Planning Optimal Paths for Multiple Robots on Graphs

Jingjin Yu   Steven M. LaValle

Abstract— **In this paper, we study the problem of optimal multi-robot path planning (MPP) on graphs. We propose two multiflow based integer linear programming (ILP) models that computes minimum last arrival time and minimum total distance solutions for our MPP formulation, respectively. The resulting algorithms from these ILP models are complete and guaranteed to yield true optimal solutions. In addition, our flexible framework can easily accommodate other variants of the MPP problem. Focusing on the time optimal algorithm, we evaluate its performance, both as a stand alone algorithm and as a generic heuristic for quickly solving large problem instances. Computational results confirm the effectiveness of our method.**

I. Introduction

Planning collision-free paths for multiple robots, an easily stated yet difficult problem, has been actively studied for decades [4, 12, 13, 20, 22, 24, 25, 28, 29, 32]. The hardness of the problem mainly resides with the coupling between the robots' paths which leads to an enormous state space and branching factor. As such, algorithms that are both complete and (distance) optimal, such as the A* [8] algorithm and its variants, do not perform well on tightly coupled problems beyond very small ones. On the other hand, faster algorithms for finding the paths generally do not provide optimality guarantees: Sifting through all feasible path sets for optimal ones greatly increases the search space, which often makes these problems intractable.

In this paper, we investigate the problem of planning optimal paths for multiple robots with individual goals. The robots have identical but non-negligible sizes, are confined to some arbitrary connected graph, and are capable of moving from one vertex to an adjacent vertex in one time step. Collision between robots is not allowed, which may occur when two robots attempt to move to the same vertex or move along the same edge in different directions. For this general setting, we propose a network flow based integer linear programming (ILP) model for finding robot paths that are time optimal or distance optimal. Our time optimality criterion seeks to minimize the number of time steps until the last robot reaches its goal; distance optimality seeks to minimize the total distance (each edge has unit distance) traveled by the robots. Taking advantage of the state of the art ILP solvers (Gurobi is used in this paper), our method can plan time optimal, collision-free paths for several dozens of robots on graphs with hundreds of vertices within minutes.

As a universal subroutine, collision-free path planning for multiple robots finds applications in tasks spanning assembly [7, 15], evacuation [18], formation control [2, 16, 21, 23, 27], localization [6], object transportation [14, 19], search and rescue [9], and so on. Given its importance, path planning for multi-robot systems has remained as a subject of intense study for many decades. Given the vast size of the available literature, we will only mention related research on discrete MPP and refer the readers to [3, 10, 11] and the references therein for a more comprehensive review of the subject.

From an algorithmic perspective, discrete MPP is a natural extension of the single robot path planning problem: One may combine the state spaces of all robots and treat the problem as a planning problem for a single robot. A* algorithm can then be used to compute distance optimal solutions to these problems. However, since naive A* scales poorly due to the curse of dimensionality, additional heuristic methods were proposed to improve the computational performance. One of the first such heuristics, Local Repair A* (LRA*) [32], plans robot paths simultaneously and performs local repairs when conflicts arise. Focusing on fixing the (locality) shortcomings of LRA*, Windowed Hierarchical Cooperative A* (WHCA*) [22] proposed to use a space-time window to allow more choices for resolving local conflicts while limiting the search space size at the same time. For additional heuristics exploring various specific local and global features, see [13, 20, 25].

Formulations of MPP problems with optimality guarantee have also been studied. The most general optimality criterion is the total path length traveled by all robots, which is consistent with the distance heuristic used by the A* algorithm. Since A* is the best possible among all such algorithms for finding distance optimal solutions, one should not expect complete and true optimal algorithms to exist that perform much better than the basic A* algorithm in all cases. Nevertheless, this does not prevent algorithms from quickly solving certain instances optimally. One such algorithm that is also complete, MGS*x*, is presented in [24] (note that the grid world formulation in [24], which allows diagonal moves in general, even in the presence of diagonal obstacles, does not carry over to general graphs or geometric models in robotics). For time optimality, for a version of the MPP problem that resembles our formulation more closely, it was shown that finding a time optimal solution is NP-hard [26], implying that our formulation is also intractable [30]. Finally, it was shown that finding the least number of moves for the $N \times N$-generalization of the 15-puzzle is NP-hard [17]. Here,

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. Steven M. LaValle is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA. E-mail: lavalle@uiuc.edu. This work was supported in part by NSF grants 0904501 (IIS Robotics) and 1035345 (Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

time optimality equals distance optimality, which is not the case in general.

The main contributions of this paper are twofold. First, adapting the constructions from [31], we develop ILP models for solving time optimal and distance optimal MPP problems. The resulting algorithms are shown to be complete. Our approach is quite general and easily accommodates other formulations of the MPP problems, including that of [24]. Second, we provide thorough computational evaluations of our models' performance: With a state-of-the-art ILP solver, our models are capable of solving large problem instances with few dozens of robots fairly fast. Such a result is in some sense the best we can hope for because the best possible algorithm for such problems cannot run in polynomial time unless $P = NP$. As an added bonus, we also show that the (time optimal) algorithm works well as a subroutine for quickly solving MPP problems (non-optimally)[1].

The rest of the paper is organized as follows. We provide problem definitions in Section II, along with a motivating example. Section III relates MPP to multiflow, establishing the equivalence between the two problems. In Section IV, ILP models are provided for obtaining time optimal and distance optimal solutions, respectively. Section V is devoted to briefly discussing basic properties of the n^2-puzzle, which is an interesting benchmark problem on its own. We evaluate the computational performance of our algorithm in Section VI and conclude in Section VII.

