2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

Constraint Manifold Subsearch for Multirobot Path Planning with Cooperative Tasks

Glenn Wagner, Jae Il Kim, Konrad Urban, Howie Choset

Abstract— **The cooperative path planning problem seeks to determine a path for a group of robots which form temporary teams to perform tasks that require multiple robots. The multi-scale effects of simultaneously coordinating many robots distributed across the workspace while also tightly coordinating robots in cooperative teams increases the difficulty of planning. This paper describes a new approach to cooperative path planning called Constraint Manifold Subsearch (CMS). CMS builds upon M*, a high performance multirobot path planning algorithm, by modifying the search space to restrict teams of robots performing a task to the constraint manifold of the task. CMS can find optimal solutions to the cooperative path planning problem, or near optimal solutions to problems involving large numbers of robots.**

I. Introduction

Many interesting problems, such as automated assembly or observation of multiple targets, involve tasks where robots may either work individually or temporarily come together to form a team. Handling the dynamic formation and dissolution of these robot teams requires solving two problems: scheduling and assigning robots to teams [10, 11, 18], and coordinating the motions of many robots, both in and out of teams. In this paper, the assignment of robots to tasks and the order in which tasks must be executed are assumed to be provided *a priori*, and the focus will be on finding high-quality, collision-free paths for large numbers of robots operating both individually and as members of teams. We term the resulting problem the *cooperative path planning* (CPP) problem.

Solving the CPP problem requires solving two qualitatively different problems: coordinating the simultaneous motion of many individual robots and teams of robots [13, 14, 23, 25, 29] and finding paths for the robots within a team that satisfy the constraints of the task being executed [2, 3, 4]. The multirobot path planning community has generated many algorithms for solving either of these problems, but not both simultaneously.

This paper presents a new algorithm for solving CPP problems called *constraint manifold subsearch* (CMS), based on the M* multirobot path planning algorithm [29]. CMS operates by temporarily merging the agents in a team into a single meta-agent whose configuration space is the *constraint manifold* that describes the robot configurations which satisfy the constraints of the corresponding cooperative task. CMS is guaranteed to find the optimal solution to the CPP problem or prove that no solution exists. Alternatively, CMS can find a solution in dramatically less time by accepting a path that may be up to a user defined constant factor more expensive than the optimal solution.

Glenn Wagner is a graduate student at the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 15213 gswagner@cmu.edu
Jae Il Kim is an undergraduate student at the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 15213 jaeilk@andrew.cmu.edu
Konrad Urban a high-school student knurban@gmail.com
Howie Choset is a professor at the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 15213 choset@cs.cmu.edu

II. Prior Work

The formation control [2, 4, 22] and cooperative manipulation [15, 17, 21, 30] communities have done significant work performing cooperative tasks with a single team of robots. Coltin and Veloso [10] and Kamio and Iba [16] have investigated problems where multiple robots independently deliver packages, exchanging packages only if the cost of delivery will be reduced or if the layout of the workspace necessitates it. However, the actual interactions between robots were brief, limited to the exchange of packages, and the planning did not account for multirobot collisions.

Ayanian and Kumar [1], Desaraju and How [12], and Bhattacharya et al. [7] developed CPP planners that could plan for systems where multiple teams form and persist for significant durations. Ayanian and Kumar [1] solved problems where robots must remain in close proximity to the other robots in its team by searching the prepares graph of controllers in a sequential composition framework [8], but this algorithm did not scale well to larger numbers of robots. Bhattacharya et al. [7] and Desaraju and How [12] both developed CPP algorithms that operate by iteratively updating the path of one robot at a time to better match the constraints imposed by the tasks. Desaraju and How [12] developed DM-RRT, a decentralized algorithm where a single robot was allowed to replan its path to better match the task constraints. DM-RRT scales well with increasing numbers of robots, but guarantees neither completeness nor optimality. Bhattacharya et al. [7] recomputed the path of the robots one at a time while gradually increasing the cost of violating task constraints. The resulting approach can be shown to eventually converge to the optimal path, but does not scale well with increasing numbers of robots.

