Feasibility of Pebble Motion on Trees. *Algorithmica*, 23(3):223–245, mar 1999. ISSN 0178-4617.

[8] Nora Ayanian. *Coordination of Multirobot Teams and Groups in Constrained Environments : Models , Abstractions , and Control Policies*. PhD thesis, University of Pennsylvania, 2011.

[9] Nora Ayanian and Vijay Kumar. Decentralized feedback controllers for multi-agent teams in environments with obstacles. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1936–1941. Ieee, may 2008. ISBN 978-1-4244-1646-2.

[10] Nora Ayanian and Vijay Kumar. Decentralized feedback controllers for multi-agent teams in environments with obstacles. *IEEE Transactions on Robotics*,

26(5):878–887, oct 2010.

[11] Nora Ayanian, Vinutha Kallem, and Vijay Kumar. Synthesis of feedback controllers for multiple aerial robots with geometric constraints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3126–3131. Ieee, sep 2011. ISBN 978-1-61284-456-5.

[12] Kianoush Azarm and Günther Schmidt. Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation. In *IEEE International Conference on Robotics and Automation*, number April, pages 3526–3533, 1997.

[13] Tucker Balch and Ronald C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998. ISSN 1042296X.

[14] Tomáš Balyo, Roman Bartak, and Pavel Surynek. Shortening Plans by Local Re-planning. In *24th International Conference on Tools with Artificial Intelligence*, Athens, Greece, 2012.

[15] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *Proceedings of the Sixth International Symposium on Combinatorial Search*, 2014.

[16] T D Barfoot and Christopher M Clark. Motion Planning for Formations of Mobile Robots. *Journal of Robotics and Autonomous Systems*, 46(2), 2004.

[17] Laura E Barnes, Mary Anne Fields, and Kimon P Valavanis. Swarm formation control utilizing elliptical surfaces and limiting functions. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics*, 39(6):1434–45, dec 2009. ISSN 1941-0492.

[18] Calin Belta and Vijay Kumar. Motion generation for formations of robots: a geometric approach. In *IEEE International Conference on Robotics and Automation*, number 3, pages 1245–1250, 2001. ISBN 0780364759.

[19] Calin Belta and Vijay Kumar. Optimal Motion Generation for Groups of Robots: A Geometric Approach. *Journal of Mechanical Design*, 126(January 2004):63, 2004. ISSN 10500472.

[20] Calin Belta, Volkan Isler, and George J Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5):864–874, 2005.

[21] Dmitry Berenson, Siddhartha S Srinivasa, Dave Ferguson, and James J Kuffner. Manipulation planning on constraint manifolds. In *IEEE International Conference on Robotics and Automation*, volume i, pages 625–632. Ieee, may 2009. ISBN 978-1-4244-2788-8.

[22] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Multi-agent path planning with multiple tasks and distance constraints. In *IEEE International*

Conference on Robotics and Automation, pages 953–959. Ieee, may 2010. ISBN 978-1-4244-5038-1.

[23] Zahy Bnaya, Roni Stern, and Ariel Felner. Multi-Agent Path Finding for Self Interested Agents. In *Proceedings of the Sixth International Symposium on Combinatorial Search*, pages 38–46, Leavenworth, Washington, USA, 2013.

[24] Robert Bohlin and Lydia E Kavraki. Path planning using lazy PRM. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 521–528. IEEE, 2000.

[25] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001. ISSN 0004-3702.

[26] Valerie Boor, Mark H Overmars, and A Frank van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1018–1023. IEEE, 1999.

[27] James Robert Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2383–2388, 2002. ISBN 0-7803-7398-7.

[28] James Robert Bruce and Manuela Veloso. Real-time multi-robot motion planning with safe dynamics. In *Multi-Robot Systems. From Swarms to Intelligent Automata*, volume III, pages 1–12. 2005. ISBN 1402033885. doi: 10.1007/1-4020-3389-3{_}13.

[29] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *IEEE International Conference on Robotics and Automation*, pages 723–730, 2011. ISBN 9781612843865.

[30] Stephen Buckley. Fast motion planning for multiple moving robots. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 322–326, may 1989.

[31] RR Burridge, Alfred A Rizzi, and Daniel E Koditschek. Sequential composition of dynamically dexterous robot behaviors. *International Journal of Robotics Research*, 18(6):534–555, 1999.

[32] Jan-P Calliess and Stephen J Roberts. Multi-Agent Planning with Mixed-Integer Programming and Adaptive Interaction Constraint Generation (Extended Abstract). In *Proceedings of the Sixth International Symposium on Combinatorial Search*, pages 207–208, Leavenworth, Washington, USA, 2013.

[33] Michal Cáp, Peter Novák, Martin Selecký, Jan Faigl, and Jií Vokínek. Asynchronous Decentralized Prioritized Planning for Coordination in Multi-Robot System. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3822–3829, 2013. ISBN 9781467363587.

