Noname manuscript No.
(will be inserted by the editor)

Dynamic Teams of Robots as Ad Hoc Distributed Computers: Reducing the Complexity of Multi-Robot Motion Planning via Subspace Selection

Michael Otte ·
Nikolaus Correll

Received: date / Accepted: date

Abstract We solve the multi-robot path planning problem using three complimentary techniques: (1) Robots that must coordinate to avoid collisions form temporary dynamic teams. (2) Robots in each dynamic team become a distributed computer by pooling their computational resources over ad hoc wireless Ethernet. (3) The computational complexity of each team's problem is reduced by carefully constraining the environmental subspace in which the problem is considered.

An important contribution of this work is a method for quicky choosing the subspace, used for (3), to which each team's problem is constrained. The heuristic is based on a tile-like pebble motion game, and returns *true* only if a subset of the environment will permit a solution to be found (otherwise it returns *false*). We perform experiments with teams of four and six CU Prairiedog robots (built on the iRobot Create platform) deployed in a large residence hall, as well as ten robots in random simulated environments.

Keywords Motion Planning · Multi Robot Team · Ad Hoc Distributed Computer · Any-Com · Dynamic Team

1 Introduction

Autonomous robots are usually equipped with on-board computers. Robots use their computers to aggregate sensor information, reason about the environment, and react to unexpected events in real-time. Most robots are also equipped with a wireless device that enables communication with other robots over a wireless network. Given these resources, there is an opportunity that nearby robots can use wireless communication to pool their computational resources, thus creating an ad hoc distributed computer (Figure 1). For some adequate level of network reliability, we expect such a distributed computer to be more capable of solving computationally challenging problems than any single robot alone.

Centralized multi-robot motion planning, which is concerned with guaranteeing the mutual safety of robots operating in close proximity to each other, is a problem well suited to being solved with such an ad hoc distributed computer. Centralized multi-robot motion planning is a computationally challenging NP-hard problem; moreover, solving it benefits all robots in a particular geographic area. Even robots that are not part of a predefined team must coordinate with nearby robots to ensure mutual safety, and thus benefit from pooling resources with nearby agents to solve their joint motion planning problem.

In this paper we leverage the fact that the computational complexity of each team's joint motion planning problem can be reduced by constraining the hypervolume of the joint configuration space in which the dynamic team's problem is defined. Such a spatial reduction is only useful if it does not prevent us from finding a solution. Thus, an important contribution of this work is the presentation of an admissible heuristic function for subspace selection that allows us to easily check if a particular subset of space is sufficiently large for solving the multi-robot path planning problem. The heuristic is based on the solvability of pebble motion games, and returns true only if the space is sufficiently large (and obstacle-free) to solve a problem with n robots.

Subspace selection and dynamic teams both reduce the size of the configuration space, but in different and complimentary ways. While dynamic teams reduce the

M. Otte
(work done at) University of Colorado at Boulder
Dept. Computer Science
E-mail: ottemw@gmail.com

Nikolaus Correll
University of Colorado at Boulder
Dept. Computer Science

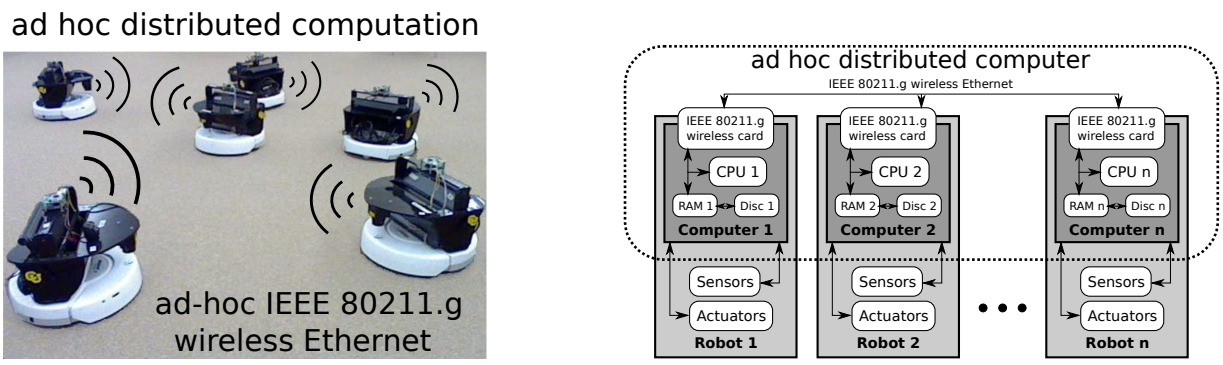


Fig. 1: Left: Six Prairiedog robots using ad hoc distributed computation to solve a motion planning problem. Right: Schematic of an ad hoc distributed computer created across a group of robots.

hypervolume of the configuration space by minimizing its dimensionality (Otte and Correll, 2013a, 2014), subspace selection reduces the hypervolume assuming that a particular dimensionality has already been selected. Thus, subspace selection is well suited for combination with dynamic teams, and we investigate their combination.

In the rest of this section we discuss more formally how subspace selection and dynamic teams reduce the size of the configuration space. The C-FOREST distributed branch and bound sampling-based motion planning technique, which is used by the dynamic teams to solve the motion problems in the resulting subspace, is also discussed.

1.1 Motivation for dynamic teams and subspace selection

The centralized multi-robot motion planning problem has complexity of the form $\mathcal{O}(\|\mathcal{X}\|^n)$, where $\|\mathcal{X}\|$ is the measure of a single robot's configuration space, and n is the number of robots that are considered. Previous work (Al-Wahedi, 2000; van Den Berg et al, 2009; Standley and Korf, 2011; Sharon et al, 2012) has shown that problem complexity can often be reduce by considering separately each non-overlapping sub-problem. In practice this amounts to only considering the joint problem of two or more robots if their individual solutions conflict, i.e., would result in a collision between two or more robots. More formally, these methods work by solving separate problems in each space $\mathcal{X}^{|R_1|}$, ..., $\mathcal{X}^{|R_k|}$ such that teams $R_1, \ldots, R_k$ do not conflict with each other and $|R_1| + \ldots + |R_k| = n$, instead of in the much larger $\mathcal{X}^{|R_1|} \times \ldots \times \mathcal{X}^{|R_k|} = \mathcal{X}^n$ full configuration space of all robots.

The method of considering a joint problem for sets of robots that conflict has been called using dynamic teams (Otte and Correll, 2014), dynamic networks (Clark et al, 2003), subdimensional expansion (Wagner and Choset, 2015; Wagner et al, 2012), etc., and results in computational complexity that can be described as $\mathcal{O}(\|\mathcal{X}\|^{|R_{\max}|})$, where $|R_{\max}| \leq n$ is the number of robots in the largest subteam. In (Otte and Correll, 2014) we experimentally observed that even the use of dynamic teams may be insufficient to achieve tractability in large environments with large $\|\mathcal{X}\|$.

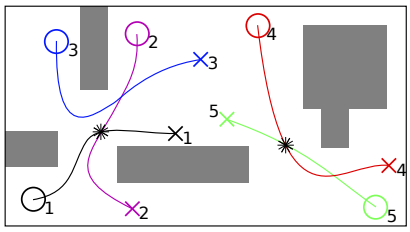
Subspace selection is a complementary means of reducing computational complexity in which each team R restricts its own $|R|$-dimensional problem to a reduced subset of space $\mathcal{S}_R \subseteq \mathcal{X}_R$ containing the potential conflicts that created the dynamic team in the first place, see Figure 2. Subspace selection decreases computational complexity whenever $\|\mathcal{S}\|^{|R_{\max}|} < \|\mathcal{X}\|^{|R_{\max}|}$, often by orders of magnitude, but at the price of global optimality.

The use of subspace selection assumes that we have a judicious way to choose $\mathcal{S}_R$, which is why we require a sufficiency heuristic for solvability of the motion planning problem in $\mathcal{S}_R$.

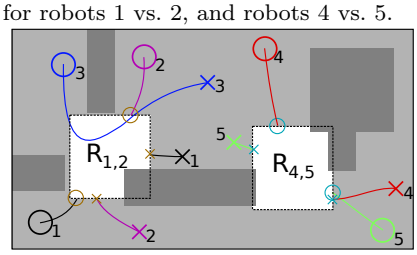
1.2 Motivation for C-FOREST

C-FOREST is a distributed branch and bound technique designed for sampling-based motion planning algorithms that is easy to parallelize across an ad hoc distributed computer (Otte and Correll, 2013b, 2014; Otte, 2011).

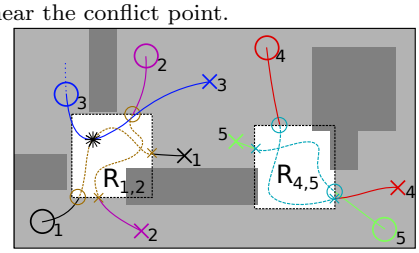
Each computational node works to solve the same planning problem by building its own (unique) random tree; thus, a forest of trees is built in parallel. C-FOREST differs from OR-parallelization (i.e., voting) in that C-FOREST's random trees communicate with each other *during* the search. Tree-to-tree communication during the search enables distributed branch and bound as well as the exchange of other useful information.



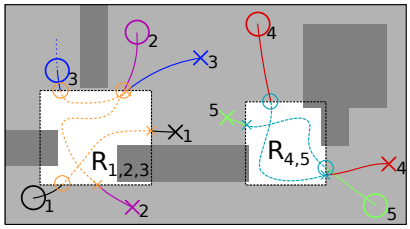
Robots 1, 2, 3, 4, and 5 start in their own teams (black, magenta, blue, red, and green, respectively). 'O's denote start locations and 'X's denote goal locations. Each robot plans its own path; however, this leads to possible collisions (black asterisks) for robots 1 vs. 2, and robots 4 vs. 5.



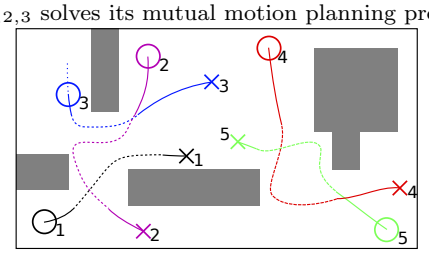
Each set of conflicting robots combines into a team (team $R_{1,2}$ is gold and $R_{4,5}$ is green). Each team forms an ad hoc distributed computer to solve its mutual motion planning problem in a reduced subset of space (non grayed-out regions) near the conflict point.



The solution of team $R_{1,2}$ conflicts with that of robot 3. Thus, team $R_{1,2,3}$ (orange) is formed by combining team $R_{1,2}$ with robot 3.



Team $R_{1,2,3}$ solves its mutual motion planning problem.



After a team has solved its mutual problem, its robots merge the multi-robot solution into their own plans and synchronize along the shared solution.

Fig. 2: Dynamic teams, based on conflicts, form ad hoc distributed computers to motion plan using subspace selection.

Whenever a better path is discovered by any random tree, the graph nodes of that path are broadcast to all other trees. This enables: (1) All nodes to prune their trees and sample space based on the best solution found by any tree—increasing search speed and focusing remaining effort on exploring helpful portions of the configuration space. (2) By inserting the nodes of the best path into their own trees, different nodes exchange data about useful explorations into new homotopy classes and are able to improve the best known path directly

Given a robot team in which each agent has its own CPU, each robot is responsible for growing (at least) one C-FOREST tree. Robots communicate wirelessly, and so the team becomes an ad hoc distributed computer to solve its joint problem using C-FOREST.

C-FOREST (like any distributed branch and bound approach) will require more communication than OR-Parallelization. That said, even if C-FOREST saturates the channel during planning such that no messages get through, then the final round of path broadcasts and agreement makes the performance essentially identical to that of OR-parallelization.

1.3 Motivation for an Any-Com algorithm

Distributed computing algorithms typically assume a reliable communication medium. In contrast, wireless communication in real world environments is inherently unreliable, and so distributed algorithms designed to run on an ad hoc distributed computer must be designed to function despite this unreliability. Any-Com algorithms (Otte and Correll, 2013a; Otte, 2011) are defined by the property of graceful performance declines versus degrading communication, and are thus well suited to this task. In (Otte and Correll, 2014; Otte, 2011) we demonstrate that C-FOREST has good any-com properties.

2 Related Work

2.1 Our previous work

In our previous work (Otte and Correll, 2013a) we presented an algorithm for distributing the computation of asymptotically optimal centralized multi-robot motion planning across a robot team. In (Otte and Correll, 2013b), we described the underlying parallelization technique, called C-FOREST, and showed that it often has super-linear speedup on a *traditional* distributed computer with *reliable* communication. In (Otte and Correll, 2014) we implemented a distributed version of

C-FOREST across a dynamic team of robots that used ad hoc distributed computing over an unreliable wireless connection. Also in (Otte and Correll, 2014) we observed that a combination of dynamic teams (to reduce problem complexity) and ad hoc distributed computing (to maximize computing power) was still unable to solve relatively simple four-robot problem when the environment was large. In (Otte and Correll, 2014) we conjectured that computational complexity would be further reduced by judiciously selecting a subset of the original problem space in which to resolve conflicts between different robots, and that this might increase the number of problem instances for which a solution could be found within a practical amount of time. In the current paper we investigate this idea, and refer to it as subspace selection.

Much of the material in the current paper originally appeared as the final chapter in the first author's PhD dissertation (Otte, 2011), but has not previously been presented at a conference or in a journal paper. The current paper also contains additional analysis that did not appear in (Otte, 2011). In particular, proofs that the solvability of a tile moving game in free space rectangles can be used as a sufficiency heuristic for solvability of the multi-robot motion planning problem in a reduced subset of the configuration space.

2.2 Previous work using subspace selection and/or dynamic teams

In (Clark et al, 2003) robots form teams that include all other robots within communication range. Whenever two different teams get close enough to communicate they merge to form a larger team. A network-level protocol ensures that all robots in a team exchange world state information, so that each robot is able to solve the centralized motion planning problem for the team. This can be described as OR-parallelization of kynodynamic PRM, in which different random solutions are computed independently but in parallel and the best one used. Each robot has a dedicated off-board computer. One difference versus our work is that communication between the computers in (Clark et al, 2003) is reliable, although range effects are simulated in (Clark et al, 2003) such that robots may only communicate with nearby robots. In contrast, our communication is wireless and unreliable.

