Artificial Intelligence 219 (2015) 1–24

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Subdimensional expansion for multirobot path planning ☆

Glenn Wagner *, Howie Choset

Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, United States

A R T I C L E   I N F O

Article history:
Received 4 March 2013
Received in revised form 28 October 2014
Accepted 2 November 2014
Available online 7 November 2014

Keywords:
Motion and path planning
Multiagent system

A B S T R A C T

Planning optimal paths for large numbers of robots is computationally expensive. In this paper, we introduce 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. We present the M* algorithm, an implementation of subdimensional expansion that adapts the A* planner to perform efficient multirobot planning. M* is proven to be complete and to find minimal cost paths. Simulation results are presented that show that M* outperforms existing optimal multirobot path planning algorithms.

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.

In this paper, we focus on issues surrounding planning paths for large numbers of robots. Such issues center on the fundamental trade-off between path quality and computational cost. The trade-off is illustrated by the differences between *coupled* and *decoupled* approaches to multirobot path planning. Coupled approaches use the high-dimensional joint configuration space of a multirobot system[1] as the search space, and can find paths that are collision-free and minimal cost, *i.e.* optimal, but at high computational cost [1,2]. For example, finding optimal paths for multirobot systems on graphs is known to be NP-hard [3]. Decoupled algorithms explore a low dimensional search space to coordinate the motion of robots on paths computed for each robot individually; they can rapidly find paths for large numbers of robots, but are not guaranteed to find a path for all solvable problems [4–8].

☆ This work is supported by the ONR MURI N00014-09-1-1031 entitled “ANTIDOTE: Adaptive Network for Threat Intrusion Detection or Termination.”

* Corresponding author. Tel.: +1 585 802 4541.
E-mail addresses: gswagner@cmu.edu (G. Wagner), choset@cs.cmu.edu (H. Choset).

[1] In a slight abuse of terminology, we define the configuration space of a multirobot system as the Cartesian product of the free configuration spaces of the individual robots. The free configuration space of a robot is the space of all positions that a robot can occupy without colliding with itself or an obstacle. We use the term “joint” configuration space to emphasize that we are referring to the configuration space of the entire system, not that of a single robot.

http://dx.doi.org/10.1016/j.artint.2014.11.001
0004-3702/© 2014 Elsevier B.V. All rights reserved.

In this paper, we introduce a new approach that combines the advantages of coupled and decoupled algorithms 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 multirobot path planning problems. Subdimensional expansion starts by finding a path for each robot in its individual configuration space. 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 colliding robots to be found with coupled planning. Although the search space may grow to cover the entire joint configuration space in the worst case, for many problems subdimensional expansion can construct a low dimensional search space that allows for efficient computation of a high quality path.

This paper combines and extends results previously published in conference papers [9,10]. We describe an implementation of subdimensional expansion for planning on arbitrary graphs called M*.[2] M* uses A* [1] as the underlying planner that both computes paths for individual robots to guide the construction of the multirobot search space, and explores the multirobot search space to find a path for the entire system. M* is guaranteed to find the optimal path. Several improved versions of M* are described which further reduce the dimensionality of the search space and reduce the time required to explore the search space. Simulation results are presented that show that M* has better performance than existing optimal multirobot path planning algorithms.

2. Prior work

We place multirobot path planning algorithms on a continuum between coupled and decoupled approaches. Coupled planning algorithms search the joint configuration space of the multirobot system, guaranteeing that the optimal path will be found. Decoupled approaches, on the other hand, compute paths separately for individual robots, then adjust the paths to avoid collisions. Decoupled approaches generally can not guarantee that a path will be found, much less the optimal path.

2.1. Coupled multirobot path planning

A straightforward multirobot path planner could use standard A* [1] to find a path for a multirobot system, resulting in a simple, coupled planner. However, the exponential growth in the number of possible joint actions would render such an approach computationally infeasible as the number of robots increases. Operator Decomposition (OD) [2] and Enhanced Partial Expansion A* (EPEA*) [12,13] are lazy variants of A* designed for multirobot path planning which delay enumerating paths that are expensive based on a heuristic, thereby dramatically reducing the effective branching factor of multirobot systems. Iterative Deepening A* (IDA*) is a general purpose heuristic depth-first search algorithm, which can be applied to the problem of finding optimal paths for systems of multiple robots. However, it is effective only when robots are packed densely enough that only a few robots can move at any time [14].

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 directly to explore the joint configuration space of multirobot systems [15–18]. The shear size of the joint configuration space of multirobot systems limits such approaches to relatively small numbers of robots. MA-RRT* [19], dRRT [20], and sPRM [21] are probabilistic planning algorithms customized for multirobot path planning that decouple planning to avoid robot–obstacle and robot–robot collisions, resulting in substantial reductions in the computational cost of finding paths.

An alternate approach to coupled planning is to recast the multirobot path planning problem as a Boolean Satisfiability (SAT) problem, for which there are very efficient general-purpose solvers. The multirobot path planning 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, defining a logical formula for which the solver tries to find a valid assignment of truth values [22–26]. SAT planners have also been used to find shortcuts to reduce the cost of non-optimal paths computed by rule-based planners [27,28].

2.2. Decoupled multirobot path planning

Decoupled approaches compute paths separately for individual robots, then adjust the paths to avoid collisions. Because the search for both the individual robot paths and the necessary adjustments to avoid collisions are performed in low-dimensional search spaces, decoupled approaches can rapidly find paths for systems containing many robots [6,7]. Velocity planners fix the paths that will be followed by each robot, then find a velocity schedule along those paths that avoids collisions [5,29–34]. Priority planners assign a priority to each robot, then plan for individual robots in decreasing order of priority, treating higher priority robots as moving obstacles [4,35–39]. The choice of priority ordering is critical, leading

[2] M* has previously been used as the name of an unrelated opponent modeling search algorithm [11].

to a number of heuristics for choosing priority orders that are likely to lead to a solution [38,40,41]. Turpin et al. [42,43] showed that for permutation invariant multirobot path planning, where any robot can move to any goal location, there is a polynomial time algorithm for choosing an assignment of robots to goals and a priority order that is guaranteed to produce a solution. The drawback of decoupled algorithms is that the search spaces employed 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 [18], excepting permutation invariant multirobot path planning.

2.3. *Intermediate multirobot path planning*

Several approaches have been developed that lie between coupled and decoupled approaches: they allow for more rich robot–robot behaviors than can be achieved with decoupled planners, while avoiding planning in the joint configuration space.

Rule based approaches use a set of stereotyped behaviors to govern robot–robot interactions, and can find paths in polynomial time. The existence of polynomial time algorithms for non-optimal multirobot path planning was first proved by Wilson [44] and Kornhauser et al. [45]. Gabriele and Helmert [46] brought this early work back to the attention of the planning community, and worked to adapt the proofs of Kornhauser et al. [45] into a practical planning algorithm. Several approaches to rule based path planning have been developed. Warehousing approaches shift robots into configurations which will not interfere with the motion of other robots [47,48]. Push and Swap [49,50], Push and Rotate [51], and the Tree-Based Agent Swapping Strategy algorithm [52] utilize behaviors that exchange the positions of two robots without disturbing other robots. While these algorithms are guaranteed to find paths in polynomial time, the stereotyped behaviors induced by the rules can lead to low quality paths. In particular, only a few robots are typically allowed to move at any given time. Parallel Push and Swap (PPAS) [53] is a variant of Push and Swap which allows all robots to move simultaneously, significantly reducing the typical makespan of plans, *i.e.* the time required for the last robot to reach its goal.

