Conflict-Free Multi-Agent Meeting

Dor Atzmon, Shahar Idan Freiman, Oscar Epshtein, Oran Shichman, Ariel Felner
Ben-Gurion University of the Negev
{dorat, freimans, oscarep, shichman}@post.bgu.ac.il, felner@bgu.ac.il

Abstract

Multi-Agent Meeting (MAM) is the problem of finding a meeting location for multiple agents and paths to that location. Practically, a solution to MAM may contain conflicting paths. A related problem that plans conflict-free paths to a given set of goal locations is the *Multi-Agent Path Finding* problem (MAPF). In this paper, we solve the *Conflict-Free Multi-Agent Meeting* problem (CF-MAM). In CF-MAM, we find a meeting location for multiple agents (as in MAM) as well as conflict-free paths (as in MAPF) to that location. We introduce two novel algorithms, which combine MAM and MAPF solvers, for optimally solving CF-MAM. We prove the optimality of both algorithms and compare them experimentally, showing the pros and cons of each algorithm.

Introduction and Background

Multi-Agent Meeting (MAM) (Yan, Zhao, and Ng 2015; Atzmon et al. 2020a) is the problem of finding a meeting location for multiple agents, and a path for each agent to that location (paths may conflict). MAM is practically applicable for finding a gathering location for multiple agents or choosing a point that is close to important locations. In other research areas, the problem is also known as *Optimal Meeting Point* (Yan, Zhao, and Ng 2015), *Smallest Enclosing Discs* (Welzl 1991), or *1-Center problem* (Megiddo 1983). In continuous Euclidean spaces, the problem is known as the *Weber problem* (Cooper 1968) and can be solved by different algorithms (Ostresh Jr 1977; Chen 1984; Rosing 1992).

On discrete graphs, the problem can be solved by (1) applying single-source shortest paths (SSSP) solver for each agent; (2) calculating the cost for reaching each location for each agent; and (3) iterating over all potential meeting locations and choosing a preferred location for a meeting. Enhancements for solving this problem have been proposed (Lanthier, Nussbaum, and Wang 2005; Yan, Zhao, and Ng 2015; Li et al. 2019a). Recently, a *Multi-Directional Heuristic Search* algorithm, called MM*, was introduced (Atzmon et al. 2020a). MM* is a state-of-the-art MAM optimal solver that searches from multiple directions (one for each agent) and is guided by a heuristic function.

In its common variant, MAM does not consider the possible physical body of the agents. Thus, a solution to MAM may contain conflicting paths for the agents, in which multiple agents occupy the same location at the same timestep or swapping locations between two consecutive timesteps. In this paper, we introduce the problem of *Conflict-Free MAM* (CF-MAM). In CF-MAM, we seek a meeting location for multiple agents and conflict-free paths to that location.

A related problem that finds conflict-free paths is the *Multi-Agent Path Finding* problem (MAPF) (Stern et al. 2019). MAPF is the problem of finding a path for each agent to a specified goal location, while avoiding conflicts with other agents. Although solving MAPF optimally is NP-hard (Yu and LaValle 2013b; Surynek 2010), some algorithms are cable of solving it optimally for many agents, e.g., *M** (Wagner and Choset 2015), *BIBOX* (Surynek 2012), *ICTS* (Sharon et al. 2013), and *Conflict-Based Search* (CBS) (Sharon et al. 2015). The latter, CBS, is a prominent algorithm that (1) plans a path for each agent, without considering other agents; and (2) repeatedly resolves conflicts by constraining each of the conflicting agents and replanning new paths. Many enhancements were introduced for CBS (Boyarski et al. 2015; Felner et al. 2018; Lam et al. 2019). In general, MAPF was investigated extensively and has many variants and extensions, including large agents (Li et al. 2019b), trains (Atzmon, Diei, and Rave 2019), convoys (Thomas, Deodhare, and Murty 2015), deadlines (Ma et al. 2018), and robustness (Atzmon et al. 2020b,c). Later in the paper, we introduce a new algorithm for CF-MAM that uses the framework of CBS.

A unique variant of MAPF is the *Permutation Invariant MAPF* problem (PI-MAPF) (Kloder and Hutchinson 2006). PI-MAPF is the problem of finding conflict-free paths to lead the agents to a set of goal locations that were not pre-assigned to the agents. This problem is also known as *Anonymous MAPF* (Ma and Koenig 2016) and *Unlabeled MAPF* (Solovey and Halperin 2016). PI-MAPF can be optimally solved in polynomial time by reducing the problem to Network-Flow (Yu and LaValle 2013a).