In (Clark et al, 2003) teams are created and joined based on communication range only, which means that teams may form when robots do not conflict. As a result teams using the method in (Clark et al, 2003) will often be larger (and will therefore need to solve more complex problems) than in our work.

The idea of starting each robot in its own team, and then combining robots into teams based on conflicts, has previously appeared in (Al-Wahedi, 2000; van Den Berg et al, 2009; Standley and Korf, 2011; Sharon et al, 2012), although the idea is frequently described as solving decoupled or independent sub-problems. The most closely related of these works is (Al-Wahedi, 2000), which uses both dynamic teams and subspace selection in conjunction with a fast-marching level set method to find paths in a simulated checkerboard environment. Differences versus our work include our use of sampling-based motion planning, parallel computation, and physical teams of robots (as well as the specific conditions under which different teams can be combined).

Previous work (van Den Berg et al, 2009) has approached the multi-robot path planning problem from a centralized graph theoretic point of view, assuming a PRM roadmap graph of the environment is provided a priori for each agent. An A* algorithm-based solution is presented by Standley and Korf (2011). Other work by Sharon et al (2012, 2015) considers a version of the problem on a predefined movement grid over the environment and allows a user-defined number of greedy attempts to resolve conflicts between agents before teams are combined; A* and a number of its descendants are used in experiments. Sharon et al (2012, 2015) also observe that conflict-based team combination optimal motion planning algorithms are, in general, exponentially complex in the size of the largest team. Differences between our work and (van Den Berg et al, 2009; Standley and Korf, 2011; Sharon et al, 2012, 2015) are that these previous ideas do not consider communication issues between agents, do not use subspace selection, and do not use the distributed resources of a multi-robot team as a distributed computer to solve the problem faced by the team.

In the current paper we solve a relaxation of asymptotically optimal motion planning in continuous space that attempts to find short paths but trades asymptotic optimality for better practical performance (this is discussed further in Section 4.2); in contrast, Standley and Korf (2011); Sharon et al (2012, 2015) use an optimal algorithm in discrete space, and van Den Berg et al (2009) uses a resolution optimal approach.

Recent work (Wagner and Choset, 2015) on M* and recursive-M* (and other related algorithms) considers a general modification to graph-based shortest path planning algorithms (with specific implementation within the A* algorithm and a number of its descendants), that handles team combination at a low level within the graph-search algorithm; experiments are performed on a predefined graph. M* and related algorithms propagate collision set data backward through nodes along

the search path (all the way to conflicting robots' start points), which are then added to the priority queue as new joint space points. Additional search effort has the effect of rebuilding the search tree while accounting for the coupled states of conflicting robots. Of the algorithms presented by Wagner and Choset (2015), recursive-M* is the most closely related to our approach. Runtime complexity is equivalent to planning in the joint configuration space of the largest team after the final backpropogation and subdimensional expansion. Recursive M* considers the joint movement of conflicting robots in the subset of the search-tree preceding nodes at which disjointed policies lead to a collision. This is a form of subspace selection (where the particular space selected is determined by start locations, collision points, and the topology of the workspace). Our work differs from Wagner and Choset (2015) in that we consider both communication issues between robots and the potential for parallel processing within a robot team. Finally, Wagner and Choset (2015) use a single robot graph that is provided over the workspace a priori (and is identical for all robots).

The sRRT algorithm (Wagner et al, 2012) applies some of the principles used in M* to extensions of RRT. sPRM, also presented by Wagner et al (2012), does the same for PRM, which basically requires providing recursive M* an input PRM graph for each robot. Our work differs in that collision set information is encoded in the heuristically chosen sub-areas located near to the point of collision (enabling each robot in the team to use its individual paths both to and from the sub-area, and often allowing smaller sub-ares to be used—but at the cost of asymptotic optimality).

A related idea is to have each agent plan its own path, have each agent share those paths with nearby robots, and then have those or other robots check for pair-wise path conflicts. In (Scerri et al, 2007) tokens that correspond to single-robot paths are passed among the robots, and robots that have possession of tokens are allowed to report when they detect pair-wise conflicting paths. The robots involved in the collision then replan. Desaraju and How (2012) also use tokens to facilitate pair-wise conflict resolution. Tokens are passed to the robots that report the most useful path modifications, and those robots are empowered to modify their teammates' conflicts. Methods that only fix pair-wise problems are incomplete, since it is possible to design pathological cases involving conflict between more than two robots that cannot be resolved by replanning for only two robots; however, such methods often work well for problems in which path conflicts are rare.

Dynamic teams based on shared problems and/or communication have also been used in the context of other (non-motion) planning problems; for example, coverage, e.g., for search-and-rescue, (Arrichiello et al, 2009; Voyles et al, 2010), surveillance (Hsieh et al, 2007), remote sensing (Allred et al, 2007), exploration (Voyles et al, 2009), and maintaining a communication link (Dixon and Frew, 2007; Elston et al, 2009), etc.

2.3 Previous work on pebble motion games

Multi-robot motion planning is shown to be a relaxation of pebble motion games (Kornhauser et al, 1984; Auletta et al, 1999; Peasgood et al, 2006; Surynek, 2009; Standley and Korf, 2011; Goldreich, 2011; Krontiris et al, 2013; Solovey and Halperin, 2014; Yu and Rus, 2015; de Wilde et al, 2014; Surynek, 2014) that are themselves a more general form of the 15-puzzle (Johnson et al, 1879; Loyd and Gardner, 1959; Wilson, 1974; Ratner and Warmuth, 1986). Pebble motion is relevant to the subspace selection used in our work because any subspace that contains enough free space to solve the n pebble motion problem is sufficiently large that a sampling-based motion planning algorithm will find a solution to the n robot problem in that subspace with probability one, in the limit, as the number of samples approaches infinity (in our work pebble size represents robot width plus a non-zero clearance term used to ensure a solution can be found with probability greater than zero).

Early work by Johnson et al (1879) showed that the 15-puzzle had two classes of random starting configurations, one that can be solved and one that cannot. This result was extended to non-separable graphs (with a few caveats) by Wilson (1974), while Kornhauser et al (1984) show that the number of moves required along an optimal solution path between any two configurations is $\mathcal{O}(n^3)$ and present a polynomial-time decision algorithm that calculates whether or not a particular problem instance is solvable (but does not provide a solution to the instance).

Finding an optimal solution to any N-by-N pebble motion puzzle (which includes the 15-puzzle) was shown to be NP-complete in (Ratner and Warmuth, 1986). Finding an optimal solution for the general pebbles on graphs problem is shown to be NP-hard by Goldreich (2011). A linear time decision algorithm for the special case that the graph is a tree is given by Auletta et al (1999), who also provide a super-polynomial algorithm for solving an instance. This idea is used to create an (incomplete) algorithm for solving a pebble motion-like version of centralized multi-robot path planning problem by Peasgood et al (2006).

From a topological level, a spanning tree with $n+1$ leaves over the motion graph of the environment is a

sufficient (but not necessary) condition to solve the centralized feasible motion planning problem with n robots (and is quick to solve), (Surynek, 2009). A mechanism for finding a solution in a number of other graphs, beyond spanning trees, is given by Surynek (2009). A cell based formulation is considered by Standley and Korf (2011), who performs extensive experimental trials using different algorithms from artificial intelligence.

Faster practical algorithms for the general case are presented in (Krontiris et al, 2013; de Wilde et al, 2014). Algorithms for the biconnected graph case appear in (Surynek, 2014) and algorithms for the c colored robot case (where any robot of a particular color may go to any location associated with the same color) appear in (Solovey and Halperin, 2014). Recent work by Yu and Rus (2015) focuses on scenarios in which a set of pebbles on a graph cycle may move along that cycle without the existence of a free space.

All of the pebble motion algorithms discussed in this subsection either assume that a graph structure is provided *a priori* (Kornhauser et al, 1984; Auletta et al, 1999; Surynek, 2009; Standley and Korf, 2011; Goldreich, 2011; Krontiris et al, 2013; Solovey and Halperin, 2014; Yu and Rus, 2015; de Wilde et al, 2014; Surynek, 2014) or calculate a graph at runtime (Peasgood et al, 2006) and then solve the pebble motion problem on that graph. In contrast, we leverage a decision test for pebble motion solvability in free space rectangles to ensure that we pick a sufficiently large subspace in which to solve the multi-robot motion planning problem via a sampling-based motion planning algorithm.

2.4 Related work for distributed computation on a team of robots

The basic idea of distributed computing over an ad hoc wireless network or within a robotic team has appeared in a handful of previous works. In (Khoo and Horswill, 2002) an inference engine is implemented across a three-computer team made up of two robots and a command module. The idea of sequentially cycling the computation related to solving a shared coordination problem across a multi-agent team is presented in (Ford et al, 2010) (i.e., the computation of what is essentially a single virtual process rotates through the CPUs of all team members).

Agile computing[1] (Suri et al, 2006, 2008; Suri and Cabri, 2014) and the "Object Interaction Language (OIL)" (Sutton et al, 2010) promote the idea of using a middleware layer to discover and share resources and services among nodes in a distributed computing network that may include ad hoc wireless. In (Johnson et al, 2008), agile computing is applied in the context of mixed teams of humans and robots. In (Otte, 2016), a distributed neural network is created over a swarm of robots and trained to classify images projected onto the swarm; each robot is responsible for maintaining a slice of neurons that communicate with their neighbors on other robots using wireless communication. Although Holland et al (2005) and Nardi et al (2006) consider the design of a particular robotic platform, it is speculated in these works that distributed computation in a robotic swarm could be enabled over a wireless communication medium.

The fact that distributed algorithms for multi-agent teams operating over unreliable communication have graceful performance declines versus degrading communication (i.e., the "Any-Com" property) has been previously observed in (Otte and Correll, 2013a, 2014; Otte, 2011; Rutishauser et al, 2009; Amstutz et al, 2009; Hollinger et al, 2011; Best et al, 2016).

[1] This is unrelated to agile software development practices.

3 Preliminaries

This section contains formal definitions of our nomenclature, followed by a high level description of our heuristic subspace selection.

3.1 Nomenclature

A robot is denoted r. We assume that there are n robots labeled $r_1, \ldots, r_n$. A (temporary) team of robots is defined as the set containing its members. For example a team containing robots $r_i, \ldots, r_k$ is denoted $R_{i,\ldots,k} = \{r_i, \ldots, r_k\}$, where we abuse the notation by dropping the set notation '{' and '}' in the subscript for the sake of readability.

The full configuration space of robot r_i is denoted $\mathcal{X}_i$. A particular configuration of robot r_i denoted x_i. Robot r_i starts at configuration $x_i^{\text{start}} \in \mathcal{X}_i$ and must achieve $x_i^{\text{goal}} \in \mathcal{X}_i$.

The full joint configuration space of a team is given by the product space of its members, i.e., the full joint configuration space of team $R = \{i, \ldots, k\}$ is $\mathcal{X}_R = \mathcal{X}_i \times \ldots \times \mathcal{X}_k$ and a point within this space is $x_R \in \mathcal{X}_R$.

This work is intimately related to different subspaces of each robot's configuration space as well as the product spaces created from them. Let $\mathcal{S}_i$ be a subset of robot r_i's configuration space, where $\mathcal{S}_i \subseteq \mathcal{X}_i$. The product space of multiple subspaces is denoted $\mathcal{S}_{i,\ldots,k} = \mathcal{S}_i \times \ldots \times \mathcal{S}_k \subseteq \mathcal{X}_{i,\ldots,k}$.

If a collision point is detected such that team $R = \{i, \ldots, k\}$ is created, then we consider the conflicting robots' joint sub-problem that is defined in $\mathcal{S}_R = \mathcal{S}_{i,\ldots,k} \subseteq \mathcal{X}_R$. In practice $\mathcal{S}_R$ is a subset of the configuration space located "near to" the collision(s) that cause(d) R to be created in the first place. The particular method of choosing $\mathcal{S}_R$ is largely up to the user, but we assume that some form of heuristic is used (e.g., those in Surynek (2009); Peasgood et al (2006); Clark et al (2003) or the one we present in Section 4) to ensure the probability a solution can be found is sufficiently high for the application being considered.

For the ease of presentation we assume that robots are physically identical such that, when multiple robots form a team, the relevant subspace of each robot's configuration space is identical. In other words, the joint motion planning problem considered by team $R = \{i, \ldots, k\}$ is in space $\mathcal{S}_R = \mathcal{S}_{i,\ldots,k}$ such that $\mathcal{S}_i = \ldots = \mathcal{S}_k$. The assumption $\mathcal{S}_i = \ldots = \mathcal{S}_k$ is not required *in general*; however, robots must agree on the particular planning space $\mathcal{S}_R$ in which the joint problem is defined.

We denote points within $\mathcal{S}_R$ as $s_R \in \mathcal{S}_R$. For a sub-problem defined in $\mathcal{S}_R$ the team starts at $s_R^{\text{start}} = s_i^{\text{start}} \times \ldots \times s_k^{\text{start}}$ where $s_i^{\text{start}} \neq s_j^{\text{start}}$ for all $i \neq j$ (robots start at different locations) and s_i^{start} is the point where robot i enters $\mathcal{S}_R$ (i.e., immediately before the conflict that caused the dynamic team to form).

Similarly, the team agrees to achieve the subgoal $s_{\mathcal{X}_R}^{\text{goal}} = s_i^{\text{goal}} \times \ldots \times s_k^{\text{goal}}$ where $s_i^{\text{goal}} \neq s_j^{\text{goal}}$ for all $i \neq j$ (robots go to different locations) and s_i^{goal} is the point where robot i exits $\mathcal{S}_R$ (i.e., immediately after the conflict that caused the dynamic team to form).

A feasible path for robot r_i is a sequence of ℓ points $P_i = x_i(1), \ldots x_i(\ell)$ such that $x_i(1) = x_i^{\text{start}}$ and $x_i(\ell) = x_i^{\text{goal}}$ and it is safe for robot i to move from point $x_i(j)$ to $x_i(j+1)$ for all j such that $i \leq j \leq \ell - 1$.