Planning in the presence of movable obstacles, either by passing among them [26, 28] or rearranging the obstacles into a specified pattern [5] is similar to the CPP problem. Each movable obstacle can be thought of as an unpowered robot that can only move as part of a team with a powered robot. Both problems feature large numbers of interacting robots, with the concomitant high-dimensional search space. However, work on movable obstacles generally considered

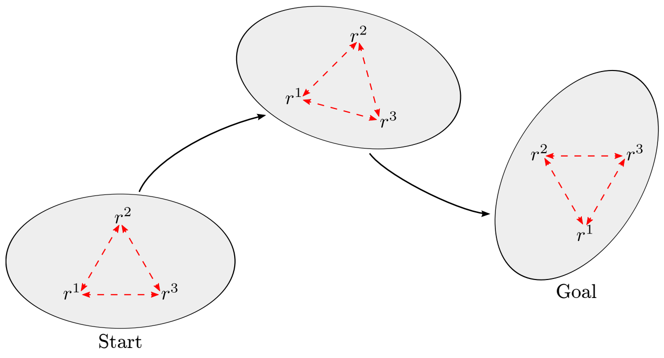


Fig. 1: A cooperative task is defined by a set of robots that will perform the task, an initial joint configuration at which the robots can start execution of the task, a joint goal configuration at which the task is considered complete, and a set of inter-robot constraints (red, dashed arrows) which must be satisfied during execution.

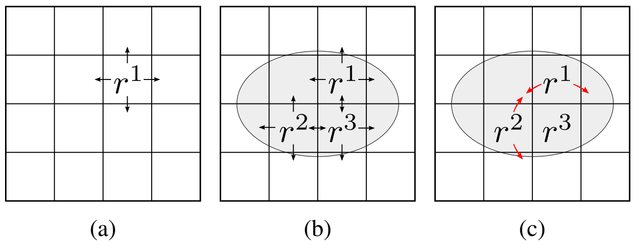


Fig. 2: Certain cooperative tasks may be incompatible with the joint configuration graph of the robots executing the task. **(a)** A regular grid is an appropriate discretization of the configuration graph of a fully-actuated planar robot. The arrows show the actions permitted by the abstraction: horizontal and vertical translation. **(b)** A team of three robots can translate a common load, represented by the ellipse, with the translation actions afforded by the discretization. **(c)** The red arrows indicate the necessary actions of the robots to rotate the load, but such actions are not afforded by a grid discretization.

only a single powered robot or team thereof, so there is only a small number of actions that could be taken at a given state. In the CPP problem, there are multiple robots and robot teams which can take independent actions, leading to a number of possible actions that grows exponentially with the number of robots.

III. Problem Definition

The objective of the CPP problem is to find a high quality, collision-free path for a set of n robots $r^i, i \in \{1, 2, \dots, n\}$ such that the robots as a whole complete a set of m cooperative tasks $\tau^j, j \in \{1, 2, \dots, m\}$. Each robot r^i has an ordered *task list* describing the order in which r^i must complete the tasks it has been assigned. Each task τ^j is represented as a tuple $\left(\mathcal{R}^j, \tau^j_{\text{start}}, \tau^j_{\text{goal}}, \mathcal{C}^j\right)$ where $\mathcal{R}^j$ is the set of robots assigned to perform the task, τ^j_{start} is the configuration of the robots in $\mathcal{R}^j$ at which the task execution can begin, τ^j_{goal} is configuration of the robots at which the task is complete, and $\mathcal{C}^j$ is a set of *task constraints* on the robots in $\mathcal{R}^j$ which must be satisfied during task execution (Figure 1). $\mathcal{R}^j$ is not a unique identifier for a set of cooperating robots, because multiple tasks may may be assigned to the same set of robots, so a *team* of robots performing a specific task τ^j is denoted γ^j.

For the purpose of planning, the configuration space of each robot is discretized. The configuration space of robot r^i is represented as the directed graph $G^i = \{V^i, E^i\}$, termed the *configuration graph*. V^i is the set of vertices in G^i which represent configurations of r^i, while E^i is the set of edges which represent valid transitions between configurations. The joint configuration space that describes the configuration of the entire system is represented by the *joint configuration graph*, the tensor product of the individual graphs.

IV. Planning on Constraint Manifolds

At its core, CMS takes an existing high-performance multirobot path planning algorithm and modifies its search space to quickly find a path that satisfies the task constraints. More specifically, CMS plans for each team in the *constraint manifold* of the associated task. The constraint manifold is the subspace of the configuration space of the constituent robots of a team that satisfy the associated task's constraints[1].

CPP can be naïvely solved by providing an existing multirobot path planner the ability to form and dissolve teams. Violations of the task constraints can then be treated simply as a robot-robot collision. For instance, if a pair of robots are carrying a rigid body and one robot moves too far away from the other, the multirobot path planning algorithm would treat that state as being invalid just as if the first robot had run into the second.