II. Multi-robot Path Planning on Graphs

A. Problem Formulation

Let $G = (V, E)$ be a connected, undirected, simple graph (i.e., no multi-edges), in which $V = \{v_i\}$ is its vertex set and $E = \{(v_i, v_j)\}$ is its edge set. Let $R = \{r_1, \ldots, r_n\}$ be a set of robots that move with unit speeds along the edges of G, with initial and goal locations on G given by the injective maps $x_I, x_G : R \to V$, respectively. The set R is effectively an index set. A *path* or *scheduled path* is a map $p_i : \mathbb{Z}^+ \to V$, in which $\mathbb{Z}^+ := \mathbb{N} \cup \{0\}$. Intuitively, the domains of the paths are discrete time steps. A path p_i is *feasible* for a single robot r_i if it satisfies the following properties: 1. $p_i(0) = x_I(r_i)$; 2. For each i, there exists a smallest $k_i^{\min} \in \mathbb{Z}^+$ such that for all $k \geq k_i^{\min}$, $p_i(k) \equiv x_G(r_i)$; 3. For any $0 \leq k < k_i^{\min}$, $(p_i(k), p_i(k+1)) \in E$ or $p_i(k) = p_i(k+1)$. We say that two paths p_i, p_j are in *collision* if there exists $k \in \mathbb{Z}^+$ such that $p_i(k) = p_j(k)$ (collision on a vertex, or *meet*) or $(p_i(k), p_i(k+1)) = (p_j(k+1), p_j(k))$ (collision on an edge, or *head-on*). If $p(k) = p(k+1)$, then the robot stays at vertex $p(k)$ between the time steps k and $k+1$.

Problem 1 (MPP on Graphs) *Given (G, R, x_I, x_G), find a set of paths $P = \{p_1, \ldots, p_n\}$ such that p_i's are feasible paths for respective robots r_i's and no two paths p_i, p_j are in collision.*

[1]The software (written in Java, including a programming interface), as well as all examples used in our evaluation, are available at `http://msl.cs.uiuc.edu/~jyu18/pe/mapp.html`.

A natural criterion for measuring path set optimality is the number of time steps until the last robot reaches its goal. This is sometimes called the *makespan*, which can be computed from $\{k_i^{\min}\}$ for a feasible path set P as

$$T_P = \max_{1 \leq i \leq n} k_i^{\min}.$$

Another frequently used objective is distance optimality, which counts the total number of edges traveled by the robots. We point out that distance optimality and time optimality cannot be satisfied at the same time in general: In Fig. 1, let the dotted straight line have length t and the dotted arc has length $1.5t$ from some large even number t. The four solid line segments are edges with unit length. Assuming that robot 1, 2 are to move from the locations marked with solid circles to the locations marked with gray dotted circles. Time optimal paths take $1.5t + 2$ time steps with a total distance of $2.5t + 4$; distance optimal paths take $2t + 3$ time steps with a total distance of $2t + 4$.

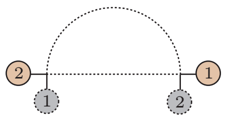


Fig. 1. Time optimality and distance optimality cannot be satisfied simultaneously for this setup.

In this paper, we work with graphs on which the only possible collisions are meet or head-on collisions. This assumption is a mild one: For example, a 2D grid with unit edge lengths is such a graph for robots with radii of no more than $\sqrt{2}/4$. As a last note, our formulation allows multiple robots to move at the same time step as long as no collision occurs. On a graph, this allows robots on any cycle to "rotate".

B. A Motivating Example

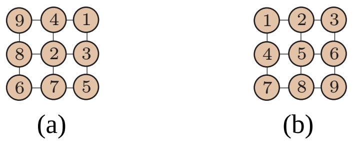


Fig. 2. a) A 9-puzzle problem. b) The desired goal state.

To better characterize what we solve in this paper, look at the example in Fig. 2. We call this problem a 9-puzzle, which is a variant of the 15-puzzle [17]; it is also related to the "H" example in [12]. Given the robots as numbered in Fig. 2(a), we want to get them into the *state* (*configuration* is also used in this paper to refer to the same, depending on the context) given in Fig. 2(b) (such a configuration is often referred to as *row major* ordering). Coming up with a feasible solution for such a highly constrained problem is non-trivial, let alone solving it with an optimality guarantee. The time optimal algorithm we present in this paper solves this problem instance under 0.1 second. The solution is given in Fig. 3. The time optimality of the solution is evident: It takes at least four steps for robot 9 to reach its goal.

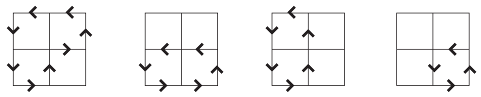


Fig. 3. A 4-step solution from our algorithm. The directed edges show the moving direction of the robots at the tail of the edges.

