2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

Efficient Formation Path Planning on Large Graphs

Max Katsev Jingjin Yu Steven M. LaValle

***Abstract*—For the task of transferring a group of robots from one formation to another on a connected graph with unit edge lengths, we provide an efficient hierarchical algorithm that can complete goal assignment and path planning for 10,000 robots on a 250,000 vertex grid in under one second. In the extreme, our algorithm can handle up to one million robots on a grid with one billion vertices in approximately 30 minutes. Perhaps more importantly, we prove that with high probability, the algorithm supplies paths with total distance within a constant multiple of the optimal total distance. Furthermore, our hierarchical method also allows these paths to be scheduled with a tight completion time guarantee. In practice, our implementation yields a total path distance less than two times of the true optimum and a much shorter completion time.**

I. Introduction

Controlling the formation of a group of robots with optimality guarantees has many practical applications. For example, in deploying a mobile sensor network, it may often be necessary to move the indistinguishable mobile sensors from one formation to another one to accommodate changes such as sensor outages. Doing so economically is then important to extend the operational efficiency and online time of the network. Although finding paths with minimum total distance can be achieved with the Hungarian algorithm [11] in $O(n^3)$ time with n being the number of robots, it is not clear how long it will take to schedule these paths so that no collision will occur between the robots. Recently, it was proven that such paths, when restricted on a graph network with unit edge lengths, can be scheduled with time $n+\ell-1$ (in which ℓ is the largest of the distances between all pairing of start/goal formation locations), which is a tight time bound [27]. Furthermore, it was shown that a minimum time constraint and a minimum distance constraint cannot be satisfied simultaneously in general [27].

Using a fully centralized approach, an $O(n^3 + nE)$ algorithm was given in [27], in which E is the number of edges of the graph. Although it is sufficient for medium size problems (it takes about 15 minutes to plan the paths for 5000 robots on a 4×10^6 vertex graph using a high-end six-core CPU), the centralized algorithm starts to slow down significantly on larger instances. In this paper, we address this problem by proposing a hierarchical method that partitions the graph into subgraphs. In a two-level approach, at the higher level, a network flow method is employed to distribute the robots so that the number of start and goal locations are balanced in every smaller subgraph. For the second stage, the algorithm from [27] is adapted to plan the paths on these subgraphs. A multi-level hierarchy can also be used. Our main theoretical contribution is showing that with high probability, the two-level hierarchical method plans paths with a total distance within a constant factor of the true optimal distance. In evaluation, our implementation generally yields a total path of length no more than double the true optimal. Moreover, the robots can be scheduled within the same time bound of $n + \ell - 1$. Our implementation, using a different scheduling algorithm to speed up the process, produces on average a schedule much better than this theoretical time bound. As an end result, in approximately 30 minutes, our algorithm can compute near distance optimal paths for one million robots on a one billion vertex graph.

The problem of formation path planning, despite its unique applications, can be viewed as a specialized version of the multi-robot path planning problem, which has remained an actively studied problem for many decades [8], [14], [16], [18], [19], [20], [23], [24], [25], [26], [29]. For more information on this general problem, see [4], [12], [13] and the references therein. When the workspace is continuous, the formation path planning problem, called *permutation invariant multi-robot path planning* in [10], is represented using a single polynomial of which the roots correspond to the unassigned configurations for the robots in the formation. Recently, a dynamic graph building approach was employed to solve the continuous multi-robot path planning problem allowing various levels of robot distinguishability [21]. A partition-based approach to the *multi-robot task allocation* problem is considered in [15].

An alternative approach to solving the continuous space problem is to build a discrete graph (roadmap) over the continuous workspace and then solve the discrete problem. Unless the workspace is very tight, the approach is often effective. Such formulations have been studied as network flow problems, for which many efficient algorithms are available [3], [9]. These algorithms, however, do not directly carry over to multi-robot formation path planning since the focus of network flow is usually on transporting goods over a road network in which the vertices of a graph represent cities. These "cities" are usually assumed to be able to hold an infinite amount of goods. On the other hand, we generally do not want multiple robots occupy the same vertex. Nevertheless, with some effort, it is possible to apply network flow algorithms to solve the formation path planning over many optimality objectives [28].