An alternative approach followed by *dynamically coupled* algorithms is to grow the search space during path planning, so that the search space can initially be very small, then grow only where necessary. Al-Wahedi [54] presented an approach in which paths are found separately for each robot, followed by coupled planning in a window around conflicts, but does not return optimal paths. The work of van den Berg et al. [55] shows how to identify the minimal sets of robots which must execute a cooperative path instead of sequentially executing single robot paths. The Increasing Cost Tree Search (ICTS) [56] 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. Conflict-Based Search (CBS) [57] constructs a set of constraints when planning for individual robots to find optimal solutions without exploring higher-dimensional spaces. Enhanced CBS (ECBS) [58] is a variant of CBS that can rapidly find near optimal solutions to problems involving large numbers of robots. Independence Detection (ID) [2] and Meta-Agent Conflict-Based Search (MA-CBS) [59] 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 paths. 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.

3. Subdimensional expansion

In multirobot path planning 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 multirobot path planning problem naturally decomposes into small subproblems, which permits optimal paths[3] to be found at low computational cost. Specifically, if the interactions between robots are sparse, the multirobot path planning problem can be split into two parts: planning paths for individual robots and optimally resolving conflicts between robots.

Subdimensional expansion is a framework for multirobot path planning 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 present in the joint policy path.

Subdimensional expansion then uses the individual policies to guide the construction of a search space of variable dimensionality, 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 the only free parameter is for how long to follow the 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

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

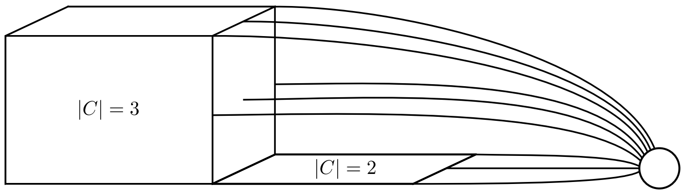


Fig. 1. Geometric visualization of the search space as embedded in the joint configuration space. The circle represents the goal configuration. The cube represents a region of the search space in which the collision set contains three robots, while the square denotes a region where the collision set 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.

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 to ensure that a path can be found the search space must be expanded along all paths explored by the underlying planner that lead to the collision. The backpropagation set is used to propagate information about a collision back along all explored paths leading to the collision. 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. For instance, if the underlying planner is A*, 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 some point on a path passing through q 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 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 (Fig. 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 multirobot path planning 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 4, 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.

Consider a team of three robots, r^1, r^2, r^3, which move on a graph representing a 4 connected grid. The X coordinates of the graph 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 $v_f^2 = B2$ and $v_f^3 = C3$ respectively (Fig. 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.

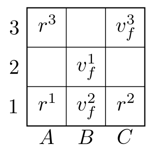
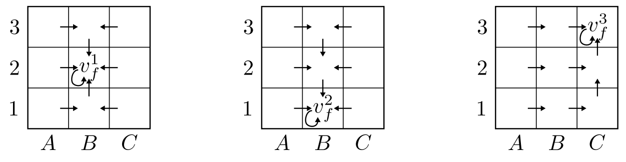
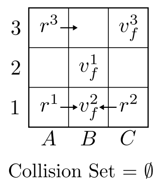


Fig. 2. Example of the working of subdimensional expansion. Robots r^1, r^2, r^3 start at $A1$, $C1$, and $A3$ respectively, with goal configurations $B2$, $B1$ and $C3$.



(a) Policy for r^1 (b) Policy for r^2 (c) Policy for r^3

Fig. 3. Subdimensional expansion starts by computing an 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.

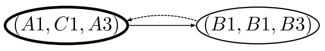


Neighbors of Expanded State
$(B1, B1, B3)$

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.

Fig. 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.

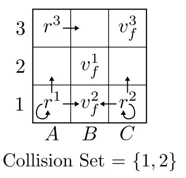
Subdimensional expansion begins by computing an individual policy for each robot (Fig. 3). The choice of policies is not unique. For instance, an alternate policy for r^1 would be to move up from $A1$ rather than right. Choice of individual policies is discussed in Section 5.4.

Once the individual policies are computed, search for the multirobot system can commence. M* maintains an open list of candidate nodes which are explored in order of f-value, the sum the cost to reach a node and a heuristic cost-to-go. When search begins the open list only contains the initial configuration, with an empty collision set (Fig. 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, $(B1, B1, B3)$ (Fig. 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 $(B1, B1, B3)$ (Fig. 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 4.2).

In the second iteration of M*, the initial configuration is once more taken from the open list, and expanded (Fig. 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 (Fig. 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 (Fig. 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 $(A2, B1, B3)$ (Fig. 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 $(A2, B1, B3)$

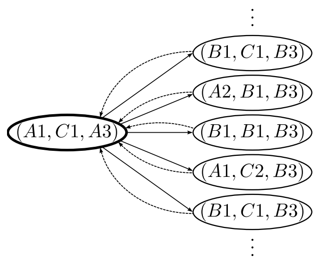
Neighbors of Expanded State
$(B1, B1, B3)$,$(A2, B1, B3)$,$(A1, B1, B3)$
$(B1, C2, B3)$,$(A2, C2, B3)$,$(A1, C2, B3)$
$(B1, C1, B3)$,$(A2, C1, B3)$,$(A1, C1, B3)$



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	$\emptyset$

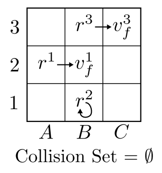
(a) Configuration at Step 2



(b) Search tree after expansion.

Fig. 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.

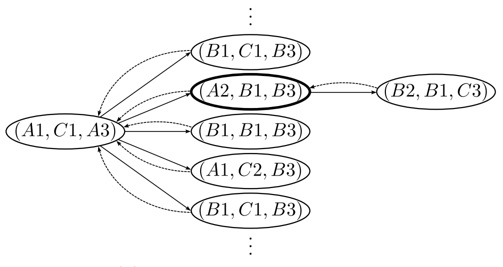
Neighbors of Expanded State
$(A2, B1, C3)$



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	$\emptyset$

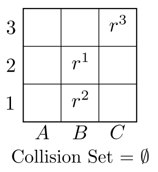
(a) Configuration at Step 3



(b) Search tree after expansion.

Fig. 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.

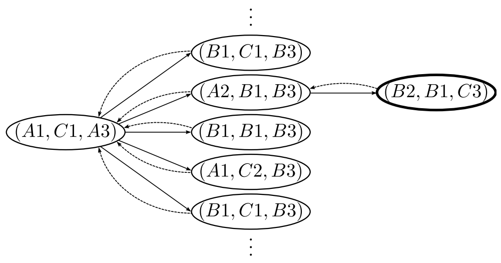
Neighbors of Expanded State



Post Expansion Open List

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

(a) Configuration at Step 4



(b) Search tree after expansion.

Fig. 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. M* terminates before generating any neighbors because the expanded state, $(B2, A2, C3)$, is the goal state, which indicates that a valid path has been found. (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.

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 (Fig. 7), which indicates that the optimal path has been found.

4. M*

In this section the M* algorithm will be described in detail. M* is an implementation of subdimensional expansion for multirobot path planning when the configuration space of each robot can be described by a graph. M* uses A* [1] as the underlying planner, because 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. M* will be shown to have the same optimality and completeness properties as A*. The section will end with a discussion of the performance of M*.

4.1. Problem definition

Consider a system of n robots r^i indexed by the set $I = \{1, \dots, n\}$. Let the free configuration space of r^i be represented by the directed 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 non-negative cost. 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 tensor product of the individual robot graphs $G = G^1 \times \cdots \times G^n$, with vertex set V and edge set E. Recall that the tensor 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_{kl} connecting v^i_k to v^i_l is present in E^i and the edge e^j_{kl} 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 $\pi^i(v_k, v_l)$ denote a path in G^i from v_k to v_l. The cost of a single robot path $g^i : \Pi^i \mapsto \mathbb{R}^+$ is the sum of the costs of the edges traversed in the path. The cost $g : \Pi \mapsto \mathbb{R}^+$ of a path $\pi(v_k, v_l)$ in G is the sum of the costs of the corresponding single robot paths $\pi^i(v^i_k, v^i_l)$, where v^i_k is the position of r^i at the joint configuration v_k,

$$g\big(\pi(v_k, v_l)\big) = \sum_{i \in I} g^i\big(\pi^i\big(v^i_k, v^i_l\big)\big). \tag{1}$$

Table 1
Symbol definitions for multirobot path planning on graphs.