A special case of PI-MAPF is the *Shared-Goal MAPF* problem (SG-MAPF; also known as *Evacuation*) (Yu and LaValle 2013a), in which all goal locations are identical, i.e., finding conflict-free paths to a single goal location. There are number of possible assumptions for how agents behave at their goal locations (Stern et al. 2019). As all agents share a single goal, SG-MAPF assumes that when an agent

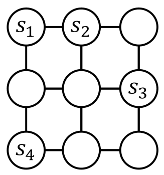
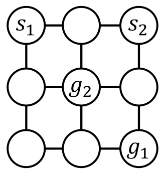
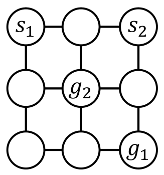
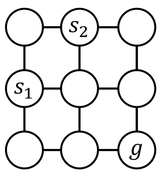
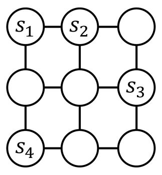
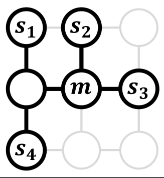
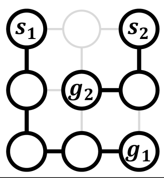
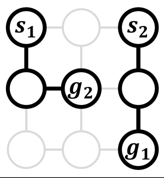
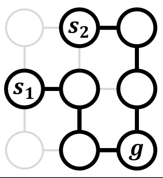
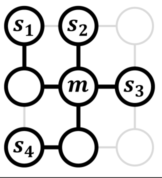
Problem	(1) MAM	(2) MAPF	(3) PI-MAPF	(4) SG-MAPF	(5) CF-MAM
Input	· Graph · Set of start locations 	· Graph · Set of start locations · Set of goal locations 	· Graph · Set of start locations · Set of goal locations 	· Graph · Set of start locations · Goal location 	· Graph · Set of start locations 
Output	Meeting location and paths (possibly conflicting) to the meeting location 	Conflict-free paths from each start location to a specified goal location 	Conflict-free paths from each start location to an unspecified goal location 	Conflict-free paths from each start location to the given goal location 	Meeting location and conflict-free paths to the meeting location 

Table 1: Related problems overview.

reaches the goal it immediately disappears. SG-MAPF can also be optimally solved by the same reduction used for PI-MAPF. We introduce a specifically designed reduction for SG-MAPF, in order to solve CF-MAM.

Table 1 summarizes all related problems. Optimal solutions to all these problems minimize some objective function. We focus on minimizing the *Sum-Of-Costs* (SOC) objective function, which is the sum of the costs of edges traversed by all agents until the agents have met.

We present two algorithms for solving CF-MAM optimally and prove their optimality and completeness. The first algorithm, CFM-CBS, uses the framework of CBS. CFM-CBS has two levels. Its low level solves the given problem as MAM (using MM*) under a given set of constraints. The high level of CFM-CBS repeatedly calls the low level, identifies conflicts, and resolves conflicts by imposing constraints on the conflicting agents. The second algorithm, called *Iterative Meeting Search* (IMS), iteratively solve the problem as SG-MAPF with different meeting locations until the optimal meeting location can be determined. We perform an experimental study that shows that each algorithm has circumstances where it performs best.

Definitions

Path-finding problems for multiple agents $A = \{a_1, \dots, a_k\}$ get as input an undirected connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a set of start locations $S = \{s_1, \dots, s_k\} \subset \mathcal{V}$. In this research, we assume that all edges have a unit cost, which is common in other conflict-free MAPF problems (Stern et al. 2019). As output, path-finding problems for multiple agents return a set of paths $\Pi = \{\pi_1, \dots, \pi_k\}$ for the agents. π_i is a path from a start location s_i to a destination location v_d. Such destination location can be known (i.e., given as input, as in the case of the various MAPF problems) or unknown (should be found). Let $N(v)$ represent the neighbors of v, i.e., $\forall v' \in N(v), (v, v') \in \mathcal{E}$. Each path $\pi_i \in \Pi$ for agent $a_i \in A$ consists of a sequence of locations, such that each two consecutive locations $v_1, v_2 \in \pi_i$ must either satisfy $v_2 \in N(v_1)$ (move action) or $v_1 = v_2 \in \mathcal{V}$ (wait action).

The cost of path π_i is denoted by $C(\pi_i)$ and equals the number of edges traversed in π_i ($C(\pi_i) = |\pi_i| - 1$). We use $\pi_i(t)$ to denote the t-th location in π_i. Thus, $\pi_i(0) = s_i$ and $\pi_i(|\pi_i|) = v_d$. We use $d(v_1, v_2)$ to denote the cost of a shortest path between the two locations v_1 and v_2. Trivially, if path π_i is a shortest path, $C(\pi_i) = d(s_i, v_d)$. Often, a set of paths that minimizes some objective function is preferred. In this work, we focus on minimizing the *Sum-Of-Costs* (SOC) objective function, which equals the sum of the costs of all paths in Π ($C(\Pi) = \sum_{\pi_i \in \Pi} C(\pi_i)$). We use C^* to denote the cost of the optimal (minimum cost) solution.