The rest of the paper is organized as follows. In Section II we introduce a formal description of the problem we are

Max Katsev and Steven M. LaValle are with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA. E-mail: katsev1@uiuc.edu, lavalle@uiuc.edu.

Jingjin Yu is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA. E-mail: jyu18@uiuc.edu.

This work is supported in part by NSF grant 0904501 (IIS Robotics), NSF grant 1035345 (CNS Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

An extended version of the paper, with complete proofs, is available at http://msl.cs.uiuc.edu/∼katsev1/.

considering. Section III describes the algorithm that is the main contribution of this paper and proves its correctness. Next, we analyze the algorithm's running time and the quality of the generated solutions in Section IV. In Section V we provide experimental data and compare our results to the optimal solutions. We conclude in Section VI.

II. Modeling Formation Path Planning on Graphs

A. Formation Path Planning with Collision Prevention

Let $G = (V, E)$ be a connected, undirected, simple graph (i.e., no multi-edges) with unit length edges. Let $X^I, X^G \subset V$ be sets of initial and goal locations on G.

Consider a robot moving along the edges of G at unit speed. We call a function $p\colon \mathbb{N} \to V$ a *scheduled path* from an *initial location* $u \in X^I$ to a *goal location* $v \in X^G$ if $p(0) = u$, there exists $T \in \mathbb{N}$ such that for all $t \geq T$ $p(t) = v$, and for all $0 \leq t < T$ either $(p(t), p(t+1)) \in E$ or $p(t) = p(t+1)$. The smallest T that satisfies this definition is called the *completion time*. Note that p induces a regular path on G if we ignore specific times when a robot visits each vertex. The *length* of p is simply the length of this induced path.

A set of n scheduled paths P is called *collision-free* if for any two paths $p_i, p_j \in P$ and for any time $t \in \mathbb{N}$, neither $p_i(t) = p_j(t)$ ("meet" type collision), nor $p_i(t) = p_j(t+1)$ and $p_i(t+1) = p_j(t)$ ("head-on" type collision).

Problem 1 (Formation Path Planning on Graphs)**.** *Given a* 3*-tuple* (G, X_I, X_G)*, find a collision-free set of* $n = |X^I| = |X^G|$ *scheduled paths* P *such that every initial and goal location is used exactly once.*

We have two metrics to evaluate the quality of a solution to Problem 1. One is *total distance traveled*, which is simply a sum of the lengths of all paths. Another is *overall completion time* or *makespan*, which is the maximum completion time among all robots. As shown in [27], it is impossible in general to satisfy both time and distance optimality simultaneously. In this paper, our main goal is to minimize the total distance, but we pay attention to the completion time as well.

B. An Example

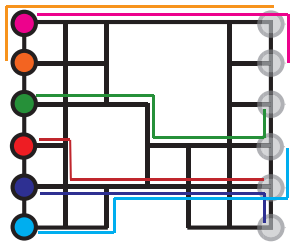


Fig. 1. A 6×7 grid with some vertices removed. The colored discs on the left represent the initial formation and the gray discs represent the goal formation. The colored paths represent the paths (not yet scheduled to avoid collision).

To better characterize Problem 1 and its solution, look at the example in Fig. 1. For the 6×7 grid with holes, we assign the top left corner coordinates $(0, 0)$ and bottom right coordinates $(6, 5)$. There are six robots with $X^I = \{(0, i-1)\}$, $X^G = \{(6, i-1)\}$, $i = 1 \ldots 6$. That is, we want to move the robots from the leftmost column to the rightmost one. A distance-optimal solution to this problem is given in Table I, corresponding to a schedule of the multi-colored paths in Fig. 1. Each main entry of the table designates the coordinates a robot should be at the given time step. The overall completion time of this solution is 8.