Similarly, a multipath $\mathbf{P}_{i,\ldots,k} = s_{i,\ldots k}(1), \ldots s_{i,\ldots,k}(\ell)$ through $\mathcal{S}_{i,\ldots,k}$ is a mutually safe set of paths for robots $i, \ldots, k$ such that $s_{i,\ldots k}(1) = s_{i,\ldots,k}^{\text{start}}$ and $s_{i,\ldots,k}(\ell) = s_{i,\ldots,k}^{\text{goal}}$ and it is safe for each robot $i, \ldots, k$ to move from $s_{i,\ldots k}(j)$ to $s_{i,\ldots k}(j+1)$ for all j such that $i \leq j \leq \ell - 1$. The multipath of team R is denoted $\mathbf{P}_R$

We assume the existence of a (family of) path and multipath measures $\|\cdot\|$ that we seek to minimize. Curve length through the joint state space is one possible choice for $\|\cdot\|$ (in other words, the one-dimensional Lebesgue measure). Ideally, given an environment with n robots, we hope to find $\mathbf{P}^*_{i,\ldots,n}$, the particular $\mathbf{P}_{i,\ldots,n}$ that minimizes $\|\mathbf{P}_{i,\ldots,n}\|$. In practice this is usually intractable, and so we settle for $\hat{\mathbf{P}}_{i,\ldots,n}$, the multipath with smallest $\|\mathbf{P}_{i,\ldots,n}\|$ that we can find given some set of practical constraints. Practical constraints include, for example, a user-defined planning time limit as well as our greedy process of increasing team size and considering higher dimensional problems *only* in the vicinity of robot-robot collision points.

3.2 Overview of heuristic subspace selection

In our implementation we use a lightweight heuristic that either: (1) affirms that the problem instance in $\mathcal{S}_R$ permits a solution or (2) returns that such a guarantee cannot be made (in which case $\mathcal{S}_R$ may or may not permit a solution). We note that this heuristic only checks for a sufficient condition that a pebble motion solution can be found, it does not solve the pebble motion problem. We continue increasing the size of $\mathcal{S}_R$ by in inflation factor δ until the heuristic guarantees that a solution can be found (or, in the worst case, until $\mathcal{S}_R = \mathcal{X}_R$). This heuristic checks if a rectangle $\Xi_{H,W}$ of height H and width W exists in $\mathcal{S}_R^{\text{w.s.}}$ such that $\Xi_{H,W}$ contains no obstacles, and where $\mathcal{S}_R^{\text{w.s.}}$ is the projection of $\mathcal{S}_R$ into the the team's two-dimensional workspace. We shall refer to obstacle-free rectangles in $\mathcal{S}_R^{\text{w.s.}}$ as "free space rectangles".

Requirements for H and W are explicitly proved in Section 5, and depend on robot width ρ and the clearance term ϵ (where $\epsilon > 0$ ensures a sampling-based solution will be found with probability greater than zero). The heuristic also checks that each robot in the subteam can safely reach $\Xi_{H,W}$ assuming other robots' start locations in the sub-problem are treated as obstacles and the projection of each robot i's start and goal in $\mathcal{S}_R^{\text{w.s.}}$ are surrounded by free space squares σ_i^{start} and σ_i^{goal}, respectively.

4 Algorithm

At a high level, an ad hoc distributed computer is formed over each team of conflicting robots. The joint problem considered by a particular team is constrained to a subset of the search space near to the collisions that caused that team to be formed in the first place. Each robot uses the beginning part of its original single robot path (which does not conflict with any other robots) to navigate to the conflict region. The robots within each subteam use the C-FOREST parallelization of an asymptotically optimal sampling-based motion planning algorithm to solve their mutual motion planning problem on their collective ad hoc distributed computer.

Mutual motion planning problem seeks to avoid robot-robot collisions. When sampling-based motion planning

Algorithm 1 Main control loop on robot r_j

1: **Initialize**()
2: **ListenerThread**()
3: **SenderThread**()
4: $P_j \leftarrow$ **CFOREST**$(\{r_j\}, \mathcal{X}_j, x_j^{\text{start}}, x_j^{\text{goal}})$
5: **loop**
6: **if** NewConflictsDetected **then**
7: **HoldPattern**()
8: **Reset**()
9: data $\leftarrow$ **waitForDataFromTeammates**()
10: $(\mathcal{S}_R, s_R^{\text{start}}, s_R^{\text{goal}}) \leftarrow$ **GetSubSpace**$(R, \mathcal{X}_R, \text{data})$
11: $\hat{\mathbf{P}}_R \leftarrow$ **CFOREST**$(R, \mathcal{S}_R, s_R^{\text{start}}, s_R^{\text{goal}})$
12: **if not** NewConflictsDetected **then**
13: $P_j \leftarrow$ **PathCombine**$(P_j, \hat{\mathbf{P}}_R, s_R^{\text{start}}, s_R^{\text{goal}})$
14: **moveAlong**(P_j)

algorithms are used to solve the mutual motion planning problem, then robot-robot collisions are treated as obstacles in the configuration space.

If an established team conflicts with another robot or team then they are merged into a larger team. After a team solves its mutual motion problem, and coordinates its movement through the conflict region, then the team structure dissolves along with the ad hoc distributed computer. Each robot is then free to continue along the remaining portion of its original single robot path, which is safe with respect to the now dissolved dynamic team, but may still conflict with other robots (necessitating the formation of another dynamic team).

The main control loop that runs on robot r appears in Algorithm 1; pseudocode for **CFOREST** is presented in Appendix A, while the details of lower level subroutines including **Initialize**(), **ListenerThread**(), **SenderThread**(), and **Reset**() are presented in Appendix C.

The main control loop starts by running an initialization routine and launching separate threads responsible for receiving and sending messages to other robots (lines 1-3). An initial path P_r for only robot r is calculated using the C-FOREST algorithm (line 4) before the main control loop starts (line 5). The algorithm continually checks if new conflicts are detected between robot r's current team and another robot or team (line 6); and if so, then all conflicting robots are combined into a new and larger team (see listener thread) and the new team solves its new multi-robot problem as a distributed computer using the distributed C-FOREST algorithm (lines 7-11).

Note that the team enters into a safe holding pattern (line 7) before beginning planning (this can, e.g., be accomplished using the method of Hsu et al (2002)). In addition to resetting motion planning data and getting all new data from relevant robots (lines 8-9), the team also selects a subset of the environment near the conflict region(s) in which to solve the multi robot motion planning algorithm (lines 9); this includes robot r's entry and exit point (s_R^{start} and s_R^{goal}) into the region in which the higher dimensional problem is defined. Although we technically sacrifice asymptotic optimality of the final (global) multipath by solving the team's joint problem in a subset of the environment, considering a reduced environment has the advantage of significantly reducing problem complexity. The reduced complexity makes it easier to find solutions, and often leads to better overall solutions being found in practice (additional discussion regarding optimality appears in Section 4.2).

After a solution is found (and assuming no new conflicts have be detected as the solution was being computed), then robot r calculates its new path using **PathCombine**$(P_j, \hat{\mathbf{P}}_R, s_R^{\text{start}}, s_R^{\text{goal}})$, which sandwiches[2] robot r's projection of the team's multipath between the old portions of its path to s_R^{start} and from s_R^{goal} (line 13). As long as conflicts are not detected, then robot r moves along its most recent path (line 14) while communicating with the rest of its team so that the team can adjust if any robot falls behind schedule. The subroutine **moveAlong**(P_j) on line 14 is responsible for removing robot j from its dynamic team once it has passed through the conflict area. In the most straightforward implementation, all robots necessarily reach the end of the multipath at the same time due to the fact that the team's subgoal is the Cartesian product of each member's exit point. That said, a straightforward modification allows each robot to leave the team as soon as it reaches its own exit point, provided it does not re-enter the conflict region, and its post-conflict region motion schedule is adjusted to accommodate collision checking with other robots.

Although robots sandwich their projection of the dynamic team's multipath into their single-robot paths as soon as the multipath is found by team, the team is not dissolved until robots reach the end of the conflict region. Thus, new conflicts with different robots will cause a dynamic team expansion of the original team, which is necessary to prevent cycles. Any new, larger dynamic team will replan in a subspace that results as a function of all of the team-member's single-robot paths. Because the post-sandwiched single-robot paths implicitly include the old team's multipath coordination constraints, the new dynamic team's subspace is implicitly dependent on the old team's multipath.

[2] Given that synchronization may be necessary to ensure that all robots move through the conflict zone safely; we assume that robots are able to hold position within their start-tile projection until all team members are ready. For complex hold patterns, this requires that the clearance term be chosen such that it is possible to align the hold patterns of different robots

Algorithm 2 GetSubSpace$(R, \mathcal{X}_R, \text{data})$

1: $\mathcal{S}_R \leftarrow$ "smallest hyperrectangle containing one robot width around all conflict points"
2: **while** $\mathcal{S}_R \neq \mathcal{X}_R$ **do**
3: $\quad \mathcal{S}_R \leftarrow$ increase $\mathcal{S}_R$ by a factor of δ
4: $\quad \mathcal{S}_R \leftarrow \mathcal{S}_R \cap \mathcal{X}_R$
5: $\quad s_R^{\text{start}} \leftarrow$ entry points of team into $\mathcal{S}_R$
6: $\quad s_R^{\text{goal}} \leftarrow$ exit points of team from $\mathcal{S}_R$
7: $\quad$ **if largeEnough**$(\mathcal{S}_R, |R|)$**and safe**$(s_R^{\text{start}}, s_R^{\text{goal}})$ **then**
8: $\quad\quad$ **return** $(\mathcal{S}_R, s_R^{\text{start}}, s_R^{\text{goal}})$
9: $\quad$ **return** $(\mathcal{X}_R, x_R^{\text{start}}, x_R^{\text{goal}})$

GetSubSpace$(R, \mathcal{X}_R, \text{data})$ is used to select $\mathcal{S}_R$, the subspace in which this robot's team R solves its mutual motion planning problem, as well as the substart s_R^{start} and subgoal s_R^{goal} locations that represent the transition point into and out of the subspace for the team, respectively. It is described in Algorithm 2, and works in an iterative fashion. It begins by setting $\mathcal{S}_R$ to be the smallest axis-aligned hyperrectangle that contains all collision points (that lead to the creation of the current team) increased by one robot width in all directions (line 1). This is calculated using the information stored in teamData, and then increasing the space's radius by a user-defined inflation factor δ (line 3). We ensure that $\mathcal{S}_R$ does not expanded beyond the original $\mathcal{X}_R$ (line 4). s_R^{start} and subgoal s_R^{goal} are then calculated for the new $\mathcal{S}_R$ (lines 5 and 6). Note that if a robot's overall start or goal locations are inside $\mathcal{S}_R$, and the paths from or to them never leave $\mathcal{S}_R$, then a robot's substart/subgoal is simply its overall start/goal, respectively. Algorithm 2 either returns when a safe rectangle is found (line 7-8) or when the search space has grown to be the entire space (line 9).

The heuristic **largeEnough**$(\mathcal{S}_R, |R|)$ is used to check if we can quickly predict the existence of a solution within $\mathcal{S}_R$ (inside the if-statement, line 7). It should be stressed that it *does not* itself find a solution, but provides a quick prediction about if a solution can be found (this prediction is allowed to produce a false negative but may not produce a false positive). It cannot predict all of the cases in which a solution can be found (as this would require solving the entire motion planning problem). The methods in Surynek (2009); Peasgood et al (2006); Clark et al (2003) are all possible candidates for **largeEnough**$(\mathcal{S}_R, |R|)$. However, in our implementation (with cylindrical robots that can rotate in place) we simply verify that a large enough rectangle exists to solve the $R+1$ pebble motion problem with square pebbles of width $\rho + \epsilon$ (and that each robot is capable of entering/leaving that rectangle while treating the start/goal locations' hold patterns of other robots as obstacles). This calculation can be done, for example, with the help of an occupancy grid map (as is done in our experiments on real robots), or by collision checking the sub-area versus obstacles in the environment (we use polygon collision checking in our simulations). In Section 5 we formalize the requirements of the rectangle, and prove that, in spaces with this property, a solution will be found with probability one.

The function **safe**$(s_R^{\text{start}}, s_R^{\text{goal}})$, which appears within the if-statement on line 7, verifies that the substart and subgoal locations do not force conflicts at the entry or exit points of $\mathcal{S}_R$. If the space is large enough that a solution can be found and there are no collisions at entry and exit points, or we are unable to find a smaller subset of $\mathcal{X}_R$ that is suitable, then the algorithm returns the particular space that should be used for planning (lines 7-8).

4.1 Soundness and completeness

Modern single-query sampling-based motion planning algorithms such as RRT, RRT*, and RRT-# are *probabilistically complete*[3] and *sound*[4] when used to solve the centralized multi-robot motion planning problem.

Without a communication guarantee it is impossible for any algorithm to safely coordinate (or form dynamic teams or use distributed computation). So, similar to *any* distributed multi-robot method, if robots cannot communicate quickly enough to maintain safety then the resulting multi-robot algorithm is both unsound and incomplete. That said, if we are allowed to assume robot communication is above a minimum finite threshold necessary to maintain safety, then our method inherits both the probabilistic completeness guarantees and soundness of whatever underlying sampling-based motion planning algorithm is called as a subroutine.

Assuming that the underlying sampling-based motion planning algorithm is probabilistically complete and robot communication is above a minimum threshold that maintains safety, then the method we present is probabilistically complete. This property is the result of two things: (1) The C-FOREST distributed algorithm trivially inherits the probabilistic completeness of the underlying sampling-based motion planning algorithm it parallelizes assuming any finite bounded level of communication. (2) The heuristic we use for subspace

[3] A probabilistically complete algorithm is an algorithm that will find and return a solution with probability one if a solution exists, but may run forever if a solution does not exist (in practice a timeout is usually used to indicate that a solution could not be found within an allotted planning time).

[4] A sound algorithm is an algorithm that will always return a correct solution when it returns a solution.

selection guarantees that probabilistic completeness is not sacrificed by using subspace selection.