Some path-finding problems restrict the returned set of paths Π to be conflict-free. We focus on two main types of conflicts (Stern et al. 2019): *vertex conflict* and *swapping conflict*. A vertex conflict $\langle a_i, a_j, v, t \rangle$ occurs between two paths π_i and π_j if the same vertex $v \in \mathcal{V}$ is occupied by both agents a_i and a_j at the same timestep t, i.e., $\pi_i(t) = \pi_j(t) = v$. A swapping conflict $\langle a_i, a_j, e, t \rangle$ occurs between two paths π_i and π_j if the same edge $e \in \mathcal{E}$ is traversed in opposite directions by both agents a_i and a_j between the same two consecutive timesteps t and $t+1$, i.e., $(\pi_i(t), \pi_i(t+1)) = (\pi_j(t+1), \pi_j(t)) = e$.

Multi-Agent Meeting and the MM* Algorithm

The *Multi-Agent Meeting* problem (MAM) (Yan, Zhao, and Ng 2015; Atzmon et al. 2020a) gets as input the tuple $\langle \mathcal{G}, S \rangle$. A *solution* to MAM is a meeting location $m \in \mathcal{V}$ and a set of paths $\Pi = \{\pi_1, \dots, \pi_k\}$ from each $s_i \in S$ to the meeting location m. MAM can be optimally solved using a multi-directional heuristic search algorithm, called MM* (Atzmon et al. 2020a).[1] In MM*, a node is a pair (a_i, v) representing an agent and its location. MM* organizes nodes in a single open-list (denoted OPEN) and a single closed-list (denoted CLOSED). OPEN is initialized with k root nodes: (a_i, s_i) representing each of the k agents and its start location. Each node is associated with a g-value. Naturally, $g(a_i, s_i) = 0$. Expanding a node (a_i, v) has two parts: (1) generating a node (a_i, v') for each $v' \in N(v)$ (the neighbors of v), setting $g(a_i, v') = g(a_i, v) + 1$, and inserting it into OPEN; and (2) moving (a_i, v) to CLOSED.

Let $f^*(a_i, v)$ be the cost of the optimal MAM solution such that a_i passes through v on its way to the meeting location. $f(a_i, v)$ is a lower bound on $f^*(a_i, v)$, i.e., $f(a_i, v) \leq f^*(a_i, v)$. In MM*, OPEN prioritizes nodes according to lower f-values. v is a *possible goal* if it has been generated from all directions, i.e., $\forall a_i \in A$, $(a_i, v) \in$ OPEN$\cup$CLOSED. If v is a possible goal, $C(v) = \sum_{a_i \in A} g(a_i, v)$. Let U be the minimum $C(v)$ among all possible goals. U is an upper bound on C^*. MM* halts if $fmin \geq U$, where $fmin$ is the minimum f-value in OPEN. This guarantees that U cannot be further improved and MM* returns an optimal solution.

Consider node (a_i, v) in OPEN. $f^*(a_i, v)$ can be disassembled into 3 items: (1) a_i reaches location v; (2) a_i continues from v to m; and (3) each of the other agents a_j travels from s_j to m. $f^*(a_i, v) = g(a_i, v) + h^*(a_i, v)$, where $h^*(a_i, v)$ is the sum of the cost of a_i to get from v to m (item 2), plus the cost of the other agents to get from their start locations to m (item 3). Formally,

$$h^*(a_i, v) = \min_{m \in \mathcal{V}} \{d(v, m) + \sum_{a_j \in A \setminus \{a_i\}} d(s_j, m)\}.$$

The optimal meeting location w.r.t. to node (a_i, v) is denoted by $m^*(a_i, v)$ (with a cost of $f^*(a_i, v)$). $h(a_i, v)$ is an admissible estimate (lower bound) of $h^*(a_i, v)$, i.e., $h(a_i, v) \leq h^*(a_i, v)$. For SOC, naturally, $f(a_i, v) = g(a_i, v) + h(a_i, v)$. Let $S_i(v)$ be the set of all start locations in S, except for s_i, which is replaced with v (the current location of a_i). Formally, $S_i(v) = S \setminus \{s_i\} \cup \{v\}$. Then,

$$h^*(a_i, v) = \sum_{v' \in S_i(v)} d(v', m^*(a_i, v)).$$

Atzmon et al. (2020a) proposed a number of admissible heuristic functions $h(a_i, v)$ for MAM. Here, we use one such function, called the *Clique heuristic*, which balances well between simplicity and efficiency. The clique heuristics assumes that for every pair of locations (v_1, v_2), there exists a classic admissible heuristic h, such that $h(v_1, v_2) \leq d(v_1, v_2)$ (e.g., Manhattan distance). The clique heuristic uses admissible estimates for every pair of locations in $S_i(v)$ to create the following admissible heuristic for MAM.

$$h(a_i, v) = \sum_{\substack{\{v_1, v_2\} \in 2^{S_i(v)} \\ v_1 \neq v_2}} \frac{h(v_1, v_2)}{k-1} \leq h^*(a_i, v).$$

Intuitively, the $k - 1$ in the denominator is a result of summing heuristics for each agent with the other $k - 1$ agents.

[1]MM* is a generalization of the bidirectional search algorithm MM (Holte et al. 2016) to multiple agents.