TABLE I

Robot	Time Step								
	0	1	2	3	4	5	6	7	8
1	0,0	1,0	2,0	3,0	4,0	5,0	6,0	6,1	6,1
2	0,1	0,0	1,0	2,0	3,0	4,0	5,0	6,0	6,0
3	0,2	1,2	2,2	3,2	3,3	4,3	5,3	6,3	6,2
4	0,3	1,3	1,4	1,4	2,4	3,4	4,4	5,4	6,4
5	0,4	1,4	2,4	3,4	4,4	5,4	6,4	6,5	6,5
6	0,5	1,5	2,5	2,4	3,4	4,4	5,4	6,4	6,3

III. Partition-Based Algorithm for Formation Path Planning[1]

A. An Exact Algorithm for the Distance-Optimal Planning Problem

We start with a brief description of Algorithm 1 [27] that finds a distance-optimal solution to Problem 1.

The algorithm consists of two parts. In the first part, the shortest paths between all pairs of initial and goal locations are found using breadth-first search, and each robot is assigned a goal location using the Hungarian algorithm [11]. The second part involves modifying these paths to avoid collisions (if necessary) and scheduling the robots' starting times.

Algorithm 1 ExactPlanner

Input: Graph G, sets of initial and goal locations X^I, X^G
Output: Set of scheduled paths on G

```
for all u ∈ X^I, v ∈ X^G do
    obtain a shortest path p_ij between u_i, v_j
pick the optimal subset P of {p_ij} that covers all locations
update P according to Lemma 7 of [27]
for all p ∈ P do
    schedule p to start at start(p) = t
    t = t + 1
```

As shown in [27], Algorithm 1 finds a solution to Problem 1, which minimizes the total distance traveled by all robots and has a bounded completion time. Additionally, the schedule generated by the algorithm has several properties that we will use later:

(P1) after the robots start moving, they never stop until they reach their corresponding goal locations;

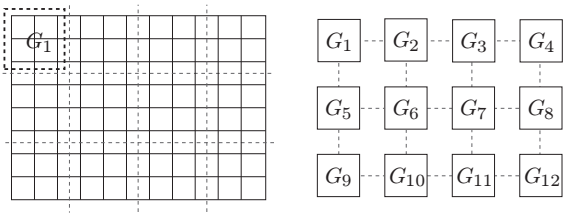
(P2) certain kinds of delays can be introduced into the schedule while still maintaining the collision-free property; more specifically, for any time t, all robots that are scheduled to start after t can be delayed by the same amount of time Δt;

[1]Although the algorithms described here operate on grid graphs, they can be extended to work for more general classes of planar graphs.

(P3) robots arrive at their goal locations in the same order that they have started moving, except for the cases when the goal location is blocking a path of another robot.

B. The Two-Level Approach to the Planning Problem

The basic idea behind our algorithm is to divide a large graph into smaller ones to avoid computing all-pairs shortest paths on the large graph. This partition creates a two-level structure: a collection of smaller subproblems and a high-level partition graph that encodes the way smaller graphs are connected. An example partition of a 9×12 grid into 3×3 grids is given in Fig. 2.



(a) A 9×12 grid with superimposed partition lines. (b) Resulting high level 3×4 structure.

Fig. 2. A partition of a 9×12 grid graph.

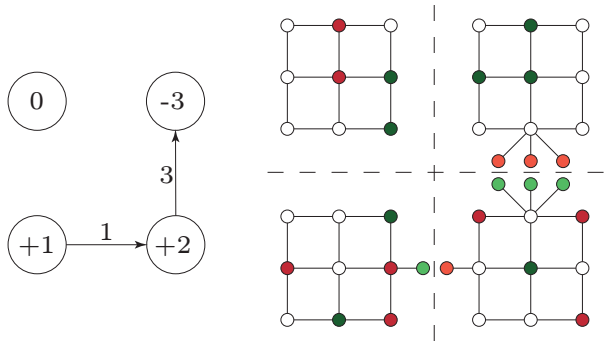
We will use $G_i = (V_i, E_i)$, $i = 1, \ldots, k$, to denote k subgraphs created by the partition, and Γ for the high-level graph. This process also induces a split on X^I and X^G. Define $X_i^I := X^I \cap V_i$ and $X_i^G := X^G \cap V_i$ to be sets of initial and goal locations that belong to the corresponding subgraphs. Finally, let $n_i^I = \left|X_i^I\right|$ and $n_i^G = \left|X_i^G\right|$.