Assuming the underlying algorithm is sound, robot communication is above a minimum threshold that maintains safety, and a solution is returned (to all robots), then the method we present is sound. Soundness is guaranteed because the resulting multipath is safe with respect to the constraints that were considered during its generation. Assuming robot communication is above a minimum threshold that maintains safety, then the same solution is returned to all robots in time to avoid collisions.

4.2 Optimality

The strongest optimality property that can be obtained with a sampling-based motion planning algorithm is *asymptotic optimality*[5]. Our use of subspace selection sacrifices any theoretical guarantees of asymptotic optimality that exist for the underlying algorithm. Indeed, the overall solution found by our algorithm is almost surely *not* asymptotically optimal (unless maximum team size is one, in which case the solution is indeed asymptotically optimal).

As a consolation, our method has the weaker property that the sub-paths between conflict regions, and the sub-multipaths within each dynamic team's conflict region *are* asymptotically optimal. We note that the concatenation of asymptotically optimal sub-paths does not yield an asymptotically optimal solution because robots/teams are unlikely to enter a new dynamic team's subspace at points along the globally optimal multipath.

Sub-path optimality is a stronger guarantee than feasibility, but not as strong as optimality. In practice, we expect the solutions found by our algorithm to be closer to optimal than those found by a an algorithm that only provides a feasibility guarantee[6]. Indeed, in the experiments presented in Section 6, our method often produces better solutions, within a finite planning time, than algorithms that are asymptotically optimal.

[5] Asymptotic optimality means that, in the limit as the number of samples approaches infinity, then the algorithm will find an optimal solution with probability one.

[6] Although, to be fair, we cannot rule out the possible existence of pathological cases for which quality might become arbitrarily worse than a feasible algorithm; however, we expect such cases to be unlikely to occur in practice (if they occur at all). And, on the other hand, it is easy to design cases where feasible algorithms are expected to yield solutions with path lengths that are arbitrarily worse than our method: for example, simply place an arbitrarily large border of free space around a continuous space position swapping problem.

5 Analysis of Subspace Selection Heuristic

Finding $\mathcal{S}_R \subseteq \mathcal{X}_R$ that permits a solution is, in the worst case, at least as hard as planning through the entire $\mathcal{X}_R$. In some cases $\mathcal{X}_R$ may be so intricate that the only way to test if a particular $\mathcal{S}_R$ permits a solution is to solve the motion planning problem in $\mathcal{S}_R$; moreover, pathological $\mathcal{X}_R$ can be designed such that any $\mathcal{S}_R \subset \mathcal{X}_R$ that permits a solution must be arbitrarily close to $\mathcal{X}_R$. Thus, we settle for using a quick heuristic test that will often *but not always* report when $\mathcal{S}_R$ permits a solution; namely, $\textbf{largeEnough}(\mathcal{S}_R, |R|)$.

We shortly prove that if $\textbf{largeEnough}(\mathcal{S}_R, |R|)$ reports that $\mathcal{S}_R$ permits a solution, then a sampling-based motion planning algorithm will find that solution with probability one.

Let $\mathcal{S}_R^{\text{w.s.}}$ denote the two-dimensional workspace projection of $\mathcal{S}_R$. We assume disc robots that can rotate in place, have a width of ρ, require a small clearance term of ϵ (to ensure a solution can be found with nonzero probability), and have the ability to stop in place. Thus, verifying that $\mathcal{S}_R$ permits a solution is reduced to solving a pebble motion problem in $\mathcal{S}_R^{\text{w.s.}}$. Similar (sufficient condition) proofs exist for more general robots, and can be derived by replacing ρ with the footprint of a safety space required to slow to a halt, turn around, or otherwise maintain movement required for safety (i.e., circling in place to maintain altitude in an aircraft). The basic idea can also be applied in higher dimensional workspaces.

Our analysis is considerably simplified by assuming that the pebbles (robots) behave like square tiles at starts, goals, and when considering robot-robot collisions. With this assumption our heuristic simply checks whether or not we can assert that all robots can access a free space rectangle $\Xi_{H,W} \subset \mathcal{S}_R^{\text{w.s.}}$ that is large enough to solve the R tile moving problem with square tiles of width $(\rho + \epsilon)$.

Let σ_i^{start} be the $(\rho + \epsilon)$ width square that contains the projecting of robot i at its start location. Let σ_k^{goal} be the $(\rho+\epsilon)$ width square that contains the projecting of robot i at its goal location. We assume that σ_i^{start} and σ_k^{goal} are open sets of space. The tile-like start and goals assumption requires that all robot sub-problem start tiles are nonintersecting and that all robot sub-problem goal tiles are nonintersecting (this is necessary to guarantee that we cannot design a pathological start or goal instance for which our heuristic declares is solvable but that is not solvable). This is overly cautious in tight environments, and can cause the heuristic to report false negatives. However, the resulting heuristic still provides sufficiency guarantees that the problem

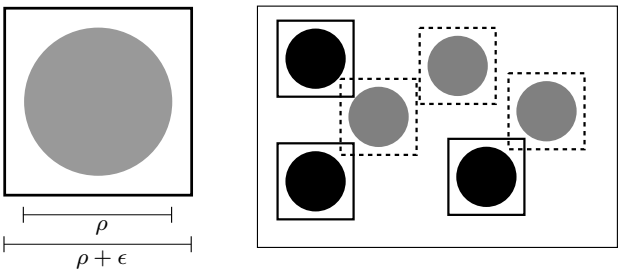


Fig. 3: Each robot is assumed to start and end in a $(\rho+\epsilon)$ by $(\rho+\epsilon)$ square (left) such that no two start tile projections (solid) intersect and no two goal tile projections (dashed) intersect (right).

is solvable in $\mathcal{S}_R$, which is all that we require for our purposes.

The heuristic **largeEnough**$(\mathcal{S}_R, |R|)$ returns true if a H by W rectangle $\Xi_{H,W}$ exists such that the following conditions hold:

1. $\Xi_{H,W}$ is large enough:
   - There exist $w, h \in \mathbb{Z}^+$ s.t. $hw \geq 2|R|$ if $|R|$ is even and $hw \geq 2|R|+1$ if $|R|$ is odd, and $W \geq (\rho+\epsilon)\max(3,w)$ and $H \geq (\rho+\epsilon)\max(3,h)$
2. A free space rectangle $\Xi_{H,W}$ exists:
   - $\Xi_{H,W} \subset \mathcal{S}_R^{\text{w.s.}}$
   - $\Xi_{H,W}$ contains only free space
3. $\Xi_{H,W}$ is accessible:
   - For all $i \in \{1,\ldots,|R|\}$ there is a single robot path to $\Xi_{H,W}$ from s_i^{start} that treats s_k^{start} for all $k \neq i$ as obstacles.
   - For all $i \in \{1,\ldots,|R|\}$ there is a single robot path from $\Xi_{H,W}$ to $s_i^{\text{goal}} \in \Xi_{H,W}$ that treats s_k^{goal} for all $k \neq i$ as obstacles.
4. Tile-like start and goals:
   - No two tile projections intersect at the start; $\sigma_i^{\text{start}} \cap \sigma_k^{\text{start}} = \emptyset$ for all $i \neq k$.
   - No two tile projections intersect at the goal; $\sigma_i^{\text{goal}} \cap \sigma_k^{\text{goal}} = \emptyset$ for all $i \neq k$ for all $i \neq k$.

We will shortly prove that a free space rectangle satisfying condition (1) is sufficient to solve the tile-moving problem. Validating condition (1) and (2) is accomplished by iteratively choosing $w = 3, \ldots \lceil\sqrt{|R|+1}\rceil$, and then calculating:

$$W = (\rho+\epsilon)\max(3,w)$$

and

$$H \geq \begin{cases} (\lceil 2|R|/w \rceil)(\rho+\epsilon) & \text{if } |R| \text{ is even} \\ (\lceil (2|R|+1)/w \rceil)(\rho+\epsilon) & \text{if } |R| \text{ is odd} \end{cases}$$

In the special case that all robots start in an axis-aligned single-file line, we require additional space to ensure that start and goal locations in the subspace can be deconflicted (this is why the '3' appears – it guarantees a buffer of at least least one robot at the edge of the subspace, which is necessary to deal with especially bad sub-problem start or goal locations).

Next, we check if either freespace rectangles $\Xi_{H,W}$ or $\Xi_{W,H}$) exist in $\mathcal{S}_R^{\text{w.s.}}$. Given a particular $\Xi_{H,W}$ an occupancy grid map can be used to check condition (2), and a simple straight line path or a simple grid-based planner can be used to verify condition (3) (Note that if start or goal locations exist in bottlenecks then the heuristic may return a false negative due to the requirement to be able to reach $\Xi_{H,W}$; we reiterate that we tolerate false negatives but not false positives). Similar methods could easily be developed to work in a triangulation or polytope representation of the environment and use more sophisticated single-robot planners to generate fewer false negatives.

Given our assumptions and conditions 1-4, we now prove that an assertion by **largeEnough**$(\mathcal{S}_R, |R|)$ is sufficient to guarantee that the problem instance in $\mathcal{S}_R$ can be solved with probability one by a sampling-based motion planning algorithm. The proof relies on three intermediate results. First, (given our assumptions and conditions 1-4) it is always possible to move the robots into a grid pattern that is h robots by w robots and located within $\Xi_{H,W}$. Next we show it is possible to re-grid the robots into any other ordering in the h by w grid without any robot leaving $\Xi_{H,W}$ (i.e., that $\Xi_{H,W}$ is sufficient to solve any instance of the $|R|$ tile moving problem). Finally, we show that some robot ordering exists over the same h by w grid such that it is possible to move robots from the grid pattern to their goal locations.

We first consider the strict tile moving game and then later relax these assumptions. The assumptions of a pure tile moving game are as follows (see Figure 3):

- All robots are square tiles of width $\rho+\epsilon$.
- All tiles start and end in $\Xi_{H,W}$; formally, $\sigma_i^{\text{start}} \subset \Xi_{H,W}$ and all $\sigma_i^{\text{goal}} \subset \Xi_{H,W}$.

We define a w by h grid embedded in $\Xi_{H,W}$ (See Figure 4). For convenience grid cells may either be identified by their two-dimensional coordinates (i,j) or by a unique grid number $iw+j$. Let the i-th row of the h by w grid be defined $\mathcal{R}_i$. Let Σ_i be the set of all robots with squares σ_i^{start} such that $\sigma_i^{\text{start}} \cap \mathcal{R}_i \neq \emptyset$. We recursively define the special row set $\hat{\Sigma}_i$ as follows:

$$\hat{\Sigma}_0 = \Sigma_0$$

$$\hat{\Sigma}_i = \Sigma_i \setminus \hat{\Sigma}_{i-1}$$

Note that $\hat{\Sigma}_i$ contains all σ_i^{start} that have their "bottom" in $\mathcal{R}_i$ bot not necessarily their "top".

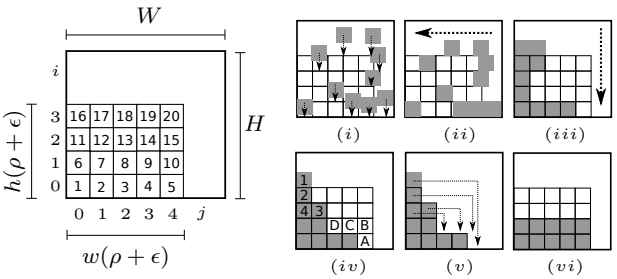


Fig. 4: Left: w by h grid embedded in freespace rectangle $\Xi_{H,W}$, grids can be labeled by position (i, j) or by number $iw + j + 1$ (shown). Right: depiction of argument used in Lemma 1; given our definition of $\Xi_{H,W}$ (from which it follows that $h, w \geq 3$ and $hw \geq |R|2$ for even $|R|$ and $hw \geq |R|2 + 1$ for odd $|R|$), it is always possible to gather the robots in the lower left corner (i-iv), and then re-position them in the $|R|$ lowest numbered grid cells (iv-vi), e.g., moving the robots labeled 1-4 into cells labeled A-D, respectively).

Lemma 1 *Assuming the tile model assumptions hold and free space $\Xi_{H,W}$ such that condition (1) is satisfied, then it is always possible to move robots (without collisions) such that they are organized in grids numbered 1 through $|R|$.*

Proof By construction we can move all robots directly downward as far as they each can go until $\hat{\Sigma}_i \subset \mathcal{R}_i$ for all $\hat{\Sigma}_i$ (See Figure 4). Next we move all robots left as far as they can each go and then down as far as they can all go. This will position all robots at the bottom left such that their are no holes in the robot mass. Finally, since there is at least $|R|$ free grids in the upper right, we can move robot one at a time until they occupy grids 1 through $|R|$

The following corollary is obtained by reversing the processes described above, and swapping start and goal locations.

Corollary 1 *Assuming the tile model assumptions hold and free space $\Xi_{H,W}$ such that condition (1) is satisfied, there always exists some labeled gridding of robots in grids numbered 1 through $|R|$ such that it is possible to move the robots (without collisions) to the labeled goal locations.*

Lemma 1 places the robots into arbitrary locations in the h by w grid, while Corollary 1 requires a particular arrangement of the robots within the h by w grid. The next two lemmas show that it is always possible to reorder robots to achieve any particular order we desire.

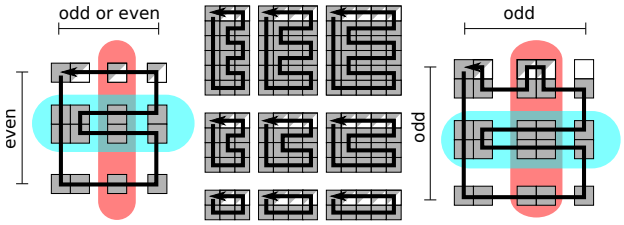


Fig. 5: For any w and h allowed by condition 1, it is possible to construct a cycle that visits each of the $|R|$ lowest numbered grid cells. Note that we only depict rows containing the $|R|$ lowest numbered cell. Also note the left case can be transposed to yield the case for odd height and even width. Rows outlined in blue can be repeated as often as required (including its sub-path) to obtain a cycle over the required number of rows. Similarly, the columns outlined in red can be repeated as often as require to obtain a cycle over w columns. Nine different examples are shown for the even row case (center). Grid cells numbered $\leq |R|$ appear gray, while the upper right grid cell in the odd-odd case is always $> |R|$ and appears white. Cells that are both gray and white may or may not have numbers in $\{1, \ldots, |R|\}$ depending on the particular $|R|$, h, and w that are used.