Symbol	Meaning
r^i	ith robot
G^i	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	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
$\pi(v_k, v_l)$	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

The task of M* is to find an optimal, collision-free path from the joint initial configuration $v_s = v^1_s \times \cdots \times v^n_s$ to the joint goal configuration $v_f = v^1_f \times \cdots \times v^n_f$, 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 instance 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 graph.

The notation in this paper 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^y_z 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^i_k refers to a vertex k describing the configuration of robot r^i. The symbols x^i_k and x^j_k 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 paper in different contexts. The definitions of symbols used in the problem definition are summarized in Table 1.

4.2. Algorithmic description

M* is broadly similar to A* [1] 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 (Figs. 4–7). 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* [8]. 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 [14,2,12]. The SIC heuristic evaluated at v_k is the sum of the costs of the individually optimal paths of all robots

$$h(v_k) = g\big(\pi_\phi(v_k, v_f)\big) \leq g\big(\pi_*(v_k, v_f)\big). \tag{2}$$

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

Algorithm 1 Pseudocode for collision set backpropagation.

```
Require: $v_k$, $C_l$, open
  {$v_k$- vertex in the backpropagation set of $v_l$}
  {$C_l$ – the collision set of $v_l$}
  {open – the open list for M*}
  if $C_l \nsubseteq C_k$ then
    $C_k \leftarrow C_k \cup C_l$
    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)
```

Table 2
Symbol definitions for M*.

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
$\pi_*(v_k, v_l)$	Minimal cost, collision-free path connecting v_k to v_l
C_k	Collision set at v_k
V^{nbh}_k	Limited neighbors of v_k
$h(v_k)$	Heuristic cost-to-go from v_k to v_f

any neighboring state in the robots individual 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 at v_k then $V^{\text{nbh}}_k = \emptyset$ to prevent paths from passing through collisions.

$$V^{\text{nbh}}_k = \left\{ v_l \;\middle|\; \begin{Bmatrix} e^i_{kl} \in E^i, & i \in C_k \\ v^i_l = \phi^i(v^i_k), & i \notin C_k \end{Bmatrix} \right\} \tag{3}$$

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^{nbh}_l. 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^{nbh}_l 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). The meanings of symbols used to describe M* are summarized in Table 2. Pseudocode for M* is provided in Algorithm 2.

4.3. Completeness and cost optimality

In this section, M* will be shown to be both complete and optimal. The description of M* given in Section 4.2 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 time and that the search graph will eventually contain the optimal path, if extant, is sufficient to prove that M* is complete and optimal.

4.3.1. 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 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). 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}.