Such an approach would face two serious problems. First, the actions afforded to a team by the joint configuration graph may not be sufficient to complete its assigned task. Figure 2 shows a simple example of a joint configuration graph that only allows horizontal and vertical translations, preventing the robot team from performing a rotation. The second problem is that the constraint manifold of a task is typically of lower dimensionality than the team configuration space in which it is embedded. For instance, the constraint manifold for ten planar robots carrying a rigid body is a 3 dimensional space, rather than a 20 or 30 dimensional space. As a result, most configurations in the team configuration space will violate task constraints. Planning a path for the team will thus require expensive, coupled planning for all of the robots in the team, and is equivalent to constructing the constraint manifold by exhaustive search.

While brute force computation of the constraint manifold is computationally expensive, the constraint manifold for a number of interesting tasks can be described in closed form. For instance, the constraint manifold for a team of planar robots carrying a rigid body is isomorphic to SE(2) and can be parametrized by the position and orientation of the body. Cohen et al. [9] showed that the constraint manifold for dual-arm manipulation by a PR-2 robot is a simple 6 dimensional subspace of the full 14 dimensional configuration space. CMS exploits such exact descriptions

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

of the constraint manifold by restricting robots executing a cooperative task to the constraint manifold.

CMS restricts a team of robots to the constraint manifold of a task by replacing the robots with a single meta-agent whose configuration space is the associated task's constraint manifold. Let $\mathcal{M}^j$ be the constraint manifold associated with the task τ^j. Let $\Psi_j : \mathcal{M}^j \to Q^{\mathcal{R}^j}$ be the embedding that maps the constraint manifold to the joint configuration space of the robots in the team. $\mathcal{M}^j_{\text{start}} = \Psi_j^{-1}(\tau^j_{\text{start}})$ is the position of the team in the constraint manifold when the robots start task τ^j, and $\mathcal{M}^j_{\text{goal}} = \Psi_j^{-1}(\tau^j_{\text{goal}})$ is the position of the team in the constraint manifold where the task is completed. When the robots in $\mathcal{R}^j$ reach τ^j_{start} and begin the task, CMS replaces the agents in the team with a single meta-agent at $\mathcal{M}^j_{\text{start}}$. Planning continues with the meta-agent until it reaches the goal configuration of the task $\mathcal{M}^j_{\text{goal}}$. Once the task is completed the team is dissolved, causing the meta-agent to be replaced by individual agents representing each robot in the team. Let $G^j_{\mathcal{M}}$ denote the *manifold graph* which provides a discretized representation of $\mathcal{M}^j$ the same way G^i represents Q^i.

V. Constraint Manifold Subsearch

CMS is implemented by extending M* [29], a multirobot path planning algorithm, to solve CPP problems by adding support for dynamic formation and dissolution of teams of robots. We proceed by giving a brief description of M*, before presenting the details of CMS.

A. M*

M* implements an approach called *subdimensional expansion* [29] to efficiently explore the joint configuration graph of a multirobot system to find an optimal, collision-free path. It attempts to decouple planning between the robots in a multirobot system by planning for each robot separately. When the seperate plans conflict, M* locally performs coupled planning using A* to coordinate the robots involved.

M* is most easily described as a variant of A* that explores the search space defined by subdimensional expansion. Recall that A* maintains an *open list* of vertices to explore. The vertices in the open list are sorted by *f-value*, the sum of the cost to reach the vertex and a *heuristic function* that estimates the cost-to-go. At each iteration, the vertex with the smallest f-value in the open list is *expanded*. The neighbors of the expanded vertex are added to the open list if the cheapest path to a given neighbor passes through the expanded vertex. The process continues until the goal vertex v_f is expanded, which indicates that an optimal path to the goal has been found for the multirobot system.

M* constructs a low dimensional search space in the joint configuration graph by only adding the *limited neighbors* of a vertex to the open list when the vertex is expanded. The limited neighbors of a vertex are determined by two constructs: the *individual policy*, which defines the default action of the robots; and the *collision set*, which is the set of robots that would collide with other robots if they follow their individual policy, and thus are allowed to explore alternative actions. The individual policy ϕ^i of a robot r^i maps the position of r^i to the action that will move r^i along an optimal path to its goal, neglecting the presence of other robots. M* computes the individual policy using A*. All robots not in the collision set of a given vertex must obey their individual policy, so if the collision set of a vertex is empty there is only a single limited neighbor.