Lemma 2 *Assuming the tile model assumptions hold and free space $\Xi_{H,W}$ such that condition (1) is satisfied, and that $|R|$ robots have been placed into the lowest numbered grids (1 to $|R|$) in $\Xi_{H,W}$, a one-dimensional cycle exists through $\Xi_{H,W}$ such that when robots follow the cycle each robot eventually visits each grid with number less than $|R|$.*

Proof The proof is by construction. There are two cases, one each for when $|R|$ is even and odd, respectively. The cases are illustrated in Figure 5, respectively. Both cases amount to moving an empty space in the reverse direction through the cycle which causes the team to move in the forward direction.

Lemma 3 *Assuming the tile model assumptions hold and free space $\Xi_{H,W}$ such that condition (1) is satisfied, and that $|R|$ robots with unique labels (e.g., IDs 1 to $|R|$) have been placed into the 1 to $|R|$ numbered grids of an h by w robot grid pattern in $\Xi_{H,W}$, it is possible to move robots into any permutation of that labeling over the same $|R|$ numbered grids.*

Proof Without loss of generality we assume that robots start randomly permuted around the cycle described in the Lemma 2 and we want to reorder them according to increasing ID around that cycle. The proof is by induction. (Base case) we choose robot 1 as the start of the ordered cycle—in which case, it takes no more than

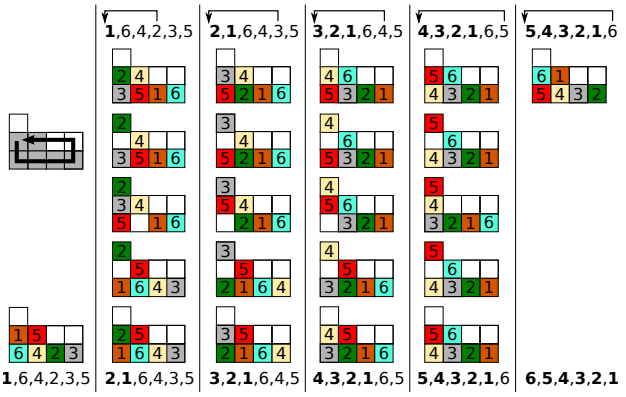


Fig. 6: Using repeated cycling and an additional grid for storage, it is possible to reorder the robots into any desired permutation around the cycle (example of the induction used in Lemma 3). In this example robots start in the order $1, 6, 4, 2, 3, 5$ around the cycle and we place them into the order $5, 6, 5, 4, 3, 2, 1$.

a single cycling of the team to place one robot in the correct location, i.e., robot 1. (Inductive Step) we assume that robots $1, \ldots, k$ have been organized in the correct order around the cycle, and we wish to place robot $k+1$ behind robot k (an example of this is illustrated in Figure 6). By Lemma 2 we know we can cycle the team until robot $k+1$ is located at position $(\lceil h/2 \rceil, 0)$. If R robots are in the the lowest $|R|$ numbered grids, then by construction grid cell $(\lceil h/2 \rceil + 1, 0)$ exists and will be empty. Thus, we can move robot $k+1$ into grid $(\lceil h/2 \rceil + 1, 0)$ which creates a free position at $(\lceil h/2 \rceil, 0)$. Next we partially cycle the team until the new free position is re-positioned behind robot k, and then cycle all robots (except robot $k+1$) and the gap until the gap is located at $(\lceil h/2 \rceil, 0)$. Finally we move robot $k+1$ into the gap.

Theorem 1 *Assuming the tile model assumptions hold and free space $\Xi_{H,W}$ such that condition (1) is satisfied, and given $|R|$ robots at (labeled) start locations $s_i^{\text{start}} \subset \Xi_{H,W}$ and (labeled) goal locations $s_i^{\text{start}} \subset \Xi_{H,W}$, then it is always possible to move all robots from start to goal.*

Proof The proof is by construction. Using Lemma 1 and 2 it is possible to move robots from (labeled) starts in $\mathcal{S}_R^{\text{w.s.}}$ to unlabeled positions in the lowest $|R|$ grids of $\Xi_{H,W}$., call the resulting permutation p_1. Similarly, using Corollary 3 we know there exists some permutation of robots p_2 within the lowest $|R|$ grids of $\Xi_{H,W}$ such that all robots can be moved to (labeled) goal locations in $\mathcal{S}_R^{\text{w.s.}}$. Finally using Lemma 3 we know it is possible to reorder robots from p_1 into p_2 without leaving $\Xi_{H,W}$.

The above Theorem assumes that robots are tiles and that all $\sigma_i^{\text{start}} \subset \Xi_{H,W}$ and $\sigma_i^{\text{goal}} \subset \Xi_{H,W}$ to begin with. These results can be relaxed to disc teams (as long as the start and goal locations still obey condition 4) and to teams with robots that do not start or end in $\Xi_{H,W}$ (as long as the pebble projection of each robot i is capable of reaching $\Xi_{H,W}$ through $\mathcal{S}_R^{\text{w.s.}}$). These extensions are made explicit in the next two lemmas.

Lemma 4 *If condition 4 holds then Theorem 1 can be relaxed such that the first tile assumption is dropped and robots are instead discs of radius $\rho + \epsilon$.*

Proof As long as start and goal locations obey the tile model (condition 4 holds), then reducing the footprint of robots from tiles to discs cannot prevent any solution from being found (since we can always move robots as if they were surrounded by a tile and still solve the problem).

Lemma 5 *Assuming $\Xi_{H,W}$ such that conditions (1), (2), (3), (4) are satisfied and all robots start in $\Xi_{H,W}$ and have projected goals in $\mathcal{S}_R^{\text{w.s.}}$, then each robot i is capable of reaching its goal.*

Proof By construction when all robots start in $\Xi_{H,W}$ we can move all robots to the lowest numbered cells using Lemma 1 and Lemma 4, then then place them, one at a time, into the highest even-numbered grids to create a checkerboard pattern. In such a checkerboard pattern, the grid cells containing robots alternate with grid cells containing only free space. Thus, some robot is able to move through any opening that is at least $\rho + \epsilon$ large, since any particular robot on the boundary can move up to $\rho + \epsilon$ before colliding with another robot. Robots can be arranged in the grid in the required order by Lemma 3, such that the robot closest to an exit is the one that needs to use it first (by construction, other robots can navigate directly to the exit after the closest robot to it has left $\Xi_{H,W}$). Thus we can move all robots that have goals entirely outside of $\Xi_{H,W}$ first, then place the remaining robots where they go (either entirely in $\Xi_{H,W}$, or intersecting the boundary of $\Xi_{H,W}$).

Reversing the process gives the following corollary.

Corollary 2 *Assuming $\Xi_{H,W}$ such that conditions (1), (2), (3), and (4) are satisfied, and that all robots end in $\Xi_{H,W}$, each robot i is capable of reaching $\Xi_{H,W}$ through $\mathcal{S}_R^{\text{w.s.}}$ from its projected start.*

Finally we prove the following theorem regarding our Heuristic.

Theorem 2 *If* **largeEnough**$(\mathcal{S}_R, |R|)$ *returns true then a sampling-based motion planning algorithm will find a solution in* $\mathcal{S}_R$ *with probability one, in the limit, as the number of samples increases.*

Proof Using Corollary 2 it is always possible to move robots from their start locations in $\mathcal{S}_R$ into $\Xi_{H,W}$, and using Lemma 5 it is also always possible to move them from $\Xi_{H,W}$ to $\mathcal{S}_R$. Thus a solution exists. The safety factor of $\epsilon > 0$ ensures that the sampling-based motion planning algorithm that we use does not need to perfectly sample any particular point to find a feasible solution. And thus, as the number of samples approaches infinity, the probability that we find a solution approaches one.

6 Experiments and Simulations

Experiments on real robots appear in Sections 6.1-6.2, and are performed using multiple Prairiedog robotic platforms (Figure 7-Left) that run the ROS operating system, and localize using the Hagisonic Stargazer, an infrared beacon-based localization system. All robots are equipped with an occupancy grid map of the environment that is used for static obstacle collision checking. Robots exchange data using IEEE 802.11g wireless in ad hoc mode. The desired speed of all robots is set to 0.2 meters per second. Motion planning is performed onboard, and multi-robot problems are solved on ad hoc distributed computers formed across the dynamic teams. The C-FOREST distributed parallelization framework (Otte and Correll, 2013b) is used on top of the asymptotically optimal Shortest Path Random Tree algorithm (presented in the Appendix of Otte (2011)). Solution lengths are measured as the sum of all individual path lengths.

Experiments in simulations appear in Section 6.2, and were coded using the Julia programming language and run on a Dell computer with an Intel i7 processor and 64 GB of memory. Simulations compare C-FOREST parallelization to OR-parallelizations and no parallelization (in no parallelization a single robot computes the solution). Each of the parallelization (and lack of parallelization) methods are tested in conjunction with the RRT, RRT*, and RRT-# sampling-based motion planning algorithms. All parallelization-algorithm combinations use dynamic teams and are evaluated both with and without subspace selection and across different levels of communication quality.

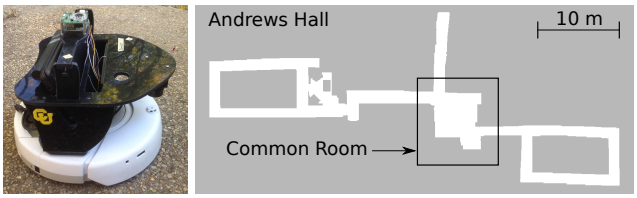


Fig. 7: Left: Prairiedog robotic platform used in experiments. Right: Occupancy grid used for static obstacle detection in the Andrews Hall and Common Room environments at the University of Colorado.

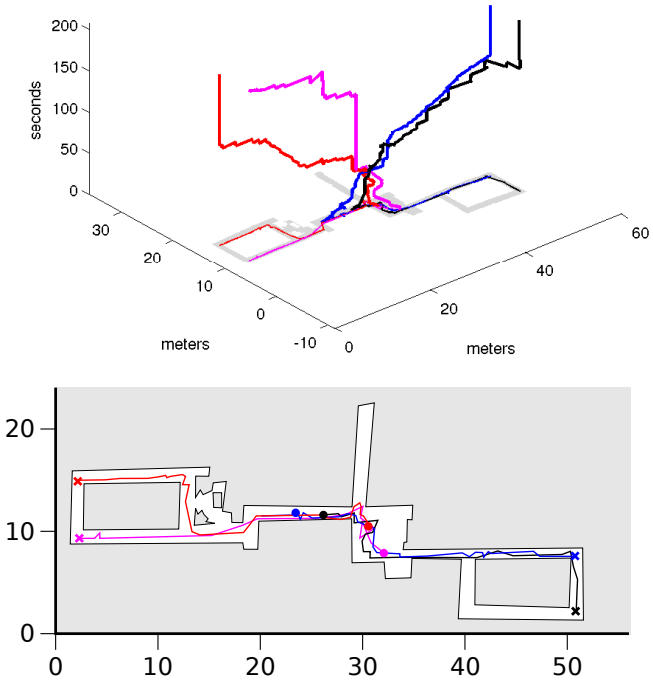


Fig. 8: Top: Actual robot paths vs. time (thick colored lines), and their projections on the free space of Andrews Hall (thin colored lines and gray, respectively). Time (measured along the z-axis) 0 corresponds to the first team formation. Bottom: projections on the free space of Andrews Hall with start and goal locations ('o's and 'x's, respectively).

Table 1: Large Andrews Hall experiment (with conflict region selection) statistics, 10 runs with 4 robots

	mean	std. dev.
In-team communication quality	63.18%	27.48%
Actual distance traveled (meters)	161.8	12.6

6.1 Large Andrews Hall experiment with subspace selection

This experiment is designed to test our conjecture from Otte and Correll (2014) that using subspace selection with a dynamic team algorithm can reduce computa-

Table 2: Six-robot experiment (with dynamic teams) statistics, 10 runs with 6 robots

	mean	std. dev.
In-team communication quality	63.81%	13.72%
Actual distance traveled (meters)	37.30	8.60

Table 3: Six-robot experiment (without dynamic teams) statistics, 10 runs with 6 robots

	mean	std. dev.
In-team communication quality	66.59%	9.21%
Actual distance traveled (meters) if an experiment was successful	64.62	18.35

tional burden versus using dynamic teams alone. That is, to test the hypothesis that planning in $\mathcal{S}_R = \mathcal{S}_{i,\ldots,k} \subseteq \mathcal{X}_R$ instead of $\mathcal{X}_R$ will improve performance in practice.

Four robots are placed as they would enter the common room/hallway in the center of the building and told to go to the corners of the opposite wing. This results in a bottle-neck condition that favors the creation of a four-robot team. Each team is allowed to plan for twice the time it takes to find an initial solution. 10 runs are performed, and all robots successfully reach their goals in every run. Figure 8 depicts robot locations versus time for a typical run. The mean and standard deviation of observed statistics over all 10 runs are displayed in Table 1.

In Otte and Correll (2014) we ran another experiment *without* subspace selection but *with* dynamic teams that provides an interesting contrast to the current experiment. The experiment in Otte and Correll (2014) is similar to the one described above in that the same environment was used (the only difference was that robots started further away from each other in the corners of the building). In the experiment in Otte and Correll (2014) all robots failed to reach the goal in all five runs, after forming a single team that was unable to solve its problem in 10 minutes (average communication quality between any two members of the same team was 60.69% with a standard deviation of 22.64%, ruling out communication failure as the reason for failure (Otte and Correll, 2014)).