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 $v_l \in V_k^{\text{nbh}}$ do
        $v_l$.back_set.append($v_k$) {Add $v_k$ to the back propagation list}
        $C_l \leftarrow C_l \cup \Psi(v_l)$
        {Update collision sets, and add vertices whose collision set changed back to open}
        backprop($v_k$, $C_l$,open)
        if $\Psi(v_l) = \emptyset$ and $v_k$.cost $+ f(e_{kl}) <$ $v_l$.cost then
            {$v_k$ is the cheapest route to $v_l$}
            $v_l$.cost ← $v_k$.cost $+ f(e_{kl})$
            $v_l$.back_ptr ← $v_k$ {Track the best path to $v_l$}
            open.insert($v_l$)
return No path exists
```

Table 3
Search graph symbols.

Symbol	Name	Meaning	A* equivalent
G	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 $G^{\text{nbh}} \setminus G^{\text{exp}}$	

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.

$$C_k = \begin{cases} \Psi(v_k) \bigcup_{v_l \in V_k} \Psi(v_l) & v_k \in G^{\text{exp}} \\ \emptyset & v_k \notin G^{\text{exp}} \end{cases} \tag{4}$$

where $V_k = \{v_l \mid \exists \pi(v_k, v_l) \subseteq G^{\text{exp}}\}$ 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 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 $v_k \in G^{\text{nbh}} \setminus G^{\text{exp}}$ will proceed along $\pi_\phi(v_k, v_f)$ until either v_f or a vertex in G^{exp}[4] is reached. The resulting path segment is denoted $\pi_\phi(v_k)$, and is represented as a subgraph G_k^{ϕ}, 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 $G^{\phi} = \bigcup_{v_k \in G^{\text{nbh}} \setminus G^{\text{exp}}} G_k^{\phi}$.

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 $G^{\text{nbh}} \setminus G^{\text{exp}}$ to v_f. By the definitions of G^{exp}, G^{nbh} and

[4] If $\pi_\phi(v_k, v_f)$ encounters a vertex in the explored graph G^{exp}, then there may be some $v_l \in \pi_\phi(v_k, v_f)$ such that $\Psi(v_l) \neq \emptyset$, with $v_l \in G^{\text{exp}}$. 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.

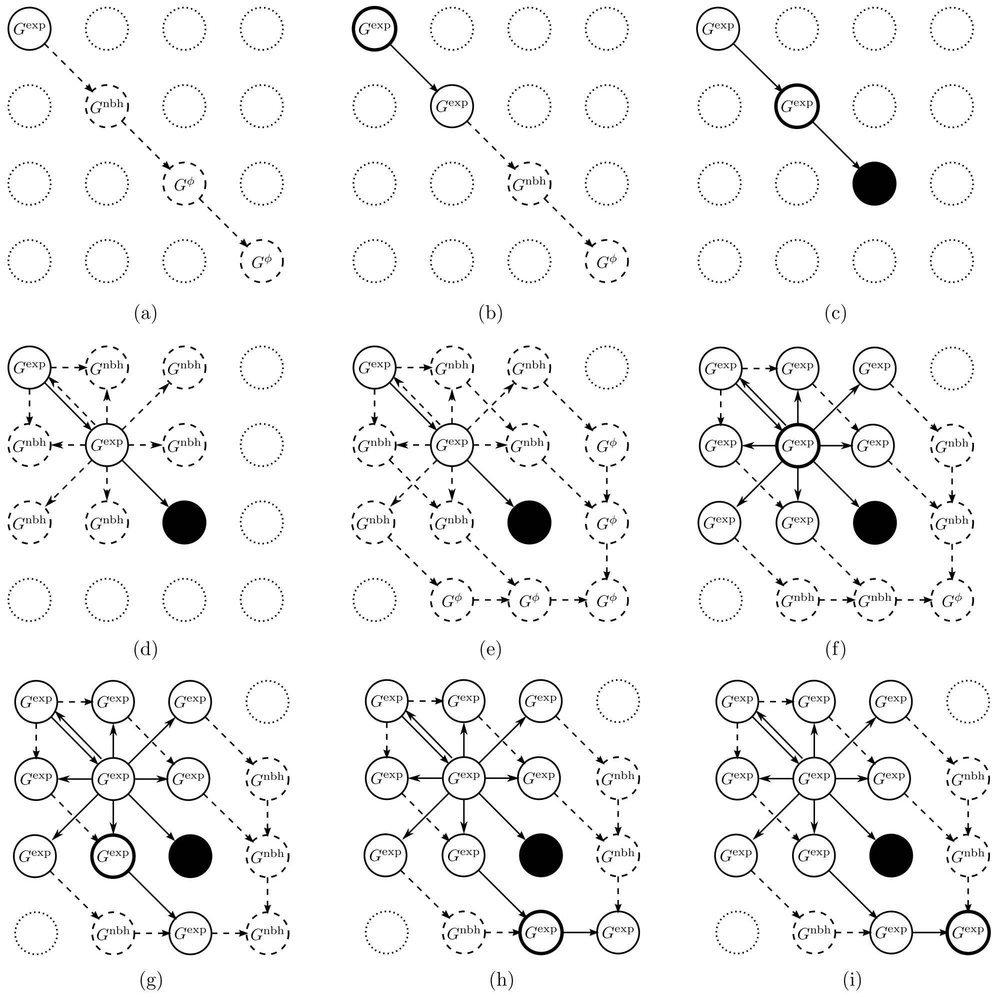


Fig. 8. The above figures depict how the explored graph $G^{\exp}$ and the search graph G^{sch} evolve in the configuration space. Vertices are represented as circles, with arrows representing directed edges. $G^{\exp}$ is depicted by solid lines, while $G^{\text{sch}} \setminus G^{\exp}$ is depicted 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.

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 Fig. 8 for an illustration of how the subgraphs change over time.

4.3.2. *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 [1], 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.

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. $\square$

Next, assume that an optimal collision-free path from v_s to v_f exists, *i.e.* the 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 $g(\pi(v_s, v_c)) + h(v_c) \le g(\pi_*(v_s, v_f))$, and $\exists v_b \in \pi(v_s, v_c)$ such that $\Psi(v_c) \nsubseteq C_b$.*

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 satisfy 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 $\Psi(v_k) \ne \emptyset$, and by (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 (4). Since $\Psi(v_k) \nsubseteq C_k$, 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.[5] $\square$

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 $g(\pi_*(v_s, v_f))$ [1]. 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. $\square$

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. $\square$

[5] We note that if the equality $g(\pi(v_s, v_c)) + h(v_c) \le g(\pi_*(v_s, v_f))$ 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.

With these auxiliary 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 configuration graph contains an optimal path $\pi_*(v_k, v_f)$, then $\forall \Omega \subset I$, $g(\pi'_\Omega(v_k, v_f)) \le g(\pi_*(v_k, v_f)$. Furthermore, if $\Omega_1 \subset \Omega_2$, then $g(\pi'_{\Omega_1}(v_k, v_f)) \le g(\pi'_{\Omega_2}(v_k, v_f))$.*

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 $\pi_*(v_k, v_f)$, $g(\pi'_\Omega(v_k, v_f)) \le g(\pi_*(v_k, v_f))$. By the same logic, if $\Omega_1 \subseteq \Omega_2$, then $g(\pi'_{\Omega_1}(v_k, v_f)) \le g(\pi'_{\Omega_2}(v_k, v_f))$. □

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 (2). Since $C_l \subseteq C_k$ by (4), Lemma 6 implies

$$g\big(\pi'_{C_k}(v_k, v_l)\big) + g\big(\pi'_{C_l}(v_l, v_f)\big) \le g\big(\pi'_{C_k}(v_k, v_f)\big) \le g\big(\pi_*(v_k, v_f)\big) \tag{5}$$

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 (5) gives the bound $g(\pi''(v_s, v_f)) \le g(\pi'_{C_s}(v_s, v_f)) \le g(\pi_*(v_s, v_f))$. 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 $\Psi(v_c) \ne \emptyset$. 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 (2), $g(\pi''(v_s, v_c)) + h(v_c) \le g(\pi''(v_s, v_f)) \le g(\pi_*(v_s, v_f))$, 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 $v_l \in G^{\text{exp}}$, 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 (5) does not apply. However, in this case the path induced by ϕ from v_l must terminate at some vertex v_c with $\Psi(v_c) \ne \emptyset$. We construct a new path by following $\pi''(v_s, v_f)$ to v_l, and then following $\pi_\phi(v_l, v_f)$ 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 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. □

4.4. 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 robot graphs. Normalized to the cost of a single A* expansion, b^n, the total cost of all M* expansions of a given vertex is

$$\sum_{i=0, i\neq 1}^{n} \left(\frac{1}{b}\right)^i \leq \sum_{i=0,}^{n} \left(\frac{1}{b}\right)^i = \frac{1-(\frac{1}{b})^{n+1}}{1-\frac{1}{b}} \leq \frac{b}{b-1} \tag{6}$$

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 configuration graph. $|V|$ is exponential in the number of robots. In practice the cost of maintaining the collision set is not significant.

