

# Optimal Multi-Robot Path Planning on Graphs: Complete Algorithms and Effective Heuristics

Jingjin Yu      Steven M. LaValle

## Abstract

We study the problem of optimal multi-robot path planning on graphs (MPP) over four distinct minimization objectives: the makespan (last arrival time), the maximum (single-robot traveled) distance, the total arrival time, and the total distance. In a related paper Yu and LaValle (2015), we show that these objectives are distinct and NP-hard to optimize. In this work, we focus on efficiently algorithmic solutions for solving these optimal MPP problems. Toward this goal, we first establish a one-to-one solution mapping between MPP and network-flow. Based on this equivalence and integer linear programming (ILP), we design novel and complete algorithms for optimizing over each of the four objectives. In particular, our exact algorithm for computing optimal makespan solutions is a first such that is capable of solving extremely challenging problems with robot-vertex ratio as high as 100%. Then, we further improve the computational performance of these exact algorithms through the introduction of principled heuristics, at the expense of some optimality loss. The combination of ILP model based algorithms and the heuristics proves to be highly effective, allowing the computation of  $1.x$ -optimal solutions for problems containing hundreds of robots, densely populated in the environment, often in just seconds.

## I. INTRODUCTION

In this paper, we study the problem of optimal *multi-robot path planning on graphs* (MPP), focusing on the design of *complete algorithms and effective heuristics*. In an MPP instance, the robots are uniquely labeled (*i.e.*, distinguishable) and are confined to an arbitrary connected graph. The robots may move from a vertex to an adjacent one in one time step in the absence of collision, which may occur when two robots simultaneously move to the same vertex or along the same edge in different directions. A distinguishing feature of our MPP formulation is that we allow robots on fully occupied cycles to rotate synchronously. Such a formulation, more appropriate for multi-robot applications, has not been widely studied (except, *e.g.*, Standley (2010); Standley and Korf (2011)). Over the basic MPP formulation, we look at four commonly studied minimization objectives: the makespan (last arrival time), the maximum (single-robot traveled) distance, the total arrival time, and the total distance. We note that these global objectives have direct relevance toward real world multi-robot applications such as autonomous warehouse systems Wurman et al. (2008). For example, minimizing makespan is equivalent to minimizing the task completion time, whereas minimizing total distance is applicable to minimizing the fuel consumption of the entire fleet of robots.

In a related paper Yu and LaValle (2015), we show that these objectives are all distinct and NP-hard to optimize, suggesting that efforts on solving optimal MPP should be directed at finding effective, near-optimal algorithms. In this work, we make an attempt toward this goal and propose a novel yet general framework for solving MPP optimally. By examining space and time dimensions jointly, we observe a one-to-one mapping between a solution for an MPP instance and that for a multi-commodity flow problem derived from the MPP problem. Based on the equivalence relationship, we can then translate the MPP problem into an integer linear programming (ILP) model, which can be subsequently solved using an ILP solver. The generality of ILP allows the encoding of all four objectives to yield complete algorithms for optimizing these objectives. From here, we take a further step and introduce several heuristics to boost the algorithmic performance at a slight loss of solution optimality. Our method is especially effective in computing near-optimal minimum makespan solutions, capable of computing  $1.x$ -optimal solutions for hundreds of robots, densely populated on the underlying graph, often in just seconds.

**Related work.** Multi-robot path planning problems, in its many formulations, have been actively studied for decades Erdmann and Lozano-Pérez (1986); Zelinsky (1992); LaValle and Hutchinson (1998); Silver (2005); Kloder and Hutchinson (2006); Ryan (2008); Jansen and Sturtevant (2008); Surynek (2009); Luna and Bekris (2011); Standley and Korf (2011); van den Berg and Overmars (2005); van den Berg et al. (2009); Solovey and Halperin (2012); Yu and LaValle (2013a); Turpin et al. (2014); Solovey et al. (2015). As a universal subroutine, collision-free path planning for multiple robots finds applications in tasks spanning assembly Halperin et al. (2000); Nnaji (1992), evacuation Rodriguez and Amato (2010), formation control Balch and Arkin (1998); Poduri and Sukhatme (2004); Shucker et al. (2007); Smith et al. (2009); Tanner et al. (2004), localization Fox et al. (2000), micro droplet manipulation Ding et al. (2001); Griffith and Akella (2005), object transportation Mataric et al. (1995); Rus et al. (1995), search and rescue Jennings et al. (1997), and so on. We refer the readers to Choset et al. (2005); Latombe (1991); LaValle (2006) and the references therein for a more comprehensive review on the general subject of multi-robot path and motion planning.

The algorithmic study of graph-based multi-robot path planning problems, the focus of this work, can be traced to as least 1879 Story (1879), in which Story makes the observation that the feasibility of the 15-puzzle Loyd (1959) is decided by the parity of the game. The 15-puzzle is a restricted MPP instance moving 15 labeled game pieces on a  $4 \times 4$  grid, from some initial configuration to some goal configuration. The restriction is that only a single game piece near the only empty vertex may move to the empty vertex in a step—multiple mobile robots could move simultaneously. A generalization of the 15-puzzle is introduced in Wilson (1974), extending the problem from 15 game pieces on a  $4 \times 4$  grid to  $n - 1$  labeled *pebbles* on an  $n$ -vertex, 2-connected graph. It is shown, together with an implied algorithm, that an instance is always feasible if the graph is non-bipartite. When the graph is bipartite (such as the 15-puzzle), all pebble configurations are split into two groups of equal size such that any two configurations in the same group form a feasible instance. A further generalization is introduced in Kornhauser et al. (1984), allowing  $p < n$  pebbles on a graph with  $n$  vertices. For this problem, an  $O(n^3)$  algorithm is provided to solve an instance or decide that the instance is infeasible.

As computer games and multi-robot systems gain popularity, concurrent movements are introduced and pebbles are replaced with robots (or agents). On the feasibility side, the MPP problem studied in this paper is shown to be solvable also in  $O(n^3)$  time Yu and Rus (2014). To distinguish the formulations, we denote the formulation that does not allow cyclic rotations of robots along fully occupied cycles as *cycle-free* MPP. Until recently, the majority of algorithmic study on MPP is on the cycle-free case. Since the problem is shown to be tractable Kornhauser et al. (1984), most algorithmic study of cycle-free MPP put some emphasis on optimality. Through the clever use of primitive operations, algorithms from Ryan (2008); Surynek (2009); Luna and Bekris (2011); Sajid et al. (2012); de Wilde et al. (2014) could quickly solve difficult problems with some form of completeness guarantees. These algorithms do not have optimality guarantees but the produced solutions are often of much better quality than the  $O(n^3)$  bound given by Kornhauser et al. (1984). For more discussion and references on sub-optimal methods, see Sajid et al. (2012); de Wilde et al. (2014).

Pushing more toward the optimality side, most algorithmic results explore ways to limit the exponential search space growth induced by multiple robots. One of the first such algorithms, Local Repair A\* (LRA\*) Zelinsky (1992), plans robot paths simultaneously and performs local repairs when conflicts arise. Focusing on fixing the (locality) shortcomings of LRA\*, Windowed Hierarchical Cooperative A\* (WHCA\*) proposes using a space-time window to allow more choices for resolving local conflicts while simultaneously limiting the search space size Silver (2005). Iterative deepening technique is shown to be applicable to MPP problems in Sharon et al. (2012). A technique called sub-dimensional expansion is shown to perform well in complex environment Wagner and Choset (2011); the robot density is however relatively low (104 cells per robot according to the paper). In Standley (2010); Standley and Korf (2011), instead of agnostic dissection of an instance, the natural idea of independence detection (ID) is explored to only consider multiple robots jointly (the source of exponential search space growth) as necessary. With operator decomposition (OD) that treats each legal move as an “operator”, the authors produced algorithms (ID,OD+ID, and related variants) that prove to be quite effective in computing total time- or distance-optimal solutions. We point out that ID and OD+ID have support for handling cycles (*i.e.*, they apply to MPP instead of cycle-free MPP). Algorithms designed for minimizing makespan have also been attempted, *e.g.*, Surynek (2012), but the solution quality degrades rapidly as the robot-vertex ratio increases.

Many approaches have also been proposed for solving multi-robot path planning problems in the continuous domain. A representative method called velocity-obstacles Kant and Zucker (1986); van den Berg et al. (2008, 2011) explicitly examines velocity-time space for coordinating robot motions. In Griffith and Akella (2005), mixed integer programming (MIP) models are employed to encode the robot interactions. A method based on the space-time perspective, similar to ours, is explored in I. Karamouzas (2012). In Peasgood et al. (2008), an A\*-based search is performed over a discrete roadmap abstracted from the continuous environment. In Solovey et al. (2014), discrete-RRT (d-RRT) is proposed for the efficient search of multi-robot roadmaps. Algorithms for discrete MPP, cycle-free or not, have also helped solving continuous problems Solovey and Halperin (2012); Krontiris et al. (2012).

**Contributions.** We study the optimal MPP formulation allowing up to  $n$  robots on a  $n$ -vertex connected graph, which we believe is better suited for multi-robot applications. The formulation is not widely studied, perhaps due to the inherent difficulty in handling cyclic rotations of robots. Besides the novelty of the problem, this work brings several algorithmic contributions. First, based on the equivalence relationship between MPP and network flow, we establish a general and novel solution framework allowing the compact encoding of optimal MPP problems using ILP models. We show that the framework readily produces complete algorithms for minimizing the makespan (last arrival time), the maximum (single-robot traveled) distance, the total arrival time, and the total distance, which are perhaps four most common global objectives for MPP. The resulting algorithms, in particular the one for computing minimum makespan, are highly effective in computing challenging problems instances with robot-vertex ratio up to 100%. Second, we introduce several principled heuristics, in particular a  $k$ -way split heuristic that divides an MPP instance over the time horizon, to give the exact algorithms a sizable performance boost at the expense of some solution optimality loss. With these heuristics, we are able to extend our algorithms to tackle problems with several hundred robots that are extremely densely populated, while at the same time maintain  $1.x$  solution optimality. Last but not least, our successful exploit of ILP to attack optimal MPP shows that integer linear programming method is competitive with direct search methods in this problem domain, especially when the number of robots becomes large. This is surprising because ILP solvers are not designed specifically for MPP.