6.2 Subspace selection with and without dynamic teams

In this experiment we use subspace selection and evaluate how using or not using dynamic teams affects performance. Two groups of three robots start at the two bottle-neck positions located at either end of the common room in Andrews Hall (Figure 7-Right). The robots are told to exchange places within each group. This placement is used to encourage the formation of two teams, such that each team has three robots. Each team is allowed to plan for twice the time it takes to find its initial solution.

In the first set of ten experiments the robots are programmed to use *both* dynamic teams and subspace selection. In the second set of ten experiments robots form a single team out of all robots in communication range, which then uses subspace selection to reduce the hypervolume of the six-robot configuration space.

All robots successfully reach their goals in every run when using both dynamic teams and subspace selection. Figure 9-Left depicts robot locations versus time for a typical run. The mean and standard deviation of observed statistics over all ten runs are shown in Table 2.

In contrast, when using subspace selection without dynamic teams, five of ten runs failed due to a five minute time-out (if an initial solution has not been found within five minutes of planning time). Figure 9-Right depicts robot locations versus time for a typical successful run. The mean and standard deviation of observed statistics over all ten runs are displayed in Table 3, where communication quality is over all ten runs, and distance traveled is over the three experiments where all robots reached the goal.

6.3 Simulations with various methods, obstacle clutter, and communication quality

We run repeated trials of the proposed method in conjunction with RRT, RRT* and RRT-# in simulation, across a variety of scenarios with different obstacle clutter and different communication qualities. Each method combination is run both with and without subspace selection. The use of simulation enables us to control communication quality, which is modeled using a Bernoulli distribution, and to run a large number of trials to generate statistics.

Robots start near the outside of a 50 meter by 50 meter environment at equal angles and must reach the opposite side while avoiding static obstacles and other robots. Random maps are generated containing random polygonal obstacles inside the 40 meter by 40 meter center part of the environment. A particular map is used once for each combination of parallelization technique, subspace selection (used or not used), underlying planning algorithm, and communication quality. The communication qualities evaluated include $p \in \{1.0, 0.5, 0.1, 0.05\}$, where p is the probability a message is successfully sent according to the Bernoulli model.

We define a 255-second timeout for each sub-problem. However, this is implemented as follows: dynamic teams

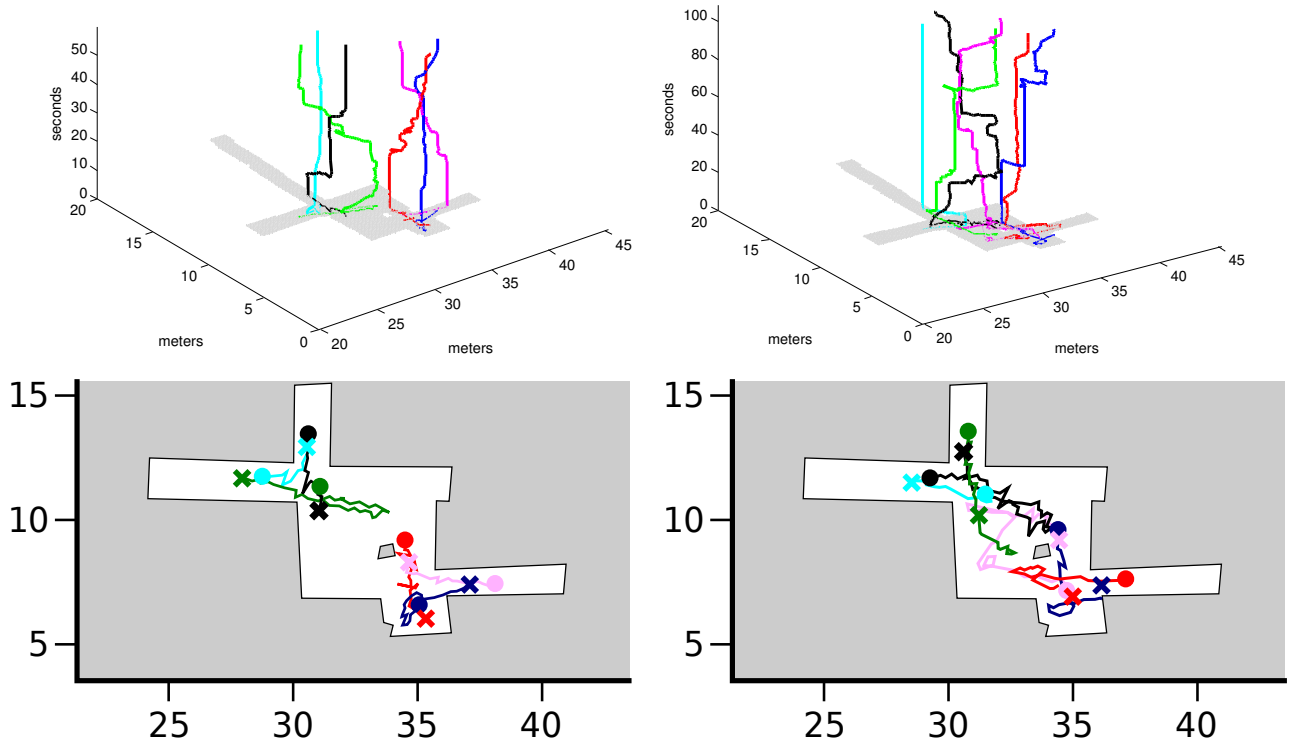


Fig. 9: Top Left: a typical solution from the six-robot experiment with subspace selection and dynamic teams. Top Right: A typical solution from the six-robot experiment without dynamic teams but still using conflict region selection. Actual robot paths versus time (thick colored lines), and their projections on the free space of the common room environment (thin colored lines and gray, respectively). Two teams of three robots each form on either side of the common room. The floor plan is included to aid visualization. Note the different scales used along the time (z) axis for left versus right. Bottom: projections on the free space of Andrews Hall with start and goal locations ('o's and 'x's, respectively).

restart from scratch if they have not found a solution in $2^{restarts}$ seconds for $restarts = 1, 2, \ldots, 7$. This idea comes from Wedge and Branicky (2008), and is used to mitigate effects of poor initial sampling. If an asymptotically optimal algorithm is used (RRT* and RRT-#), then the team continues to improve its solution as long as time remains during its current restart phase. When a feasible algorithm is used (RRT) the team stops planning as soon as the first solution is found on each robot (or at the end of the current restart if at least one robot has found a solution).

Figure 10 shows the mean summed path length (the sum of the distances traveled by all robots in the team) over all trials for the various methods. The mean value of the maximum planning time (experienced by any robot in the team) over all trials appears in figure Figure 11. All results shown use dynamic teams (we also ran experiments without dynamic teams, but none of them finished within the allotted planning time).

7 Discussion

Our experiments demonstrate that both subspace selection and dynamic teams can reduce problem complexity, thus making multi-robot motion planning problems easier to solve in practice. Indeed, our first experiment shows that using subspace selection can make the difference between being able to solve a problem within a reasonable amount of time or not — even when dynamic teams are already used to reduce problem complexity and distributed computing is used to leverage the available computational power.

Our simulations show that subspace selection improves performance across a variety of different parallelization methods (C-FOREST, OR-Parallelization, and single robot planning) and when used in conjunction with various sampling-based motion planning algorithms (RRT, RRT*, and RRT-#).

On the other hand, our experiments also show that subspace selection should be used to complement dynamic teams but not as a replacement for dynamic

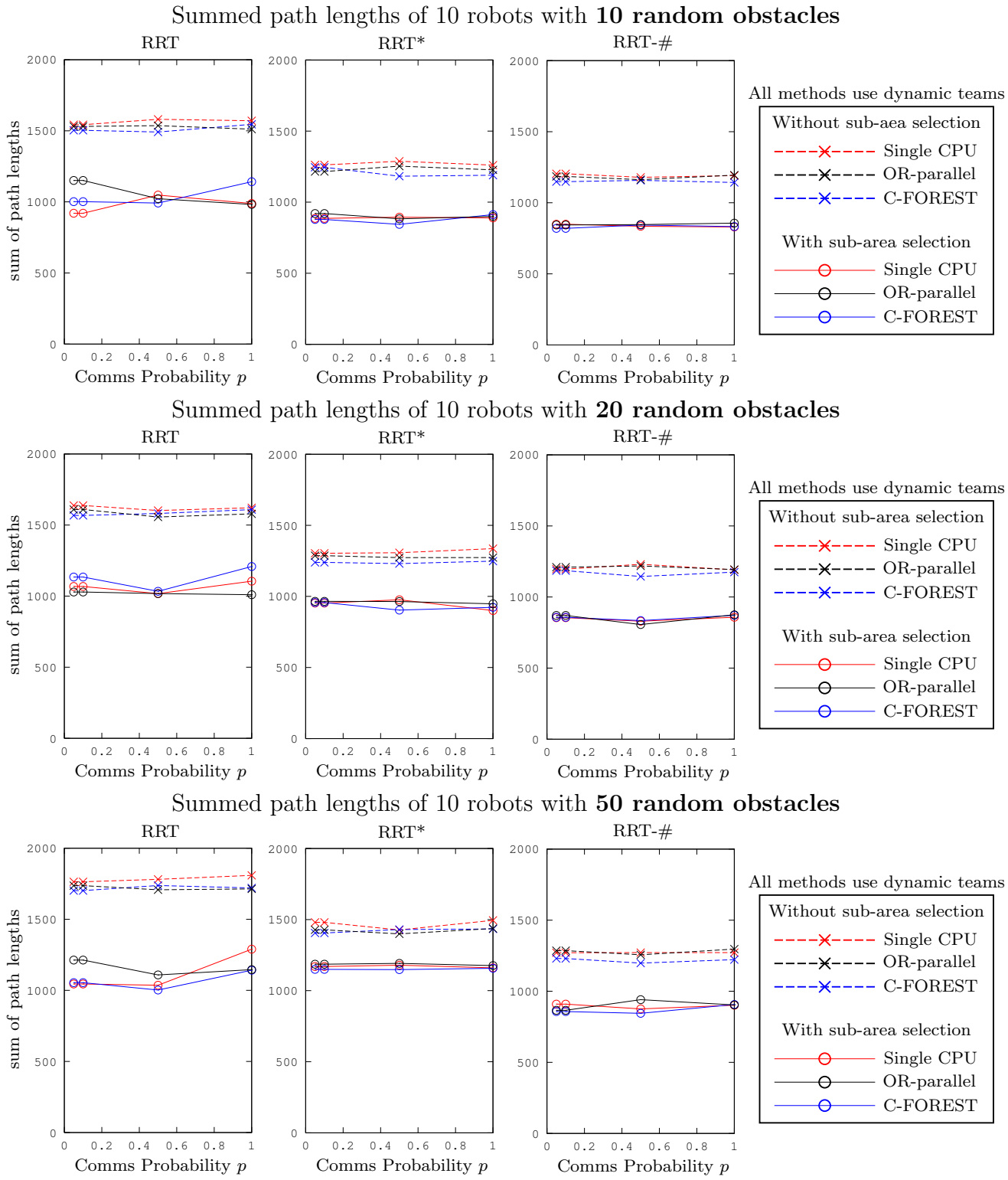


Fig. 10: The mean summed path length of all robots, averaged over 15 trials. For each trial a map is generated containing random polygonal obstacles; this map is used once for each method combination. Top to Bottom: increasing numbers of obstacles. Left to Right: different sampling-based motion planning algorithms. Colors: different parallelization methods. Solid and dashed lines represent the use of subspace selection or not, respectively. All results shown use dynamic teams (we also ran experiments without dynamic teams, but none of them finished within the allotted planning time).

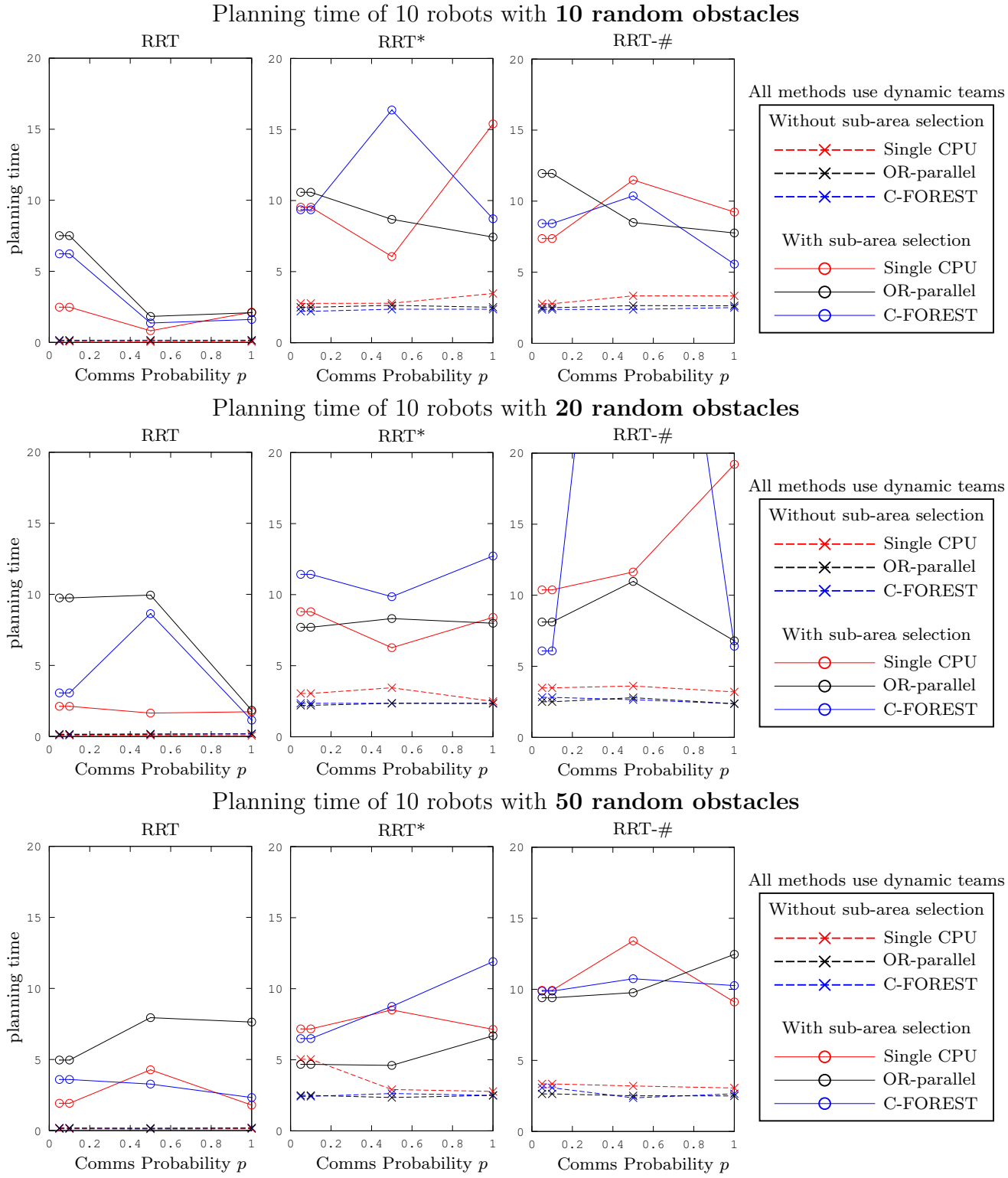


Fig. 11: The maximum planning time used by any robot in the team (summed over all path planning phases), averaged over 15 trials. For each trial a map is generated containing random polygonal obstacles; this map is used once for each method combination. Top to Bottom: increasing numbers of obstacles. Left to Right: different sampling-based motion planning algorithms. Colors: different parallelization methods. Solid and dashed lines represent the use of subspace selection or not, respectively. All results shown use dynamic teams (we also ran experiments without dynamic teams, but none of them finished within the allotted planning time).

teams. In other words, subspace selection is not a "silver bullet." This is a very intuitive result given that the complexity of the multi-robot motion planning problem is $\mathcal{O}(\|\mathcal{S}_R\|^{|R_{\max}|})$ and subspace selection reduces $\mathcal{S}_R$ while dynamic teams reduce $|R_{\max}|$.