Conflict-Free Multi-Agent Meeting

While MAM finds a meeting location and paths for multiple agents to that meeting location, it ignores conflicts between the agents. Often, path-finding problems for multiple agents, such as MAPF, are restricted to only return conflict-free paths. Here, we define the problem of finding a meeting location and conflict-free paths to the meeting location.

The *Conflict-Free Multi-Agent Meeting* problem (CF-MAM) gets as input the tuple $\langle \mathcal{G}, S \rangle$, where $\mathcal{G}$ is an undirected connected graph and S is a set of start locations. A *solution* to CF-MAM is a meeting location m and a set of conflict-free paths Π to the meeting location m. While agents must avoid conflicts on their path to m, naturally, we define that agents can arrive at m at the same timestep. This is practical, for example, in the case that agents disappear at goal (Stern et al. 2019) (e.g., robots entering a charging station or autonomous vehicles entering a garage).

In some cases, a solution to MAM is invalid for CF-MAM. In the examples for MAM and CF-MAM in Table 1 (columns 1 and 5) both get similar input. In the output example depicted for MAM, both agents a_1 and a_4 conflict at timestep 1. As CF-MAM does not allow either vertex conflicts nor swapping conflicts, this solution is invalid for CF-MAM. Thus, in the output example for CF-MAM, the path of agent a_4 is modified to a non-conflicting path.

Canceling Swapping Conflicts

Lemma 1. *Let Π be a set of paths from a set of start locations S to a meeting location m with a cost of $C(\Pi)$, such that Π only contains swapping conflicts (and no vertex conflicts). There exists a set of conflict-free paths (without vertex conflicts and without swapping conflicts) $\Pi' = \{\pi'_1, \ldots, \pi'_k\}$ from S to m with the same cost of $C(\Pi)$.*

Proof. Consider a swapping conflict $\langle a_i, a_j, e, t \rangle$ between paths π_i and π_j in Π. The agents a_i and a_j cannot conflict at timestep t with another agent, as a swapping conflict with more than two agents must result in a vertex conflict. Now, we can create two new paths π'_i and π'_j from start locations s_i and s_j, respectively, to m such that $C(\pi'_i) + C(\pi'_j) = C(\pi_i) + C(\pi_j)$ and the swapping conflict is canceled, as follows. First, for each timestep $t' \leq t$ we set $\pi'_i(t') \leftarrow \pi_i(t')$ and $\pi'_j(t') \leftarrow \pi_j(t')$. By definition of a swapping conflict, $(\pi_i(t), \pi_i(t+1)) = (\pi_j(t+1), \pi_j(t)) = e$. Thus, instead of swapping locations, we can force the agents to wait (not traverse e) and continue following the path of the other agent. This can be done by setting for each timestep $t' > t$, $\pi'_i(t') \leftarrow \pi_j(t')$ and $\pi'_j(t') \leftarrow \pi_i(t')$. Obviously, performing this mechanism maintains $C(\pi'_i) + C(\pi'_j) = C(\pi_i) + C(\pi_j)$. This can be performed repeatedly for each swapping conflict until Π' is conflict-free and obtains the cost $C(\Pi)$. □

Figure 1 presents an example of a MAM problem instance where only agents a_1, a_2, and a_3 are presented in the figure. For simplicity, we ignore the rest of the agents. Let m be a meeting location and let Π contain the paths of a_1, a_2, and a_3 such that $\pi_1 = (s_1, s_2, A, m)$, $\pi_2 = (s_2, s_1, s_2, A, m)$, and $\pi_3 = (s_3, A, m)$. Π contains the swapping conflict $\langle a_1, a_2, (s_1, s_2), 0 \rangle$, and no vertex conlicts. Following

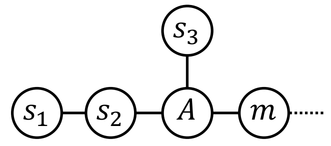


Figure 1: Example of cancelling a swapping conflict.

Lemma 1, we can construct $\pi'_1 = (s_1, s_1, s_2, A, m)$ and $\pi'_2 = (s_2, s_2, A, m)$ which resolves the swapping conflict without increasing the cost.

In the next two sections, we introduce two algorithms for CF-MAM. As a (conflict-free) solution can be constructed from a set of paths that only contains swapping conflicts (Lemma 1), both algorithms only consider vertex conflicts.

CBS-Based Solution for CF-MAM

Conflict-Based Search (CBS) (Sharon et al. 2015) is a prominent, state-of-the-art MAPF solver. It plans a set of paths that may contain conflicts and iteratively resolve them by imposing constraints on the agents and replanning new paths for the constrained agents. Here, we introduce the CFM-CBS algorithm for solving CF-MAM, which uses the framework of the CBS algorithm. CFM-CBS has two levels. The high level of CFM-CBS searches the binary *constraint tree* (CT). Each node $N \in CT$ contains: (1) a set of constraints imposed on the agents ($N.constraints$); (2) a set of paths ($N.\Pi$) that satisfies all constraints in $N.constraints$; and (3) the cost of $N.\Pi$ ($N.cost$). A *constraint* is a tuple $\langle a_i, v, t\rangle$ that prohibits agent a_i to occupy location v at timestep t. We use such constraints for resolving conflicts, as explained below. The root node of CT contains an empty set of constraints. The high level searches the CT in a best-first manner, prioritizing nodes with lower cost.

