Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics

Jingjin Yu and Steven M. LaValle

***Abstract*—We study optimal multirobot path planning on graphs (MPP) over four minimization objectives: the makespan (last arrival time), the maximum (single-robot traveled) distance, the total arrival time, and the total distance. Having established previously that these objectives are distinct and NP-hard to optimize, in this paper, we focus on efficient algorithmic solutions for solving these optimal MPP problems. Toward this goal, we first establish a one-to-one solution mapping between MPP and a special type of multiflow network. 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 that is capable of solving extremely challenging problems with robot-vertex ratios 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 slight 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

We study the problem of optimal *multirobot 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. A robot may move from a vertex to an adjacent one in one time step in the absence of collision, which occurs when two robots simultaneously move to the same vertex or along the same edge in opposing directions. A distinguishing feature is that our formulation allows robots on fully occupied cycles to rotate synchronously. Such a formulation, more appropriate for multirobot applications, has not been widely studied (except, e.g., [1], [2]). 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. These global objectives have direct relevance toward real-world multirobot applications, including autonomous warehouse systems [3]. 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 work [4], we show that these objectives are pairwise distinct and NP-hard to optimize, suggesting that efforts on solving optimal MPP should be directed at finding effective near-optimal algorithms. In this paper, we make an attempt toward this goal and propose a novel yet general framework for solving MPP optimally. Examining space and time dimensions jointly, we observe a one-to-one mapping between a solution for an MPP instance and that for a multicommodity network flow problem (multiflow[1]) derived from the MPP problem. Based on the equivalence, we translate the MPP problem into an integer linear programming (ILP) model solvable using an ILP solver. The generality of ILP allows the encoding of all four objectives to yield complete optimization algorithms. From here, we further 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: Multirobot path planning problems, in its many formulations, have been actively studied for decades [2], [5]–[19]. As a universal subroutine, collision-free path planning for multiple robots finds applications in tasks spanning assembly [20], [21], evacuation [22], formation control [23]–[27], localization [28], microdroplet manipulation [29], [30], object transportation [31], [32], search and rescue [33], etc. See [34]–[36] and the references therein for a more comprehensive review on the general subject of multirobot path and motion planning.

The algorithmic study of graph-based multirobot path planning problems, which is the focus of this paper, can be traced to 1879 [37], in which Story makes the observation that the feasibility of the 15-puzzle [38] depends on the *parity* of the game. The 15-puzzle is a restricted MPP instance moving 15 labeled game pieces on a 4×4 grid, from some initial configuration to some goal configuration. The restriction is that only a single game piece near the single empty vertex may move to the empty vertex in a step; multiple mobile robots, on the other hand, could move simultaneously. A generalization of the 15-puzzle is introduced in [39], extending the problem from 15 game pieces on a 4×4 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 nonbipartite.

Manuscript received July 13, 2015; revised April 3, 2016; accepted June 19, 2016. Date of publication August 10, 2016; date of current version September 30, 2016. This paper was recommended for publication by Associate Editor H. Kress-Gazit and Editor C. Torras upon evaluation of the reviewers' comments. This work was supported by the National Science Foundation under Grant 1328018 (National Robotics Initiative) and Grant 1617744 (Division of Information and Intelligent Systems-Robot Intelligence).

J. Yu is with the Department of Computer Science, Rutgers University, New Brunswick, NJ 08901 USA (e-mail: jingjin.yu@cs.rutgers.edu).

S. M. LaValle is with the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL 61820 USA (e-mail: lavalle@illinois.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2016.2593448

[1] *multiflow* is used here to refer the multicommodity flow problem.

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 [40], 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 multirobot 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 [41]. 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 [40], most algorithmic study of cycle-free MPP put some emphasis on optimality. Through the clever use of primitive operations, algorithms from [10], [12], [13], [42], and [43] 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.* [40]. For more discussion and references on suboptimal methods, see [42] and [43].

On 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*) [6], 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 [8]. A technique called subdimensional expansion is shown to perform well in complex environments [44]; the robot density is, however, relatively low (104 cells per robot according to the paper). In [1] and [2], instead of applying an 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). More recently, increasing cost tree search (ICTS [45]) and conflict-based search (CBS [46]) have further pushed the performance on cycle-free MPP. Algorithms designed for minimizing makespan have also been attempted, e.g., [47], but the solution quality degrades rapidly as the robot-vertex ratio increases.

Many approaches have also been proposed for solving multirobot path planning problems in the continuous domain. A representative method called velocity obstacles [48]–[50] explicitly examines velocity-time space for coordinating robot motions. In [30], mixed integer programming models are employed to encode the robot interactions. A method based on the space-time perspective, similar to ours, is explored in [51]. In [52], an A*-based search is performed over a discrete roadmap abstracted from the continuous environment. In [53], discrete-RRT is proposed for the efficient search of multirobot roadmaps. Algorithms for discrete MPP, cycle-free or not, have also helped solving continuous problems [16], [54].

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 multirobot applications. The formulation is not widely studied, perhaps due to the inherent difficulty in handling cyclic rotations of robots. Beside the novelty of the problem, this work brings several algorithmic contributions. First, based on the equivalence relationship between MPP and multiflow, 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 the four most common global objectives for MPP. The resulting algorithms, in particular the one for computing the minimum makespan, are highly effective in solving challenging problem instances with a 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 loss of solution optimality. 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 exploitation of ILP to attack optimal MPP shows that the ILP 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 optimal MPP problems and provide a brief review of network flow. We establish the equivalence relationship between MPP and network flow in Section III. We derive complete algorithms in Section IV 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 [17], [55], [56].[2] In comparison to [17] and [55], beside 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 provide a brief review of *network flow*.

[2] [56] is a preliminary poster presentation.

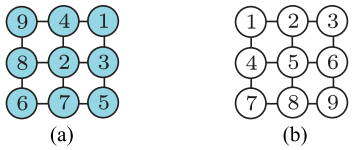


Fig. 1. (a) 9-puzzle problem. (b) Desired goal configuration.

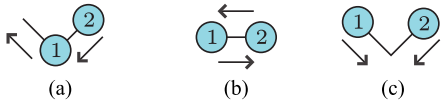


Fig. 2. Some feasible and infeasible moves for two robots. (a) Feasible synchronous move. (b) Infeasible synchronous move in which two robot collide "head-on." (c) Infeasible synchronous move in which two robots "meet" at a vertex.

A. Multirobot 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 $R = \{r_1, \ldots, r_n\}$ be a set of n robots. A *configuration* of the robots is an injective map from R to V, i.e., for a given configuration, two robots r_i and r_j $(i \neq j)$ occupy different vertices of V. At any given time step $t = 0, 1, \ldots$, the robots assume a configuration. 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 nine robots on a 3×3 grid graph. Fig. 1(b) shows a possible goal configuration, in which the robots are ordered based on *row-major ordering*.[3]

During a discrete time step, each robot may either remain stationary or move to an adjacent vertex. To formally describe a plan, let a *path* be a map $p_i : \mathbb{Z}^+ \to V$, in which $\mathbb{Z}^+ := \mathbb{N} \cup \{0\}$. A 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 $t_i^f \in \mathbb{Z}^+$ such that $p_i(t_i^f) = x_G(r_i)$.
3) For any $t \geq t_i^f$, $p_i(t) = x_G(r_i)$.
4) For any $0 \leq t < t_i^f$, $\{p_i(t), p_i(t+1)\} \in E$ 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 $(p_i(t) = p_j(t+1)) \wedge (p_i(t+1) = p_j(t))$ (head-on collision). As an illustration, Fig. 2 shows the feasible and infeasible moves for two robots during a single time step.[4] The MPPproblem is defined as follows.

Problem 1 (Multirobot path planning on graphs): Given a 4-tuple (G, R, x_I, x_G), find a set of paths $P = \{p_1, \ldots, p_n\}$ such that for all $1 \leq i < j \leq n$: (i) p_i is a feasible path for robot r_i, and (ii) p_i and p_j are not in collision.