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 EPEA* replace A* with Operator Decomposition (OD) [2] and Enhanced Partial Expansion A* (EPEA*) [12], 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 [59] 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.

5.1. *Recursive* M*

The M* algorithm described in Section 4.2 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 4.3.2 apply to rM*.

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 [60–64]. Furthermore, the resultant path will cost no more than $\epsilon \cdot g(\pi_*(v_s, v_f))$ [65]. The logic in Section 4.3.2 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.

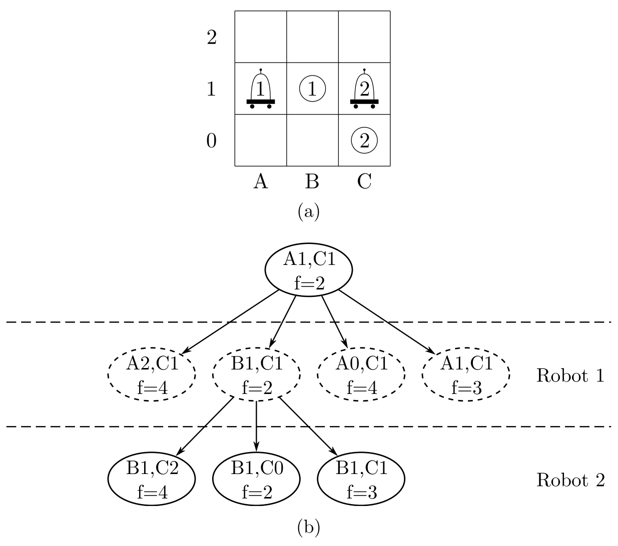


Fig. 9. Operator Decomposition is used to solve a simple, 2 robot path planning problem (a), where the robots move from vertices $A1$ and $C1$ to the goals $B1$ and $C0$. Initially, the search tree contains a single, standard vertex $\{A1, C1\}$ (b). When $\{A1, C1\}$ is expanded, four intermediate vertices, 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 nodes. The goal vertex $\{B1, C0\}$ has the lowest remaining f-value, and is expanded next, indicating that a path has been found [56, adapted].

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. Operator Decomposition (OD) [2] and Enhanced Partial Expansion A* (EPEA*) [12] 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.

5.3.1. 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 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.

Fig. 9 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.

5.3.2. 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 [12]. EPEA* seeks to eliminate the generation of *excess nodes*, 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 nodes 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. [12] report that EPEA* outperforms A* and OD when solving dense multirobot path planning problems.

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) [59] is a planning framework introduced by Sharon et al. based on their *Conflict-Based Search* (CBS) planning algorithm [57], and generalizes the earlier *Independence Detection* (ID) algorithm by Standley [2]. Conflict-Based Search explores a space of constraints on individual robots, rather than the joint configuration space of the system. Each search node 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 node 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 nodes 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 [59] 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 [59].

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.

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 (Fig. 10a). 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 (Fig. 10b). In the case of M*, r^3 is not involved in any collision, and thus will remain restricted to its individually optimal path (Fig. 10c). 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

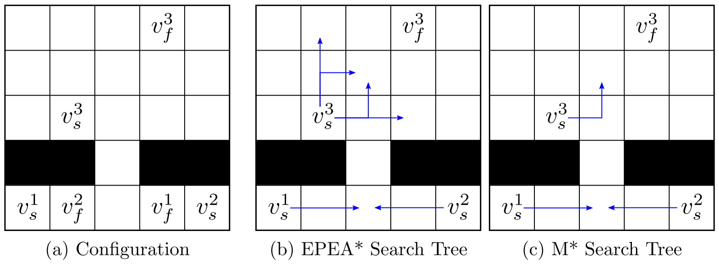


Fig. 10. Illustrative example of the computation benefit of M* compared with A*, OD, or EPEA*. (a) Robots start at v_s^1, v_s^2, and v_s^3 and move to v_f^1, v_f^2 and v_f^3 respectively. (b) EPEA* must construct a search tree containing multiple alternate paths before it can consider moving r^1 into the alcove. (c) M* does not need to consider alternate paths for r^3 before M* can consider moving r^1 into the alcove.

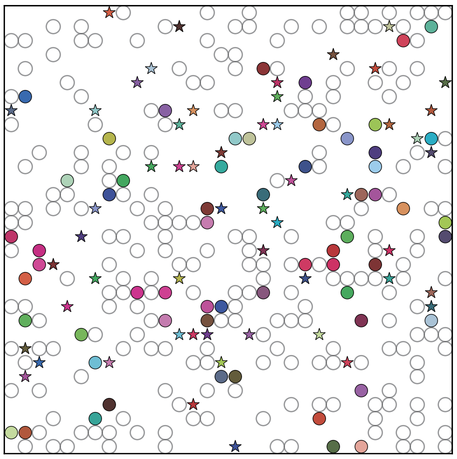


Fig. 11. A typical four-connected grid world with 32×32 cells for a test run including 40 robots. Shaded circles represent the initial positions of the robots, while goal positions are marked with stars with matching shading. Unfilled circles represent the obstacles.

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.

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 an Intel Core i5-2500 processor clocked at 3.30 GHz (Turbo Boost disabled) with 8 GB of RAM. The test environment was a 32×32, four-connected grid of cells, with a 20% probability of a given cell being marked as an obstacle, as in [2] (Fig. 11). 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.

7.1. M*, *operator decomposition and* rM*

We start by comparing A*, OD, EPEA*, M*, EPEM*, ODM*, rM*, ODrM* and EPErM* (Fig. 12). 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.

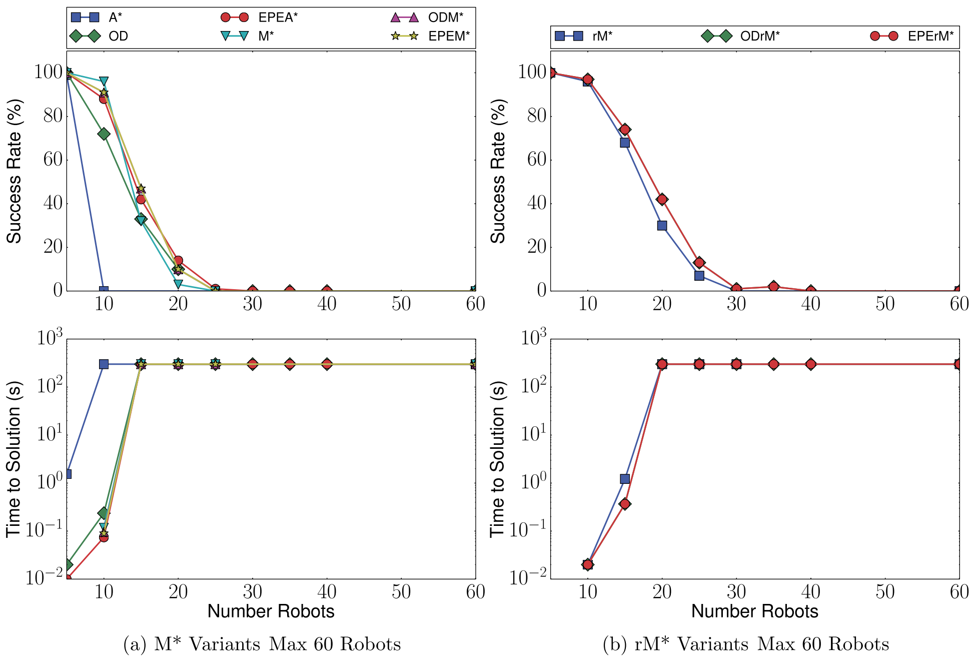


Fig. 12. Results for A*, OD, EPEA*, M*, ODM*, and EPEM* (a) and rM*, ODrM*, and EPErM* (b). The plots on top illustrate the percentage of trials in which a solution was found within 5 minutes, in a 32 × 32 four-connected grid world. The bottom graphs illustrate the median time to solution.

Table 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 32 × 32 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.

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

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, 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* (Fig. 12b). 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 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,

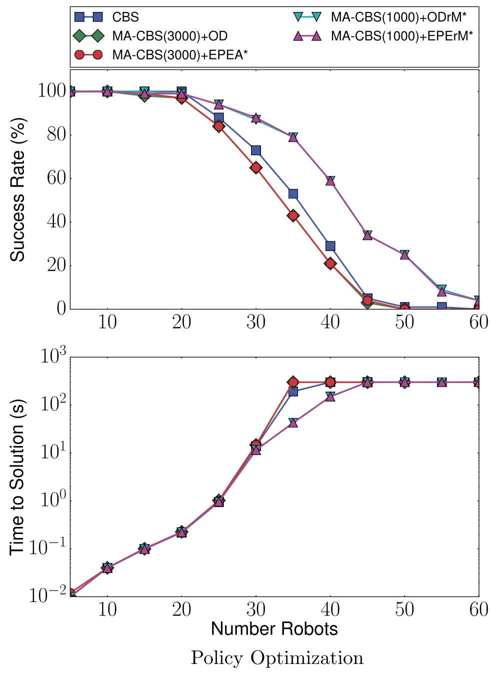


Fig. 13. Results for CBS, MA-CBS(10)+OD, MA-CBS(20)+ODrM*, and MA-CBS(30)+EPErM*. The plot on top illustrates the percentage of trials in which a solution was found within 5 minutes, in a 32×32 four-connected grid world. The bottom graph illustrates the median time to solution.

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.

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.

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 Fig. 13. 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 [59]. 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 [10].

7.3. Inflated heuristics

We tested A*, M*, EPEA* and variants of M* with a heuristic inflated by a factor of 1.1 (Fig. 14a). 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 (Fig. 14b). Inflated ODrM* and EPErM* were able to find solutions more quickly, in more cases, and with a simpler implementation than

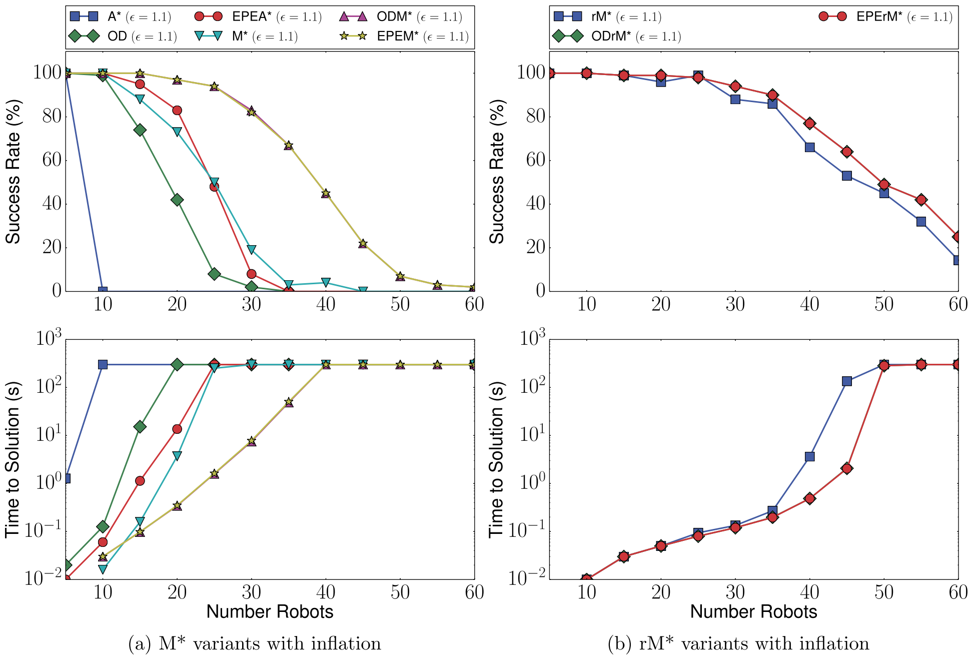