The rest of the paper is organized as follows. In Section II, we define the optimal MPP problems and provide background material on network flow. We then establish the equivalence relationship between MPP and multi-commodity flow in Section III. We derive complete algorithms in Section IV based on the equivalence and continue to describe the performance boosting heuristics in Section V. We evaluate the algorithms in Section VI and conclude in Section VII. This paper is partly based on Yu and LaValle (2013a,b,c).<sup>1</sup> In comparison to Yu and LaValle (2013a,b), besides demonstrating significantly improved computational performance due to the addition of the  $k$ -way split heuristic, we have substantially extended the generality of our ILP-based algorithmic framework, which now supports all common global time- and distance-based objectives.

## II. PRELIMINARIES

We now define MPP and the optimality objectives studied in this paper. Following the problem statements, we gave a brief review on *network flow*.

### A. Multi-Robot Path Planning on Graphs

Let  $G = (V, E)$  be a connected, undirected, simple graph, with  $V = \{v_i\}$  being the vertex set and  $E = \{(v_i, v_j)\}$  the edge set. Let  $R = \{r_1, \dots, r_n\}$  be a set of  $n$  robots. The robots move at discrete time steps (*i.e.*, at  $t = 0, 1, \dots$ ). At time step  $t = 0$ , each robot occupies a distinct vertex of  $G$ . In general, at any time step  $t = 0, 1, \dots$ , the robots assume a *configuration* that is an injective map from  $R$  to  $V$ . The *start (initial)* and *goal* configurations of the robots are denoted as  $x_I$  and  $x_G$ , respectively. Fig. 1(a) shows a possible configuration of 9 robots on a  $3 \times 3$  grid graph. Fig. 1(b) shows a possible goal configuration, in which the robots are ordered based on *row-major ordering*<sup>2</sup>.

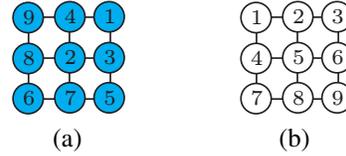


Fig. 1. a) A 9-puzzle problem. b) The desired goal configuration.

During a discrete time step, each robot may either remain stationary or move to an adjacent vertex. To formally describe a plan, let a *scheduled path* be a map  $p_i : \mathbb{Z}^+ \rightarrow V$ , in which  $\mathbb{Z}^+ := \mathbb{N} \cup \{0\}$ . A scheduled path  $p_i$  is *feasible* if it satisfies the following properties: 1)  $p_i(0) = x_I(r_i)$ . 2) For each  $i$ , there exists a smallest  $t_i \in \mathbb{Z}^+$  such that  $p_i(t_i) = x_G(r_i)$ . 3) For any  $t \geq t_i$ ,  $p_i(t) \equiv x_G(r_i)$ . 4) For any  $0 \leq t < t_i$ ,  $(p_i(t), p_i(t+1)) \in E$  or  $p_i(t) = p_i(t+1)$  (if  $p_i(t) = p_i(t+1)$ , robot  $r_i$  stays at vertex  $p_i(t)$  between the time steps  $t$  and  $t+1$ ). We say that two paths  $p_i, p_j$  are in *collision* if there exists  $k \in \mathbb{Z}^+$  such that  $p_i(t) = p_j(t)$  (meet collision) or  $(p_i(t), p_i(t+1)) = (p_j(t+1), p_j(t))$  (head-on collision). As an illustration, Fig. 2 shows the feasible and infeasible moves for two robots during a single time step<sup>3</sup>. The *multi-robot path planning on graph* (MPP) problem is defined as follows.

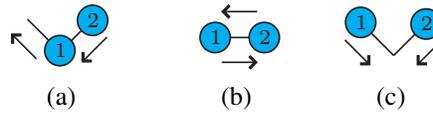


Fig. 2. Some feasible and infeasible moves for two robots. a) A feasible synchronous move. b) An infeasible synchronous move in which two robot collide “head-on”. c) An infeasible synchronous move in which two robots “meet” at a vertex.

**Problem 1 (Multi-robot Path Planning on Graphs)** Given a 4-tuple  $(G, R, x_I, x_G)$ , find a set of paths  $P = \{p_1, \dots, p_n\}$  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.

For example, Fig. 1(a) and Fig. 1(b) define an MPP problem on the  $3 \times 3$  grid. We call this particular problem the *9-puzzle* problem (a variant of the 15-puzzle Ratner and Warmuth (1990)), which readily generalizes to  $N^2$ -puzzles.

**Remark.** With a few exceptions (*e.g.*, Standley and Korf (2011)), most existing studies on discrete multi-robot path planning problems require empty vertices as swap spaces. In these formulations, in a time step, a non-intersecting chain of robots may move simultaneously only if the head of the chain is moving into a previously unoccupied vertex. In contrast, our MPP

<sup>1</sup>Yu and LaValle (2013c) is a preliminary poster presentation.

<sup>2</sup>In this paper, we generally use shaded discs to mark start locations of robots and discs without shading for goal locations.

<sup>3</sup>We assume that the graph  $G$  only allows “meet” or “head-on” collisions. The assumption is mild. For example, a (arbitrary dimensional) grid with unit edge distance is such a graph for robots of with radii of no more than  $\sqrt{2}/4$ .

formulation allows synchronized rotations of robots along fully occupied cycles (see, *e.g.*, Fig. 1 and 3). This implies that even when the number of robots equals the number of vertices, robots can still move on disjoint cycles. We note that MPP can be solved in polynomial time with feasibility test taking only linear time Yu and Rus (2014).  $\triangle$

### B. Optimal Formulations

Let  $P = \{p_1, \dots, p_n\}$  be an arbitrary feasible solution to some fixed MPP instance. For a path  $p_i \in P$ , let  $\text{len}(p_i)$  denote the length of the path  $p_i$ , which is increased by one each time when the robot  $r_i$  passes an edge. A robot, following a path  $p_i$ , may visit the same edge multiple times. Recall that  $t_i$  denotes the arrival time of robot  $r_i$ . In the study of optimal MPP formulations, we examine four common objectives with two focusing on time optimality and two focusing on distance optimality. Below, each objective is formally defined.

**Objective 1 (Makespan)** Compute a path set  $P$  that minimizes  $\max_{1 \leq i \leq n} t_i$ .

**Objective 2 (Maximum Distance)** Compute a path set  $P$  that minimizes  $\max_{1 \leq i \leq n} \text{len}(p_i)$ .

**Objective 3 (Total Arrival Time)** Compute a path set  $P$  that minimizes  $\sum_{i=1}^n t_i$ .

**Objective 4 (Total Distance)** Compute a path set  $P$  that minimizes  $\sum_{i=1}^n \text{len}(p_i)$ .

The intuitive meaning of these objectives is clear. Fig. 3 illustrates the four-step minimum makespan solution to the 9-puzzle problem from Fig. 1. The solution is optimal as it takes at least four steps for robot 9 to move to its goal.

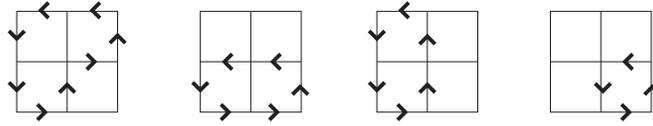


Fig. 3. A 4-step solution from our algorithm. The directed edges show the moving directions of the robots at the tail of the edges.

### C. Network Flow Review

A network  $\mathcal{N} = (G, c_1, c_2, S)$  consists of a directed graph  $G = (V, E)$  with  $c_1, c_2 : E \rightarrow \mathbb{Z}^+$  as the maps specifying capacities and costs over edges, respectively, and  $S \subset V$  as the set of sources and sinks. Let  $S = S^+ \cup S^-$ , with  $S^+$  denoting the set of source vertices,  $S^-$  denoting the set of sink vertices, and  $S^+ \cap S^- = \emptyset$ . 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  $f : E \rightarrow \mathbb{Z}^+$  that satisfies edge capacity constraints,

$$\forall e \in E, \quad f(e) \leq c_1(e), \quad (1)$$

the flow conservation constraints at non terminal vertices,

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

and the flow conservation constraints at terminal vertices,

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

The quantity  $F(f)$  is called the *value* of the flow  $f$ . The classic (single-commodity) *maximum flow* problem asks the following 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 a minimum total cost among all maximum flows. That is, we want to find a flow among all maximum flows that also minimizes the quantity

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

The network flow formulation described so far only considers a *single commodity*, corresponding to all robots being interchangeable. For general MPP formulations, the robots are distinct and must be treated as different commodities. Such

problems can be captured with the *Multi-commodity flow* or simply *multiflow*. 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

$$\forall i, \forall e \in E, \quad \sum_i f_i(e) \leq c_1(e), \quad (5)$$

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

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

Maximum flow and minimum cost flow problems may also be posed under a multiflow setup; we omit the details. Our review on network flow only touches aspects pertinent to the current work; for a thorough coverage on the subject of network flow, see Aronson (1989); Ahuja et al. (1993) and the references therein. Note that the multiflow model stated here is sometimes also referred to as *integer multiflow* because  $f_i$  must have integer values.

### III. FROM MULTI-ROBOT PATH PLANNING TO MULTIFLOW

A close algorithmic connection exists between optimal MPP and network flow problems. Maximum (single commodity) flow problems generally admit efficient (low-degree polynomial time) algorithmic solutions Ahuja et al. (1993), whereas maximum multiflow is a well-known NP-hard problem, difficult to even approximate Andrews and Zhang (2005). Mirroring the disparity between single- and multi-commodity flows, in the domain of MPP problems, if there is a single group of interchangeable robots (here, for a group, it does not matter which robot goes to which goal as long as all goal locations assigned to the group are occupied by robots from the same group), then many optimal formulations admit polynomial time algorithms Yu and LaValle (2013a). However, as soon as a single group of robots splits into two or more groups, finding optimal paths for these robots become intractable Yu and LaValle (2015). The apparent similarity between optimal MPP and multiflow is perhaps best explained through a graph-based reduction from MPP problems to network flow problems. The reduction will also form the basis of our algorithmic solution.