For a given G_i, if $n_i^I = n_i^G$, we can use ExactPlanner to solve the subproblem exactly. On the other hand, if $n_i^I \neq n_i^G$, it means that this subproblem is unbalanced and some robots have to cross the partition lines. For example, if $n_i^I > n_i^G$, we need at least $n_i^I - n_i^G$ robots that start in G_i, but eventually leave it for the goal locations in some other subgraphs.

In order to minimize the distance traveled by such robots, we consider a flow network on Γ. In this network, vertices that correspond to subproblems with $n_i^I > n_i^G$ ($n_i^I < n_i^G$) become sources (sinks) with supply (demand) equal to $\left|n_i^I - n_i^G\right|$, all edge capacities are unlimited, and edge costs are equal to 1 (varying edge costs can be used to accommodate non-uniform partitions). The solution of the min-cost flow problem [1] determines how the robots need to travel between subgraphs.

To balance the subproblems, we add artificial initial and goal locations, which simulate robots moving between subgraphs. For any directed edge (G_i, G_j) of Γ with positive flow f_{ij}, we need to add f_{ij} goal locations to G_i and equal number of initial locations to G_j. We place them in the newly created nodes, connected to the midpoints of boundaries of the corresponding subgraphs, so that they overlap in the original graph G (see Fig. 3). In other words, a robot traveling from G_i to G_j can be seen as arriving at an artificial goal location in G_i, crossing the partition line, and "appearing" at the matching artificial initial location in G_j.

Now that the subproblems are balanced, we can use Algorithm 1 to solve each G_i independently. The final step is to



(a) Flow on Γ. (b) Resulting subproblems.

Fig. 3. In order to balance the original initial and goal locations (dark red and green) we add artificial ones (light red and green) to every subgraph.

get rid of the artificial locations by "stitching" the generated local paths together. Here we can use the fact that the order in which the robots depart an artificial initial location must be the same as the order in which they have arrived to the matching goal location.

The pseudo-code for the partition-based planner is presented in Algorithm 2.

Algorithm 2 PartitionPlanner

Input: Graph G, sets of initial and goal locations X^I, X^G, number of subproblems k
Output: Set of global paths P, local schedules for each G_i

```
partition G into {G_i}
construct the high-level graph Γ
construct the flow network on Γ
use network simplex algorithm [6] to solve the min-cost flow problem
for i = 1 to k do
    add artificial initial and goal locations to G_i
    plan local paths on G_i using ExactPlanner
for all edges (G_i, G_j) of Γ with positive flow do
    for all paths leaving subgraph G_i towards G_j do
        identify a matching path entering G_j from G_i
        merge them into a single path on G
```

C. Scheduling the Global Paths

Our task is to create a schedule for the global paths on G that would respect the local schedules generated by Algorithm 1 for each individual G_i.

For every global path p, let $local(p, i)$ be the part of p that goes through the subgraph G_i, $start(p, i)$ be the starting time of $local(p, i)$ according to the local schedule of G_i, and $offset(p, i)$ be the length of the segment of p that starts at the beginning of p and ends at the beginning of $local(p, i)$ (see Fig. 4 for an example).

We create a scheduling graph SG, in which every node is a path and every edge indicates a dependency between robots' starting times. More specifically, we add a directed edge (p, q) with label i to SG if both paths p and q go through the same subgraph G_i and $local(q, i)$ is scheduled to start

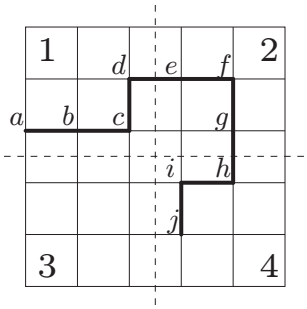


Fig. 4. A path $p = [a \dots j]$ (shown in bold) visits the subproblems G_1, G_2, and G_4. We have: $local(p,1) = [abcd]$, $local(p,2) = [efg]$, $local(p,4) = [hij]$, $offset(p,1) = 0$, $offset(p,2) = 4$, and $offset(p,4) = 7$.

immediately after $local(p,i)$. Note that if some paths have several subgraphs in common, SG is a multigraph.