Generating a CT node. Given a node N, the low level of CFM-CBS solves the given CF-MAM problem instance as a MAM problem that satisfies all constraints of node N. Such a solution can be achieved using any MAM solver, such as MM*.[2] However, to support constraints, as well as wait actions, MM* needs to be slightly modified to CF-MM*. Thus, instead of the pair (a_i, v), a node in CF-MM* is a tuple of (a_i, v, t) representing an agent and its location at timestep t. In CF-MM* an *invalid node* (a_i, v, t) is a node that violates the constraint $\langle a_i, v, t\rangle$. CF-MM* may generate invalid nodes as such nodes may be meeting locations. However, if an invalid node N is chosen for expansion, CF-MM* discards N and does not expand it.

Expanding a CT node. Once CFM-CBS has chosen a node N for expansion, it checks its paths $N.\Pi$ for conflicts. If it is conflict-free, then node N is a goal node and CFM-CBS returns its solution. Otherwise, CFM-CBS *splits* node N on one of the conflicts $\langle a_i, a_j, v, t\rangle$ by generating two children for node N. Each child node has a set of constraints that is the union of $N.constraints$ and a new constraint. One of the two children adds the new constraint $\langle a_i, v, t\rangle$ and the other child adds the new constraint $\langle a_j, v, t\rangle$.

[2]Originally, to solve MAPF, the low level of CBS finds a path for each individual agent, e.g., using A*.

Algorithm 1: High level of CFM-CBS

```
1  Main(CF-MAM problem $instance$)
2      $Root.constraints \leftarrow \{\}$
3      $Root.\Pi \leftarrow$ low-level($instance, Root.constraints$)
4      $Root.cost \leftarrow SOC(Root.\Pi)$
5      Insert $Root$ into OPEN
6      while OPEN is not empty do
7          $N \leftarrow$ Pop the node with the lowest cost in OPEN
8          if $N.\Pi$ is conflict-free then
9              return $N.\Pi$ // $N$ is goal
10         $\langle a_1, a_2, v, t\rangle \leftarrow$ get-conflict($N$)
11         $N_1 \leftarrow$ GenChild($N$,$\langle a_1, v, t\rangle$)
12         $N_2 \leftarrow$ GenChild($N$,$\langle a_2, v, t\rangle$)
13         Insert $N_1$ and $N_2$ into OPEN
14     return No Solution
15 GenChild(Node $N$, Constraint $NewCons$)
16     $N'.constraints \leftarrow N.constraints \cup \{NewCons\}$
17     $N'.\Pi \leftarrow$ low-level($instance, N'.constraints$)
18     $N'.cost \leftarrow SOC(N'.\Pi)$
19     return $N'$
```

Pseudo code. Algorithm 1 presents the pseudo code of the high level of CFM-CBS. In Lines 2-5, we generate the root CT node. Then, while OPEN is not empty, we iteratively explore CT nodes. In Line 7, we extract from OPEN the CT node N with the lowest cost. If $N.\Pi$ is conflict-free (Line 8), it is a solution and returned in Line 9. Otherwise, we get one of the conflicts in $N.\Pi$ (Line 10) and resolve it by imposing constraints and generating two new CT nodes (Lines 11-13).

Example. Figure 2 presents an example of (a) a CF-MAM problem instance with five agents; and (b) its corresponding CT, created by CFM-CBS. First, at the root CT node, we call CF-MM* under an empty set of constraints. A set of paths Π with a cost of 7 is returned, in which the agents meet at location v_2. Location v_2 is closer to a larger number of agents (agents a_3, a_4, and a_5) than other locations, and thus has a lower cost. At the root, both agents a_1 and a_2 conflict at location v_1 at timestep 1, and hence Π is not a solution. We create two CT nodes with the constraints $\langle a_1, v_1, t\rangle$ and $\langle a_2, v_1, t\rangle$, and call CF-MM* under each of the constraints. Then, one of the new CT nodes is chosen for expansion and a solution with a cost of 8 is returned (the agent waits at its start location). Notice that while the agents meet at location v_2 at the root node and at both child CT nodes, the meeting location may change under a different set of constraints.