[34] Michal Cáp, Peter Novák, Jií Vokínek, and Michal Pěchouček. Multi-agent

RRT *: Sampling-based Cooperative Pathfinding. In *Autonomous Robots and Multirobot Systems Workshop at AAMAS 2013*, 2013.

[35] Stefano Carpin and Enrico Pagello. On parallel RRTs for multi-robot systems. In *Proceedings of 8th Conference Italian Association for Artificial Intelligence*, pages 834–841. Citeseer, 2002.

[36] Hande Çelikkanat and Erol ahin. Steering self-organized robot flocks through externally guided individuals. *Neural Computing and Applications*, 19(6):849–865, mar 2010. ISSN 0941-0643.

[37] Luiz Chaimowicz, Thomas Sugar, Vijay Kumar, and Mario F M Campos. An architecture for tightly coupled multi-robot cooperation. In *IEEE International Conference on Robotics and Automation*, pages 2992–2997, 2001. ISBN 0780364759.

[38] Dong Eui Chang, Shawn C Shadden, Jerrold E Marsden, and Reza Olfati-Saber. Collision avoidance for multiple agent systems. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, number December, pages 539–543, Maui, Hawaii, 2003. ISBN 0780379241.

[39] Chien Chern Cheah, Saing Paul Hou, and Jean Jacques E. Slotine. Region-based shape control for a swarm of robots. *Automatica*, 45(10):2406–2411, oct 2009. ISSN 00051098.

[40] Minsik Cho and DZ Pan. A high-performance droplet routing algorithm for digital microfluidic biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1714–1724, 2008.

[41] Christopher M Clark, Stephen M Rock, and Jean-Claude Latombe. Motion planning for multiple mobile robots using dynamic networks. In *IEEE International Conference on Robotics and Automation*, pages 4222–4227, 2003.

[42] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. Single- and dual-arm motion planning with heuristic search. *The International Journal of Robotics Research*, nov 2013. ISSN 0278-3649.

[43] David C Conner, Howie Choset, and Alfred A Rizzi. Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies. In *Robotics: Science and Systems II*, pages 57–64, Philadelphia, PA, aug 2006. MIT Press.

[44] Iain D Couzin, Jens Krause, Richard James, Graeme D. Ruxton, and Nigel R Franks. Collective memory and spatial sorting in animal groups. *Journal of Theoretical Biology*, pages 1–11, 2002.

[45] Rongxin Cui, Bo Gao, and Ji Guo. Pareto-optimal coordination of multiple robots with safety guarantees. *Autonomous Robots*, 32(3):189–205, dec 2011. ISSN 0929-5593.

[46] Joseph C Culberson and Jonathan Schaeffer. Efficiently searching the 15-puzzle. Technical Report May, Deparment of Computing Science, University of Alberta,

Edmonton, Alberta, Canada, 1994.

[47] Henry W Davis, Anna Bramanti-Gregor, and Jin Wang. The Advantages of Using Depth and Breadth Components in Heuristc Search, 1988.

[48] Yann-Michaël De Hauwere, Peter Vrancx, and Ann Nowé. Learning what to observe in multi-agent systems. In *The 21st Benelux Conference on Artificial Intelligence*, 2009.

[49] Yann-Michaël De Hauwere, Peter Vrancx, and Ann Nowé. Learning multi-agent state space representations. In *9th International Conference on Autonomous Agents and Multiagent Systems*, pages 715–722, Toronto, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[50] Yann-Michaël De Hauwere, Peter Vrancx, and Ann Nowé. Solving Sparse Delayed Coordination Problems in Multi-Agent Reinforcement Learning. In Marek Vrancx, Peter and Knudson, Matthew and Grześ, editor, *Adaptive and Learning Agents*, volume 7113 of *Lecture Notes in Computer Science*, pages 114–133. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-28499-1{_}8.

[51] Boris de Wilde, Adriaan W ter Mors, and Cees Witteveen. Push and rotate: cooperative multi-agent path planning. In *12th International Conference on Autonomous Agents and Multiagent Systems*, pages 87–94, Saint Paul, Minnesota, 2013.

[52] Jory Denny and Nancy M Amato. The Toggle Local Planner for sampling-based motion planning. In *IEEE International Conference on Robotics and Autonomation*, pages 1779–1786. Ieee, may 2012. ISBN 978-1-4673-1405-3.

[53] Jaydev P. Desai and Vijay Kumar. Motion planning for cooperating mobile manipulators. *Journal of Robotic Systems*, 16(10):557–579, 1999. ISSN 0741-2223.

[54] Vishnu R. Desaraju and Jonathan P. How. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots*, 32(4):385–403, feb 2012. ISSN 0929-5593.

[55] Kurt Dresner and Peter Stone. A Multiagent Approach to Autonomous Intersection Management. *Journal of Artificial Intelligence Research*, 31:591–653, 2008.

[56] Andrea Rose Dubin. *A Magmatic Trigger for the Paleocene-Eocene Thermal Maximum?* . Doctoral thesis, MIT, Woods Hole Oceanographic Institution, 2015.