For example, Fig. 1(a) and (b) defines an MPP problem on the 3×3 grid. We call this particular problem the *9-puzzle* problem, which readily generalizes to N^2-puzzles. We pick the name due to its similarity with the classic 15-puzzle and $(N^2 - 1)$-puzzles [57]. Despite the superficial similarities, an N^2-puzzle has N^2 robots and does not require an empty *swap* vertex that is required for the $(N^2 - 1)$-puzzle. In an N^2-puzzle, robots may rotate synchronously along multiple disjoint nonintersecting cycles.

Remark: With a few exceptions (e.g., [2]), most existing studies on discrete multirobot path planning problems require empty vertices as swap spaces. In these formulations, in a time step, a nonintersecting chain of robots may move simultaneously only if the head of the chain is moving into a previously unoccupied vertex. In contrast, our MPP formulation allows synchronized rotations of robots along fully occupied cycles (see, e.g., Figs. 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 a feasibility test taking only linear time [41]. △

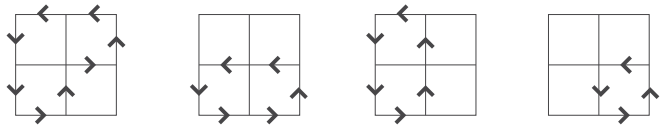


Fig. 3. Four-step solution from our algorithm for the 9-puzzle from Fig. 1. The directed edges show the moving directions of the robots at the tail of these edges.

B. Optimal Formulations

Let $P = \{p_1, \ldots, p_n\}$ be an arbitrary feasible solution to some fixed MPP instance. For a path $p_i \in P$, let $len(p_i)$ denote the length of the path p_i, which is the total number of times the robot r_i changes its residing vertex while following p_i. A robot, following a path p_i, may visit the same vertex multiple times. Recall that t_i^f 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.

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

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

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

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

A four-step minimum makespan solution to the 9-puzzle problem from Fig. 1 is illustrated in Fig. 3. The solution is optimal because robot 9 is four steps away from its goal.

C. Network Flow Review

A *network* $\mathcal{N} = (G, c_1, c_2, S)$ consists of a *directed graph* $G = (V, E)$ with $c_1, c_2 : E \to \mathbb{Z}^+$ being the maps specifying

[3] In this paper, we use shaded discs to mark start locations of robots and discs without shades for goal locations.

[4] We assume that the graph G allows only "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 with radii of no more than $\sqrt{2}/4$ (two robots traveling on adjacent edges are the closest to each other when they are in the middle of these edges).

capacities and *costs* over directed edges (arcs), 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 $S^+ \cap S^- = \varnothing$. For a vertex $v \in V$, let $\delta^+(v)$ [resp., $\delta^-(v)$] denote the set of arcs 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 arc capacity constraints

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

the flow conservation constraints at nonterminal 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

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

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

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

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 *multicommodity flow* or simply *multiflow*. Instead of having a single flow function f, we have a flow function f_i for each commodity i from a set C of commodities. The constraints (1), (2), and (3) become

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

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

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

Maximum flow and minimum cost flow problems may also be posed under a multiflow setup; we omit the details. Our review of network flows only touches aspects pertinent to this paper; for a thorough coverage on the subject of network flows, see [58] and [59] 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.

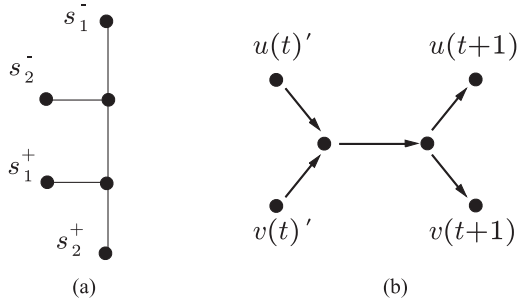


Fig. 4. (a) Simple G. (b) *Merge-split* gadget for splitting an undirected edge through time steps, for enforcing the head-on collision constraint.

III. From Multirobot 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 [59], whereas maximum multiflow is a well-known NP-hard problem, difficult to even approximate [60]. Mirroring the disparity between single- and multicommodity 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 [17]. However, as soon as a single group of robots splits into two or more groups, finding optimal paths for these robots become intractable [4]. 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-expanded* version of the graph G, which allows the explicit consideration of the interactions among the robots over space *and* time. Time expansion derived in this paper can be viewed as performing *network flow over time* [61]. To carry out this expansion, a time horizon must first be determined. 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 $0, 1, 1', \ldots,$ 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 $t, t + 1, 0 \leq t < T$, the *merge-split* gadget shown in Fig. 4(b) is added between $u(t)', v(t)'$, and $u(t + 1), v(t + 1)$

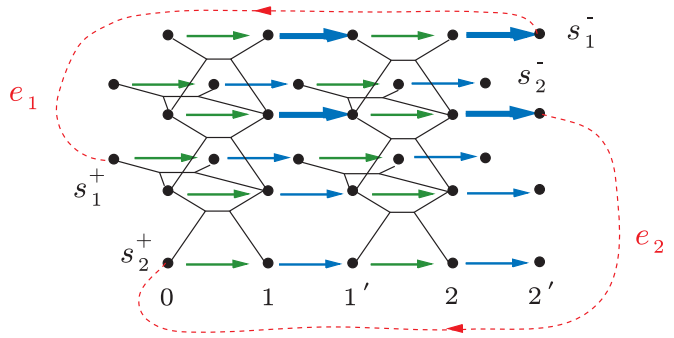


Fig. 5. Time-expanded network with an expansion time horizon of $T = 2$ over the base graph Fig. 4(a).

(arrows from the gadget are omitted from Fig. 5 since they are small). For the gadget, we assign unit capacity to all arcs, unit cost to the horizontal middle arcs, and zero cost to the other four arcs. The merge-split gadget ensures that two robots cannot travel in opposite directions on an edge of the underlying graph 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 arc between every two successive copies (i.e., we add the arcs $(v(0), v(1)), (v(1), v(1)'), \ldots, (v(T), v(T)')$). These correspond to the green and blue arcs in Fig. 5. For all green arcs, we assign them unit capacity and cost; for all blue arcs, we assign them unit capacity and zero cost. The green arcs allow robots to stay at a vertex during a time step, whereas blue arcs ensure that each vertex holds at most one robot, enforcing the meet collision constraint.

The time-expanded network in Fig. 5 (with the exception of arcs e_1 and e_2, which will become relevant shortly) 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 $P = \{p_1, \ldots, p_n\}$ (n is the number of robots) is a solution to an MPP instance. 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 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 and 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 an integer maximum multiflow on the network $\mathcal{N}$, which implies that $|f_i| = 1$ for all i's. First, we establish that any pair of flows f_i and f_j are vertex disjoint. To see this, we note that f_i and 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 arcs. If f_i and f_j share some nonsink vertex v at time step $t > 0$, both flows then must pass through the same blue arc [see Fig. 4(b)] with v being either the head or the tail vertex, which is not possible. Thus, f_i and 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 arcs) with the guarantee that no p_i, p_j will collide due to a meet collision. By the 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 the MPP defined by (G, R, x_I, x_G). ■

Remark: A multiflow problem with unit individual flows can also be viewed as a type of multipath planning problem, known as the *edge disjoint path* problem [62]. △

IV. Complete Integer Linear Programming-Based Algorithms for Optimal Multirobot Path Planning on Graphs 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 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,[5] which often target an important but limited set of problem structures, ILP-based algorithms proposed here are *agnostic* to specific problem structures. As such, ILP-based algorithms appear more capable of addressing 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, assuming a fixed time span T. That is, these models only optimize the given objective for a specific T. The discussion of the full algorithms and their completeness then follows.