Lemma 2 (Completeness)**.** *CFM-CBS is guaranteed to return a solution.*

Proof outline. CFM-CBS performs a best-first search on the CT, where the cost cannot decrease, i.e., newly generated CT nodes cannot have lower cost than the current lowest costs of any CT node in OPEN. The number of sets of paths with a cost that is *smaller than or equal to* the cost of the optimal solution is finite. By resolving a conflict at some node N, one or more such sets of paths are avoided, i.e., sets of paths that contain the resolved conflict. Thus, after resolving

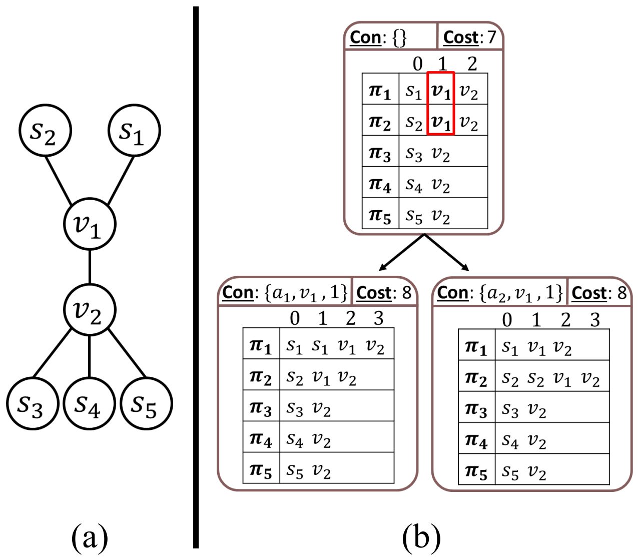


Figure 2: CF-MAM problem instance and CFM-CBS's CT.

a finite number of conflicts, the node with the lowest cost in OPEN contains a solution. □

Lemma 3 (Optimality)**.** *When CFM-CBS returns a solution, the solution has the lowest cost among all solutions.*

Proof outline. CFM-CBS never eliminates solutions by splitting a CT node. It performs a best-first search on the CT where the costs cannot decrease. Thus, the cost of an expanded node is a lower bound on the cost of all solutions, and the first expanded node with a solution contains the solution with the lowest cost. □

Iterative Meeting Solution for CF-MAM

As described above, PI-MAPF (Kloder and Hutchinson 2006) is the problem of finding conflict-free paths to a set of goal locations G that were not pre-assigned to the agents. For each agent a_i, an algorithm for PI-MAPF assigns a goal location $g_i \in G$ and finds a proper path for that agent. SG-MAPF (Yu and LaValle 2013a) is a special case of PI-MAPF in which for each agent a_i we set $g_i = g$ (the same goal location g). As PI-MAPF can be optimally solved in polynomial time using a reduction to Network Flow (Yu and LaValle 2013a), SG-MAPF can also be optimally solved in polynomial time using the same reduction.

Naively, CF-MAM can be solved by (1) solving SG-MAPF $|\mathcal{V}|$ times, each time with a different location $v \in \mathcal{V}$ as a goal location; and (2) determining the optimal meeting location, based on the cost of meeting at each location. Thus, CF-MAM can be optimally solved in polynomial time. Experimentally, this naive algorithm fails to solve many of our problems. It only solved small domain problems, and slower than the enhanced algorithm described below.

In this section, we first describe a specially designed reduction from SG-MAPF to Network Flow. Our new reduction is more suitable for SG-MAPF than the reduction presented by Yu and LaValle (2013a) for PI-MAPF, although it borrows some concepts. Then, we introduce the *Iterative-Meeting Search* algorithm (IMS), which intelligently iterates over relevant meeting locations and sets each as a goal location in SG-MAPF.

Network Flow Problems

To refresh the memory of the reader, we first provide a brief description of a *network* and a *Minimum-Cost Flow* problem (MCFP), which we use later in our reduction.

A network $N = \left\langle \overrightarrow{\mathcal{G}}, u, c, source, sink \right\rangle$ consists of a directed graph $\overrightarrow{\mathcal{G}} = (\mathcal{V}, \mathcal{E})$ with capacities u and costs c on the edges, i.e., $\forall e \in \mathcal{E}, u(e), c(e) \in \mathbb{Z}^+$, and $source, sink \in \mathcal{V}$. Let $\delta^+(v)$ and $\delta^-(v)$ denote the sets of edges entering and leaving v, respectively. Given network N, a feasible flow f ($\forall e \in \mathcal{E}, f(e) \in \mathbb{Z}^+$) must satisfy the following constraints.
(1) Edge capacity constraint,

$$\forall e \in \mathcal{E}, f(e) \leq u(e).$$

(2) Flow conservation constraint at non sink/source vertices,

$$\forall v \in \mathcal{V} \setminus \{source, sink\}, \sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e).$$

(3) Flow conservation constraint at sink and source vertices,

$$\sum_{e \in \delta^-(sink)} f(e) = \sum_{e \in \delta^+(source)} f(e).$$

Definition 1 (*Minimum-Cost Flow problem* (MCFP))**.** *Given a network N, the MCFP problem returns a minimum-cost feasible flow f, i.e.,* $\min(\sum_{e \in \mathcal{E}} c(e) \cdot f(e))$.

The cost–scaling algorithm (Goldberg and Tarjan 1990; Goldberg 1997) is a benchmark, commonly used polynomial time solver for MCFP. We used it in our experiments below.