[57] Magnus Egerstedt and Xiaoming Hu. Formation constrained multi-agent control. *IEEE Transactions on Robotics and Automation*, 17(6):947–951, 2001. ISSN 1042296X.

[58] Esra Erdem, Doga G Kisa, Umut Oztok, and Peter Schueller. A general formal framework for pathfinding problems with multiple agents. In *AAAI Conference on Artificial Intelligence*, pages 290–296, 2013.

[59] Michael Erdmann. *On Motion Planning with Uncertainty.* PhD thesis, 1984.

[60] Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. *Algorithmica*, 2(1):477–521, 1987. ISSN 0178-4617.

[61] Anders Eriksson, Martin Nilsson Jacobi, Johan Nyström, and Kolbjørn Tunstrøm. Determining interaction rules in animal swarms. *Behavioral Ecology*, 21: 1106–1111, 2010.

[62] Ariel Felner, Meir Goldenberg, Guni Sharon, Roni Stern, Tal Beja, and Robert C Holte. Partial-Expansion A * with Selective Node Generation. In *AAAI Conf*, pages 471–477, 2012.

[63] Dave Ferguson, Nidhi Kalra, and Anthony Stentz. Replanning with rrts. In *IEEE International Conference on Robotics and Automation*, pages 1243–1248. IEEE, 2006.

[64] Cornelia Ferner, Glenn Wagner, and Howie Choset. ODrM* optimal multirobot path planning in low dimensional search spaces. In *2013 IEEE International Conference on Robotics and Automation*, pages 3854–3859. Ieee, may 2013. ISBN 978-1-4673-5643-5.

[65] Carlo Ferrari, Enrico Pagello, Jun Ota, and Tamio Arai. Multirobot motion coordination in space and time. *Robotics and Autonomous Systems*, 25(3-4): 219–229, nov 1998. ISSN 09218890.

[66] Jonathan Fink, Nathan Michael, Soonkyum Kim, and Vijay Kumar. Planning and control for cooperative manipulation and transportation with aerial robots. *The International Journal of Robotics Research*, 30(3):324–334, sep 2010. ISSN 0278-3649.

[67] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1): 23–33, 1997. ISSN 10709932.

[68] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, jun 2014. ISSN 00313203.

[69] John Gaschnig. A Problem Similarity Approach to Devising Heuristics: First Results. In *International Joint Conference on Artificial Intelligence*, pages 23–29, 1979.

[70] Robert W Ghrist and Daniel E Koditschek. Safe Cooperative Robot Dynamics on Graphs. *SIAM Journal on Control and Optimization*, 40(5), 2002.

[71] Meir Goldenberg, Ariel Felner, and Nathan R Sturtevant. Optimal-Generation Variants of EPEA*. In *Symposium on Combinatorial Search*, pages 89–97, Leavenworth, Washington, USA, 2013.

[72] Juan Pablo Gonzalez and Anthony Stentz. Planning with Uncertainty in Position: an Optimal Planner. In *IEEE International Conference on Intelligent*

Robots and Systems, 2005.

[73] Michael A Goodrich, Brian Pendleton, Sean Kerman, and PB Sujit. What Types of Interactions do Bio-Inspired Robot Swarms and Flocks Afford a Human? In *Robotics: Science and Systems*, 2012.

[74] R. L. Graham, E. L. Lawler, J. K. Lenstra, and a. H G Rinnooy Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. *Annals of Discrete Mathematics*, 5(C):287–326, 1979. ISSN 01675060.

[75] D.D. Grossman. Traffic control of multiple robot vehicles. *IEEE Journal on Robotics and Automation*, 4(5):491–497, 1988. ISSN 08824967.

[76] L. C G J M Habets and Jan H. Van Schuppen. A control problem for affine dynamical systems on a full-dimensional polytope. *Automatica*, 40:21–35, 2004. ISSN 00051098.

[77] P E Hart, N J Nilsson, and B Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), jul 1968.

[78] J.E. Hopcroft, J.T. Schwartz, and M. Sharir. On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE- Hardness of the "Warehouseman's Problem". *The International Journal of Robotics Research*, 3(4): 76–88, dec 1984. ISSN 0278-3649.

[79] David Hsu, T Jiang, J Reif, and Z Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 4420–4426. IEEE, 2003.

[80] T. C. Hu. Parallel Sequencing and Assembly Line Problems. *Operations Research*, 9(6):841–848, 1961. ISSN 0030-364X.

[81] Ruoyun Huang, Yixin Chen, and Weixiong Zhang. A Novel Transition Based Encoding Scheme for Planning as Satisfiability. In *AAAI Conference on Artificial Intelligence*, pages 89–94, 2010.

[82] MR Jansen and Nathan R Sturtevant. Direction maps for cooperative pathfinding. In *AAAI Conference on Artificial Intelligence and Interactive Digitial Entertainment poster*, pages 185–190, 2008.