Fig. 14. Results for A*, OD, M*, ODM*, rM* and ODrM* with a heuristic inflated by 10% (a) and extended results to 100 robots for inflated ODM*, inflated rM*, inflated ODrM*, and MA-CBS+ODrM*(20) without an inflated heuristic (b). The plots on top illustrate the percentage of trials in which a solution was found within 5 minutes, in a 32×32 four-connected grid world. The bottom graphs illustrate the median time to solution.

MA-CBS(1000)+ODrM*, reflecting the computational benefits of relaxing the requirement to find optimal cost paths, even if only slightly.

7.4. Comparison to rule-based approaches

M* and inflated M* can find optimal or bounded 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. [53]. The PPAS code was not optimized for performance or run time (Fig. 15).

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. 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 Fig. 16. 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 (Fig. 16b), but this is an artifact of EPErM* ($\epsilon = 1.1$) 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.

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 4×4 gird world with up to 15 robots, equivalent to the 15 puzzle.

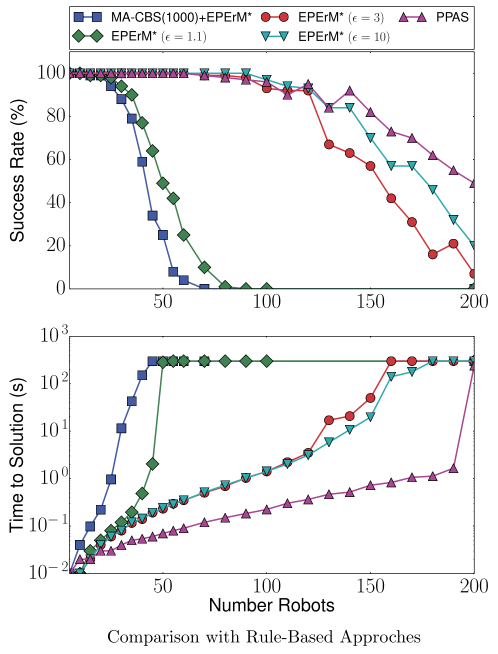


Fig. 15. Results for MA-CBS(1000)+EPErM*, inflated EPErM* with inflation factors of 1.1, 3, and 10, and Parallel Push and Swap (PPAS). The plot on top illustrates the percentage of trials in which a solution was found within 5 minutes, in a 32 × 32 four-connected grid world. The bottom graph the median time to solution. The failures of PPAS were due to the implementation being tested crashing.

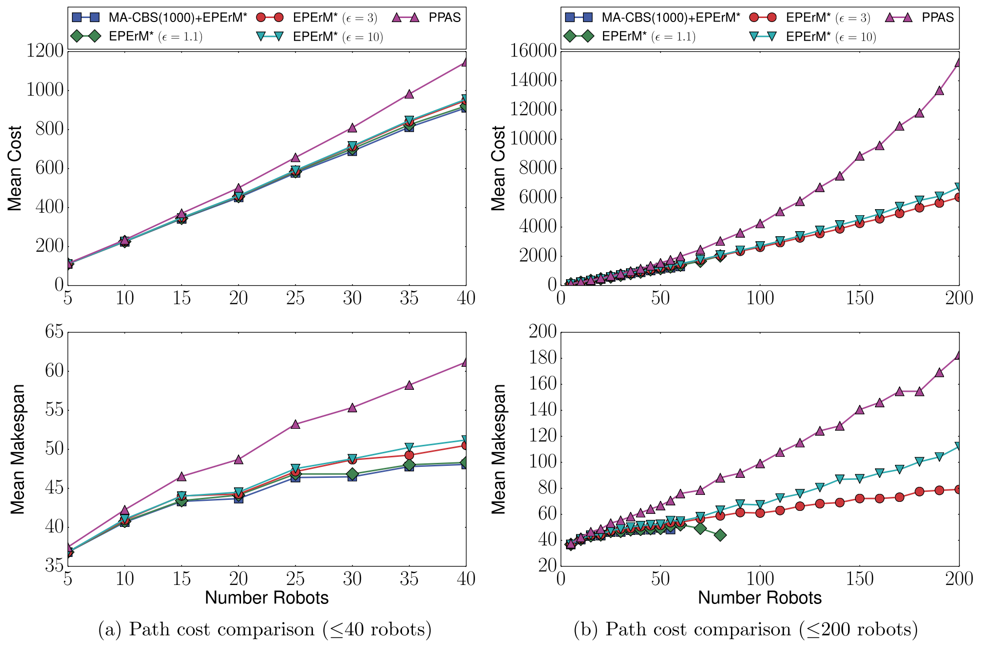


Fig. 16. Path costs for MA-CBS(1000)+EPErM*, inflated EPErM* with inflation factors of 1.1, 3, and 10, and Parallel Push and Swap (PPAS). The plots on top show the mean cost of the paths successfully found by the algorithms. The bottom plots show the mean makespan (time until all robots reach their goal). (a) Results for trials of up to 40 robots. (b) Results for trials of up to 200 robots.

Six optimal approaches were tested, EPEM*, EPErM*, CBS, MA-CBS(1000)+EPErM*, EPEA*, and MA-CBS(1000)+EPEA*[6] (Fig. 17a). 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*

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

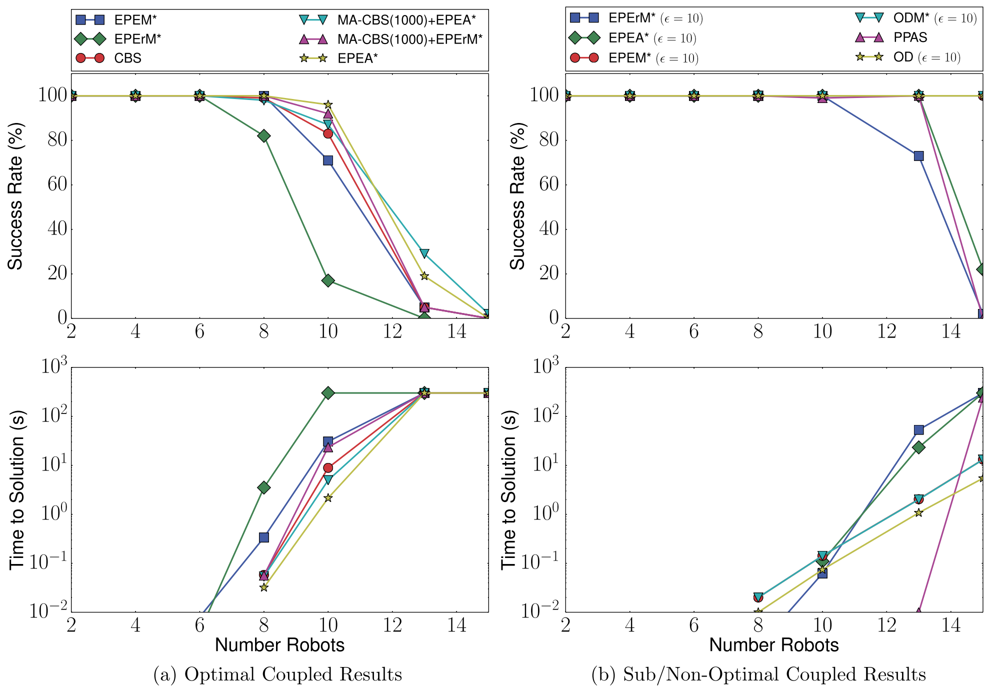


Fig. 17. Results for fully coupled tests on a 4-connected, 4×4 grid world, with (a) optimal and (b) suboptimal and non-optimal algorithms. The plots on top illustrate the percentage of trials in which a solution was found within 5 minutes, in a 32×32 four-connected grid world. The bottom graphs illustrate the median time to solution.

does outperform EPEA* for 13 robots, which we interpret as MA-CBS slightly simplifying some problems before falling back on EPEA*.

Five bounded suboptimal methods were tested; inflated EPEA*, inflated OD, inflated EPEM*, inflated EPErM*, all with an inflation factor of $\epsilon = 10$. The bounded suboptimal methods were tested against PPAS, a non-optimal, rule based method. PPAS can find solutions much faster than any of the bounded suboptimal methods, but fails on all of 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 bounded 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 $\epsilon = 1.1$ (Fig. 14a). 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. [66] 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 32×32 grid environment inflated OD with $\epsilon = 10$ performs roughly on par with EPErM* $\epsilon = 1.1$.

8. Conclusions and further work

In this paper, we presented *subdimensional expansion*, a method for constructing low-dimensional search spaces tailored to specific multirobot path planning problems. We implemented subdimensional expansion using A* as the underlying planning algorithm, resulting in the M* algorithm. While the performance of M* and its variants are marginal on fully coupled problems, in the more expansive environments for which M* was designed, M* provides considerable improvements in performance for finding minimal cost paths for multirobot systems compared to the existing state of the art. Furthermore, inflated M* can solve large problems involving 200 robots, and produce paths that cost significantly less than those generated by existing rule based planners which can also solve such large problems.

In our future work, we will consider several extensions of subdimensional expansion. Barer et al. [58] recently published a bounded-suboptimal variant of CBS termed ECBS which can outperform inflated ODrM*/EPErM* when small inflation factors are used. We believe that combining ECBS and inflated EPErM* will result in further performance improvements, similar to the improvements in optimal path planning seen in MA-CBS+ODrM*. Minimal cost paths for multirobot systems often feature robots passing very close to one another. Minor errors in plan execution can thus result in collisions, or time consuming reactive collision avoidance, which may invalidate the entire plan. Subdimensional expansion can be extended to account for uncertainty in plan execution by utilizing a collision function which treats each robot as occupying a distribution of possible states. The collision sets would then track the probability of a collision occurring, with the dimensionality of the search space increased when the probability of a collision rises above some threshold value.

While subdimensional expansion is currently formatted for multirobot systems, we plan to extend the concept of subdimensional expansion to other high-dimensional systems. There are some single robot systems with high-dimensional configuration spaces which can be decomposed into semi-independent subspaces in a similar manner to how multirobot systems can be decomposed into individual robots. For instance, changing the motion of joints of a robot arm distal to a collision cannot resolve said collision. The challenge will lie in identifying how a decomposition into largely independent subsystems can be performed for systems without the natural structure of multirobot systems.