Reducing SG-MAPF to Network Flow

Recall that the input to SG-MAPF is $\langle \mathcal{G}, S, g \rangle$. For this reduction, we need to first calculate an upper bound T on the depth (the latest timestep) l' of a solution to SG-MAPF. Let Π be a set of shortest individual paths from each start location $s_i \in S$ to g, in a relaxed variant of SG-MAPF, which allows conflicts between the agents. Let l denote the length of the longest path in Π. Similarly, let Π' be an optimal solution for (standard conflict-free) SG-MAPF and let l' denote the length of the longest path in Π'. In Π, in the worst case, the agent of the longest individual path (with a cost of l) conflicts with all other $k-1$ agents. Hence, in Π', the path of this agent may be extended by one timestep for each of the other $k-1$ agents. That is, for each of the $k-1$ conflicts the agent waits one timestep. Thus, following Yu and LaValle (2013a), $T = l + k - 1$ is a tight upper bound on l', i.e., $T \geq l'$ (for more details, see Yu and LaValle (2013a)).

We next describe our reduction. In our network, each node represents a pair of a location $v \in \mathcal{V}$ and timestep t, i.e., (v, t). Given a SG-MAPF problem instance $\langle \mathcal{G}, S, g \rangle$, we build a network by the following three steps.

Step 1. We build the network backwards from the goal. First, we build the pair (g, T) and set $t \leftarrow T$. Then, while

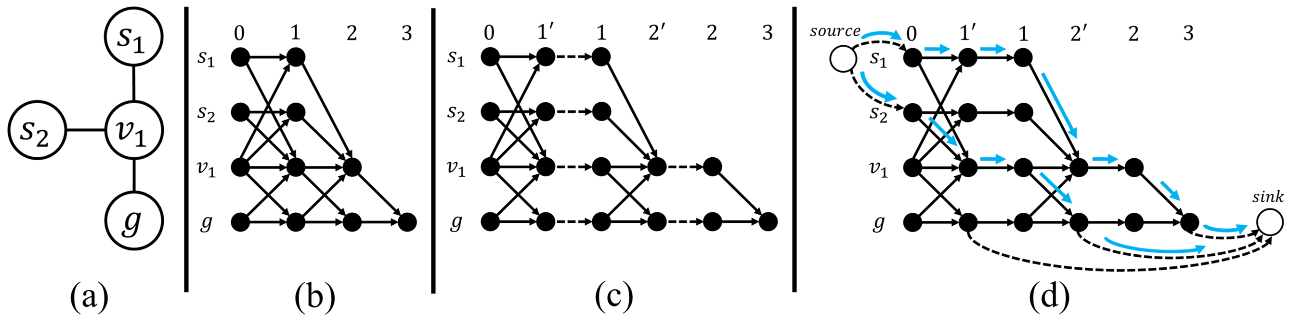


Figure 3: Reducing SG-MAPF to Network Flow. Numbers on the x-axis are timesteps and letters on the y-axis are locations.

$t > 0$, we perform the following: (i) for each node $N = (v, t)$ with timestep t, for each location $v' \in \{v\} \cup N(v)$ (either v or a neighbour of v in $\mathcal{G}$) we create a new node $N' = (v', t-1)$ and a directed edge (N', N) with a capacity and cost of 1. These edges correspond to traversing edges on $\mathcal{G}$, which costs 1 and can be performed by a single agent at a given timestep (capacity 1); (ii) set $t \leftarrow t-1$ and go back to (i) until $t = 0$. Figure 3(a) presents a SG-MAPF problem instance with two agents ($k = 2$). As the length of the longest individual path is 2, then $l = 2$. Thus, $T = l + k - 1 = 3$ is an upper bound on the longest path in the optimal solution. Figure 3(b) presents the corresponding network, built after executing step 1. The construction of this network begins at the bottom-right side, at node $(g, 3)$, and progresses left. In this figure, moving horizontally corresponds to wait actions and moving diagonally corresponds to move actions.

Step 2. Figure 3(c) shows the network after executing step 2, which is done as follows. To prevent agents from occupying the same location at the same timestep, we split each node $N = (v, t)$ at each timestep $0 < t < T$ to two nodes: $N = (v, t)$ and $N' = (v, t')$ (t with an apostrophe). All in-edges of node N, are transferred to enter node N' (solid edges). In addition, we add an edge (N', N) (dashed edges) with a capacity of 1 and a cost of 0. Dashed edges force that only one agent may enter the N nodes due to their capacity.

Step 3. Finally, as depicted in Figure 3(d), we add a source vertex and a sink vertex. For each start location $s_i \in S$ we add an edge $(source, s_i)$ with a capacity of 1 and a cost of 0. For all nodes $N' = (g, t')$, as well as for node $N' = (g, T)$, we add an edge $(N', sink)$ with a capacity of k and a cost of 0. All these edges are the dashed edges in the figure. The bold blue arrows show the solution returned by executing an MCFP solver. This solution corresponds to a solution of $\pi_1 = (s_1, s_1, v_1, g)$ and $\pi_2 = (s_2, v_1, g)$, which costs 5.