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* arcs 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 arcs of G', and let the n loopback arcs take the first n indices, with $e_j, 1 \leq j \leq n$, being the arc connecting the goal vertex of r_j to the start vertex of r_j. For example, for the G' in

[5]Here, the term heuristics does not refer to the admissible heuristic used by A* itself. Instead, it refers to ways of breaking the search space of a multirobot path planning problem into disjoint, smaller subspaces.

Fig. 5, e_1 and e_2 are the loopback arcs for r_1 and r_2, respectively. The loopback arcs have unit capacities and zero costs. Next, for each arc $e_j \in G'$ (including the loopback arcs), create n binary variables $x_{1,j}, \ldots, x_{n,j}$ corresponding to the flow through that arc, one for each robot $1 \leq i \leq n$. 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 arc capacity constraints and one flow conservation constraint

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

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

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

The objective function is

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

B. Minimizing the Maximum Single-Robot Traveled Distance

For minimizing the maximum distance traveled by any 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 goals, a maximum flow may be forced through the constraint

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

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

$$\forall 1 \leq i \leq n, \sum_{e_j \in G', j > n} c_2(e_j) \cdot x_{i,j} \leq x_{\max}. \tag{12}$$

For a fixed i, the left side of (12) represents the distance traveled by robot r_i. Note that $x_{i,j} = 0$ for $j < n$. The objective function is then simply

$$\min x_{\max}. \tag{13}$$

C. Minimizing the Total Arrival Time

For minimizing the total arrival time, the network and variables from the minimum makespan ILP-model and constraints (8), (9), and (11) are inherited. 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 certain existing variables. Recall that for each arc $e_j = (v(t), v(t)') \in G'$ (e.g., the four extra bold blue arcs in Fig. 5), a variable $x_{i,j}$ is created. There are nT such variables. Here, we give these variables a second index x_i^t. That is, x_i^t is the binary variable indicating whether arc $(v(t), v(t)') \in G'$, $v = x_G(r_i)$ 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, $x_i^T = 1$ 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

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

Effectively, (14) performs a logical AND 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. Thus, $T - \sum_{t=1}^{T} y_i^t$ is the time spent by r_i. To minimize the total arrival time, the objective function can be expressed as

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

Algorithm 1: MPP-ILP-OPTIMIZATION

Input: (G, R, x_I, x_G) and obj, the objective
Output: $P = \{p_1, \ldots, p_n\}$, the optimal solution

```
T_min ← Initial time span (an underestimate, e.g., 0)
T_max ← Estimated time span containing optimal solution
P ← ∅
for T from T_min to T_max do
    Build ILP model for obj with time span T
    Attempt solving model using an optimizer
    P ← best solution found so far
end
return P
```

D. Minimizing the Total Distance

From the ILP model for minimizing the maximum distance, we need to change only 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

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

E. Algorithm Structure and Completeness

The general algorithm structure for optimally solving MPP is outlined in Algorithm 1. Here, we assume that the problem (G, R, x_I, x_G) is feasible, which can be readily checked [41]. Then, we make conservative estimates on the minimum time span $T_{\min}$ and the maximum time span $T_{\max}$ such that there must be a $T \in [T_{\min}, T_{\max}]$ corresponding to the time span for an optimal solution. The algorithm then simply searches through $[T_{\min}, T_{\max}]$ to locate T and the optimal solution. Denoting the specific algorithms for Objectives 1–4 as MINMAKESPAN, MINMAXDIST, MINTOTALTIME, and MINTOTALDIST, respectively, we now fill in how to estimate $T_{\min}$ and $T_{\max}$ for them.

While $T_{\min} = 0$ always works, we can do better by setting $T_{\min}$ as the maximum over all robots the shortest path length for each robot, ignoring all other robots. This $T_{\min}$ clearly applies to all four objectives. For MINMAKESPAN, we may set $T_{\max}$ to be the time needed for a feasible solution, which can be computed with [41]. In the worst case, $T = O(|V|^3)$. Note that

this establishes the completeness of MINMAKESPAN and we may assume we have an optimal T_{opt} for MINMAKESPAN.

For MINMAXDIST, we set $T_{\max} = nT_{\text{opt}}$. This is true because at each time step, at least one robot must move a distance of one. After $nT_{\text{opt}} + 1$ time, by the pigeonhole principle, some robot must have moved $T_{\text{opt}} + 1$ steps. However, since T_{opt} is optimal for MINMAKESPAN, there exists a solution in which no robots travel more than T_{opt}. This yields a contradiction.

The same $T_{\max} = nT_{\text{opt}}$ works for MINTOTALTIME for a different reason. After $nT_{\text{opt}} + 1$ time steps, at least one robot must have spent this much time. On the other hand, the total time from the optimal MINMAKESPAN solution is no more than nT_{opt}. So the optimal solution for MINTOTALTIME will not require a time span of more than nT_{opt}. For MINTOTALDIST, $T_{\max} = nT_{\text{opt}}$ also works. After $nT_{\text{opt}} + 1$ time steps, the total distance traveled by all robots must exceed nT_{opt}. However, the optimal solution for MINTOTALTIME will not require more than nT_{opt} total number of moves. We have shown that the four algorithms are all complete.

Proposition 2: MINMAKESPAN, MINMAXDIST, MINTOTALTIME, and MINTOTALDIST are all complete.

V. HEURISTICS FOR EFFECTIVE COMPUTATION OF NEAR-OPTIMAL SOLUTIONS

We have established that ILP-based algorithms are complete and always produce optimal solutions. Their performance in handling relatively small but highly constrained problems are in fact quite good (see Section VI). As problem sizes grow (i.e., as the graph G and the number of robots n increase), however, the computation time grows rapidly. From a practical point of view, it may be far more desirable to quickly compute a good quality but suboptimal solution than to wait longer for the optimal solution. In this section, we introduce several heuristics to accomplish this goal, with a particular focus on computing solutions with the minimum makespan.

A. Building More Compact Models

To extract the best performance out of a solver, it is beneficial to have a lean model (i.e., fewest columns and rows). So far, our focus has been to provide a general multiflow-based framework so that the ILP models can be easily built. During the model translation stage, the models can be further simplified. The heuristics discussed in this section aim at making the representation of constraints (8) and (9) more compact. As such, they apply to all objectives.

Better encoding of the collision constraints: In building the network flow model (e.g., Fig. 5), we used a merge-split gadget [see Fig. 4(b)] for enforcing the head-on collision constraint and extra time steps (e.g., the blue arcs in Fig. 5) for avoiding meet collisions. When we translate them 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 arcs instead of five. Also, the blue edges are gone. 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 arcs,

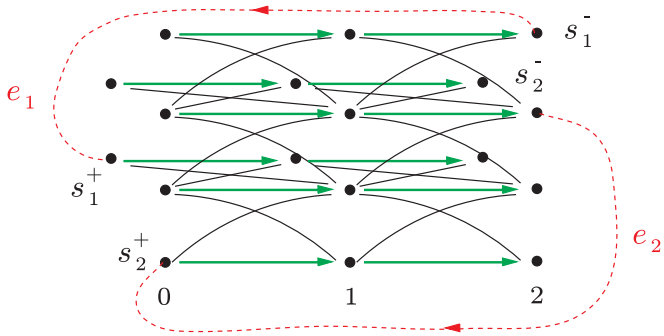


Fig. 6. More compact representation of the network flow graph from Fig. 5.

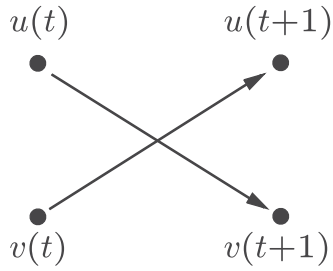


Fig. 7. Simplified *merge-split* gadget for enforcing the head-on collision constraint.

vertices such as $v(t)'$ are 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

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

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

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

The new ILP model is roughly half of the size of the original model.