Acknowledgements

We would like to thank Cornelia Ferner for her assistance with policy optimization. We would also like to thank Professor Bekris and Qandeel Sajid for making their Parallel Push and Swap code available.

References

[1] P. Hart, N. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (2) (1968) 100–107.
[2] T. Standley, Finding optimal solutions to cooperative pathfinding problems, in: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI), Atlanta, GA, USA, 2010, pp. 173–178.
[3] D. Ratner, M. Warmuth, Finding a shortest solution for the $N \times N$ extension of the 15-puzzle is intractable, J. Symb. Comput. 10 (1990) 111–137.
[4] M. Erdmann, T. Lozano-Perez, On multiple moving objects, Algorithmica 2 (1) (1987) 477–521, ISSN 0178-4617.
[5] K. Kant, S. Zucker, Toward efficient trajectory planning: the path–velocity decomposition, Int. J. Robot. Res. 5 (3) (1986) 72–89, ISSN 0278-3649.
[6] S. Leroy, J.-P. Laumond, T. Siméon, Multiple path coordination for mobile robots: a geometric algorithm, in: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 1-55860-613-0, 1999, pp. 1118–1123.
[7] M. Saha, P. Isto, Multi-robot motion planning by incremental coordination, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China, 2006, pp. 5960–5963.
[8] D. Silver, Cooperative pathfinding, in: Proceedings of the 1st Conference on Artificial Intelligence and Interactive Digital Entertainment, Marina del Rey, California, USA, 2005, pp. 23–28.
[9] G. Wagner, H. Choset, M*: a complete multirobot path planning algorithm with performance bounds, in: Proceedings of the IEEE International Conference on Intelligent Robots and Systems, San Francisco, CA, USA, 2011, pp. 3260–3267.
[10] C. Ferner, G. Wagner, H. Choset, ODrM*: optimal multirobot path planning in low dimensional search spaces, in: Proceedings of the IEEE/RSJ International Conference on Robotics and Automation, Karlsruhe, Germany, 2013.
[11] D. Carmel, S. Markovitch, Incorporating opponent models into adversary search, in: AAAI/IAAI, vol. 1, Portland OR, USA, 1996.
[12] A. Felner, M. Goldenberg, G. Sharon, R. Stern, T. Beja, N. Sturtevant, J. Schaeffer, R. Holte, Partial-expansion A* with selective node generation, in: Proceedings of the AAAI Conference on Artificial Intelligence, Toronto, Canada, 2012.
[13] M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. Sturtevant, R. Holt, J. Schaeffer, Enhanced partial expansion A*, J. Artif. Intell. Res. 50 (2014) 141–187, http://dx.doi.org/10.1613/jair.4171.
[14] R. Korf, Depth-first iterative-deepening: an optimal admissible tree search, Artif. Intell. 27 (1) (1985) 97–109, http://dx.doi.org/10.1016/0004-3702(85)90084-0, ISSN 00043702.
[15] S. Carpin, E. Pagello, On parallel RRTs for multi-robot systems, in: Proceedings of the 8th Conference of the Italian Association for Artificial Intelligence, 2002, pp. 834–841. Citeseer.
[16] D. Ferguson, N. Kalra, A. Stentz, Replanning with RRTs, in: Proceedings of the IEEE International Conference on Robotics and Automation, Orlando, FL, USA, 2006, pp. 1243–1248.
[17] G. Sánchez, J. Latombe, On delaying collision checking in PRM planning: application to multi-robot coordination, Int. J. Robot. Res. 21 (1) (2002) 5.
[18] G. Sanchez, J. Latombe, Using a PRM planner to compare centralized and decoupled planning for multi-robot systems, in: Proceedings of the IEEE International Conference on Robotics and Automation, vol. 2, Washington D.C., USA, 2002, pp. 2112–2119.
[19] M. Cáp, P. Novák, J. Vokřínek, M. Pěchouček, Multi-agent RRT*: sampling-based cooperative pathfinding, in: Autonomous Robots and Multirobot Systems Workshop at AAMAS 2013, 2013.
[20] K. Solovey, O. Salzman, D. Halperin, Finding a needle in an exponential haystack: discrete RRT for exploration of implicit roadmaps in multi-robot motion planning, in: WAFR, 2014.
[21] G. Wagner, M. Kang, H. Choset, Probabilistic path planning for multiple robots with subdimensional expansion, in: Proceedings of the IEEE/RSJ International Conference on Robotics and Automation, 2012.
[22] R. Huang, Y. Chen, W. Zhang, A novel transition based encoding scheme for planning as satisfiability, in: AAAI Conference on Artificial Intelligence, 2010, pp. 89–94.
[23] H. Kautz, B. Selman, Unifying SAT-based and graph-based planning, in: IJCAI, 1999.
[24] P. Surynek, An SAT-based approach to cooperative path-finding using all-different constraints, in: Proceedings of Symposium on Combinatorial Search, 2012, pp. 191–192.
[25] P. Surynek, Optimal cooperative path-finding with generalized goals in difficult cases, in: Tenth Symposium of Abstraction, Reformulation, and Approximation, 2013, pp. 119–122.
[26] E. Erdem, D.G. Kisa, U. Oztok, P. Schueller, A general formal framework for pathfinding problems with multiple agents, in: AAAI Conference on Artificial Intelligence, 2013, pp. 290–296.
[27] T. Balyo, R. Bartak, P. Surynek, Shortening plans by local re-planning, in: 24th International Conference on Tools with Artificial Intelligence, Athens, Greece, 2012.
[28] P. Surynek, Solving abstract cooperative path-finding in densely populated environments, in: Computational Intelligence, vol. 30, 2012.
[29] R. Cui, B. Gao, J. Guo, Pareto-optimal coordination of multiple robots with safety guarantees, Auton. Robots 32 (3) (2011) 189–205, ISSN 0929-5593.
[30] J. Peng, S. Akella, Coordinating multiple robots with kinodynamic constraints along specified paths, Int. J. Robot. Res. 24 (4) (2005) 295–310, ISSN 0278-3649.
[31] S. LaValle, S. Hutchinson, Optimal motion planning for multiple robots having independent goals, IEEE Trans. Robot. Autom. 14 (6) (1998) 912–925.

[32] S. Leroy, J.-P. Laumond, T. Siméon, Multiple path coordination for mobile robots: a geometric algorithm, in: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 1-55860-613-0, 1999, pp. 1118–1123.