III. Multi-robot Path Planning and Multiflow

A. Network Flow

In this subsection we provide a summary of the network flow problem formulation pertinent to the introduction of our algorithm. For surveys on network flow, see [1, 5]. A *network* $\mathcal{N} = (G, c_1, c_2, S)$ consists of a directed graph $G = (V, E)$ with $c_1, c_2 : E \to \mathbb{Z}^+$ as the maps defining the capacities and costs on edges, respectively, and $S \subset V$ as the set of sources and sinks. We let $S = S^+ \cup S^-$, with S^+ denoting the set of sources and S^- denoting the set of sink vertices. For a vertex $v \in V$, let $\delta^+(v)$ (resp. $\delta^-(v)$) denote the set of edges of G going to (resp. leaving) v. A feasible (static) S^+, S^--flow on this network $\mathcal{N}$ is a map $f : E \to \mathbb{Z}^+$ that satisfies edge capacity constraints,

$$\forall e \in E, \quad f(e) \le c_1(e), \tag{1}$$

the flow conservation constraints at non terminal vertices,

$$\forall v \in V \backslash S, \quad \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = 0, \tag{2}$$

and the flow conservation constraints at terminal vertices,

$$\begin{aligned} F(f) &= \sum_{v \in S^+} \Big(\sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e) \Big) \\ &= \sum_{v \in S^-} \Big(\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \Big). \end{aligned} \tag{3}$$

The quantity $F(f)$ is called the *value* of the flow f. The classic (single-commodity) *maximum flow* problem asks the question: Given a network $\mathcal{N}$, what is the maximum $F(f)$ that can be pushed through the network? The *minimum cost maximum flow* problem further requires the flow to have minimum total cost among all maximum flows. That is, we want to find a flow among all maximum flows that also minimizes the quantity

$$\sum_{e \in E} c_2(e) \cdot f(e). \tag{4}$$

The above formulation concerns a single commodity, which corresponds to all robots being inter exchangeable. For MPP, the robots are not inter exchangeable and must be treated as different commodities. *Multi-commodity flow* or *multiflow* captures the problem of flowing different types of commodities through a network. Instead of having a single flow function f, we have a flow function f_i for each commodity i. The constraints (1), (2), and (3) become

$$\forall i, \forall e \in E, \quad \sum_i f_i(e) \le c_1(e), \tag{5}$$

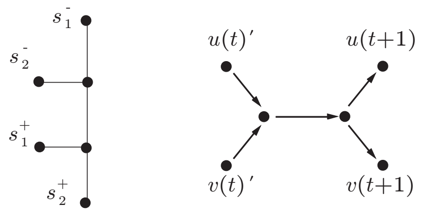
$$\forall i, \forall v \in V \backslash S, \quad \sum_{e \in \delta^+(v)} f_i(e) - \sum_{e \in \delta^-(v)} f_i(e) = 0, \tag{6}$$

$$\begin{aligned} \forall i, \quad & \sum_{v \in S^+} \Big(\sum_{e \in \delta^-(v)} f_i(e) - \sum_{e \in \delta^+(v)} f_i(e) \Big) \\ = & \sum_{v \in S^-} \Big(\sum_{e \in \delta^+(v)} f_i(e) - \sum_{e \in \delta^-(v)} f_i(e) \Big). \end{aligned} \tag{7}$$

Again, maximum flow and minimum cost flow problems can be posed for a multiflow setup.

B. Equivalence between MPP and multiflow

Viewing robots as commodities, we may connect MPP and multiflow. This relationship (Theorem 2) was stated in [31] without full proof, which is provided here for completeness. To make the presentation clear, we use as an example the simple graph G in Fig. 4(a), with initial locations $\{s_i^+\}, i = 1, 2$ and goal locations $\{s_i^-\}, i = 1, 2$. An instance of Problem 1 is given by $(G, \{r_1, r_2\}, x_I : r_i \mapsto s_i^+, x_G : r_i \mapsto s_i^-)$. We now convert this problem to a network flow problem, $\mathcal{N}' = (G', c_1, c_2, S^+ \cup S^-)$. Given the graph G and a natural number T, we create $2T+1$ copies of vertices from G, with



(a) (b)

Fig. 4. a) A simple G. b) A gadget for splitting an undirected edge through time steps.

indices $0, 1, 1', \ldots$, as shown in Fig. 5. For each vertex $v \in G$, denote these copies $v(0) = v(0)', v(1), v(1)', v(2), \ldots, v(T)'$. For each edge $(u, v) \in G$ and time steps $t, t+1$, $0 \le t < T$, add the gadget shown in Fig. 4(b) between $u(t)', v(t)'$ and $u(t+1), v(t+1)$ (arrows from the gadget are omitted from Fig. 5 since they are small). For the gadget, we assign unit capacity to all edges, unit cost to the horizontal middle edge, and zero cost to the other four edges. This gadget ensures that two robots cannot travel in opposite directions on an edge in the same time step. To finish the construction of Fig. 5, for each vertex $v \in G$, we add one edge between every two successive copies (i.e., we add the edges $(v(0), v(1)), (v(1), v(1)'), \ldots, (v(T), v(T)')$). These correspond to the green and blue edges in Fig. 5. For all green edges, we assign them unit capacity and cost; for all blue edges, we assign them unit capacity and zero cost.

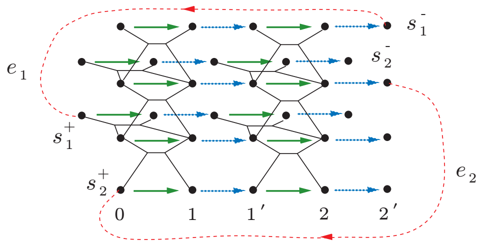


Fig. 5. The time-expanded network ($T = 2$).