[83] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998. ISSN 00043702.

[84] K Kant and S W Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research*, 5(3): 72, 1986. ISSN 0278-3649.

[85] Shin Kato, Sakae Nishiyama, and Jun'ichi Takeno. Coordinating Mobile Robots by Applying Traffic Rules. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1535–1541, 1992. ISBN 0780307372.

[86] Max Katsev, Jingjin Yu, and Steven M Lavalle. Efficient formation path plan-

ning on large graphs. In *IEEE International Conference on Robotics and Automation*, volume 0904501, pages 3606–3611, 2013. ISBN 9781467356435.

[87] Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *International Joint Conference on Artificial Intelligence*, 1999.

[88] Lydia E Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configurations spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, jun 1996.

[89] Mokhtar M Khorshid, Robert C Holte, and Nathan R Sturtevant. A Polynomial-Time Algorithm for Non-Optimal Multi-Agent Pathfinding. In *Proceedings of the Symposium on Combinatorial Search*, 2011.

[90] Stephen Kloder and Seth A Hutchinson. Path planning for permutation-invariant multirobot formations. *IEEE Transactions on Robotics*, 22(4):650–665, 2006.

[91] Ross A Knepper, Todd Layton, John Romanishin, and Daniela Rus. IkeaBot : An Autonomous Multi-Robot Coordinated Furniture Assembly System. In *IEEE International Conference on Robotics and Automation*, 2013. ISBN 9781467356428.

[92] Sven Koenig and Maxim Likhachev. D* Lite. In *AAAI Conference on Artificial Intelligence*, pages 476–483, 2002.

[93] Y. Koga and Jean-Claude Latombe. On multi-arm manipulation planning. In *IEEE International Conference on Robotics and Automation*, pages 945–952, San Diego, CA, 1994. IEEE Comput. Soc. Press. ISBN 0-8186-5330-2.

[94] Jelle R Kok and Nikos Vlassis. Using the Max-Plus Algorithm for Multiagent Decision Making in Coordination Graphs. In Yasutake Bredenfeld, Ansgar and Jacoff, Adam and Noda, Itsuki and Takahashi, editor, *RoboCub 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2006.

[95] Jelle R Kok, Pieter Jan 't Hoen, Bram Bakker, and Nikos Vlassis. Utile Coordination: Learning interdependencies among cooperative agents. In *IEEE Symposium on Computational Intelligence and Games*, 2005.

[96] RE Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, sep 1985. ISSN 00043702.

[97] Richard E Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1): 41–78, 1993. ISSN 0004-3702.

[98] DM Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Symposium on Foundations of Computer Science*, pages 241—-250. M. I. T., 1984.

[99] K Madhava Krishna, Henry Hexmoor, and Srinivas Chellappa. Reactive Navigation of Multiple Moving Agents by collaborative Resolution of Conflicts.

Journal of Robotic Systems, pages 249–269, 2005.

[100] Athanasios Krontiris, Ryan Luna, and Kostas E Bekris. From Feasibility Tests to Path Planners for Multi-Agent Pathfinding. In *Symposium on Combinatorial Search*, number Surynek 2009, pages 114–122, 2013.

[101] James J Kuffner and Steven M LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, number April, pages 995–1001, 2000. ISBN 0780358864.

[102] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP : Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Robotics: Science and Systems IV*, 2008.

[103] Jean-claude Latombe. On delaying collision checking in PRM planning: Application to multi-robot coordination. *The International Journal of Robotics Research*, 21(1):5–26, 2002.

[104] Jean-Claude Latombe, Anthony Lazanas, and Shashank Shekhar. Robot motion planning with uncertainty in control and sensing. *Artificial Intelligence*, 52(1): 1–47, 1991. ISSN 00043702.

[105] Steven M Lavalle and Seth A Hutchinson. Optimal Motion Planning for Multiple Robots having Independent Goals. *IEEE Transactions on Robotics and Automation*, 14(6):912–925, dec 1998.

[106] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. In *Proceedings IEE International Conference on Robotics and Automation*, 1999.

[107] Naomi Ehrich Leonard and Edward Fiorelli. Virtual Leaders , Artificial Potentials and Coordinated Control of Groups. In *IEEE Conference on Decision and Control*, number December, pages 2968–2973, Orlando, Florida USA, 2001. ISBN 0780370619.

[108] Stéphane Leroy, Jean-Paul Laumond, and Thierry Siméon. Multiple Path Coordination for Mobile Robots: A Geometric Algorithm. In *International Joint Conference on Artificial Intelligence*, pages 1118–1123, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-613-0.

[109] M Anthony Lewis and Kar-han Tan. High Precision Formation Control of Mobile Robots Using Virtual Structures. *Autonomous Robots*, 403:387–403, 1997.

[110] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime Dynamic A*: An Anytime, Replanning Algorithm. In *International Conference on Automated Planning and Scheduling*, 2005.