The collision set C_k for a given vertex v_k in the joint configuration graph is the set of robots r^i that would collide with another robot at some successor of v_k if restricted to their individual policies. Therefore, the robots in the collision set of a vertex are allowed to consider any possible action when the vertex is expanded. To compute the collision set, M* maintains a backpropagation set for each vertex v_k which is the set of vertices that have been expanded when their limited neighbors included v_k. When a robot-robot collision is found at v_k, the involved robots are added to the collision set of each vertex in the backpropagation set of v_k. This action is repeated recursively until the colliding robots have been added to the collision set of every predecessor of v_k. A vertex is added back to the open list whenever its collision set changes, because changing the collision set expands the limited neighbors of a vertex.

The choice of heuristic function is critical in any A*-like algorithm. M* uses the sum of the cost-to-go of the individual policies as the heuristic function. In inflated M*, the heuristic is multiplied by a user-specified $\epsilon > 1$, resulting in a path that may cost up to a factor of ϵ more than the optimal path, but which generally can be found much faster.

For a more detailed description of M*, see [29].

B. Algorithm Description

CMS makes the following modifications to M* to solve CPP problems. The joint configuration graph is replaced with the *augmented joint configuration graph*, which tracks the status of task execution/completion, and contains edges corresponding to team formation and dissolution. Secondly, multiple individual policies are computed for each robot that describe how the robot can reach the start configuration of each of its assigned tasks. Individual policies are also computed for each team of robot that describe how the team can reach the goal configuration of its task. Finally, the heuristic function and the collision set are modified to account for the formation and dissolution of teams.

The optimal behavior of a robot depends on the next task it is assigned or the task it is currently executing. Therefore, CMS performs search on the augmented joint configuration graph G^{aug}, which extends the joint configuration graph to track the status of task execution/completion. G^{aug} is composed of the union of *active graphs* and a team graph. Each active graph $G^{\text{active}}(\Gamma^{\text{active}}, \mathcal{T}^{\text{complete}})$ describes the state of the system when a set of *active* teams Γ^{active} are performing tasks, and a set of tasks $\mathcal{T}^{\text{complete}}$ have been finished. The edges of the team graph represent the formation or dissolution of teams and connect different active graphs.

$$G^{\text{aug}} = \bigcup_{(\Gamma^{\text{active}}, \mathcal{T}^{\text{complete}})} G^{\text{active}}(\Gamma^{\text{active}}, \mathcal{T}^{\text{complete}}) \bigcup G^{\text{team}}$$

The active graph for a specific $(\Gamma^{\text{active}}, \mathcal{T}^{\text{complete}})$ pair is defined as

$$G^{\text{active}}(\Gamma^{\text{active}}, \mathcal{T}^{\text{complete}}) = \prod_{\gamma^i \in \Gamma^{\text{active}}} G^i_{\mathcal{M}} \times \prod_{r^j \in \mathcal{R}^{\text{active}}} G^j$$

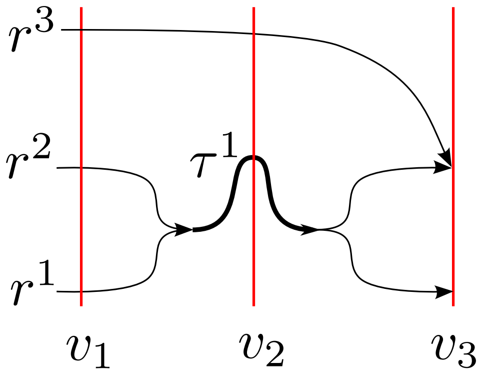
where $\mathcal{R}^{\text{active}}$ is the set of active robots that are not part of an active team. Each vertex in the $G^{\text{active}}(\Gamma^{\text{active}}, \mathcal{T}^{\text{complete}})$ is identified by the tuple $(\Gamma^{\text{active}}, \mathcal{R}^{\text{active}}, q^{\text{robot}}, q^{\text{team}}, \mathcal{T}^{\text{complete}})$, where $q^{\text{robot}} \in \prod_{r^i \in \mathcal{R}^{\text{active}}} Q^i$ is the position of the active robots, and $q^{\text{team}} \in \prod_{\gamma^i \in \Gamma^{\text{active}}} \mathcal{M}^i$ is the position of the active teams.