Fig. 5 (with the exception of edges e_1 and e_2, which are not relevant until Section IV), called a *time-expanded network*

[1], is the desired G'. For the set S, we may simply let $S^+ = \{v(0) : v \in \{s_i^+\}\}$ and $S^- = \{v(T)' : v \in \{s_i^-\}\}$. The network $\mathcal{N}' = (G', c_1, c_2, S^+ \cup S^-)$ is now complete; we have reduced Problem 1 to an integer maximum multiflow problem on $\mathcal{N}'$ with each robot from R as a single type of commodity.

Theorem 2 *Given an instance of Problem 1 with input parameters (G, R, x_I, x_G), there is a bijection between its solutions (with maximum number of time steps up to T) and the integer maximum multiflow solutions of flow value n on the time-expanded network $\mathcal{N}'$ constructed from (G, R, x_I, x_G) with T time steps.*

PROOF. (Injectivity) Assume that $P = \{p_1, \ldots, p_n\}$ is a solution to an instance of Problem 1. For each p_i and every time step $t = 0, \ldots, T$, we mark the copy of $p_i(t)$ and $p_i(t)'$ (recall that $p_i(t)$ corresponds to a vertex of G) at time step t in the time-expanded graph G'. Connecting these vertices of G' sequentially (there is only one way to do this) yields one unit of flow f_i on $\mathcal{N}'$ (after connecting to appropriate source and sink vertices in S^+, S^-, which is trivial). It is straightforward to see that if two paths p_i, p_j are not in collision, then the corresponding flows f_i, f_j on $\mathcal{N}'$ are vertex disjoint paths and therefore do not violate any flow constraint. Since any two paths in P are not in collision, the corresponding set of flows $\{f_1, \ldots, f_n\}$ is feasible and maximal on $\mathcal{N}'$.

(Surjectivity) Assume that $\{f_1, \ldots, f_n\}$ is a integer maximum multiflow on the network $\mathcal{N}'$ with $|f_i| = 1$. First we establish that any pair of flows f_i, f_j are vertex disjoint. To see this, we note that f_i, f_j (both are unit flows) cannot share the same source or sink vertices due to the unit capacity structure of $\mathcal{N}'$ enforced by the blue edges. If f_i, f_j share some non-sink vertex v at time step $t > 0$, both flows then must pass through the same blue edge (see Fig. 4(b)) with v being either the head or tail vertex, which is not possible. Thus, f_i, f_j are vertex disjoint on $\mathcal{N}'$. We can readily convert each flow f_i to a corresponding path p_i (after deleting extra source vertex, sink vertices, vertices in the middle of the gadgets, and tail vertices of blue edges) with the guarantee that no p_i, p_j will collide due to a meet collision. By construction of $\mathcal{N}'$, the gadget we used ensures that a head-on collision is also impossible. The set $\{p_1, \ldots, p_n\}$ is then a solution to Problem 1. □

C. Accommodating other formulations

Our network flow based approach for encoding the MPP problem is fairly general; we illustrate this using two examples. The first is the grid world formulation from [24], which allows (single) diagonal crossings. That is, for vertices $v_1, \ldots, v_4$ on the four corners of a square cell with v_1, v_3 and v_2, v_4 diagonal to each other, respectively, it is possible for a robot to move from v_1 to v_3 provided that v_3 is unoccupied and the v_2-v_4 diagonal is not used in the same time step. To include this constraint in the ILP model, we may simply add the gadget structure in Fig. 6 to the time-expanded network construction. The inclusion of the gadget will allow a single diagonal crossing; the extra paths do not create an issue since no two robots can go through a single vertex at the same time step (enforced by the blue dotted edges in Fig. 5).

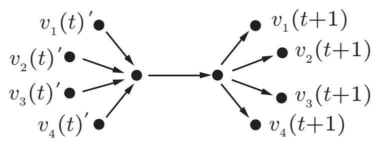


Fig. 6. A gadget for allowing diagonal crossings.

For a second example, in some MPP formulations, head-on collisions may be allowed. For instance, two adjacent CPUs may exchange two units of data in parallel but no single CPU may hold multiple units of data. To allow this, we simply do not use the gadget from Fig. 4(b) when the time-expanded network is constructed.

IV. ALGORITHMIC SOLUTIONS FOR OPTIMAL MULTI-ROBOT PATH PLANNING

Given the time-expanded network $\mathcal{N}' = (G', c_1, c_2, S^+ \cup S^-)$, it is straightforward to create an integer linear programming (ILP) model with different optimality objectives. We investigate two objectives in this section: Time optimality or makespan (the time when the last robot reaches its goal) and distance optimality (the total distance traveled by all robots).

A. Time optimality

Time optimal solutions to Problem 1 can be obtained using a maximum multiflow formulation. As a first step, we introduce a set of n *loopback* edges to G' by connecting each pair of corresponding goal and start vertices in S, from the goal to the start. For convenience, denote these loopback edges as $\{e_1, \ldots, e_n\}$ (e.g., edges e_1, e_2 in Fig. 5). These edges have unit capacity and zero cost. Next. for each edge $e_j \in G'$, create n binary variables $x_{1,j}, \ldots, x_{n,j}$ corresponding to the flow through that edge, one for each robot. $x_{i,j} = 1$ if and only if robot r_i passes through e_j in G'. The variables $x_{i,j}$'s must satisfy two edge capacity constraints and one flow conservation constraint,

$$\begin{aligned} \forall e_j \in G', \qquad & \sum_{i=1}^{n} x_{i,j} \le 1 \\ \forall 1 \le i, j \le n, i \ne j, \qquad & x_{i,j} = 0, \end{aligned} \tag{8}$$

$$\forall v \in G' \text{ and } 1 \le i \le n, \sum_{e_j \in \delta^+(v)} x_{i,j} = \sum_{e_j \in \delta^-(v)} x_{i,j}. \tag{9}$$