To describe the reduction, we use as an example the undirected graph  $G$  in Fig. 4(a), with start vertices  $\{s_i^+\}, i = 1, 2$  and goal vertices  $\{s_i^-\}, i = 1, 2$ . An instance of MPP is given by  $(G, \{r_1, r_2\}, x_I : r_i \mapsto s_i^+, x_G : r_i \mapsto s_i^-)$ . We will reduce the problem to a network flow problem  $\mathcal{N} = (G', c_1, c_2, S)$ . The reduction proceeds by constructing a network that is a *time-*

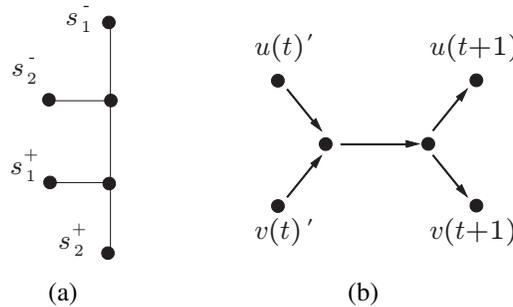


Fig. 4. a) A simple  $G$ . b) A *merge-split* gadget for splitting an undirected edge through time steps, for enforcing the head-on collision constraint.

*expanded* version of the graph  $G$ , which then allows the explicit consideration of the interactions among the robots over space *and* time. To carry out this expansion, a time horizon must first be decided. For different optimality objectives, the expansion time horizon, some natural number  $T$ , is generally different; for now we assume that  $T$  is fixed.

To begin building the network, we create  $2T + 1$  copies of  $G$ 's vertices, with indices  $0, 1, 1', \dots$ , as shown in Fig. 5. For each vertex  $v \in G$ , denote these copies  $v(0) = v(0)', v(1), v(1)', v(2), \dots, v(T)'$ . For each edge  $(u, v) \in G$  and time steps  $t, t + 1, 0 \leq t < T$ , the *merge-split* gadget shown in Fig. 4(b) is added 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. The merge-split gadget ensures that two robots cannot travel in opposite directions on an edge in the same time step, which prevents head-on collision between two robots. 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  $(v(0), v(1)), (v(1), v(1)'), \dots, (v(T), v(T)')$ ). 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. The green edges allow

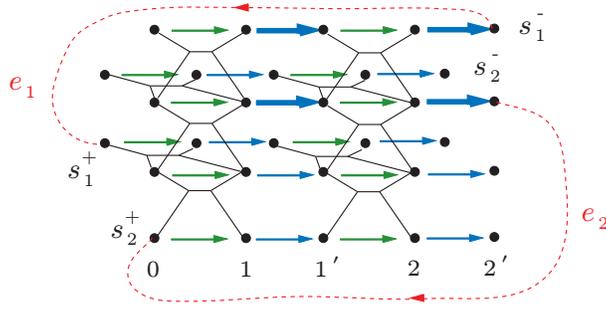


Fig. 5. The time-expanded network with an expansion time horizon of  $T = 2$  over the base graph Fig. 4(a).

robots to stay at a vertex during a time step whereas blue edges ensure that each vertex holds at most one robot, enforcing the meet collision constraint.

Fig. 5 (with the exception of edges  $e_1$  and  $e_2$ , which will become relevant shortly), the time-expanded network, is the desired  $G'$ . For the set  $S = S^+ \cup S^-$ , we may simply let  $S^+ = \{v(0) : v \in \{s_i^+\}\}$  and  $S^- = \{v(T) : v \in \{s_i^-\}\}$ .  $\mathcal{N} = (G', c_1, c_2, S)$  is now complete; we have reduced MPP to an integer multiflow problem on  $\mathcal{N}$  with each robot from  $R$  as a single type of commodity.

**Theorem 1** *Let  $(G, R, x_I, x_G)$  be an MPP instance. There is a bijection between its solution set (with a 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  $P = \{p_1, \dots, p_n\}$  ( $n$  is the number of robots) is a solution to an instance of MPP. For each  $p_i$  and every time step  $t = 0, \dots, T$ , 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 a unique 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  $\{f_1, \dots, f_n\}$  is feasible and maximal on  $\mathcal{N}$ .

**Surjectivity.** Assume that  $\{f_1, \dots, f_n\}$  is an integer maximum multiflow on the network  $\mathcal{N}$  with  $|f_i| = 1$  for all  $i$ 's. 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  $\{p_1, \dots, p_n\}$  is then a solution to MPP.  $\square$

**Remark.** A multiflow problem can be reduced to an MPP problem (on a directed graph) as well. In particular, if all edges in the network have unit capacity and there is a single unit of each type of commodity, then a multiflow problem is a multi-robot path planning problem, often known as the *edge disjoint path* problem Robertson and Seymour (1995).  $\triangle$

#### IV. COMPLETE, INTEGER LINEAR PROGRAMMING-BASED ALGORITHMS FOR OPTIMAL MPP PROBLEMS

Because optimizing MPP solutions over Objectives 1-4 are computationally intractable, reducing MPP to multiflow problems does not make these optimal MPP problems any easier. However, with a network flow formulation (see Section II-C), it becomes possible to establish integer linear programming (ILP) models for optimal MPP formulations. These ILP models can then be solved with powerful linear programming packages. In comparison to A\*-based algorithms augmented with heuristics which often target important but a limit set of problem structures, ILP-based algorithms proposed here are *agnostic* to specific problem structures. As such, ILP-based algorithms appear more capable at attacking a wider range of MPP problems and in particular difficult MPP instances in which the robot-vertex ratio is high. In this section, we build ILP models for each of Objectives 1-4.

##### A. Minimizing the Makespan

A minimum makespan solution to an MPP instance  $I = (G, R, x_I, x_G)$  can be computed using a *maximum multiflow* formulation. Fixing a time span  $T$ , let  $\mathcal{N} = (G', c_1, c_2, S)$  be the time-expanded network for  $I$ , a set of  $n$  *loopback* edges are

added to  $G'$  by connecting each pair of corresponding start and goal vertices in  $S$ , from the goal to the start. We use  $e_j$ 's to denote edges of  $G'$  and let the  $n$  loopback edges take the first  $n$  indices, with  $e_j, 1 \leq j \leq n$  being the edge connecting the goal vertex of  $r_j$  to the start vertex of  $r_j$ . For example, for the  $G'$  in Fig. 5,  $e_1$  and  $e_2$  are the loopback edges for  $r_1$  and  $r_2$ , respectively. The lookback edges have unit capacities and zero costs. Next, for each edge  $e_j \in G'$  (including the lookback edges), create  $n$  binary variables  $x_{1,j}, \dots, x_{n,j}$  corresponding to the flow through that edge, one for each robot. That is,  $x_{i,j} = 1$  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,

$$\forall e_j, \quad \sum_{i=1}^n x_{i,j} \leq 1, \quad (8)$$

$$\forall 1 \leq i, j \leq n, i \neq j, \quad x_{i,j} = 0,$$

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

The objective function is

$$\max \sum_{1 \leq i \leq n} x_{i,i}. \quad (10)$$

For each fixed  $T$ , a solution to the above ILP model with an objective equaling  $n$  means that a feasible solution to MPP is found. We are to find the minimal  $T$  that yields such a feasible solution. To do this, start with  $T$  being the maximum over all robots the shortest possible path length for each robot, ignoring all other robots. An ILP model for this  $T$  is then built and tested for a feasible solution. If the model is not feasible,  $T$  is increased and the procedure repeated. The first feasible  $T$  is the optimal  $T$ . Once a feasible model with the minimum expansion time horizon  $T$  is found, the robots' paths can be extracted based on the proof of Theorem 1. The algorithm is in fact 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 ( $T = O(|V|^3)$  is big enough Yu and Rus (2014)). Denoting the resulting algorithm as MINMAKESPAN we have shown the following.

**Proposition 2** *Algorithm MINMAKESPAN is a complete algorithm for finding minimum makespan solutions for MPP.*

### B. Minimizing the Maximum Single-Robot Traveled Distance

For minimizing the maximum distance traveled by any single robot, the network and variable creation remains the same as the minimum makespan setup; constraints (8) and (9) remain unchanged. Because we want to send all robots to their goal, a maximum flow may be forced through the additional constraint

$$\forall 1 \leq i \leq n, \quad x_{i,i} = 1. \quad (11)$$

To encode the min-max objective function, we introduce an additional integer variable  $x_{max}$  and add the constraint

$$\sum_{e_j \in G', j > n} c_2(e_j) \cdot x_{i,j} \leq x_{max} \quad (12)$$

for all  $1 \leq i \leq n$ . For a fixed  $i$ , the left side of (12) represents the distance traveled by robot  $r_i$ . The objective function is then simply

$$\min x_{max}. \quad (13)$$

Denoting the algorithm as MINMAXDIST, we have

**Proposition 3** *Algorithm MINMAXDIST is a complete algorithm for finding solution to MPP that minimize the maximum distance traveled by a single robot.*

**Remark.** Assuming that we have a minimum makespan solution, we can better bound the time horizon  $T$  needed for MINMAXDIST. Let  $t_{min}$  be the minimum makespan computed by MINMAKESPAN, setting  $T = nt_{min}$  is then sufficient for finding the min  $x_{max}$ . To see that this is true, we first note that  $\min x_{max} \leq t_{min}$  because the minimum makespan solution requires robots to synchronize their moves, which may force some robots to travel unnecessarily. Then, the  $n$  robots cannot move a total distance of more than  $n \min x_{max} \leq nt_{min}$  because no robot may travel more than  $\min x_{max}$  edges and therefore cannot use more than  $n \min x_{max}$  time in total.  $\triangle$

### C. Minimizing the Total Arrival Time

The ILP model for minimizing the total arrival time is more involved in the way the objective function is constructed. First, the network and all variables from the minimum makespan ILP-model are inherited. We also inherit constraints (8), (9), and (11). To represent the objective function, for each time step  $1 \leq t \leq T$  and each  $v := x_G(r_i)$ ,  $1 \leq i \leq n$ , we create a binary variable  $y_i^t$ . Then, we give new indices to some variables that are already created. Recall that for each edge  $e_j = (v(t), v(t')) \in G'$  (e.g., the four extra bold blue edges in Fig. 5), a variable  $x_{i,j}$  is created. There are a total of  $nT$  such variables. Here, we give these variable a second index  $x_i^t$ . That is,  $x_i^t$  is the binary variable indicating whether edge  $(v(t), v(t')) \in G'$ ,  $v := x_G(r_i)$ , is used by robot  $r_i$ .