G^{team} represents the formation and dissolution of teams and connects different active graphs. The edges in G^{team} are formed by identifying vertices v_k in some active graph G^{active}_l at which a team γ^i is at the task goal $\mathcal{M}^i_{\text{start}}$, or the set of robots assigned to a team γ^j have reached the configuration $\mathcal{T}^j_{\text{start}}$ where the team can be formed. Copies of all out edges of v_k in G^{active}_l are added G^{team} to where the action of the team or set of robots are replaced with the dissolution or formation of a team respectively. Dissolution of a team adds the associated task to $\mathcal{T}^{\text{complete}}$ while formation of a team adds a team to Γ^{active}, so edges in G^{team} connect different active graphs in G^{aug}. Note that multiple teams can take formation or dissolution actions in a single edge.

CMS computes a separate individual policy ϕ^i_j to guide each robot r^i to the location where r^i can start each of its assigned tasks τ^j. At any given vertex of G^{aug} a robot obeys the individual policy associated with its next assigned task. Similarly, an individual policy ϕ^i is computed for each task τ^i that drives the associated team γ^i to the task's goal configuration. If all of the robots involved in a task are in position to begin execution, the individual policy for all involved robots is to take the formation action. If one or more robots are not yet in position, then the policy for the other robots is to remain in place.

A slight issue arises when all the robots in a team are in position to start the associated task, but one or more of these robots are in the collision set. To ensure completeness and optimality, CMS must explore all possible actions of the robots in the collision set. Therefore, CMS explores the team formation action, which is only valid if taken by all the robots in the team, and all other possible actions of the robots in the collision set. In the latter case, the robots not in the collision set obey their individual policy as if the robots in the collision set were not in position to form the team.

CMS uses a version of the sum of cost-to-go heuristic used by M* [29], modified to account for the cost of performing all future tasks. More specifically, let the cost of preparing a task τ^i from a vertex $v_k \in G^{\text{aug}}$ be the sum of either the costs-to-go of the individual policies of the robots $r^j \in \mathcal{R}^i$ from their current position if τ^i is the next task to be executed by r^i or the cost of following the individual policy of r^i from



$C_1 = \{r^1, r^2, r^3\} \quad C_2 = \{\tau^1, r^3\} \quad C_3 = \{r^2, r^3\}$

Fig. 3: Example of how collision sets are computed in constraint manifold subsearch (CMS). CMS encounters the first robot-robot collision at v_3 between robots r^2 and r^3, so the collision set of v_3 is $C_3 = \{r^2, r^3\}$. During the collision set update, C_3 is added to C_2, but r^2 is part of γ^1 at v_2, so γ^1 is added to C_2 instead. C_2 is then added to C_1, but γ^1 is not active at v_1, so the constituent robots of γ^1, r^1, and r^2 are added to C_1 instead.

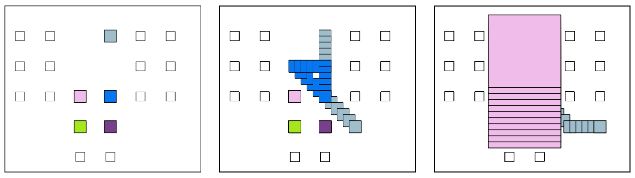
the goal of the preceeding task to the start of τ^i, if τ^i is not the next task assigned to r^i. The heuristic cost used by CMS is then the sum of the cost of preparing and executing all current and future tasks.

The final set of modifications is to how the collision sets are computed. Recall that in M*, when the collision set C_k of a vertex v_k changes the robots in C_k are added to the collision set of every vertex in the backpropagation set of v_k. In CMS, the collision set consists of a set of robots C^{robot} and a set of teams C^{team}. When a robot would be added to the collision set of a vertex in which it is not active, the team containing the robot is added to C^{team} instead. Similarly, if a team would be added to the collision set of a vertex where the team is not active, the robots composing the team are added to C^{robot}. Formally, let $C_l = \left(C^{\text{robot}}_l, C^{\text{team}}_l\right)$ be added to the collision set of a vertex $v_k = (\Gamma^{\text{active}}_k, \mathcal{R}^{\text{active}}_k, q^{\text{robot}}_k, q^{\text{team}}_k, \mathcal{T}^{\text{complete}}_k)$. The new collision set $C_{k'}$ is given by

$$\begin{aligned}
C^{\text{robot}}_{k'} =& C^{\text{robot}}_k \cup \left(C^{\text{robot}}_l \cap \mathcal{R}^{\text{active}}_k\right) \\
&\cup \left\{r^i | r^i \in \gamma^j,\ \gamma^j \in C^{\text{team}}_l \setminus \Gamma^{\text{active}}_k\right\} \\
C^{\text{team}}_{k'} =& C^{\text{team}}_k \cap \left(C^{\text{team}}_l \cap \Gamma^{\text{active}}_k\right) \\
&\cup \left\{\gamma^j | \exists\, r^i \in \gamma^j,\ r^i \in C^{\text{robot}}_l \setminus \mathcal{R}^{\text{active}}_k\right\}
\end{aligned}$$

See Fig 3 for an example.