The objective function is

$$\max \sum_{1 \le i \le n} x_{i,i}. \tag{10}$$

For each fixed T, the solution to the above ILP problem equaling n means that a feasible solution to Problem 1 is found. We are to find the minimal T that yields such a feasible solution. To do this, we start with T being the maximum over all robots the shortest possible path length for each robot, ignoring all other robots. We then build the ILP model for this T and test for a feasible solution. If the model is not feasible, we increase T and try again. The first feasible T is the optimal T. The robots' paths can be extracted based on the proof of Theorem 2. The algorithm is complete: Since the problem is discrete, there is only a finite number of possible states. Therefore, for some sufficiently large T, there must either be a feasible solution or we can pronounce that none can exist. Calling this algorithm MINMAKESPAN (time optimal MPP), we have shown the following.

Proposition 3 *Algorithm* MINMAKESPAN *is complete and returns a solution with minimum makespan to Problem 1 if one exists.*

B. Distance optimality

Distance optimality objective can be encoded using minimum cost maximum multiflow. Constraints (8) and (9) remain; to force a maximum flow, let $x_{i,i} = 1$ for $1 \leq i \leq n$. The objective is given by

$$\min \sum_{e_j \in G', j>n, 1 \leq i \leq n} c_2(e_j) \cdot x_{i,j}. \tag{11}$$

The value given by (11), when feasible, is the total distance of all robots' paths. Let T_t denote the optimal T produced by MINMAKESPAN (if one exists), then a distance optimal solution exists in a time-expanded network with $T = nT_t$ steps. Calling this algorithm MINTOTALDIST (distance optimal MPP), we have

Proposition 4 *Algorithm* MINTOTALDIST *is complete and returns a solution with minimum total path length to Problem 1 if one exists.*

Due to the large number of steps needed in the time-expanded network, MINTOTALDIST, in its current form, is not very fast in solving problems with many robots. Therefore, our evaluation in this paper focuses on MINMAKESPAN which, on the other hand, is fairly fast in solving some very difficult problems. MINTOTALDIST, however, still proves useful in providing time optimal and near distance optimal solutions using the outputs of MINMAKESPAN, as shown in Subsection VI-C.

V. PROPERTIES OF THE n^2-PUZZLE

The example problem from Fig. 2 easily extends to an $n \times n$ grid; we call this class of problems the n^2-puzzle. Such problems are highly coupled: No robot can move without at least three other robots moving at the same time. At each step, all robots that move must move synchronously in the same direction (per cycle) on one or more disjoint cycles (see e.g., Fig. 3). To put into perspective the computational results on n^2-puzzles that follow, we make a characterization of the state structure of the n^2-puzzle for $n \geq 3$ (the case of $n = 2$ is trivial).

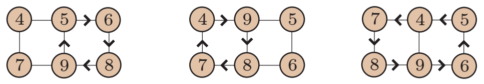


Fig. 7. A 3-step procedure for exchanging robots 8 and 9.

Proposition 5 *All states of the 9-puzzle are connected via legal moves.*

PROOF. We show that any state of a 9-puzzle can be moved into the state shown in Fig. 2(b). From any state, robot 5 can be easily moved into the center of the grid. We are left to show that we can exchange two robots on the border without affecting other robots. This is possible due to the procedure illustrated in Fig. 7. □

Larger puzzles can be solved recursively: We may first solve the top and right side of the puzzle and then the left over smaller square puzzle. For a 16-puzzle, Fig. 8 outlines the procedure, consisting of six main steps:

1) Move robots 1 and 2 to their respective goal locations, one robot at a time (first 1, then 2).
2) Move robots 3 and 4 (first 3, then 4) to the lower left corner (top-middle figure in Fig. 8).
3) Move robots 3 and 4 to their goal location together via counterclockwise rotation along the cycle indicated in the top-middle figure in Fig. 8.
4) Move robot 8 to its goal location.
5) Move robots 12 and then 16 to the lower left corner.
6) Rotate robots 12 and 16 to their goal locations.

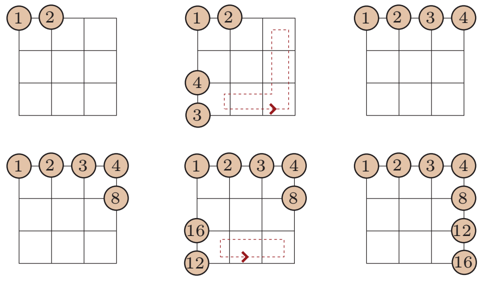


Fig. 8. A solution scheme for solving top/left sides of the 16-puzzle.

It is straightforward to see that larger puzzles can be solved similarly. We have thus outlined the essential steps for proving Proposition 6 below; a more generic proof can be written using generators of permutation groups, which we omit here due to its length. Proposition 6 implies that, for $n \geq 3$, all instances of n^2-puzzles are solvable. The constructive proofs of Proposition 5 and 6 lead to recursive algorithms for solving any n^2-puzzle (clearly, the solution is not time/distance optimal in general).