Reachability analysis: In the time-expanded graph, there are redundant binary (arc) variables that can never be true because some arcs are never reachable. For example, in Fig. 6, at $t = 0$, the only outgoing arcs that can possibly be used are those originating 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 the 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

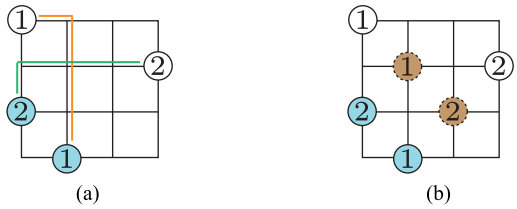


Fig. 8. (a) Simple two-robot problem. (b) Time-divided instances.

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 robot-vertex ratio is not critically high (i.e., not approaching one), relatively small ILP models do not present much challenge for ILP solvers. This is true even when there are a large number of robots (i.e., in the hundreds). 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.

In Fig. 8(a), we have a planning problem for two robots on a 3×3 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 equal or nearly 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 unoccupied location 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 both requiring a time expansion with two steps each, effectively making the individual ILP model roughly half the size of the original one that required a time expansion with four steps. In general, we may divide a problem into arbitrarily many smaller instances in the time domain.

If a problem is divided into k subproblems, we call the resulting heuristic a *k-way split*. Because the division is over time, there is no interaction between the divided instances. Once we obtain a solution for each child instance, the solutions can be glued together by concatenating the results. In practice, this heuristic dramatically improves algorithm performance without significant negative impact on path makespans; we observe a consistent speedup in computational experiments.

Remark: The k-way split heuristic, by design, is particularly apposite for the makespan objective, owing to the additive nature of the makespan objective over the split subproblems. Beside the 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 subproblems. As an example, suppose that a two-way split is carried out with each subproblem having a time horizon of $T/2$. If a robot r_i does not move in the solution to the first subproblem (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 subproblem (i.e., $T/2 \leq t \leq T$), then r_i will contribute at least $T/2$ to the total arrival time. Nevertheless, the 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. △

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.[6]

After examining a large number of existing approaches including OD+ID, ID, WHCA*, ICTS, CBS, and Cobopt, we focus our comparison on ID-based anytime algorithm and Cobopt due to problem similarity and our emphasis on 100% success rate.[7] We point out that there are various differences between our robotics-based MPP formulation and these methods to which we compare. OD+ID and ID support cycles, whereas CBS and WHCA* appear to be designed for cycle-free MPP, as they 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. Except for Cobopt, most of these methods are designed for optimizing total time and total distance optimal objectives, and do not naturally extend to the makespan. However, the associated makespans produced by these algorithms are usually of good quality. Among these, 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, ICTS, CBS, and WHCA* do not scale well when the robot-vertex ratio goes beyond 10%. Cobopt is designed for makespan.

We implemented all algorithms (MinMakespan, MinMaxDist, MinTotalTime, and MinTotalDist) in the Java programming language. We take advantage of multicore CPUs when the k-way split heuristic is being used. Also, Gurobi [64], the ILP solver used in our implementation, can engage multiple

[6]For completeness, 15-puzzle and 24-puzzle have been successfully attempted [63]. Beside, differences in not allowing cyclic rotation, the solutions also do not support multiple vacant cells.

[7]Some of these algorithms were evaluated without requiring that all robots reach their goals. For example, in the WHCA* work [8], 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, which is a stringent requirement for robot path planning.

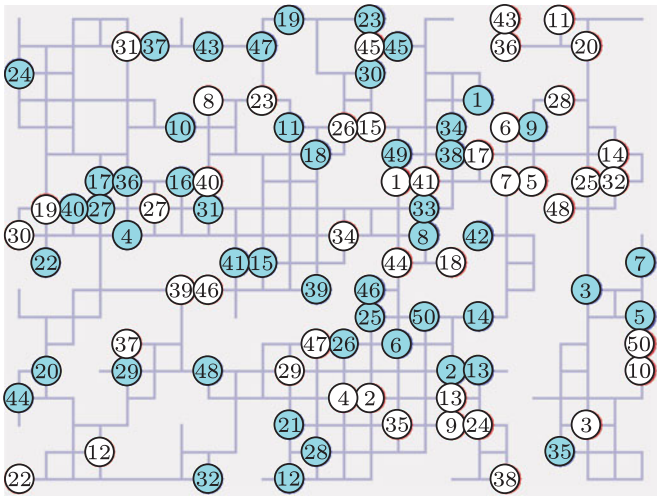


Fig. 9. Typical 24×18 grid instance with 25% vertices removed to simulate obstacles and 50 start and goal locations. Note that the connectivity of the graph is low in some areas. For example, the lower right corner blob is only singly-connected to the rest of the graph. Some (blue shaded) start locations may overlap with some (not shaded) goal locations.

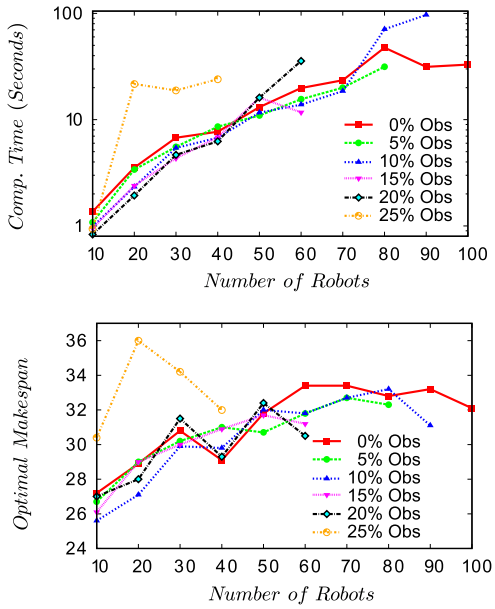


Fig. 10. [top] Average computation time of the exact MINMAKESPAN algorithm over instances on a 24×18 grid with randomly placed obstacles and start/goal locations. [bottom] The (average) optimal makespan.

cores automatically for hard problems. We ran all our tests on a MacBook Pro laptop computer (Intel Core i7-4850HQ, 16 GB memory). We obtained the C code implementing OD + ID, ID, and WHCA*among others from Trevor Standley. We modified (the original code supports only 32×32 grids) and compiled the code as a 64-bit windows executable under MSVC 2010 with all speed optimization flags turned on. C# code for CBS was retrieved from the authors' online repository.[8] The comparison to COBOPT uses the result provided in [47], which covers only 8×8 and 16×16 grids.

A. Performance of MINMAKESPAN and k-Way Split Heuristic

We begin our experimental evaluation focusing on MINMAKESPAN and the k-way split heuristic. For this purpose, we use as the base graph a 24×18 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 computational results, each data point in the figures is an average over ten 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 s for each problem instance. If an instance takes more than 600 s to produce a result, the instance is stopped and we move to the next obstacle percentage. This also means that a data point is given only if each of the ten instances is completed within 600 s. 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.

[8]https://bitbucket.org/eli.boyarski/.

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 s. Notably, for 50 robots and obstacles up to 20%, the MINMAKESPAN algorithm is able to complete in about 15 s 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 depends heavily 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 in an large jump of the computation time. Examining the instance reveals that a narrow corridor-like setting is present, similar to the scenario from Fig. 9, which requires two robots to pass through the corridor from opposing directions. This induces a much larger T in the time expansion step, causing a significant increase in computation time.

The k-way split heuristic brings a significant performance boost, allowing a much higher robot-vertex ratio in general. In our tests, we are often able to double or even triple the supported robot density. For the 24×18 grid, we evaluated the k-way split heuristic for k up to 16. The four-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 overestimate.