[111] Magnus Lindhé, Petter Ogren, and Karl Henrik Johansson. Flocking with obstacle avoidance: A new distributed coordination algorithm based on voronoi partitions. In *IEEE International Conference on Robotics and Automation*, number April, pages 1785–1790, 2005. ISBN 078038914X.

[112] Laura Lindzey, Ross A Knepper, Howie Choset, and Siddhartha S Srinivasa.

The Feasible Transition Graph: Encoding Topology and Manipulation Constraints for Multirobot Push-Planning. In *Workshop on the Algorithmic Foundations of Robotics*, page 16, 2014.

[113] S Lloyd and Y J Ng. Black hole computers. *Scientific American*, 291(5):52–61, 2004. ISSN 0036-8733.

[114] Tomas Lozano-Perez, M T Mason, and R H Taylor. Automatic Synthesis of Fine-Motion Strategies for Robots. *The International Journal of Robotics Research*, 3(1):3–24, 1984. ISSN 0278-3649.

[115] Ryan Lukeman, Yue-Xian Li, and Leah Edelstein-Keshet. Inferring individual rules from collective behavior. *Proceedings of the National Academy of Sciences of the United States of America*, 107:12576–12580, 2010.

[116] Ryan Luna and Kostas E Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *International Joint Conference on Artificial Intelligence*, 2011.

[117] B Maren, B Wolfram, and T Sebastian. Constraint-based Optimization of Priority Schemes for Decoupled Path Planning Techniques. *KI 2001: Advances in Artificial Intelligence*, pages 78—-93, 2001.

[118] Nik A Melchior and Reid Simmons. Particle RRT for path planning with uncertainty. In *IEEE Conference on Robotics and Automation*, number April, pages 10–14, Roma, Italy, 2007. ISBN 1424406021.

[119] Francisco S Melo and Manuela Veloso. Learning of Coordination: Exploiting Sparse Interactions in Multiagent Systems. In *International Conference on Autonomous Agents and Multiagent Systems*, may 2009.

[120] Francisco S Melo and Manuela Veloso. Decentralized MDPs with sparse interactions. *Artificial Intelligence*, 175(11):1757–1789, jul 2011. ISSN 00043702.

[121] Nathan Michael, Calin Belta, and Vijay Kumar. Controlling three dimensional swarms of robots. In *IEEE International Conference on Robotics and Automation*, number May, pages 964–969. Ieee, 2006. ISBN 0-7803-9505-0.

[122] Nathan Michael, Michael M Zavlanos, Vijay Kumar, and George J Pappas. Distributed multi-robot task assignment and formation control. In *IEEE International Conference on Robotics and Automation*, pages 128–133. IEEE, 2008.

[123] Scott A Miller, Zachary A Harris, and Edwin K P Chong. Coordinated guidance of autonomous UAVs via nominal belief-state optimization. In *Proceedings of the American Control Conference*, pages 2811–2818, 2009. ISBN 9781424445240.

[124] G. Ayorkor Mills-Tettey, Anthony Stentz, and M. Bernardine Dias. DD* Lite: Efficient Incremental Search with State Dominance. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1032–1038, 2006. ISBN 1577352815.

[125] Joseph Moore and Russ Tedrake. Control synthesis and verification for a perching UAV using LQR-Trees. In *IEEE Conference on Decision and Control*, pages

3707–3714. Ieee, dec 2012. ISBN 978-1-4673-2066-5.

[126] Umashankar Nagarajan, George Kantor, and Ralph Hollis. Integrated motion planning and control for graceful balancing mobile robots. *The International Journal of Robotics Research*, 32(9-10):1005–1029, jul 2013. ISSN 0278-3649.

[127] Reza Olfati-Saber, W B Dunbar, and Richard M Murray. Cooperative control of multi-vehicle systems using cost graphs and optimization. In *Proceedings of the American Control Conference*, pages 2–7, 2003. ISBN 0780378962.

[128] C Spence Oliver, Mahesh Saptharishi, John M Dolan, Ashitey Trebi-Ollennu, and Pradeep Khosla. Multi-robot path planning by predicting structure in a dynamic environment. In *IFAC special session on Collaboration and Data Fusion in Distributed Mobile Robotic Systems*, 2000.

[129] Christos H Papadimitriou and John N Tsitsiklis. The Complexity of Markov Decision Processes. *Mathematics of Operations Research*, 12(3):441–, 1987.

[130] Sachin Patil, Jur Van Den Berg, Sean Curtis, Ming Lin, and Dinesh Manocha. Directing crowd simulations using navigation fields. *IEEE transactions on visualization and computer graphics*, 17(2):244–54, feb 2011. ISSN 1941-0506.

[131] Sachin Patil, Jur Van Den Berg, and Ron Alterovitz. Estimating probability of collision for safe motion planning under Gaussian motion and sensing uncertainty. In *IEEE International Conference on Robotics and Automation*, pages 3238–3244, 2012. ISBN 9781467314039.