Proposition 6 *All states of an n^2-puzzle, $n \geq 3$ are connected via legal moves.*

Corollary 7 *All instances of the n^2-puzzle, $n \geq 3$, are solvable.*

By Proposition 6, since all states of a n^2-puzzle for $n \geq 3$ are connected via legal moves, the state space of searching an n^2-puzzle equals n^2 *factorial*. For 16-puzzle and 25-puzzle, $16! > 10^{13}, 25! > 10^{25}$. Large state space is one of the three reasons that make finding a time optimal solution to the n^2-puzzle a difficult problem. The second difficulty comes from the large branching factor at each step. For a 9-puzzle, there are 13 unique cycles, yielding a branching factor of 26 (clockwise and counterclockwise rotations). For the 16-puzzle, the branching factor is around 500. This number balloons to over 10^4 for the 25-puzzle. This suggests that on typical commodity personal computer hardware (assuming a 1GHz processor), a baisc breadth first search algorithm will not be able to go beyond depth of 3 for the 16-puzzle and depth 2 for the 25-puzzle in reasonable amount of time. Moreover, enumerating these cycles is a non-trivial task. The third difficulty is the lack of obvious heuristics: Manhattan distances of robots to their respective goals prove to be a bad one. For example, given the initial configuration as that in Fig. 2(a), the first step in the optimal plan from Fig. 3 gets robots 1, 3, 4, 6, 8, 9 closer to their respective goals while moving robots 2, 7 farther. On the other hand, rotating counterclockwise along the outer cycle takes robots 1, 3, 4, 5, 6, 8, 9 closer and only moves robot 7 farther. However, if we instead take this latter first step, the optimal plan afterwards will take 5 more steps.

VI. Solutions and Evaluation

Our experimentation in this paper focuses on MinMakespan with the main goal being evaluating the comparative efficiency of our approach rather than pushing for best computational performance. As such, our implementation is Java based and did not directly take advantage of multi-core technology. We note that, Gurobi, the ILP solver used in our implementation, can engage multiple cores automatically for hard problems. We ran our code on an Intel Q6600 quad-core machine with a 4GB JavaVM.

A. Time optimal solution to n^2-puzzles

The first experiment we performed was evaluating the efficiency of the algorithm MinMakespan for finding time optimal solutions to the n^2-puzzle for $n = 3, 4, 5$, and 6. We ran Algorithm MinMakespan on 100 randomly generated n^2-puzzle instances for $n = 3, 4, 5$. For the 9-puzzle, computation on all instances completed successfully with an average computation time of 1.36 seconds per instance. To compare the computational result, we implemented a (optimal) BFS algorithm. The BFS algorithm is heavily optimized: For example, cycles of the grid are precomputed and hard coded to save computation time. Since the state space of the 9-puzzle is small, the BFS algorithm is capable of optimally solving the same set of 9-puzzle instances with an average computation time of about 0.89 seconds per instance.

Once we move to the 16-puzzle, the power of general ILP solvers becomes evident. MinMakespan solved all 100 randomly generated 16-puzzle instances with an average computation time of 18.9 seconds. On the other hand, the BFS algorithm with a priority queue that worked for the 9-puzzle ran out of memory after a few minutes. As our result shows that an optimal solution for the 16-puzzle generally requires 6 time steps, it seems natural to also try bidirectional search, which cuts down the total number states stored in memory. To complete such a search, one side of the bidirectional search generally must reach a depth of 3, which requires storing about 3×10^7 states, each taking 64 bits of memory. This turns out to be too much for a 4GB JavaVM: A bidirectional search ran out of memory after about 10 minutes in general. To be sure, we also coded part of the same search algorithm in C++ with STL. Reaching a search depth 3 on one side takes about a minute with a memory footprint of 1.5GB, suggesting a minimum running time of more than one minute.

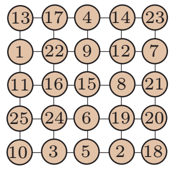


Fig. 9. An instance of a 25-puzzle problem solved by MinMakespan.

For the 25-puzzle, without a good heuristic, bidirectional search cannot explore a tiny fraction of the fully connected state space with about 10^{25} states. On the other hand, MinMakespan again consistently solves the 25-puzzle, with an average computational time under 2 hours over 100 randomly created problems. Fig. 9 shows one of the solved instances with a 7-step solution given in Fig. 10. Note that 7 steps is obviously the least possible since it takes at

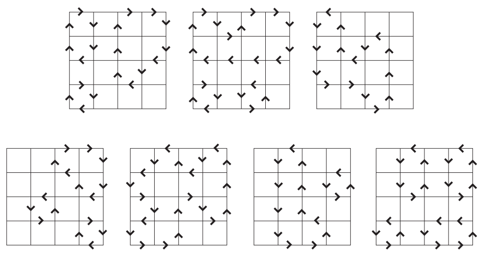


Fig. 10. An optimal 7-step solution (from left to right, then top to bottom) to the 25-puzzle problem from Fig. 9, by MinMakespan in about 30 minutes.

least 7 steps to move robot 10 to its desired goal. We also briefly tested MinMakespan on the 36-puzzle. While we had some success here, MinMakespan generally does not seem to solve a randomly generated instance of the 36-puzzle within 24 hours, which has 3.7×10^{41} states and a branching factor of well over 10^6.