Graph SG is a directed acyclic graph (see the next section for details) and allows a topological ordering [5]. Therefore, we can traverse the nodes of SG and compute the earliest possible starting time for each global path, which results in Algorithm 3. In lines 7–10 we find a starting time for q such that the delay between robots entering $local(p,i)$ and $local(q,i)$ is greater or equal to $start(q,i) - start(p,i)$ (i.e., the delay required by the output of Algorithm 1). According to (P2), this ensures that each local schedule remains valid.

Algorithm 3 PARTITIONSCHEDULER

Input: Graph G, collection of subgraphs $\{G_i\}$, set of global paths P, local schedules for each G_i
Output: Set of scheduled paths

1: construct the scheduling graph SG with P as vertices
2: perform the topological ordering of P according to SG
3: **for all** $q \in P$ **do**
4: **if** q has in-degree 0 **then**
5: schedule q to start at $start(q) = 0$
6: **else**
7: **for all** incoming edges (p,q) **do**
8: $i =$ label of (p,q)
9: $t_p = start(p) + (start(q,i) - start(p,i)) + (offset(p,i) - offset(q,i))$
10: schedule q to start at $start(q) = \max_p t_p$

D. Correctness

Theorem 1. *Algorithms 2 and 3 generate a valid solution to Problem 1.*

Proof Idea: We need to show that the resulting schedule is collision-free. Since Algorithm 3 respects the local schedules for each G_i, collisions between robots that are located in the same subgraph are impossible. Additionally, since the robots travel between the subgraphs according to the min-cost flow on Γ, they always cross the partition lines in the same direction, which excludes head-on collisions between robots switching subgraphs. Finally, we prove that the scheduling graph is acyclic by using (P3), combined with the fact that the network simplex method [6] for the min-cost flow problem produces a solution restricted to a spanning tree of Γ. ■

IV. ALGORITHM ANALYSIS

In this section, we assume that graph $G = (V,E)$ is a uniform square grid of size $m \times m$ with unit distance between adjacent vertices and that initial and goal locations are chosen at random, independently from each other, according to a uniform distribution on V.

As usual, c and C will denote positive universal constants whose values might change between expressions. Sometimes we will use a parameter, C_k, to indicate that the value depends on k, the number of subgraphs in the partition, but not on any other inputs to the algorithm.

A. Time and Distance Bounds for the Generated Paths

Let $D(k)$ be the total distance traveled by all robots in the solution generated by Algorithm 2 for a fixed value of k, and OPT be the cost of the distance-optimal solution. We say that a parametrized event $E(n)$ happens *with high probability* if it happens with probability $1 - O(n^{-c})$.

Theorem 2.

$$D(k) \le C_k \cdot OPT$$

with high probability as $n \to \infty$.

Proof Idea: First, we extend a fundamental result from the matching theory [2] to derive the bounds on the optimal solution cost:

Lemma 3. *With high probability,*

$$cm\sqrt{n \log n} \le OPT \le Cm\sqrt{n \log n}.$$

Unfortunately, the lemma cannot be applied directly to the subproblems on G_i, since artificial locations are not uniformly distributed. Therefore, we consider a minor modification of Algorithm 2 that attempts to construct as many paths as possible without resorting to artificial locations. Next, we bound the total length of the paths that are limited to a single subgraph by $C_k m\sqrt{n \log n}$, by applying Lemma 3 to them. The total length of the long-distance paths, which traverse multiple subgraphs, is bounded by showing that, with high probability, there is at most $C_k\sqrt{n \log n}$ of such paths, each of length no more than $4m$. Finally, we show that, since such a modification does not improve the solution,

$$D(k) \le C_k m\sqrt{n \log n} \le C_k \cdot OPT,$$

with high probability, as needed. ■

Theorem 4. *It is possible to use the path set P produced by Algorithm 2 to construct a solution to Problem 1 with the same total distance traveled as P and completion time at most $n + \ell - 1$, in which*

$$\ell = \max_{u \in X^I, v \in X^G} dist(u,v).$$

Proof Idea: Simply apply the scheduling approach used in Algorithm 1 to the paths from P. ■

Observation 5. Although the scheduling algorithm from Theorem 4 provides a guaranteed bound for the completion time, this approach is inefficient for large problems. Therefore, in

practice, we instead employ the scheduler given by Algorithm 3. Even though we do not have a non-trivial theoretical upper bound for the completion time for this algorithm, results of Section V demonstrate that, in most cases, the completion times for this algorithm are, in fact, better than the upper bound of Theorem 4.