Note that a collision between two single robots could eventually result in every robot in the system being added to the collision set of some predecessor state, especially if there are many team formation and dissolution events. In such an event, finding an optimal solution would be very computationally expensive. Inflating the heuristic function à la inflated M* biases search towards the final state of the system, which provides a soft limit on how far back in the search CMS will look for an alternate path around collisions, limiting the effective size of the collision set, at the cost of



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

Fig. 4: Small white squares are obstacles, while colored squares represent individual robots, and large squares represent teams. **(a)** The four robots that start at the bottom must carry a large square object to the top of the corridor. **(b)** However, the robots must delay starting the task to allow the single robot that starts at other end of the corridor to pass. **(c)** Once the single robot is clear, the task can be completed.

finding a path of bounded suboptimality.

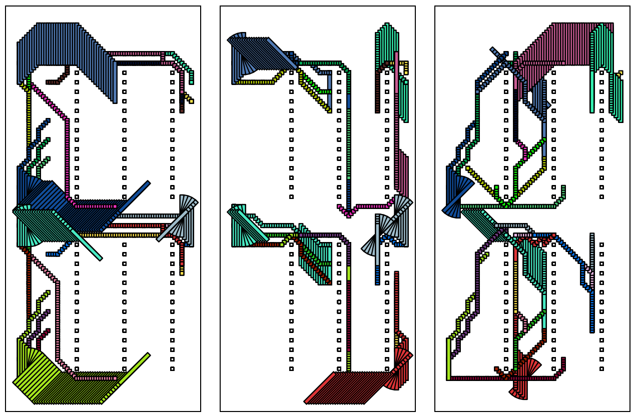
In practice, CMS uses Operator Decomposition M* (ODM*) [14, 29], a variant of M* which replaces A* with Operator Decomposition (OD) [25], and which differs from M* only in implementation details. The proofs of optimality and completeness of M* [29] apply to CMS with minimal modification. Like M*, the worst case complexity of CMS grows exponentially with the number of robots, which is to be expected as optimal multirobot path planning on graphs is known to be NP-complete [20].

VI. Results

We validate the performance of CMS in simulation. The cooperative tasks take the form of moving large, rigid, rectangular loads, which are present in the workspace only while being carried by robots. The load must be prevented from contacting obstacles or non-carrying robots. The constraint manifolds corresponding to the tasks are isomorphic to SE(2). Individual robots move on an 8-connected grid where diagonal movements cost $\sqrt{2}$. A robot may wait for zero cost at its final destination or at the start configuration of its next task if the other robots assigned to the task are not yet in position. All other actions carry cost one. Teams move on a similar grid with the cost of a given action multiplied by the number of robots in the team. Teams can also take an action to rotate by $\pm 45°$ at cost equal to the number of constituent robots.

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

In the first simulation, four robots start at one end of a corridor opposite a single robot (Figure 4a). The four robots must carry a large, square object to the top of the corridor. However, while the team is carrying the load it cannot move out of the way of the single robot, which must reach the bottom of the environment. Therefore, the robots must delay executing the task until the single robot has cleared the corridor (Figure 4b). Once the single robot has cleared the corridor, the task can be successfully executed (Figure 4c). CMS required 0.3 seconds to compute the optimal solution to this problem.



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

Fig. 5: Small white squares are obstacles, while colored squares represent individual robots and rectangles represent teams. Eight teams of three robots each must pick up rectangular loads from depots on the periphery, and deliver them to drop points between the rows of obstacles. Each team must make two such deliveries. Path segments are shown for three time windows, where the entire path is 121 units in duration. Please consult the uploaded video for a better depiction of the path.

The second simulation consists of eight teams of three robots each, that must pick up long rectangular loads from depots on the periphery of the workspace, and deliver them to positions between rows of obstacles (Figure 5). Each team must deliver two loads before returning to their initial positions. To compute a solution in reasonable time, an inflation factor of 1.2 was used. The only obvious suboptimality is one instance when a team is blocked by a single robot for a brief period of time. The blockage occurs because resolving the interaction required a heuristically unfavorable action, whose consideration was delayed by the inflated heuristic. CMS took 77 seconds to compute the solution.