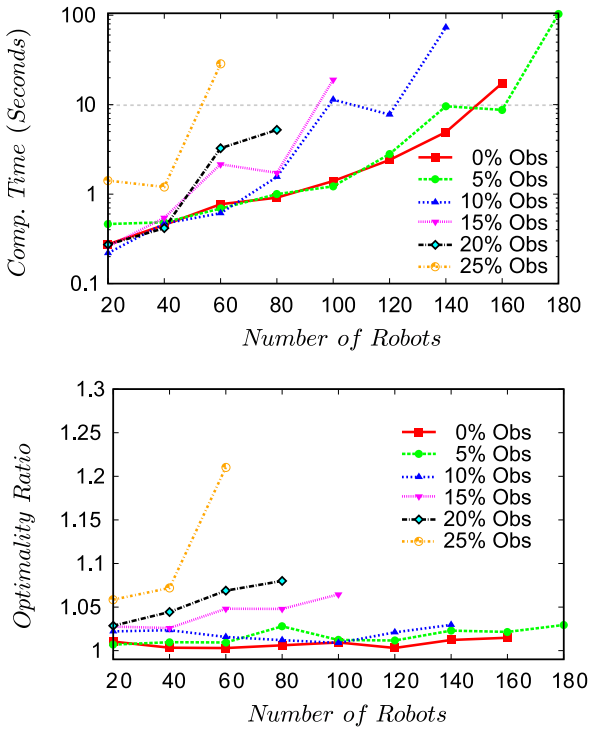


Fig. 11. [top] Average computation time of the MINMAKESPAN algorithm (with four-way split heuristic) over instances on a 24×18 grid with randomly placed obstacles and start/goal locations. [bottom] The achieved (conservatively estimated) optimality ratio.

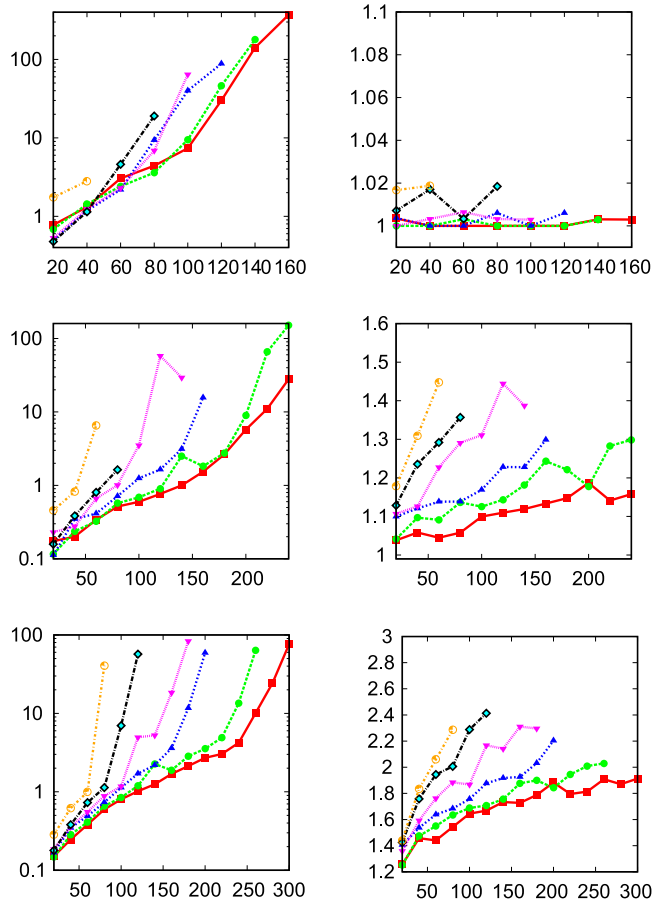


Fig. 12. Performance of the MINMAKESPAN algorithm with k-way split for $k = 2$ (top row), 8 (middle row), and 16 (bottom row). The computation time and optimality ratio plots for each k are shown side by side. The x-axis for all plots represents the number of robots. The y-axis for the plots on the left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

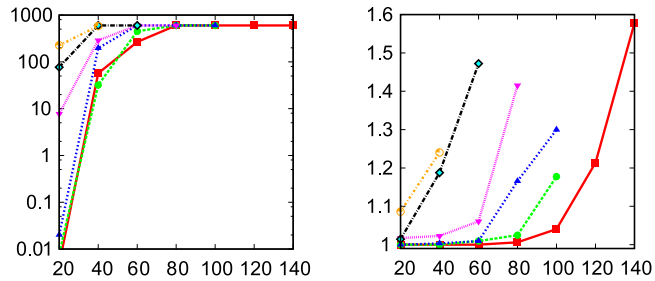


Fig. 13. Performance of the ID-based anytime algorithm over the same set of problem instances. The plot setup is the same as that from Fig. 12.

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 within 10 s. Even when there are 25% obstacles, we could solve the problem consistently for up to 60 robots in about 40 s. 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.

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. To improve clarity, axis labels are omitted. The omitted keys are the same as those from Fig. 10, representing different obstacle percentages. In conjunction with Figs. 10 and 11, as k increases, we observe a general trend of reduced computation time at the expense of loss of optimality. 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-s time limit (we also attempted OD + ID, CBS, and WHCA*, which are unable to consistently go past 40 robots under the same setup; we used 21×21 grid for CBS). 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 two-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 suboptimal.

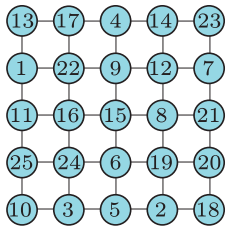


Fig. 14. Instance of a 25-puzzle problem solved by MINMAKESPAN.

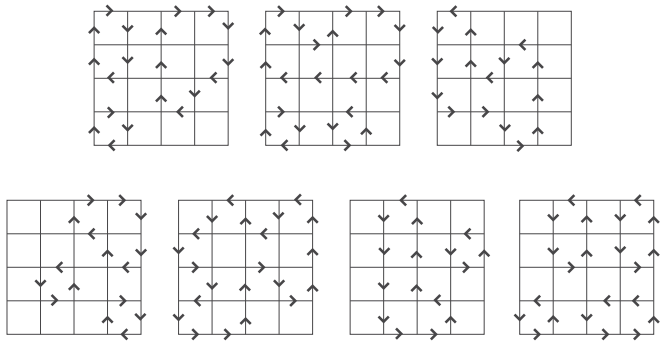


Fig. 15. Optimal seven-step solution (from left to right, then top to bottom) to the 25-puzzle problem from Fig. 14, by MINMAKESPAN in about 15 min.

B. Minimum Makespan Solution to N^2-Puzzles

Next, we evaluate the efficiency of the MINMAKESPAN algorithm 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. Note that an N^2-puzzle instance is always solvable for $n \geq 3$ [41]; this means that all states are connected in the state (search) space. We ran the MINMAKESPAN algorithm 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 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 s 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 s. 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 5×10^8 states (the branching factor is over 1000), each taking 64 bits of memory. This translates into over 4 GB of raw memory and over 8 GB of working memory, which is more than the JavaVM can handle: A bidirectional search ran out of memory after about 10 min in general. We also experimented with C++ implementation using STL libraries, which yields a similar result (i.e., ran out of memory before reaching a search depth of 3).

For the 25-puzzle, without a good heuristic, bidirectional search cannot explore even 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 s over 100 randomly created problems. Fig. 14 shows one of the solved instances with a seven-step solution given in Fig. 15. Note that seven steps are the least possible as it takes at least seven steps to move robot 10 to its desired goal. We also 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 h, which has 3.7×10^{41} states and a branching factor of well over 10^6.

As a comparison, CBS and WHCA* cannot solve the 9-puzzle. OD + ID and ID and can solve only the 9-puzzle but could not solve any 16-puzzles in 600 s.

C. Minimum Makespan on 8×8, 16×16, and 32×32 Grids

In this section, we evaluate MINMAKESPAN with underlying graphs that are 8×8 grids, 16×16 grids, and 32×32 grids with 20% obstacles. In addition to further demonstrating the effectiveness of MINMAKESPAN, this allows us to better compare our results. 8×8 and 16×16 grids are used as the environment for evaluation in [47]. 32×32 grids with 20% obstacles are used for evaluation in [2] and [8].