[132] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Welsley, 1984.

[133] Mike Peasgood, John McPhee, and Christopher M Clark. Complete and scalable multi-robot planning in tunnel environments. *Computer Science and Software Engineering*, 1, 2006.

[134] Jufeng Peng and Srinivas Akella. Coordinating Multiple Robots with Kinodynamic Constraints Along Specified Paths. *International Journal of Robotics Research*, 24(4):295–310, 2005. ISSN 0278-3649.

[135] Andrés Santiago Pérez-Bergquist and Anthony Stentz. K2: An efficient approximation algorithm for globally and locally multiply-constrained planning problems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1385–1391, 2005. ISBN 0780389123.

[136] Mike Phillips, Andrew Dornbush, Sachin Chitta, and Maxim Likhachev. Anytime incremental planning with E-Graphs. In *IEEE International Conference on Robotics and Automation*, pages 2444–2451. Ieee, may 2013. ISBN 978-1-4673-5643-5.

[137] Jim Pitman. *Probibility.* Springer-Verlag New York, 1 edition, 1993.

[138] Robert Platt, Leslie Kaelbling, Tomas Lozano-Perez, and Russ Tedrake. Non-Gaussian belief space planning: Correctness and complexity. In *IEEE International Conference on Robotics and Automation*, pages 4711–4717, 2012. ISBN

9781467314039.

[139] Ira Pohl. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *International Joint Conference on Artificial intelligence*, pages 12–17, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

[140] Josep M Porta, Nikos Vlassis, Matthijs T J Spaan, and Pascal Poupart. Point-Based Value Iteration for Continuous POMDPs. *Journal of Machine Learning Research*, 7:2329–2367, 2006. ISSN 10450823.

[141] S Prentice and Nicholas Roy. The Belief Roadmap: Efficient Planning in Belief Space by Factoring the Covariance. *The International Journal of Robotics Research*, 28(11-12):1448–1465, 2009. ISSN 0278-3649.

[142] Oliver Purwin, Raffaello DAndrea, and Jin-Woo Lee. Theory and implementation of path planning by negotiation for decentralized agents. *Robotics and Autonomous Systems*, 56(5):422–436, may 2008. ISSN 09218890.

[143] Daniel Ratner and Manfre Warmuth. Finding a Shortest Solution for the NxN Extension of the 15-PUZZLE is Intractable. In *AAAI Conference on Artificial Intelligence*, pages 168–172, Philadelphia, PA, USA, 1986.

[144] Ralf Regele and Paul Levi. Cooperative multi-robot path planning by heuristic priority adjustment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5954–5959. Ieee, oct 2006. ISBN 1-4244-0258-1.

[145] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34, aug 1987. ISBN 0897912276.

[146] Daniel Walter Rowlands. *Xenon Difluoride Etching and Molecular Oxygen Oxidation of Silicon by Reactive Scattering*. Masters thesis, MIT, 2015.

[147] Daniela Rus, Bruce Donald, and Jim Jennings. Moving Furniture with Teams of Autonomous Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 235–242, Pittsburgh, PA, 1995. ISBN 0818671084.

[148] Malcolm Ryan. Multi-robot path planning with sub-graphs. In *19th Australasian Conference on Robotics and Automation*, 2006.

[149] Malcolm Ryan. Constraint-based multi-robot path planning. In *IEEE International Conference on Robotics and Automation*, pages 922–928. Ieee, may 2010. ISBN 978-1-4244-5038-1.

[150] Mitul Saha and P Isto. Multi-Robot Motion Planning by Incremental Coordination. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5960–5963, 2006.

[151] Qandeel Sajid, Ryan Luna, and Kostas E Bekris. Multi-Agent Pathfinding with Simultaneous Execution of Single-Agent Primitives. In *Symposium on Combinatorial Search*, pages 88–96, 2012.

[152] G Sánchez and Jean-Claude Latombe. On delaying collision checking in PRM

planning: Application to multi-robot coordination. *The International Journal of Robotics Research*, 21(1):5, 2002.

[153] Gildardo Sanchez and Jean-claude Latombe. Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. *IEEE International Conference on Robotics and Automation*, 2(May):2112–2119, 2002.

[154] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. Pruning Techniques for the Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. In *Symposium on Combinatorial Search*, pages 150–157, 2011.

[155] Guni Sharon, Roni Stern, Meir Goldenberg, Ariel Felner, and I Beer-Sheva. The increasing cost tree search for optimal multi-agent pathfinding. *International Joint Conference on Artificial Intelligence*, 2(i):662–667, 2011.

[156] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-Based Search for Optimal Multi-Agent Path Finding. In *AAAI Conference on Artificial Intelligence*, 2012.

[157] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Meta-Agent Conflict-Based Search For Optimal Multi-Agent Path Finding. *Symposium on Combinatorial Search*, pages 97–104, 2012.

[158] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195: 470–495, feb 2013. ISSN 00043702.