To investigate how CMS scales with the number of robots and tasks, we generated randomized trials in the same environment as the second simulation. Each trial consisted of between one and eight teams, each consisting of three robots. Each team was randomly assigned a pick up location, and depending upon the trial, 1, 2, or 3 load-carrying tasks with randomly assigned drop-off locations. 100 trials were generated for every number of team and tasks. CMS was run with an inflation factor of 1.2 for a maximum of five minutes, at which point failure was declared if no solution had been found. The success rate and median time required to find a solution are shown in Figure 6. The median time to find a solution grows sub-exponentially for up to 5 teams, at which point the time to find a solution grows dramatically for trials involving two or three tasks.

For any set of parameters, the problems that took longest to solve featured interactions between large numbers of

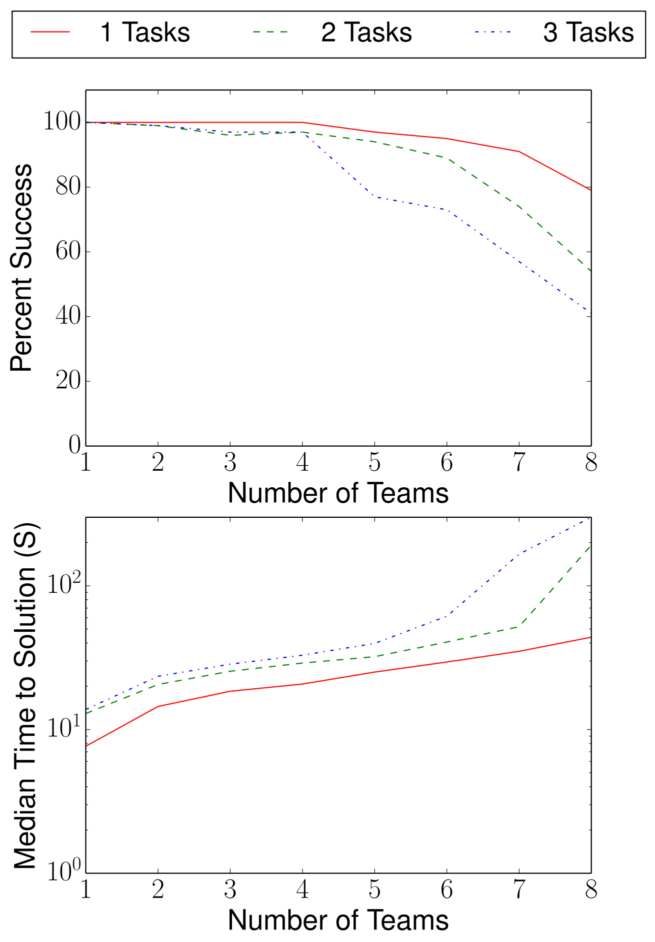


Fig. 6: Success rate (left) and median time to find a path (right) for randomized trials of CMS. Trials consisted of between one and eight teams, each consisting of three robots. Each team was randomly assigned a pick up location, and between one and three drop-off locations (tasks)

robots or teams moving in opposite directions. Large interactions force a large collision set, and thus search in a high-dimensional space. Groups of robots moving in opposite directions require sustained, expensive coordination while they maneuver around one another. If the robots are moving in the same direction then a short delay is often enough to prevent further collisions as follow-the-leader behavior then emerges. Predicting whether a given problem will be hard or easy is difficult, as the presence and geometry of robot-robot interactions depends on exactly when robots pass through a given region. However, less bi-directional traffic through constrained bottlenecks generally leads to shorter computation times.

VII. Conclusion and Future Work

We present the Constraint Manifold Subsearch, a new algorithm for solving the cooperative path planning problem. CMS can compute optimal or bounded suboptimal paths for systems with large numbers of robots that can perform cooperative tasks.

As currently implemented, CMS couples planning for all robots in the collision set, even if the robots in question form multiple physically separated and disjoint sets. To resolve this problem, we will adapt recursive M* [29], which plans separately for each disjoint subset of colliding robots, for use with CMS. In addition to reducing computation time, we believe that using recursive M* will prevent robot teams from getting "stuck" on individual robots as observed in the second simulation trial.

CMS has been implemented for only a single cooperative task: multiple mobile robots carrying a single object. In the future, we will pursue techniques for automatically constructing the constraint manifold [6, 9, 24, 27] to apply CMS to more types of tasks. Of special interest is the task of cooperatively carrying an object through an environment that is sufficiently cluttered that robots must disconnect and reconnect to the object clear obstacles [19].