B. Complexity Analysis

In Algorithm 2, lines 1–3 take linear time in the size of the problem. Line 4 can be implemented to run in $O(k^3 \log k)$ using the algorithm from [17]. Each subgraph contains, on average, $O(\frac{n}{k})$ balanced and $O(\sqrt{\frac{n}{k}})$ unbalanced locations. The total number of unbalanced locations in G, $O(\sqrt{nk})$, is an upper bound on the number of artificial locations in any subproblem. Therefore, every iteration of line 6 takes $O(\sqrt{nk})$ time. Every subproblem passed to EXACTPLANNER is a $\frac{m}{k} \times \frac{m}{k}$ grid with $O(\frac{n}{k} + \sqrt{nk})$ robots. However, since there is only a finite number of places where artificial locations can be placed in each G_i (at most one for each neighboring subgraph), we can reuse some of the BFS results, bringing the running time of line 7 to $O((\frac{n}{k} + \sqrt{nk})^3 + \frac{nm^2}{k^2}) = O(\frac{n^3}{k^3} + n^{\frac{3}{2}}k^{\frac{3}{2}} + \frac{nm^2}{k^2})$ per iteration. Lines 8 to 11 are executed $O(k)$ times and take $O(\frac{n}{k} + \sqrt{nk})$ time per iteration. The scheduling graph in Algorithm 3 contains n vertices, each of in-degree $O(k)$, therefore its running time is $O(nk)$. Adding everything together, we conclude that the combined running time of Algorithms 2 and 3 is $O(m^2 + k^3 \log k + \frac{n^3}{k^2} + n^{\frac{3}{2}}k^{\frac{5}{2}} + \frac{nm^2}{k})$.

Observation 6. The algorithms for the assignment and min-cost flow problems are extremely efficient, despite their high asymptotic running time bounds. In practice, Algorithm 2 spends most of the time inside EXACTPLANNER, computing the shortest paths using BFS. Therefore, its actual running time is dominated by the $O\left(\frac{nm^2}{k}\right)$ term, a claim that is well supported by the data of the next section.

V. EXPERIMENTAL EVALUATION

In this section we perform an experimental analysis of our algorithms. All of them have been implemented in C++, using the LEMON library [7] and Cyrill Stachniss' implementation of the Hungarian algorithm [22], and run on a computer with Intel Core i7-3930K 3.2GHz processor and 64GB RAM. We used square 2D grids as base graphs in all of the examples, initial and goal locations were selected randomly, according to a uniform distribution on V. The algorithms make no assumptions regarding the graph structure and, therefore, have to search for the shortest paths using BFS. All of the results are averaged over 5 runs.

The first two sets of experiments are intended to compare the performance of Algorithms 2 and 3 to that of Algorithm 1. The results are listed in Tables II and III. It can be seen that, in line with Observation 6, the partition-based algorithm is approximately $O(k)$ times faster than the exact one, in which k is the number of subproblems. In all cases, the solutions found had the total length no more than 140% of the optimal. Interestingly, increasing the number of subproblems does not lead to a significant (if at all) increase of the total length, while drastically improving the performance.

TABLE II

Grid size	Robots	Running time, sec	Total distance	Completion time
100×100	500	0.58	2942.8	536.2
	1000	1.18	4152.6	1025.2
	5000	22.7	7213.8	5007.2
500×500	500	7.7	15214.6	702
	1000	14.8	21556.6	1154.8
	5000	102.8	51359.2	5102.2
2000×2000	500	91.7	61341	1338.4
	1000	188.6	85224.2	1612.8
	5000	1014.3	210614.4	5468.6

Results of Algorithm 1 for various problem sizes, averaged over 5 runs each.