Given a network with a fixed  $T$ , if constraints (8), (9), and (11) can be satisfied, then there is a feasible solution to the original MPP problem. In this case,  $x_i^T \equiv 1$  for  $1 \leq i \leq n$ . We let  $y_i^T = x_i^T$ . Then, each  $y_i^t$ ,  $1 \leq t < T$  is defined recursively over  $x_i^t$  and  $y_i^{t+1}$  as

$$y_i^t \geq y_i^{t+1} + x_i^t - 1, \quad y_i^t \leq y_i^{t+1}, \quad y_i^t \leq x_i^t. \quad (14)$$

The constraint (14) effectively performs the logical *and* operation over  $x_i^t$  and  $y_i^{t+1}$  and stores the result in  $y_i^t$ . In the end, the smallest  $t$  for which  $y_i^t = 1$  is the time robot  $r_i$  reaches its goal (and stops). Therefore, for each  $1 \leq i \leq n$ ,  $\sum_{t=1}^T y_i^t$  is the number of time steps from the time  $r_i$  arrives at its goal until time  $T$ .  $T - \sum_{t=1}^T y_i^t$  is then the time spent by  $r_i$ . To minimize the total arrival time, the objective function can be expressed as

$$nT - \sum_{1 \leq i \leq n, 1 \leq t \leq T} y_i^t. \quad (15)$$

Denoting the resulting algorithm (with a sufficiently large  $T$ ) as `MINTOTALTIME`, we have

**Proposition 4** *Algorithm `MINTOTALTIME` is a complete algorithm for finding minimum total arrival time solutions for MPP.*

### D. Minimizing the Total Distance

From the ILP model for minimizing the maximum distance, we only need to change the objective function for computing a minimum total distance solution. We do not need the variable  $x_{max}$  and simply update the objective function to

$$\min \sum_{e_j \in G', j > n, 1 \leq i \leq n} c_2(e_j) \cdot x_{i,j}. \quad (16)$$

Denoting the algorithm as `MINTOTALDIST`, we have

**Proposition 5** *Algorithm `MINTOTALDIST` is a complete algorithm for finding minimum total distance solutions for MPP.*

Again,  $T = nt_{min}$  is sufficient for building a network that contains a minimum total distance solution, if one exists.

## V. HEURISTICS FOR EFFECTIVE COMPUTATION OF NEAR-OPTIMAL SOLUTIONS

In Section IV, in constructing the ILP models for Objectives 1-4, our goal is to show the universal applicability of the network flow model (e.g., Fig. 5) toward many different optimal MPP formulations. The resulting ILP-based algorithms are complete and always produce the optimal solution in principle. As will be shown in Section VI, such algorithms perform very well in handling relatively small but extremely challenging problems. Nevertheless, as the problem size grows (i.e., as the graph  $G$  and the number of robots  $n$  get larger), the computation time needed by ILP solvers grows rapidly. From a practical point of view, it may be far more desirable to quickly compute a good quality but sub-optimal solution than waiting forever for the optimal solution. In this section, we introduce several heuristics to accomplish just that, with a particular focus on computing solutions with minimum makespan.

### A. Building More Compact Models

To get the best performance out of a solver, it is beneficial to have a lean model (i.e., fewer columns and rows). So far, our focus has been to provide a general network flow based framework so that the ILP models can be easily built. When it comes to translating the models to an ILP solver, they can be further simplified. The heuristics discussed in this subsection aim at making the representation of the constraints (8) and (9) more compact in the resulting ILP models. As such, they apply to all the optimality objectives.

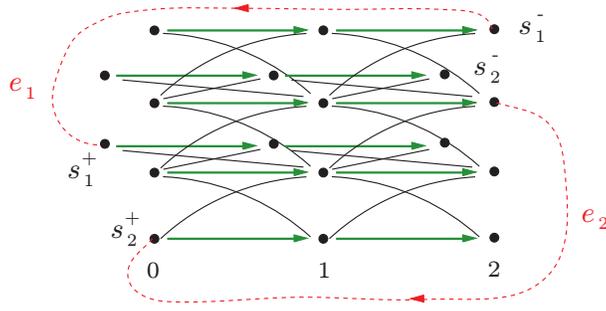


Fig. 6. A more compact representation of the network flow graph from Fig. 5.

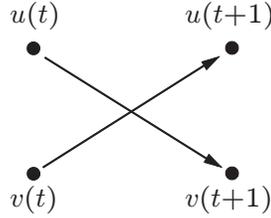


Fig. 7. The simplified *merge-split* gadget for enforcing the head-on collision constraint.

*Better encoding of the collision constraints:* Recall that in building the network flow model (e.g., Fig. 5), we used a merge-split gadget (Fig. 4(b)) for enforcing the head-on collision constraint and extra time steps (e.g., the blue edges in Fig. 5) for avoiding meet collisions. When we translate this into linear constraints, these structures can be simplified to yield the more compact structure illustrated in Fig. 6.

In the newer structure, each merge-split gadget now has two edges instead of five. Also, the blue edges are removed. The updated gadget for an edge  $(u, v) \in E$  between time steps  $t$  and  $t + 1$  is shown in Fig. 7 (note that due to the removal of the blue edges, vertices such as  $v(t)'$  is no longer needed). That is, instead of five, only two variables are needed for each robot. Denoting these binary variables as  $x_{i,(u(t),v(t+1))}$  and  $x_{i,(v(t),u(t+1))}$  for a robot  $r_i$ , the head-on collision constraint for a single gadget can be readily encoded as

$$\sum_{i=1}^n x_{i,(u(t),v(t+1))} + \sum_{i=1}^n x_{i,(v(t),u(t+1))} \leq 1. \quad (17)$$

Then, to enforce the meet constraint, for example at a vertex  $v(t)$ , we simply require that at most one outgoing edge from  $v(t)$  may be used, i.e.,

$$\sum_{e_j \in \delta^+(v(t)), 1 \leq i \leq n} x_{i,j} \leq 1. \quad (18)$$

Overall, the newer ILP model is roughly half of the size of the original model.

*Reachability analysis:* In the time-expanded graph, there are redundant binary (edge) variables that can never be true because some edges are never reachable. For example, in Fig. 6, at  $t = 0$ , the only outgoing edges that can possibly be used are those originates from  $s_1^+$  and  $s_2^+$ . The rest can be safely removed. In general, for each robot  $r_i$ , based on its reachability from its start vertex and to its goal vertex, a sizable number of binary variables  $x_{i,j}$ 's can be deleted.

### B. Divide-and-Conquer Over Time Domain

In evaluating the ILP model-based algorithm for optimal makespan computation, we observe that the ILP solver running time appears to grow exponentially as the size of the model grows. This prevents the algorithm from performing well over instances with more than a few tens of robots. The observation, while hampering the effectiveness of the exact algorithm, turns out to offer a useful insight toward a highly effective heuristic. We find that, when the overall size of the ILP model is relatively small and the robot-vertex ratio is not approaching 1, even when there are a large number of robots, the model usually does not present much challenge for ILP solvers. To apply the ILP model-based method to more challenging problems (e.g., solving problems with hundreds of robots quickly), we simply limit the size of the individual ILP model fed to the solver. One way to achieve this is through *divide-and-conquer over the time domain*. We use a simple example (see Fig. 8) to illustrate the idea.

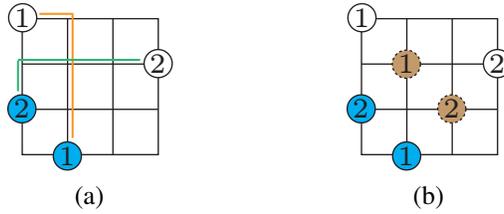


Fig. 8. a) A simple two-robot problem. b) The time-divided instances.

In Fig. 8(a), we have a simple planning problem for two robots on a  $3 \times 3$  grid. To execute the heuristic, we first compute a shortest path for every pair of start and goal locations. In this case, we get the orange and green paths for robots 1 and 2, respectively. Then, if we decide to split the problem into two smaller problems, for each of the paths, it is split into two (generally) equal length pieces and the middle node is set as the intermediate goal. In our example, we may do this for robot 1 by setting the intermediate goal location at  $(1, 1)$  from the top-left corner (the brown disc labeled 1 in Fig. 8(b)). For robot 2, because the middle location coincides with that of robot 1, we pick an alternative location that is not already occupied as the intermediate goal for robot 2, in this case  $(2, 2)$  from the top-left corner. The intermediate goals for the first instance will also serve as the start locations of the second instance. This yields two child instances with both requiring a time expansion with 2 steps each, effectively making the individual ILP model roughly half the size of the original one, which requires a time expansion with 4 steps. In general, we may divide a problem into arbitrarily many smaller instances in the time domain.

If a problem is divided in this manner to  $k$  sub problems, we call the resulting heuristic a  $k$ -way split. Because the division is over time, there is no interaction between the individual, smaller instances. Once we obtain the solution for each child instance, the solutions can be glued together by concatenating the results. In practice, this simple heuristic dramatically improves algorithm performance without heavy negative impact on path optimality in terms of makespan; we observe a consistent speedup in computational experiments.

**Remark.** The  $k$ -way split heuristic, by design, is particularly suitable for the makespan objective. This is the case due to the additive nature of the makespan objective over the split sub-problems. Besides makespan, the heuristic also applies to distance objectives (*i.e.*, Objectives 2 and 4) quite well, as long as the time horizon required for finding distance optimal solution does not differ greatly from the time horizon required for minimum makespan solution. The heuristic does not directly apply to Objective 3 because total time is not additive over the split sub-problems. As an example, suppose that a 2-way split is carried out with each sub-problem having a time horizon of  $T/2$ . If a robot  $r_i$  does not move in the solution to the first sub-problem (*i.e.*,  $0 \leq t \leq T/2$ ), it contributes 0 to the total distance. However, if  $r_i$  moves even a single step in the solution to the second sub-problem (*i.e.*,  $T/2 \leq t \leq T$ ), then  $r_i$  will contribute at least  $T/2$  to the total arrival time. Nevertheless,  $k$ -way split is still helpful in this case as we may use it to quickly compute an initial  $T$  for performing the time expansion.  $\triangle$

## VI. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of our optimal and near-optimal MPP algorithms with an emphasis on MINMAKESPAN. Our performance evaluation covers a broad spectrum of typical problem settings. For each setting, we push the limit on the robot-vertex ratio—to as high as 100%. To the best of our knowledge, the majority of the settings with high robot-vertex ratio have never been attempted with much success prior to our study.