References

[1] Nora Ayanian and Vijay Kumar. Decentralized feedback controllers for multi-agent teams in environments with obstacles. *IEEE Transactions on Robotics*, 26(5): 878–887, October 2010.

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

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

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

[5] Ohad Ben-Shahar and Ehud Rivlin. Practical pushing planning for rearrangement tasks. *IEEE Transactions on Robotics and Automation*, 14(4):549–565, 1998.

[6] Dmitry Berenson, Siddhartha S. Srinivasa, Dave Ferguson, and James J. Kuffner. Manipulation planning on constraint manifolds. In *IEEE International Conference on Robotics and Automation*, volume i, pages 625–632. Ieee, May 2009.

[7] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Multi-agent path planning with multiple tasks and distance constraints. In *IEEE International Conference on Robotics and Automation*, pages 953–959. Ieee, May 2010.

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

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

[10] Brian Coltin and Manuela Veloso. Optimizing for transfers in a multi-vehicle collection and delivery problem. In *Proceedings of Distributed Autonomous Robotic Systems*, 2012.

[11] Brian Coltin and Manuela Veloso. Online Pickup and Delivery Planning with Transfers for Mobile Robots. *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 8–13, 2013.

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

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

[14] Cornelia Ferner, Glenn Wagner, and Howie Choset. ODrM* optimal multirobot path planning in low dimensional search spaces. *2013 IEEE International Conference on Robotics and Automation*, pages 3854–3859, May 2013.

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

[16] Shotaro Kamio and Hitoshi Iba. Random sampling algorithm for multi-agent cooperation planning. *IEEE International Conference on Intelligent Robots and Systems*, pages 1265–1270, 2005.

[17] Y. Koga and Jean-Claude Latombe. On multi-arm manipulation planning. In *IEEE International Conference on Robotics and Automation*, pages 945–952, San Diego, CA, 1994. IEEE Comput. Soc. Press.

[18] Somchaya Liemhetcharat and Manuela Veloso. Synergy graphs for configuring robot team members. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 111–118, 2013.

[19] Laura Lindzey, Ross Alan Knepper, Howie Choset, and Siddhartha Srinivasa. The Feasible Transition Graph: Encoding Topology and Manipulation Constraints for Multirobot Push-Planning. In *The Eleventh International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, page 16, 2014.

[20] Daniel Ratner and Manfre Warmuth. Finding a Shortest Solution for the NxN Extension of the 15-PUZZLE is Intractable. In *AAAI Conference on Artificial Intelligence*, pages 168–172, Philadelphia, PA, USA, 1986.

[21] Daniela Rus, Bruce Donald, and Jim Jennings. Moving Furniture with Teams of Autonomous Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 235–242, Pittsburgh, PA, 1995.

[22] RO Saber, W. B. Dunbar, and Richard M. Murray. Cooperative control of multi-vehicle systems using cost graphs and optimization. In *Proceedings of the American Control Conference*, pages 2–7, 2003.

[23] Guni Sharon, Roni Stern, Ariel Felner, and Nathan Sturtevant. Meta-Agent Conflict-Based Search For Optimal Multi-Agent Path Finding. *Proceedings of the Fifth Annual Symposium on Combinatorial Search*, pages 97–104, 2012.

[24] Thierry Siméon, Juan Cortés, Anis Sahbani, and Jean Paul Laumond. A general manipulation task planner. In *Algorithmic Foundations of Robotics V*, volume 7 of *Springer Tracts in Advanced Robotics*, pages 311–327. Springer Berlin Heidelberg, 2004.

[25] Trevor Standley. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[26] Mike Stilman and James Kuffner. Planning Among Movable Obstacles with Artificial Constraints. *The International Journal of Robotics Research*, 27(11-12): 1295–1307, 2008.

[27] I A Sucan and L E Kavraki. On the advantages of task motion multigraphs for efficient mobile manipulation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4621–4626. IEEE, 2011.

[28] Jur van Den Berg, Mike Stilman, James Kuffner, Ming Lin, and Dinesh Manocha. Path planning among movable obstacles: A probabilistically complete approach. In *Algorithmic Foundation of Robotics VIII*, volume 57 of *Springer Tracts in Advanced Robotics*, pages 599–614. Springer Berlin Heidelberg, 2010.

[29] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219(0):1–24, 2015.

[30] Atsushi Yamashita, Tamio Arai, Jun Ota, and Hajime Asama. Motion planning of multiple mobile robots for cooperative manipulation and transportation. *IEEE Transactions on Robotics and Automation*, 19(2):223–237, April 2003.