B. Time optimal solutions for grid graphs

For problems in which not all graph vertices are occupied by robots, MinMakespan can handle much larger instances. In a first set of tests on this subject, a grid size of 20×15 is used with varying percentage of obstacles (simulated by removed vertices) and robots for evaluating the effect of these factors. A typical set up is illustrated in Fig. 11. The computation time (in seconds) and the average number of optimal time steps (in parenthesis) are listed in Table I. The numbers are averages over 10 randomly

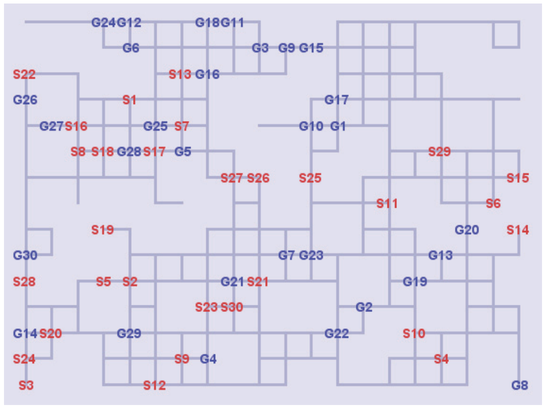


Fig. 11. A 20×15 grid with 20% verices removed (modeling obstacles) and 30 start/goal pairs. The start locations are marked with strings beginning with “S” and the goal locations are marked with strings beginning with “G”.

TABLE I

% obs	Number of robots				
	10	20	30	40	50
5	2.5(22)	7.3(24)	16.7(27)	23.6(26)	70.7(27)
10	2.1(21)	7.8(24)	13.1(26)	20.4(26)	48.6(26)
15	3.9(25)	6.2(24)	13.8(26)	32.8(27)	126(28)
20	2.4(24)	7.7(27)	21.9(28)	39.3(26)	173(27)
25	2.7(27)	8.1(28)	24.8(30)	68.0(28)	$253(30)^4$
30	3.0(31)	$29.9(34)^9$	$234(44)^5$	$80.6(29)^3$	N/A

created instances. For each run, a maximum of 1000 seconds is allowed (such limits, somewhat arbitrary, were chosen to manage the expected running time of the entire set of experiments; our complete algorithms should terminate eventually). Entries with superscript numbers suggest the 10 runs did not all finish within the given time. The superscript numbers represent the successful runs on which the statistics were computed. “N/A” means no instance finished within the allowed time. From the results, we observe that the percentage of randomly placed obstacles does not affect the problem difficulty, as measured by computational time, in a monotonic way. On one hand, more obstacles remove more vertices from the grid, making the problem size smaller, reducing the computational difficulty. On the other hand, as more obstacles are introduced, the reduced connectivity of the graph makes the problem harder. In particular, the 20×15 grid setting suddenly becomes a hard problem with 30% obstacles. The difficulty is also reflected by the average number of steps in an optimal solution: Longer time means reduced availability of alternative paths.

TABLE II

% obs	Number of robots				
	10	20	30	40	50
20	14.4(41)	34.6(45)	43.7(44)	87.5(47)	$402(49)^9$

In a second test on even larger problems, 32×32 grids with 20% obstacles were tried. For between 10 and 50 robots with an increment of 10, 10 random instances each were created; each instance is allowed to run a maximum of half an hour. The statistics, similarly composed as that in Table I, is listed in Table II. We observe that the problem is similar in difficulty to the 20×15 grid setting with 25% obstacles, but much simpler than that with 30% obstacles.

C. *Distance optimality of time optimal solutions*

Although MINTOTALDIST is not yet practical for computing distance optimal solutions alone, it can be used for computing distance optimal solutions for a fixed time expansion length T. That is, we first find a time optimal solution, which gives us the smallest time-expanded network containing feasible solutions. We then run MINTOTALDIST on this network. For evaluation, we used the same 20×15 instances with 5-25% obstacles and 10-30 robots (MINTOTALDIST could not finish most instances with 30% obstacles or 40+ robots in 200 seconds, the cutoff time). We used the first 5 of every 10 instances for each obstacle/robot combination. For each fixed number of obstacles, instances of different numbers of robots are combined. The result is listed in Table III. We allow MINTOTALDIST to run for at most 200 seconds per instance. Note that unlike MINMAKESPAN, even when MINTOTALDIST does not find the optimal solution, it generally produces feasible solution which sometimes is a near optimal solution. These are included in the result. “Time” entries are average time, in seconds, used by MINTOTALDIST. “Disjoint” entries are the average path lengths for all robots if we were to plan each shortest path ingoring other robots. The distance optimal solutions must produce a length no less than this. The next two lines are average path lengths from MINMAKESPAN and MINTOTALDIST algorithms. As we can see, MINMAKESPAN alone yields path length 50% than optimal; MINTOTALDIST, on the other hand, provided time optimal solutions that are near distance optimal ($< 1\%$ difference). For more than half of the instances, MINTOTALDIST produced true distance optimal solutions. In fact, MINTOTALDIST produced true distance optimal solutions for 42 out of the 45 instances with 5-15% obstacles.