When applicable, we also compare our results with the state-of-the-art found in the literature. In particular, we have examined OD+ID, ID, WHCA\*, and COBOPT Surynek (2012), among others. OD+ID and ID support cycles whereas WHCA\* appears to be designed for cycle-free MPP as it could not solve any 9-puzzle. The problem definition for COBOPT suggests it solves MPP but it employs a cycle-free subroutine for finding feasible solutions. OD+ID, ID, and WHCA\* are designed for optimizing total time and total distance optimal objectives, and do not naturally extend to makespan computation. However, the associated makespan produced by these algorithms are usually of good quality. Among these three, our experiments show that ID-based anytime algorithm is the most versatile due to its IDA\*-like incremental structure. On the other hand, OD+ID and WHCA\* do not scale well when the robot-vertex ratio goes beyond 10-20%, depending on the particular problem setting.<sup>4</sup> COBOPT is designed for makespan computation.

We implemented all algorithms (MINMAKESPAN, MINMAXDIST, MINTOTALTIME, and MINTOTALDIST) in the Java programming language. We take advantage of multi-core CPUs when the  $k$ -way split heuristic is being used. Also, Gurobi Gurobi Optimization (2014), the ILP solver used in our implementation, can engage multiple cores automatically for hard problems. We ran all our tests on a MacBook Pro laptop computer (Intel Core i7-4850HQ, 16GB memory). We thank Trevor Standley for sharing the C code implementing OD+ID, ID, and WHCA\*, among others. We modified (the original code

<sup>4</sup>Some of these algorithms were evaluated without requiring all robots reach their goals. For example, in the WHCA\* work Silver (2005), if an instance with  $n$  robots is solved for  $p < n$  robots, the problem is counted as partially solved. We require each instance to be fully solved to be counted as a success.

supports only  $32 \times 32$  grid) and compiled the code as a 64-bit windows executable under MSVC 2010 with all speed optimization flags turned on. The comparison to COBOPT uses the result provided in Surynek (2012), which covers only  $8 \times 8$  and  $16 \times 16$  grids.

#### A. Performance of MINMAKESPAN and $k$ -way Split Heuristic

We begin our experimental evaluation focusing on the MINMAKESPAN algorithm and the  $k$ -way split heuristic. For this purpose, we use as the based graph a  $24 \times 18$  grid with varying number of vertices (0-25%) removed to simulate obstacles. The connectivity of the graph is always maintained. We note that with 25% vertices removed, the graph is already sparsely connected at some places (see *e.g.*, Fig. 9), making solving these problems optimally a challenging task. In presenting the

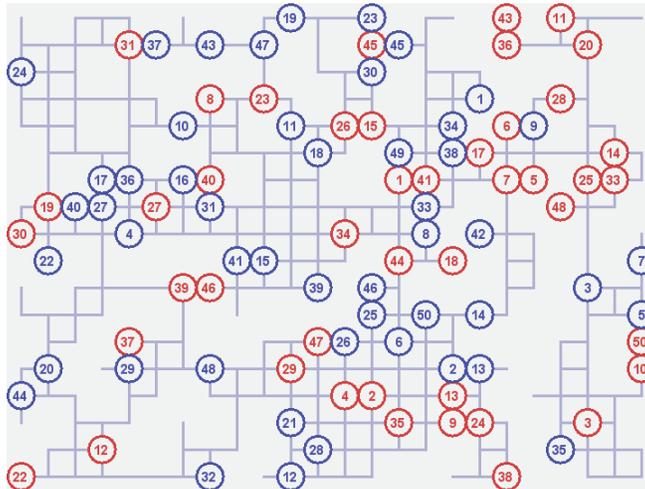


Fig. 9. A typical  $24 \times 18$  grid instance with 25% vertices removed to simulate obstacles and 50 start and goal locations. Note that the connectivity of the graph is low at some areas. For example, the lower right corner blob is only singly-connected to the rest of the graph. Some (blue) start locations may overlap with some (red) goal locations.

computational results, each data point in the figures is an average over 10 sequentially, randomly generated instances. For each obstacle percentage, we start from the lowest number of robots (usually 10 or 20) and allocate a maximum of 600 seconds for each problem instance. If an instance takes more than 600 seconds to produce a result, the instance is failed and we move to the next obstacle percentage. This also means that a data point is given only if each of the 10 instances is completed within 600 seconds. Over the same set of problem instances, the MINMAKESPAN algorithm is executed in the exact manner (which produces optimal makespan solutions) and with the  $k$ -way split heuristic.

The exact makespan computation result is summarized in Fig. 10. For all obstacle settings, the MINMAKESPAN algorithm computes optimal makespan solutions consistently for up to 100 robots with an average computation time of no more than 100 seconds. notably, for 50 robots and obstacles up to 20%, the MINMAKESPAN algorithm is able to complete in about 15 seconds in all cases. From the top plot of Fig. 10, we observe that for each fixed obstacle percentage, the computation time appears to grow exponentially with respect to the number of robots. The computational difficulty of a particular problem instance also heavily depends on the actual optimal makespan. For example, a problem instance in the case of 25% and 20 robots has a particularly long makespan (see the bottom plot of Fig. 10), resulting an unexpected jump of the computation time.

Whereas the exact MINMAKESPAN algorithm is reasonably efficient, the  $k$ -way split heuristic brings a significant performance boost, allowing a much higher robot-vertex density in general. In our tests, we are often able to double or triple the supported robot-vertex density. For the  $24 \times 18$  grid, we evaluated the  $k$ -way split heuristic for  $k$  up to 16. The 4-way split performance is illustrated in Fig. 11. In the figure, we measure optimality using a conservatively estimated *optimality ratio*. To obtain this, we divide the objective value returned by the optimizer over a conservative estimate (a lower bound). For makespan, this lower bound estimate is obtained by first computing the shortest path for each robot ignoring all other robots. The minimum makespan estimate is obtained by taking the maximum length over all these shortest paths. Clearly, the optimality ratio obtained this way is an underestimate.

We make three comments over Fig. 11. First, from the top plot, we observe that our method is highly effective in terms of computation time, capable of computing minimum makespan solutions for up to 180 robots, which translates to a maximum robot-vertex ratio of 44%. The majority of the cases are solved in under 10 seconds. Even when there are 25% obstacles, we could solve the problem consistently for up to 60 robots in about 40 seconds. Second, we again observe an exponential relationship between computation time and the number of robots. Third, all computed solutions are very close to being optimal, with all but one case having an optimality ratio of below 1.1. The average minimum makespan for these instances is about 35.

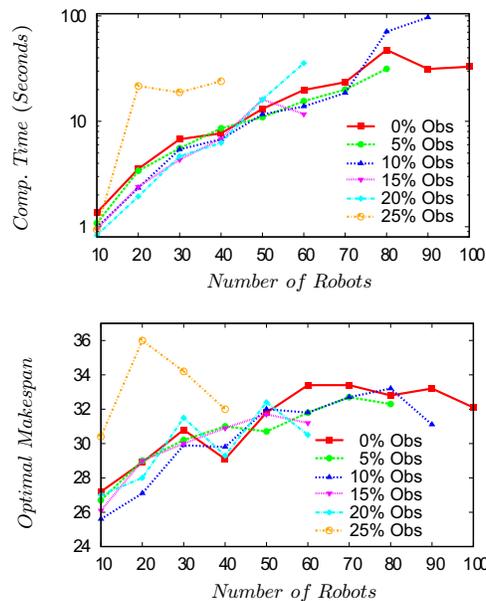


Fig. 10. [top] Average computation time of the exact MINMAKESPAN algorithm over instances on a  $24 \times 18$  grid with randomly placed obstacles and start/goal locations. [bottom] The (average) optimal makespan.

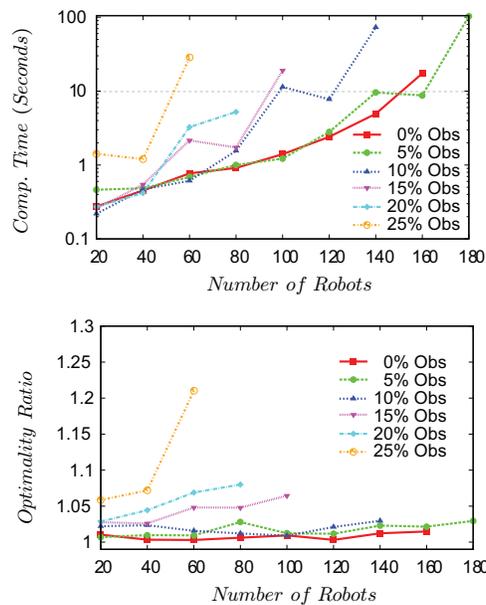


Fig. 11. [top] Average computation time of MINMAKESPAN algorithm (with 4-way split heuristic) over instances on a  $24 \times 18$  grid with randomly placed obstacles and start/goal locations. [bottom] The achieved (conservatively estimated) optimality ratio.

The rest of the  $k$ -way split evaluation is presented in Fig. 12, in which the computation time and optimality ratio are shown side by side without the axis labels. We also omit the key of the plots, which is the same as those from Fig. 10, representing different obstacle percentages. In conjunction with Fig. 10 and Fig. 11, as  $k$  increases, we observe a general trend of reduced computation time at the expense of some optimality loss. With 16-split, MINMAKESPAN can solve problems with 300 robots, corresponding to a robot-vertex ratio of 69%.

For comparison, we ran ID-based anytime algorithm over the same set of instances with a 600-second time limit (we also attempted OD+ID and WHCA\*, which do not go past 40 robots under the same setup). The result is plotted in Fig. 13. ID actually performs quite well for up to 60 robots, which can be attributed to its A\* root with minimum overhead as compared to our method. However, the performance of ID degrades faster—it does not scale well beyond 100 robots in our tests. MINMAKESPAN, with 2-way split, readily outperforms ID when there are 40 or more robots. Conceivably, it may be possible to combine ID and  $k$ -way split to make it run faster. However, adding  $k$ -way split to ID will inevitably make the overall makespan more sub-optimal.