TABLE III

Grid size	Robots	Sub-problems	Running time, sec	Distance ratio	Completion time
100×100	500	25	0.031	1.31	160
		100	<0.016	1.30	90.4
	1000	25	0.059	1.30	246
		100	0.028	1.33	105.6
	5000	25	0.37	1.31	637.6
		100	0.12	1.33	268.6
500×500	500	25	0.15	1.32	499.6
		100	0.078	1.31	407
	1000	25	0.61	1.32	561.8
		100	0.15	1.34	461.2
	5000	25	3.2	1.32	1162
		100	0.38	1.33	817.8
2000×2000	500	25	2.1	1.29	1961.4
		100	1.4	1.30	1728.6
	1000	25	3.5	1.30	1905.2
		100	2.6	1.30	1399.4
	5000	25	26.1	1.28	2540.6
		100	5.0	1.32	2022.2

Results of Algorithms 2 and 3 for various problem sizes, averaged over 5 runs each. Column 5 shows the ratio of the total distances for the generated solutions to the optimal distances.

Due to its two-level nature, our algorithm is able to efficiently check that robots' paths on the subproblem graph Γ do not intersect, therefore, their starting times do not need to depend on each other. As a result, in all but three cases, the completion times were better than those of the distance-optimal solutions found by Algorithm 1. Note that Algorithm 1 does not optimize for the completion time and its solutions can be improved with respect to this metric by scheduling multiple robots to depart at the same time when it does not cause conflicts [27]. Here we are not claiming that Algorithm 3 achieves better completion times than modified Algorithm 1, but rather that it usually performs better than the upper bound from Theorem 4.

Next, we evaluate the performance of the new algorithms on large problems, for which Algorithm 1 is impractical. The results can be seen in Table IV. Here again, we observe that, despite the lack of strong guarantees, overall completion times of the solutions are significantly lower than in Theorem 4. The algorithms scale very well and can tackle graphs with 10^9 vertices and 10^6 robots. The amount of memory needed to store the whole graph becomes the main bottleneck. Although we do not have distance-optimal solutions to compare the cost with (it would take many days or even months to run the EXACTPLANNER on problems of this size), the rate of

TABLE IV

Grid size	Robots	Sub-problems	Running time, sec	Total distance	Completion time
10000× 10000	50000	2500	41.3	4.70×10^6	4924.4
	100000		76.8	7.47×10^6	6689.2
20000× 20000	50000	2500	193.1	9.16×10^6	8144.6
	100000		339.1	1.36×10^7	10565.4
30000× 30000	1000000	10000	1959.5	7.20×10^7	14050.8

Results of Algorithms 2 and 3 for large problems, averaged over 5 runs each.

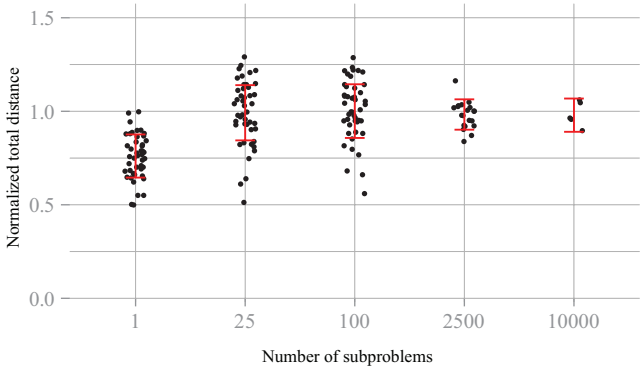


Fig. 5. Total distance values for different values of k, normalized by division by $m\sqrt{n \log n}$. Error bars correspond to one standard deviation.

growth of the total distance is consistent with Lemma 3 (see Fig. 5). This suggests that the average approximation factor of Algorithm 2 remains close to 1 even for the large number of subproblems.

VI. Conclusions and Future Work

In this paper we introduced a two-level partition-based algorithm for planning collision-free paths for multiple indistinguishable robots on a graph. The main focus of the algorithm is efficiency; however, it is still able to generate solutions that are close to optimal in terms of the total distance traveled and have low completion times.