At a high level, our work provides more evidence that teams of robots can pool their computational resources to create ad hoc distributed computers, and that the resulting ad hoc distributed computers can be used to help solve the team's mutual motion planning problem. The observed communication quality in our experiments with real robots was 65%. In simulations we tested communication rates as low as 5% and found that C-FOREST and OR-Parallel implementations both functioned well. These observations support our belief that any-com algorithms are well suited to run on a robot team's ad hoc distributed computer, and reinforce our earlier results from Otte (2011); Otte and Correll (2013a, 2014).

In Experiment 2 we observe that the multipath found by the six-robot team had individual robots wander much further away from the center of the conflict region than the combined solutions found by two teams of three robots each. This is likely due to the fact that, when more robots (and thus dimensions) exist in the configuration space, planning algorithms required more time to compute a solution that has a particular level of efficiency for all robots. This suggests that subspace selection may also provide a secondary means of reducing overall problem complexity when used with dynamic teams. Namely, by constraining replanning to small regions around conflicts, the resulting multipath solutions tend to wander less around the environment; this reduction of wandering reduces the snowballing effect that occurs when robots spread out to avoid collisions and then inadvertently cause new out-of-team conflicts that increase dynamic team size and problem complexity. We note that Experiment 2 was not designed to test this particular phenomenon, and therefore this should be considered an unproven conjecture.

It is worth mentioning that the use of dynamic teams and subspace selection will not reduce the complexity of worst-case problems, e.g., in which all robots form a single team and it is impossible to plan in any subspace smaller than the full product space over each robot's full configuration space. Finally, the heuristic subspace selection method that we use, which leverages a result from pebble motion, requires the existence of a convex region of space large enough to solve the pebble motion problem. Therefore, the heuristic will not be useful in environments that lack adequately convex subsets of free space. This may occur, for example, in highly cluttered environments.

8 Summary and Conclusions

Previous work by ourselves and others has shown the computational benefits of breaking a multi-robot motion planning problem into sub-problems based on dynamic teams, such that higher dimensional problems involving more robots must only be solved when necessary to avoid collisions. The current paper complements and extends this body of previous work by showing how heuristic subspace selection based on pebble motion provides a different and complementary mechanism to reduce problem complexity.

The subspace selection heuristic quickly checks if a subset of the environment contains a subset of obstacle-free space large enough to solve a tile moving game, where tile size is defined as a function of robot radius and maneuvering constraints. If the existence of a tile game solution can be proved, then a sampling-based motion planning algorithm will solve the multi-robot problem in the same subset of space with probability one. The heuristic does not solve the tile moving game, it just checks if enough free space exists such that the game can be solved. However, this allows us to used a much smaller configuration space when solving the multi-robot motion planning problem with a sampling-based motion planning algorithm.

We run two sets of experiments with actual robots in which conflicting robots form dynamic teams, and all robots in a team work together to solve their communal motion planning problem. The dynamic teams pool their computational resources into an ad hoc distributed computer that runs the C-FOREST distributed branch and bound framework for sampling-based motion planning. This provides additional evidence that an ad hoc computer can be formed at runtime by a robotic team using unreliable wireless communication, and also that any-com algorithms are well suited to run on such an architecture.

We find that subspace selection improves performance when used with a variety of parallelization techniques and sampling-based motion planning algorithms. Moreover, subspace selection is often necessary to solve a problem within a reasonable amount of time. Subspace selection is complimentary to, and not a replacement for, dynamic teams and distributed computation.

References

Al-Wahedi K (2000) A hybrid local-global motion planner for multi-agent coordination. Masters thesis, Case Western Reserve University

Allred J, Hasan AB, Panichsakul S, Pisano W, Gray P, Huang J, Han R, Lawrence D, Mohseni K (2007)

Sensorflock: an airborne wireless sensor network of micro-air vehicles. In: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, pp 117–129

Amstutz P, Correll N, Martinoli A (2009) Distributed boundary coverage with a team of networked miniature robots using a robust market-based algorithm. Annals of Mathematics and Artifcial Intelligence Special Issue on Coverage, Exploration, and Search 52(2-4):307–333

Arrichiello F, Das J, Heidarsson H, Pereira A, Chiaverini S, Sukhatme GS (2009) Multi-robot collaboration with range-limited communication: Experiments with two underactuated ASVs. In: International Conference on Field and Service Robots

Auletta V, Monti A, Parente M, Persiano P (1999) A linear-time algorithm for the feasibility of pebble motion on trees. Algorithmica 23(3):223–245

Best G, Cliff O, Patten T, Mettu R, Fitch R (2016) Decentralised monte carlo tree search for active perception. In: International Workshop on the Algorithmic Foundations of Robotics (WAFR), San Francisco, USA

Clark CM, Rock SM, Latombe JC (2003) Motion planning for multiple mobile robots using dynamic networks. In: Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on, IEEE, vol 3, pp 4222–4227

de Wilde B, Ter Mors AW, Witteveen C (2014) Push and rotate: a complete multi-agent pathfinding algorithm. Journal of Artificial Intelligence Research 51:443–492

van Den Berg J, Snoeyink J, Lin MC, Manocha D (2009) Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In: RSS

Desaraju VR, How JP (2012) Decentralized path planning for multi-agent teams with complex constraints. Autonomous Robots 32(4):385–403

Dixon C, Frew EW (2007) Maintaining optimal communication chains in robotic sensor networks using mobility control. In: International Conference on Robot Communication and Coordination

Elston J, Frew E, Lawrence D, Gray P, Argrow B (2009) Net-centric communication and control for a heterogeneous unmanned aircraft system. Journal of Intelligent and Robotic Systems 56(1-2):199–232

Ferguson D, Stentz A (2006) Anytime rrts. In: Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, IEEE, pp 5369–5375

Ford KM, Allen J, Suri N, Hayes PJ, Morris R (2010) Pim: A novel architecture for coordinating behavior of distributed systems. AI Magazine 31(2):9

Gammell JD, Srinivasa SS, Barfoot TD (2014) Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In: Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on, IEEE, pp 2997–3004

Goldreich O (2011) Finding the shortest move-sequence in the graph-generalized 15-puzzle is np-hard. In: Studies in Complexity and Cryptography, Springer-Verlag, pp 1–5

Holland O, Woods J, De Nardi R, Clarck A (2005) Beyond swarm intelligence: the ultraswarm. IEEE Swarm Intelligence Symposium

Hollinger G, Yerramalli S, Singh S, Mitra U, Sukhatme G (2011) Distributed coordination and data fusion for underwater search. In: IEEE International Conference on Robotics and Automation, pp 349–355

Hsieh MA, Chaimowicz L, Cowley A, Grocholsky B, Keller J, Kumar V, Taylor CJ, Endo Y, Arkin R, Jung B, Wolf DF, Sukhatme GS, MacKenzie D (2007) Adaptive teams of autonomous aerial and ground robots for situational awareness. Journal of Field Robotics 24(11):991–1014

Hsu D, Kindel R, Latombe JC, Rock S (2002) Randomized kinodynamic motion planning with moving obstacles. The International Journal of Robotics Research 21(3):233–255

Johnson M, Intlekofer Jr K, Jung H, Bradshaw JM, Allen J, Suri N, Carvalho M (2008) Coordinated operations in mixed teams of humans and robots. In: Proceedings of the First IEEE Conference on Distributed Human-Machine Systems

Johnson WW, Story WE, et al (1879) Notes on the 15 puzzle. American Journal of Mathematics 2(4):397–404

Khoo A, Horswill I (2002) An efficient coordination architecture for autonomous robot teams. In: Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on, vol 1, pp 287 – 292 vol.1

Kornhauser D, Miller G, Spirakis P (1984) Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: 25th Annual Symposium on Foundations of Computer Science, 1984., pp 241–250, DOI 10.1109/SFCS.1984.715921

Krontiris A, Luna R, Bekris KE (2013) From feasibility tests to path planners for multi-agent pathfinding. In: Sixth Annual Symposium on Combinatorial Search

Loyd S, Gardner M (1959) Mathematical Puzzles, vol 1. Courier Corporation

Nardi RD, Holland O, Woods J, Clark A (2006) Swarmav: A swarm of miniature aerial vehicles. Technical Report

Otte M (2011) Any-Com multi-robot path planning. PhD thesis, University of Colorado at Boulder

Otte M (2016) Collective cognition & sensing in robotic swarms via an emergent group mind. In: International Symposium on Experimental Robotics (ISER), Tokyo, Japan

Otte M, Correll N (2013a) Any-Com Multi-robot Path-Planning: Maximizing Collaboration for Variable Bandwidth, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 161–173. DOI 10.1007/978-3-642-32723-0_12, URL http://dx.doi.org/10.1007/978-3-642-32723-0_12

Otte M, Correll N (2013b) C-FOREST: Parallel shortest-path planning with super linear speedup. IEEE Transactions on Robotics 29:798–806

Otte M, Correll N (2014) Any-Com Multi-robot Path-Planning with Dynamic Teams: Multi-robot Coordination under Communication Constraints, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 743–757. DOI 10.1007/978-3-642-28572-1_51, URL http://dx.doi.org/10.1007/978-3-642-28572-1_51

Peasgood M, McPhee J, Clark C (2006) Complete and scalable multi-robot planning in tunnel environments. IFAC Proceedings Volumes 39(20):26–31

Ratner D, Warmuth M (1986) Finding a shortest solution for the n n extension of the 15-puzzle is intractable. In: AAAI, pp 168–172

Rutishauser S, Correll N, Martinoli A (2009) Collaborative coverage using a swarm of networked miniature robots. Robotics and Autonomous Systems 57(5):517–525

Scerri P, Owens S, Yu B, Sycara K (2007) A decentralized approach to space deconfliction. In: Information Fusion, 2007 10th International Conference on, IEEE, pp 1–8

Sharon G, Stern R, Felner A, Sturtevant NR (2012) Meta-agent conflict-based search for optimal multi-agent path finding. In: SOCS

Sharon G, Stern R, Felner A, Sturtevant NR (2015) Conflict-based search for optimal multi-agent pathfinding. Artificial Intelligence 219:40–66

Solovey K, Halperin D (2014) k-color multi-robot motion planning. The International Journal of Robotics Research 33(1):82–97

Standley T, Korf R (2011) Complete algorithms for cooperative pathfinding problems. In: IJCAI, pp 668–673

Suri N, Cabri G (2014) Adaptive, Dynamic, and Resilient Systems. Mobile Services and Systems, Taylor & Francis, URL https://books.google.com/books?id=EBzcBQAAQBAJ

Suri N, Rebeschini M, Breedy M, Carvalho M, Arguedas M (2006) Resource and service discovery in wireless ad-hoc networks with agile computing. In: Military Communications Conference, 2006. MILCOM 2006. IEEE, IEEE, pp 1–7

Suri N, Marcon M, Quitadamo R, Rebeschini M, Arguedas M, Stabellini S, Tortonesi M, Stefanelli C (2008) An adaptive and efficient peer-to-peer service-oriented architecture for manet environments with agile computing. In: Network Operations and Management Symposium Workshops, 2008, IEEE, pp 364–371

Surynek P (2009) An application of pebble motion on graphs to abstract multi-robot path planning. In: 21st IEEE International Conference on Tools with Artificial Intelligence, IEEE, pp 151–158

Surynek P (2014) Solving abstract cooperative path-finding in densely populated environments. Computational Intelligence 30(2):402–450

Sutton DJ, Klein P, Otte M, Correll N (2010) Object interaction language (oil): An intent-based language for programming self-organized sensor/actuator networks. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

Voyles R, Povilus S, Mangharam R, Li K (2010) Reconode: A reconfigurable node for heterogeneous multi-robot search and rescue. In: IEEE International Workshop on Safety, Security and Rescue Robotics

Voyles RM, Bae J, Larson AC, Ayad MA (2009) Wireless video sensor networks for sparse, resource-constrained, multi-robot teams. Intelligent Service Robotics 2(4)

Wagner G, Choset H (2015) Subdimensional expansion for multirobot path planning. Artificial Intelligence 219:1–24

Wagner G, Kang M, Choset H (2012) Probabilistic path planning for multiple robots with subdimensional expansion. In: Robotics and Automation (ICRA), 2012 IEEE International Conference on, IEEE, pp 2886–2892

Wedge NA, Branicky MS (2008) On heavy-tailed runtimes and restarts in rapidly-exploring random trees. In: AAAI Conference on Artificial Intelligence

Wilson RM (1974) Graph puzzles, homotopy, and the alternating group. Journal of Combinatorial Theory, Series B 16(1):86–96

Yu J, Rus D (2015) Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In: Algorithmic Foundations of Robotics XI, Springer, pp 729–746

Algorithm 3 CFOREST$(R, \mathcal{S}_R, s_R^{\text{start}}, s_R^{\text{goal}})$

```
1: bestLen ← ∞
2: while stopping criteria not met do
3:    s_R ← SamplePoint(S_R ∩ G(bestLen))
4:    if found better path P̂_R then
5:       bestLen ← ||P̂_R||
6:       broadcast(P̂_R)
7:    if P̂_R ← receive better path from other robot then
8:       for s_R ∈ P̂_R s.t. s_R not in tree do
9:          insert s_R in tree and do any tree maintenance
            // e.g., according to RRT-#, RRT*, etc.
10: return P̂_R
```

A Description of C-FOREST distributed branch and bound

In this section we describe the C-FOREST algorithm. C-FOREST is a distributed branch and bound technique for parallelizing sampling-based motion planning algorithms across multiple CPUs. C-FOREST is designed for use with asymptotically optimal sampling-based motion planning algorithms like RRT-# and RRT*.