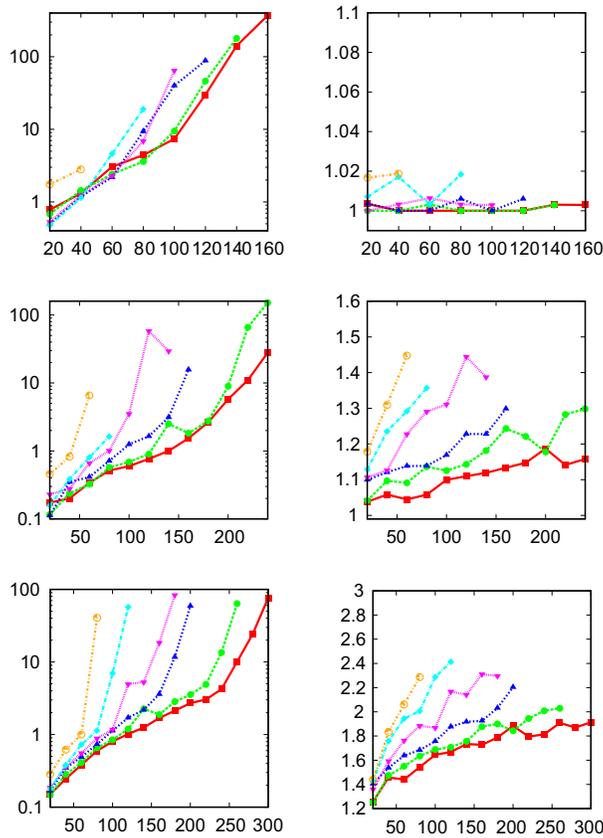


Fig. 12. Performance of the MINMAKESPAN algorithm with  $k$ -way split for  $k = 2$  (top row), 8 (middle row), and 16 (bottom row). The computation time and optimality ratio plots for each  $k$  are shown side-by-side. The  $x$ -axis for all plots represents the number of robots. The  $y$ -axis for the plots on the left represents the computation time in seconds. The  $y$ -axis for the plots on the right represents the optimality ratio. The keys for all plots are the obstacle percentage, identical to that of Fig. 10.

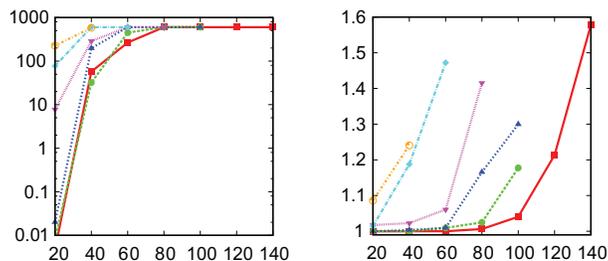


Fig. 13. Performance of the ID-based anytime algorithm over the same set of problem instances. The plot setup is the same as that from Fig. 12.

### B. Minimum Makespan Solution to $N^2$ -puzzles

Next, we evaluate the efficiency of the algorithm MINMAKESPAN for finding minimum makespan solutions to the  $N^2$ -puzzle for  $n = 3, 4, 5$ , and 6. These problems have a robot-vertex ratio of 100%, making them highly constrained and extremely challenging. Note that an  $N^2$ -puzzle instance is always solvable for  $n \geq 3$  (see the Appendix); this means that all states are connected in the state (search) space. 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 0.46 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 1.08 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 4.2 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 of 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 over  $5 \times 10^8$  states (the branching factor is over 1000), each taking 64 bits of memory. This translates into over 4GB of raw memory and over 8GB of working memory, which is more than the JavaVM can handle: A bidirectional search ran out of memory after about 10 minutes in general. We also experimented with C++ implementation using STL libraries, which yields similar result (*i.e.*, ran out of memory before reaching a search depth of 3).

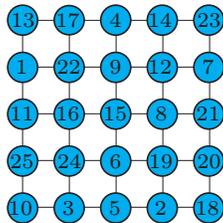


Fig. 14. An instance of a 25-puzzle problem solved by MINMAKESPAN.

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 of 391.6 seconds over 100 randomly created problems. Fig. 14 shows one of the solved instances with a 7-step solution given in Fig. 15. Note that 7 steps is obviously the least possible since it takes at least 7 steps to move robot 10

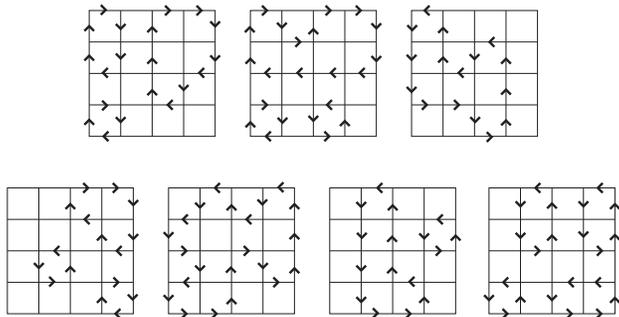


Fig. 15. An optimal 7-step solution (from left to right, then top to bottom) to the 25-puzzle problem from Fig. 14, by MINMAKESPAN in about 15 minutes.

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  $3.7 \times 10^{41}$  states and a branching factor of well over  $10^6$ .

As comparison, WHCA\* can not solve 9-puzzle. OD+ID, and ID and can only solve 9-puzzle consistently and cannot solve any 16-puzzles in 600 seconds.

### C. Minimum Makespan on $8 \times 8$ , $16 \times 16$ , and $32 \times 32$ Grids

In this section, we evaluate MINMAKESPAN with underlying graphs that are  $8 \times 8$  grids,  $16 \times 16$  grids, and  $32 \times 32$  grids with 20% obstacles. In addition to further demonstrating the effectiveness of MINMAKESPAN, this allows us to better compare our results.  $8 \times 8$  and  $16 \times 16$  grids are used as the environment for evaluation in Surynek (2012).  $32 \times 32$  grids with 20% obstacles are used for evaluation in Silver (2005); Standley and Korf (2011).

For  $8 \times 8$  and  $16 \times 16$  grids, the instances are constructed using the same procedure stated in Section VI-A. Again, each data point is an average over 10 sequentially randomly created instances. Given the size of  $8 \times 8$  and  $16 \times 16$  grids, we limit  $k$  to 8; using 16-way split can solve more instances but incur an average solution optimality between 2-4 optimal. Each instance is given a time limit of 600 seconds. The outcome of these experiments is plotted in Fig. 16, along with the result from running ID. OD+ID and WHCA\* cannot consistently solve the instances with 20 robots in 600 seconds. Makespan, instead of optimality ratio, is used in Fig. 16 for easy comparison with the results from Surynek (2012).

We observe that, over the  $8 \times 8$  grid, with 2-way split heuristic, MINMAKESPAN can solve problems with 50 robots to almost true optimal solutions in just 10 seconds. With 4-way split, MINMAKESPAN can further push to 60 robots (robot-vertex ratio of 94%) with solutions that are within 1.7-optimal. ID can only handle up to 30 robots. As reported in Surynek (2012), COBOPT generally takes more than half an hour to produce its final solution when there are 24 or more robots.<sup>5</sup> The solution

<sup>5</sup>We did not directly run COBOPT over our randomly created instances. However, the instances in Surynek (2012) are created in an identical manner. Therefore, given that similarly powered computers are used, the computation time and solution makespan are directly comparable between ours and those from Surynek (2012).

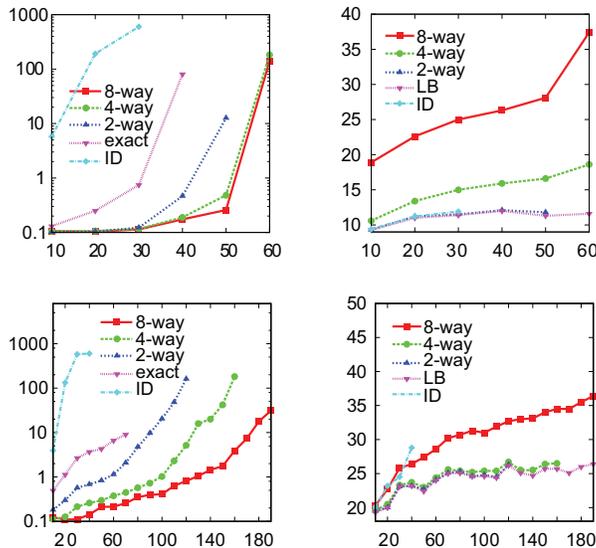


Fig. 16. Performance of the MINMAKESPAN algorithm with  $k$ -way split for  $k = 2-8$  and the ID-based anytime algorithm. The computation time and solution makespan plots are shown side-by-side. The  $x$ -axis for all plots represents the number of robots. The  $y$ -axis for the plots on the left represents the computation time in seconds. The  $y$ -axis for the plots on the right represents the solution makespan. Note that we did not plot the makespan computed by the exact MINMAKESPAN algorithm. Instead, the lower bound (LB) estimate of makespan is plotted (in magenta color). [top] Result on the  $8 \times 8$  grid. [bottom] Result on the  $16 \times 16$  grid.

quality also degrades quickly as the number of robots increases. For example (Fig. 2 and Fig. 3 in Surynek (2012)), at 50 robots, COBOPT takes over an hour to produce a solution with a makespan of over 160, whereas our 2-way split heuristic yields a near-optimal makespan of 11.8 in just 10 seconds.

Over the  $16 \times 16$  grid, MINMAKESPAN is able to handle instances with 190 (robot-vertex ratio of 74%) robots with 8-way split while at the same yielding solutions that are always less than 1.4-optimal. When switched to 4-way split, MINMAKESPAN can consistently solve problems with up to 160 robots to no more than 1.03-optimal. In comparison, ID can solve instances with up to 80 robots to relatively good quality. Taking on average an hour of computation, COBOPT can handle up to 128 robots; the solution quality is quite poor. For example (Fig. 4 and Fig. 5 in Surynek (2012)), for about 100 robots, the computed makespan by COBOPT is at 200 whereas the optimal makespan is about 25. Across  $8 \times 8$  and  $16 \times 16$  grids, we generally observe a speedup of over 100 times when MINMAKESPAN (with the 4-way split heuristic) is compared with COBOPT. At the same time, our method yields solutions with much smaller makespan.