Several questions remain regarding the proposed algorithms. For example, Theorem 2 does not provide a specific value for the approximation factor. However, experimental results suggest that it is quite low, at least on average. Additionally, it would be interesting to obtain a strong bound on the completion times for the paths scheduled by PartitionScheduler.

Although most of the computations in Algorithm 2 are performed on a local subgraph, it still requires global information in order to deal with locally unbalanced initial or goal locations and to construct the collision-free schedule. A distributed version of the algorithm that only relies on communications between neighboring robots would significantly improve the applicability.

Finally, another avenue of research would be to extend the results to continuous domains. It can be done, for example, by constructing a grid or a roadmap in the continuous space and planning the paths in the resulting discrete graph.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[2] M. Ajtai, J. Komlós, and G. Tusnády, "On optimal matchings," *Combinatorica*, vol. 4, no. 4, pp. 259–264, 1984.

[3] J. E. Aronson, "A survey on dynamic network flows," *Annals of Operations Research*, vol. 20, no. 1, pp. 1–66, 1989.

[4] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction To Algorithms*. MIT Press, 2001.

[6] G. B. Dantzig, *Linear Programming and Extensions*. Princeton University Press, 1963.

[7] B. Dezső, A. Jüttner, and P. Kovács, "LEMON – an open source C++ graph template library," *Electronic Notes in Theoretical Computer Science*, vol. 264, no. 5, pp. 23–45, Jul. 2011.

[8] M. A. Erdmann and T. Lozano-Pérez, "On multiple moving objects," in *Proceedings IEEE International Conference on Robotics & Automation*, 1986, pp. 1419–1424.

[9] L. R. Ford and D. R. Fulkerson, *Flows in Networks*. New Jersey: Princeton University Press, 1962.

[10] S. Kloder and S. Hutchinson, "Path planning for permutation-invariant multirobot formations," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 650–665, 2006.

[11] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, p. 83–97, 1955.

[12] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer, 1991.

[13] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, also available at http://planning.cs.uiuc.edu/.

[14] S. M. LaValle and S. A. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Trans. on Robotics and Automation*, vol. 14, no. 6, pp. 912–925, Dec. 1998.

[15] L. Liu and D. A. Shell, "Large-scale multi-robot task allocation via dynamic partitioning and distribution," *Autonomous Robots*, vol. 33, no. 3, pp. 291–307, Jun. 2012.

[16] R. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011, pp. 294–300.

[17] J. B. Orlin, "A polynomial time primal network simplex algorithm for minimum cost flows," in *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '96. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1996, p. 474–481.

[18] M. R. K. Ryan, "Exploiting subgraph structure in multi-robot path planning," *Journal of Artificial Intelligence Research*, vol. 31, pp. 497–542, 2008.

[19] D. Silver, "Cooperative pathfinding." in *The 1st Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005, pp. 23–28.

[20] T. Simeon, S. Leroy, and J.-P. Laumond, "Path coordination for multiple mobile robots: a resolution-complete algorithm," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 1, pp. 42–49, Feb. 2002.

[21] K. Solovey and D. Halperin, "k-color multi-robot motion planning," in *The Tenth International Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2012.

[22] C. Stachniss, "C implementation of the hungarian method," Sep. 2004. [Online]. Available: http://www.informatik.uni-freiburg.de/~stachnis/misc.html

[23] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011, pp. 668–673.

[24] P. Surynek, "A novel approach to path planning for multiple robots in bi-connected graphs," in *Proceedings IEEE International Conference on Robotics and Automation*, 2009, pp. 3613–3619.

[25] J. van den Berg and M. Overmars, "Prioritized motion planning for multiple robots," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[26] J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha, "Centralized path planning for multiple robots: Optimal decoupling into sequential plans," in *Proceedings Robotics: Science and Systems*, 2009.

[27] J. Yu and S. M. LaValle, "Distance optimal formation control on graphs with a tight convergence time guarantee," in *Proceedings IEEE International Conference on Decision and Control*, 2012, to be published.

[28] ——, "Multi-agent path planning and network flow," in *The Tenth International Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2012.

[29] A. Zelinsky, "A mobile robot exploration algorithm," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 6, pp. 707–717, 1992.