For 8×8 and 16×16 grids, the instances are constructed using the same procedure stated in Section VI-A. Again, each data point is an average over ten sequentially randomly created instances. Given the size of 8×8 and 16×16 grids, we limit k to 8; using 16-way split can solve more instances but incurs an average solution optimality between 2 and 4. Each instance is given a time limit of 600 s. The outcome of these experiments is plotted in Fig. 16, along with the result from running ID. CBS, OD + ID, and WHCA* cannot consistently solve the instances with 20 robots in 600 s. Makespan, instead of optimality ratio, is used in Fig. 16 for easy comparison with the results from [47].

We observe that, over the 8×8 grid, with the two-way split heuristic, MINMAKESPAN can solve problems with 50 robots to almost true optimal solutions in just 10 s. With the four-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 [47], COBOPT generally takes more than half an hour to produce its final solution when there are 24 or more robots.[9] The solution quality also degrades quickly as the number of robots increases. For example (Fig. 2 and Fig. 3 in [47]), at 50 robots, COBOPT takes over an

[9]We did not directly run COBOPT over our randomly created instances. However, the instances in [47] 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 [47].

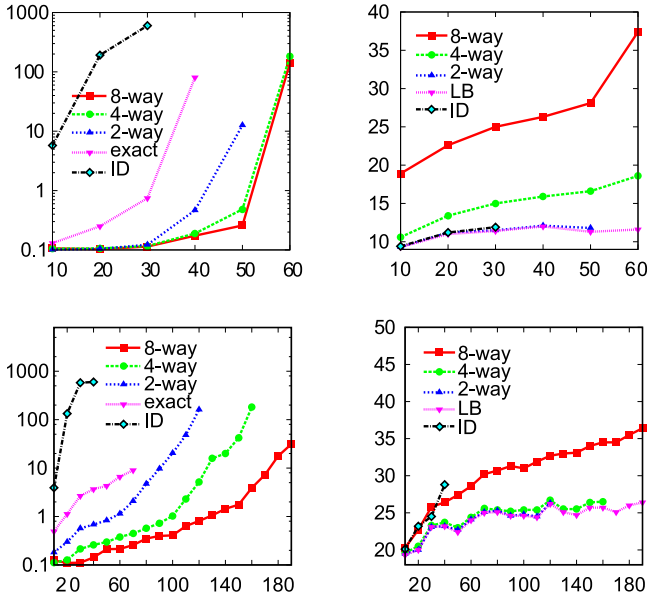


Fig. 16. Performance of the MINMAKESPAN algorithm with k-way split for $k = 2$–8 and the ID-based anytime algorithm. The computation time and solution makespan plots are shown side by side. The x-axis for all plots represents the number of robots. The y-axis for the plots on the left represents the computation 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 8×8 grid. [bottom] Result on the 16×16 grid.

hour to produce a solution with a makespan of over 160, whereas our two-way split heuristic yields a near-optimal makespan of 11.8 in just 10 s.

Over the 16×16 grid, MINMAKESPAN is able to handle instances with 190 (robot-vertex ratio of 74%) robots with the eight-way split, while at the same time yielding solutions that are always less than 1.4-optimal. When switched to the four-way split, MINMAKESPAN can consistently solve problems with up to 160 robots to no worse 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 makespans are also highly suboptimal. For example, (see [47, Figs. 4 and 5]), for about 100 robots, the computed makespan by COBOPT is at 200, whereas the optimal makespan is about 25. Across 8×8 and 16×16 grids, we observe a speedup of over 100 times when MINMAKESPAN (with the four-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 32×32 grid with 20% vertices randomly removed.[10] For completeness, we also perform an evaluation of this setup. We randomly generated the instance as before, ran the test, and plotted the result in Fig. 17. From the figure, we observe a pattern consistent with experiments on the 8×8, 16×16, and 24×18 grids.

[10]Some work (e.g., [2]) also adopts an 8-connected model. That is, each vertex is on the grid is connected to its 8-neighborhood. This causes the unit cost to be assigned to all edges, although a diagonal edge should have length $\sqrt{2}$ times than that of a nondiagonal 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 the 8-connected model. Our tests show that we can in fact compute near-optimal makespan for 400 robots on 32×32 grids assuming 8-connectivity.

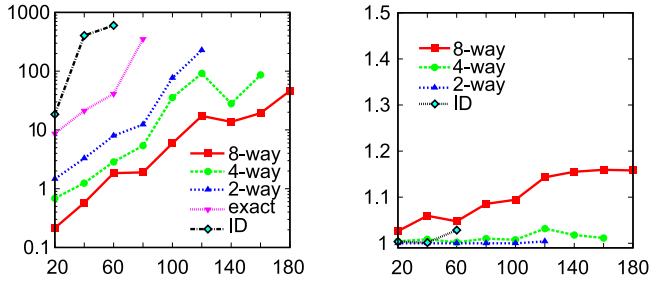


Fig. 17. Performance of MINMAKESPAN and the ID-based anytime algorithm over 32×32 grid. The axis setup is the same as that from Fig. 12.

D. Algorithm Performance Over All Objectives

Finally, we evaluate the performance of our algorithms at optimizing all objectives. 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 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 minimize the 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 the minimum makespan is the same as that for the 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 [65]. For computing total time (and distance) optimal solutions, a branch-and-bound algorithm always computes a feasible solution first and then iteratively improves over the feasible solution. This naturally leads to the steadily improving solution quality commonly observed in an anytime algorithm. The anytime property of MINTOTALTIME and MINTOTALDIST allows us to set a desired suboptimal threshold to reduce the computation time. Note that the same cannot be said for MINMAKESPAN because a feasible solution is also an optimal solution for the minimum makespan case.

Our next set of results focuses on the MINTOTALTIME algorithm (see 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 s) and required suboptimality 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,

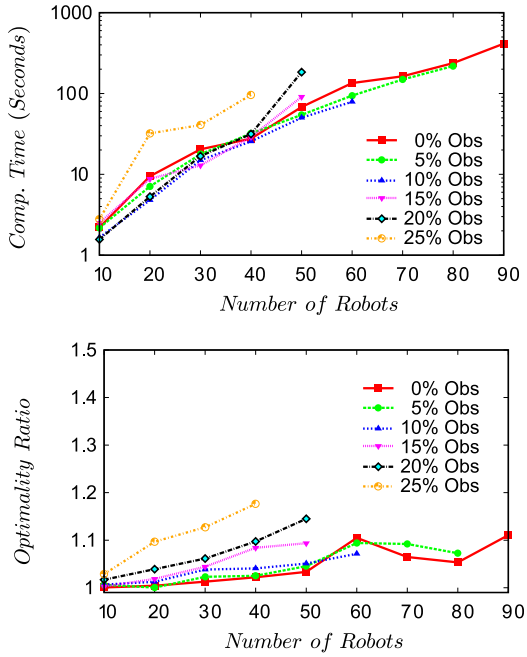


Fig. 18. [top] Average computation time of the MINTOTALTIME algorithm over instances on a 24×18 grid with randomly placed obstacles and start/goal locations. [bottom] The achieved (conservatively estimated) optimality ratio.

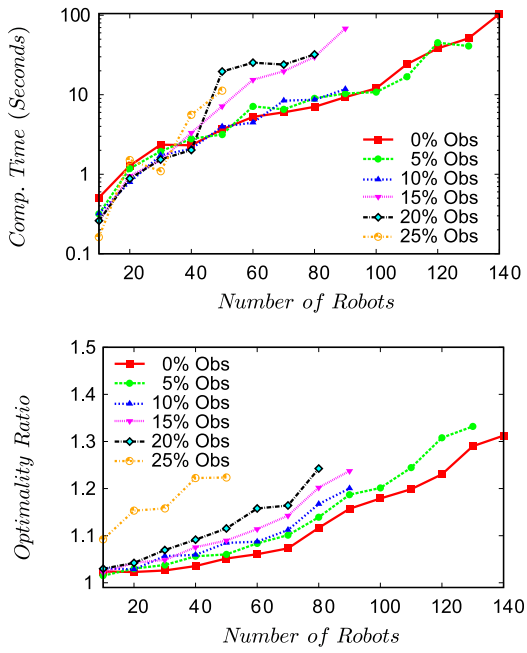


Fig. 19. [top] Average computation time of the MINTOTALDIST algorithm over instances on a 24×18 grid with randomly placed obstacles and start/goal locations. [bottom] The achieved (conservatively estimated) optimality ratio.

capable of computing 1.1-optimal solutions for up to 100 robots in the allocated time.