TABLE III

	% obs				
	5	10	15	20	25
Time	26.3	23.3	42.7	57.2	81.6
Disjoint	12.20	11.75	12.03	12.80	12.84
MINTOTALDIST	12.20	11.75	12.05	12.85	12.92
MINMAKESPAN	16.47	16.60	17.59	18.83	19.33

D. *Using* TOMPP *as a generic heuristic*

In the last experiment, we exploit MINMAKESPAN as a *generic* heuristic for locally resolving path conflicts for large problem instances. By *generic*, we mean that the heuristic is not coded to any specific robot/grid setting. In our algorithm, paths are first planned for single robots (ignoring other robots). Afterwards, the robots are moved along these paths until no further progress can be made. We then detect on the graph where progress are stalled and resolve the conflict locally using MINMAKESPAN. For every conflict, we apply MINMAKESPAN to its neighborhood of distance 2. The above steps are repeated until a solution is found. The process can be made into a complete algorithm by allowing the local neighborhood to grow gradually. For evaluation, we ran the above algorithm on a 32×32 grid with 20% obstacles. We allow each instance to run a maximum of 30 seconds. The results, each as an average over 100 runs for a certain number of robots, are listed in Table IV (keep in mind that our implementation is Java based, which should see a speedup if implemented in C++). While we did not make side-by-side comparisons with the literature due to (seemingly small but) important differences in problem formulation, the computation time and completion rate of our algorithm appear comparable with the state of the art results from other authors.

TABLE IV

	Number of Robots					
	25	50	75	100	125	150
Running time (s)	0.04	0.15	0.32	1.37	3.85	10.3
Fully solved	100	100	100	100	98	95
% goals reached	100.0	100.0	100.0	100.0	99.4	98.6

VII. CONCLUSION AND OPEN PROBLEMS

In this paper, we introduced a multiflow based ILP algorithm for planning optimal, collision-free paths for multiple robots on graphs. We provided complete ILP algorithms for solving time optimal and distance optimal MPP problems. Our experiments confirmed that MINMAKESPAN is a feasible method for planning time optimal paths for tightly coupled problems as well as for larger problems with more free space. Moreover, we showed that MINMAKESPAN can serve as a good heuristic for solving large problem instances efficiently. For distance optimality, MINTOTALDIST, when combined with MINMAKESPAN, produces time optimal solutions that are often near distance optimal.

Many interesting open problems on optimal MPP remain; we mention two here. First, the ILP algorithms have ample room for performance improvements. On one hand, the ILP model can be make leaner. For example, it is clear that some $x_{i,j}$'s will never be set to 1; these should be removed from the model. On the other hand, our application of the Gurobi solver is fairly rudimentary - we simply feed the model to the solver as a mixed integer program (MIP) without specifying any other optimization options. Therefore, it would not be surprising that tuning the parameters of the solver greatly improves its performance on MPP problems. Secondly, while MINMAKESPAN could solve hard MPP problems such as the 25-puzzle, ILP solvers are nevertheless not tailored for such problems. Thus, we expect that tailored methods, such as heuristic based search, to solve problems like n^2-puzzles even faster. Looking closely at how ILP solvers work on these problems should provide insights that help building these heuristics.

REFERENCES

[1] J. E. Aronson. A survey on dynamic network flows. *Annals of Operations Research*, 20(1):1–66, 1989.

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

[3] 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*. MIT Press, Cambridge, MA, 2005.

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

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

[6] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autom. Robots*, 8(3):325–344, June 2000.

[7] D. Halperin, J.-C. Latombe, and R. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26(3-4):577–601, 2000.

[8] P. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

[9] J. S. Jennings, G. Whelan, and W. F. Evans. Cooperative search and rescue with a team of mobile robots. In *Proceedings IEEE International Conference on Robotics & Automation*, 1997.

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

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

[12] S. M. LaValle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE Trans. on Robotics and Automation*, 14(6):912–925, December 1998.

[13] 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*, pages 294–300, 2011.

[14] M. J. Matarić, M. Nilsson, and K. T. Simsarian. Cooperative multi-robot box pushing. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 556–561, 1995.

[15] B. Nnaji. *Theory of Automatic Robot Assembly and Programming*. Chapman & Hall, 1992.

[16] S. Poduri and G. S. Sukhatme. Constrained coverage for mobile sensor networks. In *Proceedings IEEE International Conference on Robotics & Automation*, 2004.

[17] D. Ratner and M. Warmuth. The (n^2-1)-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10:111–137, 1990.

[18] S. Rodriguez and N. M. Amato. Behavior-based evacuation planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 350–355, 2010.

[19] D. Rus, B. Donald, and J. Jennings. Moving furniture with teams of autonomous robots. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 235–242, 1995.

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

[21] B. Shucker, T. Murphey, and J. K. Bennett. Switching rules for decentralized control with simple control laws. In *American Control Conference*, July 2007.

[22] D. Silver. Cooperative pathfinding. In *The 1st Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 23–28, 2005.

[23] B. Smith, M. Egerstedt, and A. Howard. Automatic generation of persistent formations for multi-agent networks under range constraints. *ACM/Springer Mobile Networks and Applications Journal*, 14(3):322–335, June 2009.

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

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

[26] P. Surynek. An optimization variant of multi-robot path planning is intractable. In *The Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 1261–1263, 2010.

[27] H. Tanner, G. Pappas, and V. Kumar. Leader-to-formation stability. *IEEE Transactions on Robotics and Automation*, 20(3):443–455, Jun 2004.

[28] 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.

[29] 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.

[30] J. Yu. Diameters of permutation groups on graphs and linear time feasibility test of pebble motion problems. *arXiv:1205.5263*, 2012.

[31] J. Yu and S. M. LaValle. Multi-agent path planning and network flow. In *The Tenth International Workshop on Algorithmic Foundations of Robotics*, 2012.

[32] A. Zelinsky. A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(6):707–717, 1992.