A classical test scenario is the 4-connected  $32 \times 32$  grid with 20% vertices randomly removed.<sup>6</sup> For completeness, we also perform a brief evaluation of this setup. We randomly generate the instance as before, run the test, and plot the result in Fig. 17. From the figure, we observe a pattern consistent with experiments on the  $8 \times 8$ ,  $16 \times 16$  and  $24 \times 18$  grids.

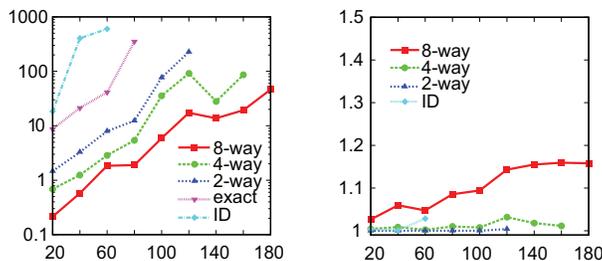


Fig. 17. Performance of MINMAKESPAN and the ID-based anytime algorithm over  $32 \times 32$  grid. The axis setup is the same as that from Fig. 12.

#### D. Algorithm Performance over All Objectives

Last, we evaluate the performance of our algorithms at optimizing all Objectives 1-4. The result on MINMAKESPAN is already presented in Section VI-A, which we also use as the solution to optimizing Objective 2 (*i.e.*, we simply use MINMAKESPAN

<sup>6</sup>Some work (*e.g.*, Standley and Korf (2011)) also adopts an 8-connected model. That is, each vertex is on the grid is connected to its 8-neighborhood. This causes unit cost to be assigned to all edges, although a diagonal edge should have length  $\sqrt{2}$  times that of a non-diagonal edge. Since we are modeling robots in this work, we do not discuss the 8-connected model here. However, we mention that our algorithms easily extend to 8-connected model. Our tests show that we can in fact compute near-optimal makespan for 400 robots on  $32 \times 32$  grids assuming 8-connectivity.

in place of MINMAXDIST due to its superior performance). Note that no change to the plots are needed here because, on one hand, we can use a (near)-optimal makespan solution as a solution to minimizing maximum single-robot traveled distance. This is true because the makespan of a solution is always no less than the minimum maximum single-robot distance. On the other hand, the lower bound estimate for minimum makespan is the same as that for minimum maximum single-robot distance.

Before moving to MINTOTALTIME and MINTOTALDIST, we note that these algorithms possess some properties of an *anytime algorithm*, which is of practical importance. In solving these ILP models, the solver generally uses variations of the *branch-and-bound* algorithm Land and Doig (1960). For computing total time (and distance) optimal solutions, a branch-and-bound algorithm always try to find a feasible solution first and then iteratively improve over the feasible solution. This naturally leads to improving solution quality commonly observed in an anytime algorithm. The anytime property of MINTOTALTIME and MINTOTALDIST allows us to set a desired sub-optimal threshold to reduce the computation time. Note that the same cannot be said for MINMAKESPAN because a feasible solution here is the optimal solution.

Our next set of results focuses on the MINTOTALTIME algorithm (Fig. 18). The general setup is the same as that used in Section VI-A. In particular, the same set of problem instances is used. In our experiment, we limit both time (600 seconds) and required sub-optimality threshold (automatically adjusted) to achieve a balanced performance. The lower bound estimate for computing optimality ratio is obtained by summing over the individual shortest path lengths. The actual optimal total time is about 15 times the number of robots (*i.e.*, 150 for 10 robots and 1500 for 100 robots), regardless of the percentage of obstacles. We observe that the MINTOTALTIME algorithm is fairly effective, capable of computing 1.1-optimal solutions for up to 100 robots in the allocated time.

Similar outcome is also observed in the performance evaluation of the MINTOTALDIST algorithm with the 4-way split heuristic (Fig. 19). The optimal total distance is again about 15 times the number of robots. In comparison with the total time optimal case, due to the 4-way split heuristic, the MINTOTALDIST algorithm is faster but produced solutions that are more sub-optimal but still quite good.

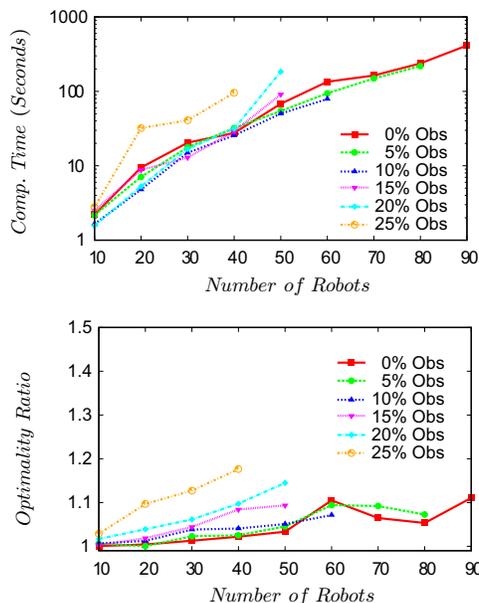


Fig. 18. [top] Average computation time of MINTOTALTIME algorithm over instances on a  $24 \times 18$  grid with randomly placed obstacles and start/goal locations. [bottom] The achieved (conservatively estimated) optimality ratio.

For comparison, we run MINTOTALTIME, MINTOTALDIST, and ID (total time and total distance versions) on  $24 \times 18$  grids with 0-20% obstacles with a maximum time limit of 600 seconds. The setting is slightly different from that used in obtaining Fig. 18 and 19; we do not set a sub-optimal threshold here. The result is plotted in Fig. 20. In the case of total time (Objective 3), ID could solve more instances. For the instances that are solved, the achieved optimality is similar. For total distance (Objective 4), we observe similar outcome. Here, ID could produce one more data point; the achieved optimality by both methods are again comparable (and good). Overall, MINTOTALTIME, and MINTOTALDIST are competitive with ID.

## VII. CONCLUSION

In this paper, we propose a general algorithmic framework for solving MPP problems optimally or near-optimally. Through an equivalence relationship between MPP and network flow, we provide ILP model-based algorithms for minimizing the makespan (last arrival time), the maximum (single-robot traveled) distance, the total arrival time, and the total distance. In conjunction with additional heuristics, our algorithmic solutions are highly effective, capable of computing near-optimal solutions for hundreds

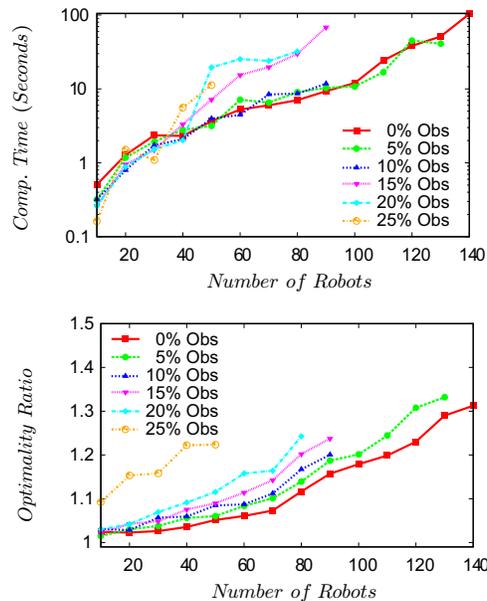


Fig. 19. [top] Average computation time of MINTOTALDIST algorithm over instances on a  $24 \times 18$  grid with randomly placed obstacles and start/goal locations. [bottom] The achieved (conservatively estimated) optimality ratio.

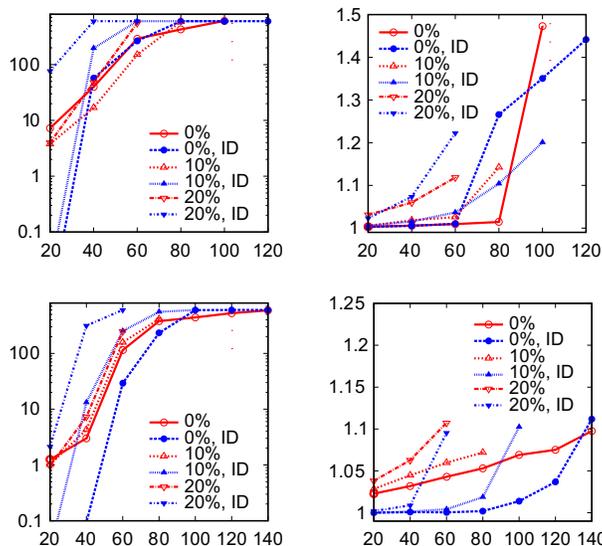


Fig. 20. Performance of MINTOTALTIME, MINTOTALDIST, and the ID-based anytime algorithm over  $24 \times 18$  with 0-20% obstacles. Each instance is allowed 600 seconds of time. The axis setup is the same as that from Fig. 12. [top] MINTOTALTIME versus total time ID; the red lines correspond to data for MINTOTALTIME. [bottom] MINTOTALDIST versus total distance ID; the red lines correspond to data for MINTOTALDIST.

of robots in seconds in scenarios with very high robot-vertex ratio. In pushing for high performance algorithms aim at solving MPP optimally or near-optimally, our eventual goal is to apply these algorithms to multi-robot path planning problems in continuous domains—our preliminary work toward this goal has begun to show promising results, producing algorithms that can compute solutions for around a hundred disc robots in 2D environments with holes.

## REFERENCES

- J. Yu and S. M. LaValle, “Optimal multi-robot path planning on graphs: Structure and computational complexity,” 2015, reference to be updated.
- T. Standley, “Finding optimal solutions to cooperative pathfinding problems,” in *Proceedings AAAI National Conference on Artificial Intelligence*, 2010, pp. 173–178.
- T. Standley and R. Korf, “Complete algorithms for cooperative pathfinding problems,” in *Proceedings International Joint Conference on Artificial Intelligence*, 2011, pp. 668–673.
- P. R. Wurman, R. D’Andrea, and M. Mountz, “Coordinating hundreds of cooperative, autonomous vehicles in warehouses,” *AI Magazine*, vol. 29, no. 1, pp. 9–19, 2008.
- M. A. Erdmann and T. Lozano-Pérez, “On multiple moving objects,” in *Proceedings IEEE International Conference on Robotics & Automation*, 1986, pp. 1419–1424.