Similar outcomes are also observed in the performance evaluation of the MINTOTALDIST algorithm with the four-way split heuristic (see Fig. 19). The optimal total distance is again about 15 times the number of robots. In comparison with the total time

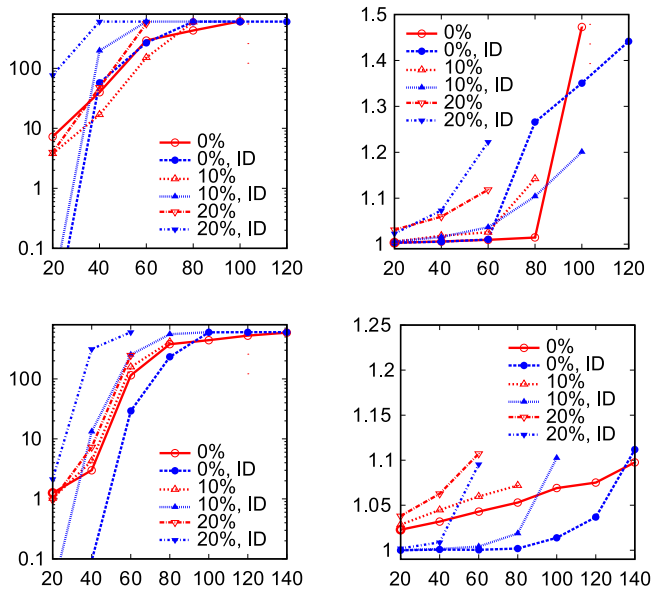


Fig. 20. Performance of MINTOTALTIME MINTOTALDIST, and the ID-based anytime algorithm over 24×18 with 0–20% obstacles. Each instance is allowed 600 s of time. The axis setup is the same as that from Fig. 12. [top] MINTOTALTIME versus total time ID; the red lines correspond to data for MINTOTALTIME. [bottom]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

optimal case, due to the four-way split heuristic, the MINTOTALDIST algorithm is faster but produced solutions that are more suboptimal but still quite good.

For comparison, we run MINTOTALTIME, MINTOTALDIST, and ID (total time and total distance versions) on 24×18 grids with 0–20% obstacles with a maximum time limit of 600 s. The setting is slightly different from that used in obtaining Figs. 18 and 19; we do not set a suboptimal 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 outcomes. 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 multiflow, 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. With additional heuristics, our algorithmic solutions are highly effective, capable of computing near-optimal solutions for hundreds of robots in seconds in scenarios with very high robot-vertex ratio. In pushing for high-performance algorithms aiming at solving MPP optimally or near-optimally, our eventual goal is to apply these algorithms to multirobot path planning problems in continuous domains. To this end, preliminary work has begun

to show promising results, producing algorithms that can compute solutions for around a hundred disc robots in bounded two-dimensional environments with holes [66].

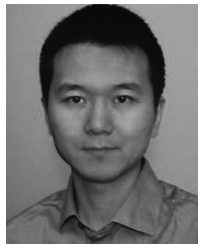
ACKNOWLEDGMENT

The authors would like to thank E. Boyarski and T. Standley for sharing their respective source codes. They also thank the reviewers for their help in improving the quality of the paper.

REFERENCES

[1] T. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proc. AAAI Nat. Conf. Artif. Intell.*, 2010, pp. 173–178.
[2] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *Proc. Int. Joint Conf. Artif. Intell.*, 2011, pp. 668–673.
[3] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Mag.*, vol. 29, no. 1, pp. 9–19, 2008.
[4] J. Yu and S. M. LaValle, "Optimal multi-robot path planning on graphs: Structure and computational complexity," ArXiv, 2015. [Online]. Available: http://arxiv.org/pdf/1507.03289v1.pdf
[5] M. A. Erdmann and T. Lozano-Pérez, "On multiple moving objects," in *Proce. IEEE Int. Conf. Robot. Autom.*, 1986, pp. 1419–1424.
[6] A. Zelinsky, "A mobile robot exploration algorithm," *IEEE Trans. Robot. Autom.*, vol. 8, no. 6, pp. 707–717, Dec. 1992.
[7] S. M. LaValle and S. A. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Trans. Robot. Autom.*, vol. 14, no. 6, pp. 912–925, Dec. 1998.
[8] D. Silver, "Cooperative pathfinding," in *Proc. 1st Conf. Artif. Intell. Interact. Digit. Entertainment*, 2005, pp. 23–28.
[9] S. Kloder and S. Hutchinson, "Path planning for permutation-invariant multirobot formations," *IEEE Trans. Robot.*, vol. 22, no. 4, pp. 650–665, Aug. 2006.
[10] M. R. K. Ryan, "Exploiting subgraph structure in multi-robot path planning," *J. Artif. Intell. Res.*, vol. 31, pp. 497–542, 2008.
[11] R. Jansen and N. Sturtevant, "A new approach to cooperative pathfinding," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2008, pp. 1401–1404.
[12] P. Surynek, "A novel approach to path planning for multiple robots in bi-connected graphs," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, pp. 3613–3619.
[13] R. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Proc. Int. Joint Conf. Artif. Intell.*, 2011, pp. 294–300.
[14] J. van den Berg and M. Overmars, "Prioritized motion planning for multiple robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2005, pp. 430–435.
[15] J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha, "Centralized path planning for multiple robots: Optimal decoupling into sequential plans," presented at the *Robotics, Science Systems*, Seattle, WA, USA, 2009.
[16] K. Solovey and D. Halperin, "k-color multi-robot motion planning," in *Proc. Workshop Algorithmic Found. Robot.*, 2012, pp. 191–207.
[17] J. Yu and S. M. LaValle, "Multi-agent path planning and network flow," in *Algorithmic Foundations of Robotics X (ser. Springer Tracts in Advanced Robotics)*, vol. 86. Berlin, Germany: Springer-Verlag, 2013, pp. 157–173.
[18] M. Turpin, K. Mohta, N. Michael, and V. Kumar, "CAPT: Concurrent assignment and planning of trajectories for multiple robots," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 98–112, 2014.
[19] K. Solovey, J. Yu, O. Zamir, and D. Halperin, "Motion planning for unlabeled discs with optimality guarantees," presented at the *Robotics, Science Systems*, Rome, Italy, 2015.
[20] D. Halperin, J.-C. Latombe, and R. Wilson, "A general framework for assembly planning: The motion space approach," *Algorithmica*, vol. 26, nos. 3/4, pp. 577–601, 2000.
[21] B. Nnaji, *Theory of Automatic Robot Assembly and Programming*. London, U.K.: Chapman & Hall, 1992.
[22] S. Rodriguez and N. M. Amato, "Behavior-based evacuation planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 350–355.
[23] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Trans. Robot. Autom.*, vol. 14, no. 6, pp. 926–939, Dec. 1998.
[24] S. Poduri and G. S. Sukhatme, "Constrained coverage for mobile sensor networks," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, pp. 165–171.
[25] B. Shucker, T. Murphey, and J. K. Bennett, "Switching rules for decentralized control with simple control laws," in *Proc. Am. Control Conf.*, Jul. 2007, pp. 1485–1492.
[26] B. Smith, M. Egerstedt, and A. Howard, "Automatic generation of persistent formations for multi-agent networks under range constraints," *Mobile Netw. Appl. J.*, vol. 14, no. 3, pp. 322–335, Jun. 2009.
[27] H. Tanner, G. Pappas, and V. Kumar, "Leader-to-formation stability," *IEEE Trans. Robot. Autom.*, vol. 20, no. 3, pp. 443–455, Jun. 2004.
[28] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "A probabilistic approach to collaborative multi-robot localization," *Auton. Robots*, vol. 8, no. 3, pp. 325–344, Jun. 2000.
[29] J. Ding, K. Chakrabarty, and R. B. Fair, "Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 20, no. 12, pp. 1463–1468, Dec. 2001.
[30] E. J. Griffith and S. Akella, "Coordinating multiple droplets in planar array digital microfluidic systems," *Int. J. Robot. Res.*, vol. 24, no. 11, pp. 933–949, 2005.
[31] M. J. Matarić, M. Nilsson, and K. T. Simsarian, "Cooperative multi-robot box pushing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 1995, pp. 556–561.
[32] D. Rus, B. Donald, and J. Jennings, "Moving furniture with teams of autonomous robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 1995, pp. 235–242.
[33] J. S. Jennings, G. Whelan, and W. F. Evans, "Cooperative search and rescue with a team of mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1997, pp. 193–200.
[34] H. Choset *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA, USA: MIT Press, 2005.
[35] J.-C. Latombe, *Robot Motion Planning*. Boston, MA, USA: Kluwer, 1991.
[36] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006. [Online]. Available: http://planning.cs.uiuc.edu/
[37] E. W. Story, "Note on the '15' puzzle," *Am. J. Math.*, vol. 2, pp. 399–404, 1879.
[38] S. Loyd, *Mathematical Puzzles of Sam Loyd*. New York, NY, USA: Dover, 1959.
[39] R. M. Wilson, "Graph puzzles, homotopy, and the alternating group," *J. Combinatorial Theory, B*, vol. 16, pp. 86–96, 1974.
[40] D. Kornhauser, G. Miller, and P. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," in *Proc. IEEE Symp. Found. Comput. Sci.*, 1984, pp. 241–250.
[41] J. Yu and D. Rus, "Pebble motion on graphs with rotations: Efficient feasibility tests and planning," presented at the *Workshop Algorithmic Foundations Robotics*, Istanbul, Turkey, 2014.
[42] Q. Sajid, R. Luna, and K. E. Bekris, "Multi-agent pathfinding with simultaneous execution of single-agent primitives," presented at the *5th Symp. Combinatorial Search*, Niagara Falls, ON, Canada, 2012.
[43] B. de Wilde, A. W. ter Mors, and C. Witteveen, "Push and rotate: A complete multi-agent pathfinding algorithm," *J. Artif. Intell. Res.*, vol. 51, pp. 443–492, 2014.
[44] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 3260–3267.
[45] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 195, pp. 470–495, 2013.
[46] E. Boyarski *et al.*, "ICBS: The improved conflict-based search algorithm for multi-agent pathfinding," presented at the *8th Annu. Symp. Combinatorial Search*, Ein Gedi, Israel, 2015.
[47] P. Surynek, "Towards optimal cooperative path planning in hard setups through satisfiability solving," in *Proc. 12th Pacific Rim Int. Conf. Artif. Intell.*, 2012 pp. 564–576.
[48] K. Kant and S. Zucker, "Towards efficient trajectory planning: The path velocity decomposition," *Int. J. Robot. Res.*, vol. 5, no. 3, pp. 72–89, 1986.
[49] J. van den Berg, M. C. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2008, pp. 1928–1935.
[50] J. van den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 3475–3482.
[51] A. F. van der Stappen, I. Karamouzas, and R. Geraerts, "Space-time group motion planning," in *Proc. 10th Workshop Algorithmic Found. Robot.*, 2012, pp. 227–243.