[33] J. Peng, S. Akella, Coordinating multiple robots with kinodynamic constraints along specified paths, Int. J. Robot. Res. 24 (4) (2005) 295–310, ISSN 0278-3649.
[34] T. Siméon, S. Leroy, J.-p. Laumond, Path coordination for multiple mobile robots: a resolution-complete algorithm, IEEE Trans. Robot. Autom. 18 (1) (2002) 42–49.
[35] M. Čáp, P. Novák, M. Selecký, J. Faigl, J. Vokřínek, Asynchronous decentralized prioritized planning for coordination in multi-robot system, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, ISBN 9781467363587, 2013, pp. 3822–3829.
[36] V.R. Desaraju, J.P. How, Decentralized path planning for multi-agent teams with complex constraints, Auton. Robots 32 (4) (2012) 385–403, ISSN 0929-5593.
[37] R. Regele, P. Levi, Cooperative multi-robot path planning by heuristic priority adjustment, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, ISBN 1-4244-0258-1, 2006, pp. 5954–5959.
[38] J. van den Berg, M. Overmars, Prioritized motion planning for multiple robots, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, Alberta, Canada, 2005, pp. 2217–2222.
[39] A. Geramifard, P. Chubak, V. Bulitko, Biased cost pathfinding, in: AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Marina del Rey, California, ISBN 978-1-57735-235-8, 2006, pp. 112–114.
[40] S. Buckley, Fast motion planning for multiple moving robots, in: Proceedings of IEEE International Conference on Robotics and Automation, vol. 1, 1989, pp. 322–326.
[41] B. Maren, B. Wolfram, T. Sebastian, Constraint-based optimization of priority schemes for decoupled path planning techniques, in: KI 2001: Advances in Artificial Intelligence, 2001, pp. 78–93.
[42] M. Turpin, N. Michael, V. Kumar, Trajectory planning and assignment in multirobot systems, in: International Workshop on Algorithmic Foundations of Robotics (WAFR), 2012.
[43] M. Turpin, N. Michael, V. Kumar, Concurrent assignment and planning of trajectories for large teams of interchangeable robots, in: IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, ISBN 9781467356428, 2013, pp. 834–840.
[44] R.M. Wilson, Graph puzzles, homotopy, and the alternating group, J. Comb. Theory, Ser. B (1974) 86–96.
[45] D. Kornhauser, G. Miller, P. Spirakis, Coordinating pebble motion on graphs, the diameter of permutation groups, and applications, in: Proceedings of the 25th Symposium on Foundations of Computer Science (FOCS), Singer Island, FL, USA, 1984, pp. 241–250.
[46] R. Gabriele, M. Helmert, Non-optimal multi-agent pathfinding is solved (since 1984), in: Symposium on Combinatorial Search, 2012, pp. 173–174.
[47] M. Peasgood, J. McPhee, C. Clark, Complete and scalable multi-robot planning in tunnel environments, in: Proceedings of the First IFAC Workshop on Multivehicle Systems, Bahia, Brazil, 2006, pp. 75–80.
[48] K. Wang, A. Botea, MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees, J. Artif. Intell. Res. 42 (2011) 55–90.
[49] R. Luna, K. Bekris, Push and swap: fast cooperative path-finding with completeness guarantees, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Barcelona, Spain, 2011, pp. 294–300.
[50] A. Krontiris, R. Luna, K.E. Bekris, From feasibility tests to path planners for multi-agent pathfinding, in: Proceedings of the Sixth International Symposium on Combinatorial Search, 2013, pp. 114–122.
[51] B. de Wilde, A.W. ter Mors, C. Witteveen, Push and rotate: cooperative multi-agent path planning, in: Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Saint Paul, Minnesota, 2013, pp. 87–94.
[52] M. Khorshid, R. Holte, N. Sturtevant, A polynomial-time algorithm for non-optimal multi-agent pathfinding, in: Proceedings of the Symposium on Combinatorial Search, Barcelona, Spain, 2011, pp. 76–83.
[53] Q. Sajid, R. Luna, K.E. Bekris, Multi-agent pathfinding with simultaneous execution of single-agent primitives, in: Proceedings of the Fifth Annual Symposium on Combinatorial Search, 2012, pp. 88–96.
[54] K. Al-Wahedi, A hybrid local–global motion planner for multi-agent coordination, Master's thesis, Case Western Reserve University, 2000.
[55] J. van den Berg, J. Snoeyink, M. Lin, D. Manocha, Centralized path planning for multiple robots: optimal decoupling into sequential plans, in: Proceedings of the Robotics: Science and Systems, vol. 2, 2009.
[56] G. Sharon, R. Stern, M. Goldenberg, A. Felner, The increasing cost tree search for optimal multi-agent pathfinding, in: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, San Francisco, CA, USA, 2011, pp. 662–667.
[57] G. Sharon, R. Stern, A. Felner, N. Sturtevant, Conflict-based search for optimal multi-agent path finding, in: Proceedings of the AAAI Conference on Artificial Intelligence, Toronto, Ontario, Canada, 2012.
[58] M. Barer, G. Sharon, R. Stern, A. 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.
[59] G. Sharon, R. Stern, A. Felner, N. Sturtevant, Meta-agent conflict-based search for optimal multi-agent path finding, in: Proceedings of the Symposium on Combinatorial Search, Niagara Falls, Ontario, Canada, 2012.
[60] I. Pohl, The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving, in: Proceedings of the 3rd International Joint Conference on Artificial Intelligence, 1973, pp. 12–17.
[61] B. Bonet, H. Geffner, Planning as heuristic search, Artif. Intell. 129 (1–2) (2001) 5–33, ISSN 0004-3702.
[62] R. Korf, Linear-space best-first search, Artif. Intell. 62 (1) (1993) 41–78, ISSN 0004-3702.
[63] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison–Wesley, 1984.
[64] J. Gaschnig, Performance measurement and analysis of certain search algorithms, Ph.D. thesis, Carnegie-Mellon University, 1979.
[65] H. Davis, A. Bramanti-Gregor, J. Wang, The advantages of using depth and breadth components in heuristic search, in: Methodologies for Intelligent Systems, vol. 3, 1989, pp. 19–28.
[66] M. Goldenberg, A. Felner, N. Sturtevant, Optimal-generation variants of EPEA*, in: Proceedings of the Sixth International Symposium on Combinatorial Search, Leavenworth, Washington, USA, 2013, pp. 89–97.