[159] David Silver. Cooperative pathfinding. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 23–28, 2005.

[160] Thierry Siméon, Stéphane Leroy, and Jean-paul Laumond. Path Coordination for Multiple Mobile Robots : A Resolution-Complete Algorithm. *IEEE Transactions on Robotics and Automation*, 18(1):42–49, 2002.

[161] Thierry Siméon, Juan Cortés, Anis Sahbani, and Jean-Paul Laumond. A general manipulation task planner. In Jean-Daniel Boissonnat, Joel Burdick, Ken Goldberg, and Seth A Hutchinson, editors, *Workshop on the Algorithmic Foundations of Robotics*, volume 7 of *Springer Tracts in Advanced Robotics*, pages 311–327. Springer Berlin Heidelberg, 2004. ISBN 3540404767. doi: 10.1007/978-3-540-45058-0{_}19.

[162] Reid Simmons. The curvature-velocity method for local obstacle avoidance. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3375 –3382 vol.4, 1996. ISBN 0-7803-2988-0.

[163] Jamie Snape, Jur van den Berg, Stephen J Guy, and Dinesh Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011.

[164] Kiril Solovey and Dan Halperin. k-Color Multi-Robot Motion Planning. In *Workshop on the Algorithmic Foundations of Robotics*, volume 255827, pages 1–23, 2012.

[165] Matthijs T j Spaan and Francisco S Melo. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *International Conference on Autonomous Agents and Multiagent Systems*, number Aamas, pages 525–532, 2008.

[166] WM Spears and DF Gordon. Using artificial physics to control agents. In *International Conference on Information Intelligence and Systems*, pages 281–288, Bethesda, MD, 1999.

[167] Trevor Standley. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *AAAI Conference on Artificial Intelligence*, 2010.

[168] Trevor Standley and R Korf. Complete algorithms for cooperative pathfinding problems. In *International Joint Conference on Artificial Intelligence*. IJCAI, 2011.

[169] Anthony Stentz. Optimal and Efficient Path Planning for Unknown and Dynamic Environments. *International Journal of Robotics and Automation*, 10: 89–100, 1993.

[170] Anthony Stentz. CD*: A Real-Time Resolution Optimal Re-Planner for Globally Constrained Problems. In *AAAI Conference on Artificial Intelligence*, pages 605–612, 2002.

[171] I A Sucan and Lydia E Kavraki. On the advantages of task motion multigraphs for efficient mobile manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4621–4626. IEEE, 2011.

[172] I A Sucan and Lydia E Kavraki. Mobile manipulation: Encoding motion planning options using task motion multigraphs. In *IEEE International Conference on Robotics and Automation*, pages 5492–5498. IEEE, 2011.

[173] Wen Sun and Lg Torres. Safe Motion Planning for Imprecise Robotic Manipulators by Minimizing Probability of Collision. In *International Symposium on Robotics Research*, pages 1–16, 2013.

[174] Pavel Surynek. An application of pebble motion on graphs to abstract multi-robot path planning. In *IEEE International Conference on Tools with Artificial Intelligence*, number 201, pages 151–158. Ieee, nov 2009. ISBN 978-1-4244-5619-2.

[175] Pavel Surynek. Solving Abstract Cooperative Path-Finding in Densely Populated Environments. *Computational Intelligence*, 30(2), 2012.

[176] Pavel Surynek. An SAT-Based Approach to Cooperative Path-Finding Using All-Different Constraints. In *Symposium on Combinatorial Search*, pages 191–192, 2012.

[177] Pavel Surynek. Optimal Cooperative Path-Finding with Generalized Goals in Difficult Cases. In *Tenth Symposium of Abstraction, Reformulation, and Approximation*, pages 119–122, 2013.

[178] Petr Švestka and Mark H Overmars. Coordinated path planning for multiple

robots. *Robotics and Autonomous Systems*, 23(3):125–152, 1998.

[179] Russ Tedrake, Ian R. Manchester, Mark Tobenkin, and John W Roberts. LQR-trees: Feedback Motion Planning via Sums-of-Squares Verification. *The International Journal of Robotics Research*, 29(8):1038–1052, apr 2010. ISSN 0278-3649.

[180] Paolo Toth and Daniele Vigo, editors. *The Vehcile Routing Problem*. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. ISBN 0898715792.

[181] Peter Trautman and Andreas Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 797–803. Ieee, oct 2010. ISBN 978-1-4244-6674-0.

[182] Matthew Turpin, Nathan Michael, and Vijay Kumar. Trajectory Planning and Assignment in Multirobot Systems. In *Workshop on Algorithmic Foundations of Robotics*, 2012.

[183] Matthew Turpin, Nathan Michael, and Vijay Kumar. Concurrent Assignment and Planning of Trajectories for Large Teams of Interchangeable Robots. In *IEEE International Conference on Robotics and Automation*, pages 834–840, Karlsruhe, 2013. ISBN 9781467356428.