[52] M. Peasgood, C. Clark, and J. McPhee, "A complete and scalable strategy for coordinating multiple robots within roadmaps," *IEEE Trans. Robot.*, vol. 24, no. 2, pp. 283–292, Apr. 2008.
[53] 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 *Proc. Int. Workshop Algorithmic Found. Robot.*, 2014, pp. 591–607.
[54] A. Krontiris, Q. Sajid, and K. Bekris, "Towards using discrete multiagent pathfinding to address continuous problems," presented at the *Workshop Multi-Agent Pathfinding*, Toronto, ON, Canada, 2012.
[55] J. Yu and S. M. LaValle, "Planning optimal paths for multiple robots on graphs," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 3612–3617.
[56] J. Yu and S. M. LaValle, "Fast, near-optimal computation for multi-robot path planning on graphs," in *Proc. 17th AAAI Conf. Late-Breaking Develop. Field Artif. Intell.*, 2013, pp. 155–157.
[57] D. Ratner and M. Warmuth, "The $(n^2 - 1)$-puzzle and related relocation problems," *J. Symbolic Comput.*, vol. 10, pp. 111–137, 1990.
[58] J. E. Aronson, "A survey on dynamic network flows," *Ann. Oper. Res.*, vol. 20, no. 1, pp. 1–66, 1989.
[59] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, 1993.
[60] M. Andrews and L. Zhang, "Hardness of the undirected edge-disjoint paths problem," in *Proc. 37th Annu. ACM Symp. Theory Comput.*, 2005, pp. 276–283.
[61] L. R. Ford and D. R. Fulkerson, "Constructing maximal dynamic flows from static flows," *Oper. Res.*, vol. 6, pp. 419–433, 1958.
[62] N. Robertson and P. D. Seymour, "Graph minors. XIII. The disjoint paths problem," *J. Combinatorial Theory, B*, vol. 63, no. 1, pp. 65–110, 1995.
[63] A. Felner, R. E. Korf, R. Meshulam, and R. C. Holte, "Compressed pattern databases." *J. Artif. Intell. Res.*, vol. 30, pp. 213–247, 2007.
[64] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2014. [Online]. Available: http://www.gurobi.com
[65] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.
[66] J. Yu and D. Rus, "An effective algorithmic framework for near optimal multi-robot path planning," in *Proc. Int. Symp. Robot. Res.*, 2015.



Jingjin Yu received the B.S. degree from the University of Science and Technology, Hefei, China, in 1998, the M.S. degrees in chemistry from the University of Chicago, Chicago, IL, USA, in 2000, in mathematics from the University of Illinois at Chicago, Chicago, in 2001, and in computer science from the University of Illinois at Urbana-Champaign, Champaign, IL, in 2010, and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, Champaign, in 2013.

He is an Assistant Professor in the Department of Computer Science at Rutgers University, New Brunswick, NJ, USA. He was with the University of Illinois at Urbana-Champaign, Champaign, as a Postdoctoral Researcher. He was a Postdoctoral Researcher with Massachusetts Institute of Technology from 2013 to 2015, with a joint appointment with Boston University from 2013 to 2014. He is broadly interested in the areas of robotics and control, focusing on issues related to computational complexity, and the design of efficient algorithms with provable guarantees.

Steven M. LaValle received the Ph.D. degree in electrical engineering from the University of Illinois, Champaign, IL, USA, in 1995.

He is a Professor in the Department of Computer Science at the University of Illinois. From 1995 to 1997, he was a Postdoctoral Researcher and a Lecturer with the Department of Computer Science, Stanford University, Stanford, CA, USA. From 1997 to 2001, he was an Assistant Professor with the Department of Computer Science, Iowa State University, Ames, Iowa, USA. His research interests include robotics, virtual reality, sensing, planning algorithms, computational geometry, and control theory. He is mostly known for his introduction of the Rapidly-Exploring Random Tree algorithm, which is widely used in robotics and other engineering fields. He was also an early Founder and Chief Scientist of Oculus VR, acquired by Facebook in 2014, where he developed patented tracking technology for consumer virtual reality. He led a team of perceptual psychologists to provide principled approaches to virtual reality system calibration, health and safety, and the design of comfortable user experiences. He also authored the books *Planning Algorithms*, *Sensing and Filtering*, and *Virtual Reality*.