C-FOREST works by having each CPU build its own random tree. Whenever a more optimal path is discovered by the tree on any CPU, then that path is broadcast to all CPUs so that it can be incorporated into the trees being built on other CPUs.

C-FOREST works for two reasons. (1) It enables "lucky" random exploration (that discovers better homotopy classes or basins of attraction) to be shared across all trees during the planning process so that all trees benefit from the good fortune of any tree. (2) It allows us to ignore portions of the space that cannot possibly lead to better solutions.

(2) is often accomplished with the help of a heuristic. For example, if the length of the geodesic from start to a new point plus the length of the geodesic from the new point to the goal is longer than the best path — then the new point need not be considered because it cannot be part of a path that is shorter than the best one already found. Given the length of the best known path, it is possible to sample directly from the portion of space that yields points shorter than the best solution. The latter heuristic was first used in Any-Time RRT (Ferguson and Stentz, 2006), and was previously used in the C-FOREST distributed branch and bound RRT* in our previous work (Otte and Correll, 2013b; Otte, 2011); more recently, it was used in conjunction with RRT* for the single-robot case in the Bit* algorithm (Gammell et al, 2014).

It is important to note that C-FOREST only provides benefits after the first solution is found (by any CPU). Thus, while C-FOREST can be used with feasible planning algorithms like RRT, it behaves like OR-parallelization for feasible planning.

For ease of presentation, in this section we assume that the robots in the dynamic team R are labeled $r_1, \ldots, r_{|R|}$ such that $R = \{r_1, \ldots, r_{|R|}\}$. Each robot r_1 grows a unique random tree through the team's combined configuration space $\mathcal{S}_R$ from s_R^{start} to s_R^{goal}.

The quantity $G(\text{bestLen})$ represents the subset of space in which $\textbf{geodesic}(s_R^{\text{start}}, s_R) + \textbf{geodesic}(s_R, s_R^{\text{start}}) < \text{bestLen}$, where $\textbf{geodesic}(A, B)$ returns the length of the geodesic from A to B.

B Messages used in low level subroutines

Robots broadcast messages to each other to communicate their intentions, to solve their centralized motion planning problems using C-FOREST distributed computation, and to coordinate their progress along the agreed upon multipaths. Let the overall message be denoted M. In our algorithm presentation we use the C/C++ language convention that sub-fields of the message data structure are denoted using the '.' token. Messages from robot r_i contain the following sub-fields, (similar quantities are also tracked locally on each robot and for brevity we omit repeating their definitions, but they appear without the 'M.' prefix in the presentation of our algorithms):

- M.epoch is the current planning epoch of r_i.
- M.data contains data about each of the j robots that r_i knows about (either directly or indirectly via other robots).
- M.path contains P_i the path r_i is using to navigate (that is, the subset between its current location and x_i^{goal})— including its time parameterization.
- M.behindSchedule is the amount of time that the sending robot's *team* is behind schedule (with respect to the time parameterization of P_i).
- M.data[k] stores data about r_j, the k-th robot r_i knows about.
  - M.data[k].id stores j
  - M.data[k].start stores x_j^{start}.
  - M.data[k].goal stores x_j^{goal}.
- $M.R$ contains a list of all robots in the current team.
- M.teamData contains additional data about each of the robots in r_i's current team.
  - M.teamData[k] stores data about the k-th robot in r_i's current team. Note that each robot places itself into position $k = 0$ when populating message data.
  - M.teamData[k].id the k-th robot's id j
  - M.teamData[k].start stores s_j^{start}.
  - M.teamData[k].goal stores s_j^{goal}.
  - M.teamData[k].epoch stores the k-th robot's epoch.
- M.bestSolution is the best solution $\hat{\mathbf{P}}_R$ (found so far) currently known to r_i for the sub-problem currently being solved.
- M.bestLen is the length of the best solution (found so far), $\|\mathbf{P}_R^*\|$.
- M.bestID is the ID of the robot that generated the best solution (found so far).
- M.agreeSet is the subset of the current team that believes $\hat{\mathbf{P}}_R$ is the best solution (found so far).
- M.finalSet is the subset of the current team that has submitted a final solution (is done planning for the current sub-problem).
- M.movingSet is the subset of the current team that has started moving.

C Description of lower level subroutines

The initialization routine **Initialize**() appears in Algorithm 4. Each robot starts in its own team and initializes its team's multipath to contain its starting location so that if other robot's paths conflict with this starting location, they will detect that a team combination should occur.

The reset routine **Reset**() is described in Algorithm 5 and run before a new motion planning sub-problem is solved;

Algorithm 4 Initialize()

1: $R = \{r_j\}$
2: $P_j = \{x_j^{\text{start}}\}$
3: epoch $= 0$
4: data[0].id $\leftarrow j$
5: data[0].start $\leftarrow x_j^{\text{start}}$
6: data[0].goal $\leftarrow x_j^{\text{goal}}$
7: **Reset**()

Algorithm 5 Reset()

1: NewConflictsDetected $\leftarrow$ **false**
2: bestSolution $\leftarrow \emptyset$
3: bestLen $\leftarrow \infty$
4: bestID $\leftarrow j$
5: agreeSet $= \emptyset$
6: finalSet $= \emptyset$
7: movingSet $= \emptyset$
8: $s_R^{\text{goal}} = \emptyset$
9: timeBehindSchedule $\leftarrow 0$

Algorithm 6 waitForDataFromTeammates()

1: **while** $\exists\ r_j \in R$ s.t. **NeedData**$(r_j, \varepsilon, \text{data})$ **do**
2: **sleep**$(1/\omega)$

Algorithm 7 ListenThread()

1: **loop**
2: $M \leftarrow$ **recieveMessage**()
3: data $\leftarrow$ **combineData**(data, M.data)
4: **if mergeAllNewConflicts**$(M, R, \text{teamData})$ **then**
5: NewConflictsDetected $\leftarrow$ **true**
6: **else if senderInTeam**(M) **then**
7: **if** M.moving **and not** moving **then**
8: **processMoveMessage**(M)
9: **else if runningCFOREST**() **then**
10: **processCFORESTMessage**(M)
11: **if** M.bestLen $= \|\hat{\mathbf{P}}_R\|$ **and** M.bestID $=$ bestID **then**
12: agreeSet $\leftarrow$ agreeSet $\cup$ M.agreeSet
13: **if not runningCFOREST**() **then**
14: finalSet $\leftarrow$ finalSet $\cup$ M.finalSet

it is responsible for re-initializing all C-FOREST related data (lines 1-4) as well as all data that is used to coordinate movement with a team (lines 5-9).

waitForDataFromTeammates() appears in algorithm 6 and causes the calling process to sleep until this robot has received data from all members of its current team.

BroadcastThread() is responsible for sending messages to other robots. It does not appear in pseudocode but, simply packages all data related to movement and planning into a message, and then broadcasts that message on an open channel (to which all robots listen). The broadcast thread is set to run no faster than ω hz for user-defined ω so that the communications channel is not saturated. If, in practice, a more sophisticated ad hoc communication protocol is used, then this can be replaced appropriately.

ListenThread() is responsible for listening for incoming messages from other robots and appears in Algorithm 7. Each time a new message is received (line 2) the data is combined with the local data on the robot to reflect the most up-to-

Algorithm 8 mergeAllNewConflicts$(M, R, \text{teamData})$

1: oldEpoch
2: **if senderInThisTeam**(M) **then**
3: **if** M.epoch $>$ epoch **then**
4: epoch $= M$.epoch
5: **else if conflict**$(M.\text{path}, P_r, \hat{\mathbf{P}}_R)$ **or** $(M.R \cap R \neq \emptyset$ **and** M.epoch $>$ epoch**) then**
6: epoch $= 1 + \max(\text{epoch}, M.\text{epoch})$
7: **if** epoch $\neq$ oldEpoch **then**
8: **for** $k = 0$ to $|M.R| - 1$ **do**
9: **if** M.teamData$[k]$.id $\notin R$ **then**
10: $R \leftarrow R \cup \{M.\text{teamData}[k].\text{id}\}$
11: teamData$[\text{end} + 1] \leftarrow M$.teamData$[k]$
12: **return true**
13: **return false**

data epoch and path information for nearby robots in general (line 3). The subroutine **mergeAllNewConflicts** that appears in the if-statement on Line 4 is responsible for detecting when a team merge operation must take place in response to new message data (and is described in Algorithm 8). The Boolean flag NewConflictsDetected is set to true if this happens (line 5). If the sending robot is already in the receiving robot's team (check on line 6), then message passing is used to synchronize the team's start of movement (lines 7-8 and 13-14), or to compute and improve the C-FOREST solution (lines 9-12). Note that agreeSet is maintained as the set of robots that agree on the best solution found so far in the C-FOREST algorithm, and finalSet is similarly maintained as the set of robots that agree on the final solution. These sets are used to ensure the team reaches consensus on a solution before moving along any solution.

mergeAllNewConflicts$(M, R, \text{teamData})$ appears in Algorithm 8 and is responsible for detecting when a robot's team must be merged with other robots and teams. The epoch count of a team is always increased to the maximum epoch of any of its robots (line 2-4) to ensure that all robots in a team are computing on the most up-to-date problem faced by that team. Two conditions (both on line 5) can cause a new problem to need to be solved (in which case the epoch number is incremented). The first is if a robot not currently in this robot's team has a conflicting path; the second is if the sending robot advertises that its own team includes a robot from the receiving robot's team and it has a higher epoch number (we assume that agents are honest, and so the sending robot must have detected that a robot in the receiving robot's team is conflicting). In the latter case we must increment the epoch to account for the unlikely event that multiple teams are merging at the same time; doing this ensures that the new problem has an epoch number larger than any problem that may have previously been solved by any of the new team's members. If the epoch number has been increased, then all relevant new robots are merged into the receiving robot's team (lines 7-11) and the algorithm returns true. Otherwise the algorithm returns false.

The subroutine **moveAlong**(P_r) is responsible for controlling robot r's movement along P_r such that team movement along their shared multipath is coordinated. It appears in Algorithm 9. We note that various coordination mechanisms could be used, the one presented here is simple to implement but may be too naive for many problems (e.g., such as those involving fixed-wing aircraft). This version works well for ground vehicles moving at low speeds. All robots broadcast how far they are behind schedule, and each member of

Algorithm 9 moveAlong(P_r)

```
1: while not NeedToReset do
2:    if RobotsAgree()then
3:       B_r = time this robot is behind schedule along P_r
4:       if B_r < behindSchedule then
5:          stop moving along path
6:       else if B_r > behindSchedule then
7:          behindSchedule = B_r
8:          continue moving along path
9:       else
10:         continue moving along path
11:   if RobotAt(s_r^goal) then
12:      R = {r}
```

Algorithm 10 RobotsAgree()

```
1: if moving or (finalSet = R) or
   (bestID = r and agreeSet = R) then
2:    moving ← true
3:    return true
4: return false
```

Algorithm 11 processMoveMessage(M)

```
1: for k = 0 to |M.teamData| − 1 do
2:    if M.teamData[k].id = r then
3:       if M.teamData[0].epoch = epoch then
4:          bestSolution ← M.bestSolution
5:          bestLen ← M.bestLen
6:          bestID ← M.bestID
7:          agreeSet ← M.agreeSet ∪ {r}
8:          finalSet ← M.finalSet ∪ {r}
9:          moving ← true
10:      return
```

the team waits, when necessary, to match the progress of the robot that is the most behind schedule.

When a team has finished computing a solution to the multi-robot planning problem then **RobotsAgree**() shown in Algorithm 10, is used to reach consensus on the particular solution that the team uses (since some robots may not yet have received the best solution found by any member of the team). This routine helps with the synchronization of the beginning of movement, which can be initiated by any robot that discovers all team members agree on the final solution (line 1) or by a robot that discovered what it thinks is the best solution—as long as all robots agreed it was the best solution at some previous point. We assume that all robots are honest and seek to use the group optimal solution. However, in rare situations this alternative mechanism for starting movement may cause the team to use a slightly outdated solution; although, it will not cause conflicts/collisions due to the fact that this robot could not possibly have agreed that some other (better) solution was the final solution if it believes its own (worse) solution is still the best. In practice this decreases consensus time by one team-wise message propagation by allowing this relaxed form of consensus to be computed in parallel to the motion planning solution (this is particularly helpful in low-communication environments). Note that if any robot discovers that one of its teammates has started moving, then it also starts moving according to the same solution (even if it previously did not believe it was the best). It is better to agree on a lower quality solution than for two different subsets of the team to move along conflicting solutions.

processMoveMessage(M) in Algorithm 11 is responsible for updating the robot's data to reflect the movement agreements of its current team. On lines 1-3 the receiving root checks to make sure that the sending robot believes they are in the same team (line 2) and that they agree on the current epoch (line 3).