- A. Zelinsky, "A mobile robot exploration algorithm," *IEEE Transactions on Robotics & Automation*, vol. 8, no. 6, pp. 707–717, 1992.
- S. M. LaValle and S. A. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Transactions on Robotics & Automation*, vol. 14, no. 6, pp. 912–925, Dec. 1998.
- D. Silver, "Cooperative pathfinding," in *The 1st Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005, pp. 23–28.
- S. Kloder and S. Hutchinson, "Path planning for permutation-invariant multirobot formations," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 650–665, 2006.
- M. R. K. Ryan, "Exploiting subgraph structure in multi-robot path planning," *Journal of Artificial Intelligence Research*, vol. 31, pp. 497–542, 2008.
- R. Jansen and N. Sturtevant, "A new approach to cooperative pathfinding," in *In International Conference on Autonomous Agents and Multiagent Systems*, 2008, pp. 1401–1404.
- P. Surynek, "A novel approach to path planning for multiple robots in bi-connected graphs," in *Proceedings IEEE International Conference on Robotics & Automation*, 2009, pp. 3613–3619.
- R. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Proceedings International Joint Conference on Artificial Intelligence*, 2011, pp. 294–300.
- J. van den Berg and M. Overmars, "Prioritized motion planning for multiple robots," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots & Systems*, 2005.
- J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha, "Centralized path planning for multiple robots: Optimal decoupling into sequential plans," in *Robotics: Science and Systems*, 2009.
- K. Solovey and D. Halperin, " $k$ -color multi-robot motion planning," in *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2012.
- J. Yu and S. M. LaValle, "Multi-agent path planning and network flow," in *Algorithmic Foundations of Robotics X, Springer Tracts in Advanced Robotics (STAR)*. Springer Berlin/Heidelberg, 2013, vol. 86, pp. 157–173.
- M. Turpin, K. Mohta, N. Michael, and V. Kumar, "CAPT: Concurrent assignment and planning of trajectories for multiple robots," *International Journal of Robotics Research*, vol. 33, no. 1, pp. 98–112, 2014.
- K. Solovey, J. Yu, O. Zamir, and D. Halperin, "Motion planning for unlabeled discs with optimality guarantees," in *Robotics: Science and Systems*, 2015, to appear.
- D. Halperin, J.-C. Latombe, and R. Wilson, "A general framework for assembly planning: The motion space approach," *Algorithmica*, vol. 26, no. 3-4, pp. 577–601, 2000.
- B. Nnaji, *Theory of Automatic Robot Assembly and Programming*. Chapman & Hall, 1992.
- S. Rodriguez and N. M. Amato, "Behavior-based evacuation planning," in *Proceedings IEEE International Conference on Robotics & Automation*, 2010, pp. 350–355.
- T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Transactions on Robotics & Automation*, vol. 14, no. 6, pp. 926–939, 1998.
- S. Poduri and G. S. Sukhatme, "Constrained coverage for mobile sensor networks," in *Proceedings IEEE International Conference on Robotics & Automation*, 2004.
- B. Shucker, T. Murphey, and J. K. Bennett, "Switching rules for decentralized control with simple control laws," in *American Control Conference*, Jul 2007, pp. 1485–1492.
- 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*, vol. 14, no. 3, pp. 322–335, Jun. 2009.
- H. Tanner, G. Pappas, and V. Kumar, "Leader-to-formation stability," *IEEE Transactions on Robotics & Automation*, vol. 20, no. 3, pp. 443–455, Jun 2004.
- D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "A probabilistic approach to collaborative multi-robot localization," *Autonomous Robots*, vol. 8, no. 3, pp. 325–344, Jun. 2000.
- J. Ding, K. Chakraborty, and R. B. Fair, "Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 20, no. 12, pp. 1463–1468, 2001.
- E. J. Griffith and S. Akella, "Coordinating multiple droplets in planar array digital microfluidic systems," *International Journal of Robotics Research*, vol. 24, no. 11, pp. 933–949, 2005.
- M. J. Matarić, M. Nilsson, and K. T. Simsarian, "Cooperative multi-robot box pushing," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots & Systems*, 1995, pp. 556–561.
- D. Rus, B. Donald, and J. Jennings, "Moving furniture with teams of autonomous robots," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots & Systems*, 1995, pp. 235–242.
- 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.
- 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.
- J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer, 1991.
- S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, also available at <http://planning.cs.uiuc.edu/>.
- E. W. Story, "Note on the '15' puzzle," *American Journal of Mathematics*, vol. 2, pp. 399–404, 1879.
- S. Loyd, *Mathematical Puzzles of Sam Loyd*. New York: Dover, 1959.
- R. M. Wilson, "Graph puzzles, homotopy, and the alternating group," *Journal of Combinatorial Theory (B)*, vol. 16, pp. 86–96, 1974.
- D. Kornhauser, G. Miller, and P. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," in *Proceedings IEEE Symposium on Foundations of Computer Science*, 1984, pp. 241–250.
- J. Yu and D. Rus, "Pebble motion on graphs with rotations: Efficient feasibility tests and planning," in *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2014.
- Q. Sajid, R. Luna, and K. E. Bekris, "Multi-agent pathfinding with simultaneous execution of single-agent primitives," in *Fifth Annual Symposium on Combinatorial Search*, 2012.
- B. de Wilde, A. W. ter Mors, and C. Witteveen, "Push and rotate: a complete multi-agent pathfinding algorithm," *Journal of Artificial Intelligence Research*, pp. 443–492, 2014.
- G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Conflict-Based Search for Optimal Multi-Agent Path Finding," in *Proc of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.
- G. Wagner and H. Choset, "M\*: A complete multirobot path planning algorithm with performance bounds," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots & Systems*, 2011, pp. 3260–3267.
- P. Surynek, "Towards optimal cooperative path planning in hard setups through satisfiability solving," in *Proceedings 12th Pacific Rim International Conference on Artificial Intelligence*, 2012.
- K. Kant and S. Zucker, "Towards efficient trajectory planning: The path velocity decomposition," *International Journal of Robotics Research*, vol. 5, no. 3, pp. 72–89, 1986.
- J. van den Berg, M. C. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proceedings IEEE International Conference on Robotics & Automation*, 2008, pp. 1928–1935.
- J. van den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *Proceedings IEEE International Conference on Robotics & Automation*, 2011, pp. 3475–3482.

- A. F. v. d. S. I. Karamouzas, R. Geraerts, “Space-time group motion planning,” in *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2012.
- M. Peasgood, C. Clark, and J. McPhee, “A complete and scalable strategy for coordinating multiple robots within roadmaps,” *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 283–292, 2008.
- K. Solovey, O. Salzman, and D. Halperin, “Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning,” in *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2014.
- A. Krontiris, Q. Sajid, and K. Bekris, “Towards using discrete multiagent pathfinding to address continuous problems,” *Workshop on Multi-agent Pathfinding (in conjunction with AAAI)*, 2012.
- J. Yu and S. M. LaValle, “Planning optimal paths for multiple robots on graphs,” in *Proceedings IEEE International Conference on Robotics & Automation*, 2013, pp. 3612–3617.
- , “Fast, near-optimal computation for multi-robot path planning on graphs,” in *Proceedings AAAI National Conference on Artificial Intelligence*, 2013, late breaking papers.
- D. Ratner and M. Warmuth, “The  $(n^2 - 1)$ -puzzle and related relocation problems,” *Journal of Symbolic Computation*, vol. 10, pp. 111–137, 1990.
- J. E. Aronson, “A survey on dynamic network flows,” *Annals of Operations Research*, vol. 20, no. 1, pp. 1–66, 1989.
- R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.
- M. Andrews and L. Zhang, “Hardness of the undirected edge-disjoint paths problem,” in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*. ACM, 2005, pp. 276–283.
- N. Robertson and P. D. Seymour, “Graph minors. XIII. the disjoint paths problem,” *Journal of combinatorial theory, Series B*, vol. 63, no. 1, pp. 65–110, 1995.
- I. Gurobi Optimization, “Gurobi optimizer reference manual,” 2014. [Online]. Available: <http://www.gurobi.com>
- A. H. Land and A. G. Doig, “An automatic method of solving discrete programming problems,” *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.

## APPENDIX

### A. Properties of the $N^2$ -puzzle

The example problem from Fig. 1 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).

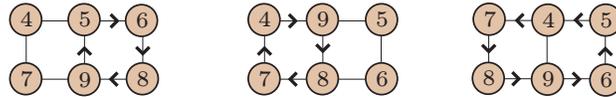


Fig. 21. A 3-step procedure for exchanging robots 8 and 9.

**Proposition 6** *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. 1(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. 21.  $\square$

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. 22 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. 22).
- 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. 22.
- 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.

It is straightforward to see that larger puzzles can be solved similarly. We have thus outlined the essential steps for proving Proposition 7 below; a more generic proof can be written using generators of permutation groups, which we omit here due to its length. Proposition 7 implies that, for  $N \geq 3$ , all instances of  $N^2$ -puzzles are solvable. The constructive proofs of Proposition 6 and 7 lead to recursive algorithms for solving any  $N^2$ -puzzle (clearly, the solution is not time/distance optimal in general).

**Theorem 7** *All states of an  $N^2$ -puzzle,  $N \geq 3$  are connected via legal moves.*

**Corollary 8** *All instances of the  $N^2$ -puzzle,  $N \geq 3$ , are solvable.*

By Proposition 7, 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

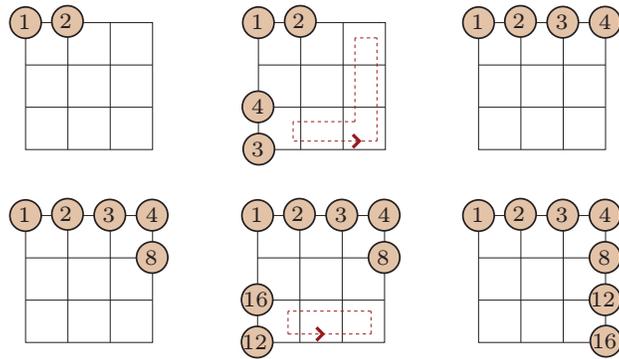


Fig. 22. A solution scheme for solving top/left sides of the 16-puzzle.

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 basic 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. 1(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 afterward will take 5 more steps.