[184] Alphan Ulusoy, Stephen L Smith, Xu Chu Ding, and Calin Belta. Robust multi-robot optimal path planning with temporal logic constraints. In *IEEE International Conference on Robotics and Automation*, pages 4693–4698, Saint Paul, Minnesota, USA, may 2012. Ieee. ISBN 978-1-4673-1405-3.

[185] Alphan Ulusoy, Stephen L Smith, and Calin Belta. Optimal Multi-Robot Path Planning with LTL Constraints : Guaranteeing Correctness Through Synchronization. In M. Ani Hsieh and G Chirikjian, editors, *Distributed Autonomous Robotic Systems*, volume 104 of *Spring Tracts in Advanced Robotics*. 2014. ISBN 9783642551468.

[186] Jur van den Berg and Mark H Overmars. Prioritized Motion Planning for Multiple Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2217–2222, 2005.

[187] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal Velocity Obstacles for real-time multi-agent navigation. In *IEEE International Conference on Robotics and Automation*, pages 1928–1935. Ieee, may 2008. ISBN 978-1-4244-1646-2.

[188] Jur van den Berg, Jack Snoeyink, Ming Lin, and Dinesh Manocha. Centralized Path Planning for Multiple Robots: Optimal Decoupling into Sequential Plans. In *Robotics: Science and Systems*, 2009.

[189] Jur van den Berg, Pieter Abbeel, and Ken Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913, 2011. ISSN

0278-3649.

[190] Jur van den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-Body Collision Avoidance. In *Robotics Research*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 3–19. 2011.

[191] Pradeep Varakantham, Jun-young Kwak, Matthew Taylor, Janusz Marecki, Paul Scerri, and Milind Tambe. Exploiting Coordination Locales in Distributed POMDPs via Social Model Shaping. In *International Conference on Automated Planning and Scheduling*, pages 313–320, 2009.

[192] Prasanna Velagapudi, Katia Sycara, and Paul Scerri. Decentralized prioritized planning in large multirobot teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4603–4609, 2010. ISBN 9781424466764.

[193] Tamás Vicsek, Andras Czirók, Eshel Ben-Jacob, Inon Coehn, Ofer Shochet, A Czirók, Eshel Ben-Jacob, I Cohen, and Ofer Shochet. Novel Type of Phase Transition in a System of Self-Driven Particles. *Physical Review Letters*, 75(6): 4–7, 1995.

[194] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219(0):1–24, 2015. ISSN 0004-3702.

[195] Glenn Wagner, Misu Kang, and Howie Choset. Probabilistic Path Planning for Multiple Robots with Subdimensional Expansion. In *IEEE/RSJ International Conference on Robotics and Automation*, may 2012.

[196] Jing Wang and Suparerk Premvuti. Distributed traffic regulation and control for multiple autonomous mobile robots operating in discrete space. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1619–1624. Ieee, 1995. ISBN 0-7803-1965-6.

[197] Ko-Hsin Cindy Wang, Adi Botea, and Philip Kilby. On Improving the Quality of Solutions in Large-Scale Cooperative Multi-Agent Pathfinding. In *Symposium on Combinatorial Search*, pages 209–210, 2011.

[198] Steven A Wilmarth, Nancy M Amato, and Peter F Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *IEEE International Conference on Robotics and Automation*, number May, pages 1024–1031, 1999.

[199] Richard M Wilson. Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory, Series B*, pages 86–96, 1974.

[200] Peter R Wurman, Raffaello D Andrea, and Mick Mountz. Coordinating Hundreds of Cooperative , Autonomous Vehicles in Warehouses. *AI Magazine*, 29 (1):9–20, 2008.

[201] Chairit Wuthishuwong, Ansgar Traechtler, and Torsten Bruns. Safe trajectory planning for autonomous intersection management by using vehicle to infrastructure communication. *EURASIP Journal on Wireless Communications and Networking*, 2015(33), 2015. ISSN 1687-1499.

[202] Atsushi Yamashita, Tamio Arai, Jun Ota, and Hajime Asama. Motion planning of multiple mobile robots for cooperative manipulation and transportation. *IEEE Transactions on Robotics and Automation*, 19(2):223–237, apr 2003. ISSN 1042-296X.

[203] Peng Yang, Randy A. Freeman, and Kevin M. Lynch. Multi-agent coordination by decentralized estimation and control. *IEEE Transactions on Automatic Control*, 53(11):2480–2496, 2008.

[204] Jingjin Yu and Steven M Lavalle. Multi-agent Path Planning and Network Flow. In Emilio Frazzoli, Tomas Lozano-Perez, Nicholas Roy, and Daniela Rus, editors, *Workshop on the Algorithmic Foundations of Robotics*, volume 86 of *Springer Tracts in Advanced Robotics*, pages 157–173, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-36278-1.

[205] Jingjin Yu and Steven M LaValle. Structure and intractability of optimal multi-robot path planning on graphs. *Proceedings AAAI National Conference on Artificial Intelligence*, 2013.