Independence detection. To determine T, as described above, we need to calculate all shortest individual paths Π, e.g., using a breadth-first search from g. As the runtime of an MCFP solver is mainly influenced by T (Yu and LaValle 2013a), before reducing SG-MAPF we execute an *independence-detection* mechanism (Standley 2010). Such mechanism detects paths in Π that can be kept for the returned solution and not be passed to the reduction. For this purpose, we first identify conflicts between the agents in Π. We only reduce agents that either conflict or may conflict after the reduction, based on the correspond upper bound T. For example, assume agents $\{a_1, \ldots, a_4\}$ with shortest individual paths Π of lengths $4, 4, 5$, and 7, respectively, in which the paths of a_1 and a_2 conflict. We know that, in the optimal solution, one of agents a_1 or a_2 may have a path of length 5 after the reduction. Thus, we add a_3 to the group of agents that will be reduced. However, a_4 will not conflict with the other three agents, as the longest path of the three agents can only reach a length of 6 (Yu and LaValle 2013a) and they will be at least one step ahead on their way to g.

There are a number of differences between our reduction and the reduction from PI-MAPF, introduced by Yu and LaValle (2013a). (1) In our reduction the graph $\mathcal{G}$ is constructed by performing a single breadth-first search from g, instead of k breadth-first searches (from the k start locations) in their reduction. (2) We apply independence detection to construct a smaller network flow. (3) Following Lemma 1, there is no need for avoiding swapping conflicts, which requires a special step by Yu and LaValle (2013a). Therefore, optimally follows. Moreover, Yu and LaValle (2013a) proved that PI-MAPF, as well as SG-MAPF, always have a valid solution. Thus, CF-MAM too can always be solved.

Iterative Meeting Search

We now present the *Iterative Meeting Search* algorithm (IMS). It has two levels. The high level of IMS iteratively examines possible meeting locations until the optimal meeting location can be determined. This is done by a best-first search on possible meeting locations. We describe this process below. The low level sets each possible meeting location as a goal location of SG-MAPF and applies a network-flow solver to solve it using our reduction.

Algorithm 2 describes the pseudo code of the high level of IMS. First, it initializes OPEN, CLOSED, and an upper bound on the cost of the optimal solution U ($U \geq C^*$) with infinity (Line 2). The high level performs a best-first search, starting from only one of the start locations s_i ((a_i, s_i) is inserted to OPEN; Line 3). We explain how s_i can be selected below. While OPEN is not empty, an expansion cycle is performed in Lines 4-16. Each expansion cycle starts by extracting the node (a_i, v) with the lowest f-value (the same f-value as in MM*) from OPEN (Line 5). As MAM is a relaxed problem of CF-MAM, for the same input $\langle \mathcal{G}, S \rangle$, the cost C' of the optimal solution for MAM is a lower bound on the cost C^*

Algorithm 2: High level of IMS

```
1  Main(CF-MAM problem $instance$)
2      Init OPEN, CLOSED; $U \leftarrow \infty$
3      Insert $(a_i, s_i)$ into OPEN // only a single start location
4      while OPEN is not empty do
5          Extract $(a_i, v)$ from OPEN // with lowest $f(a_i, v)$
6          $U \leftarrow \min(U, low\text{-}level(instance, v))$
7          if $f(a_i, v) \geq U$ then
8              return $U$
9          foreach $v' \in N(v)$ do
10             if CLOSED contains $(a_i, v')$ then
11                 continue
12             else if OPEN contains $(a_i, v')$ then
13                 if $g(a_i, v') \leq g(a_i, v) + 1$ then
14                     continue
15             Insert $(a_i, v')$ into OPEN
16         Insert $(a_i, v)$ into CLOSED
17     return $U$
```

of the optimal solution for CF-MAM, i.e., $C' \leq C^*$. Thus, since f is a lower bound on C', it is also a lower bound on C^*. For each node (a_i, v) selected for expansion, the high level calls the low level to calculate the cost of meeting at location v (by the above reduction). Then, U is updated with the lowest cost found (Line 6).[3] As U is an upper bound on the cost of the optimal solution C^*, if $fmin \geq U$ then IMS halts and the optimal solution is found ($C^* = U$), where $fmin$ is the lowest f in OPEN (Lines 7-8). Otherwise, in case the optimal solution is still not found, for each neighbor v' of v, the high level inserts (a_i, v') to OPEN and moves (a_i, v) to CLOSED (Lines 10-16). Note that the node (a_i, v') is not inserted to OPEN in case it is either in CLOSED or in OPEN with a lower or equal cost (Lines 10-14).

Starting the search. In our experiments, we started the search from the start location with the highest *closeness centrality* among all start locations in S, where the closeness centrality of a start location s_i can be estimated by $\sum_{s_j \in S \setminus \{s_i\}} \frac{1}{h(s_i, s_j)}$, where h is an admissible heuristic in the underlying graph $\mathcal{G}$ between any two points. We found that IMS with this start location perform better than random. This is reasonable as such a location is usually closer to the optimal meeting location. Future work can investigate different start locations for IMS.

Lemma 4 (Completeness). *IMS is guaranteed to return a solution.*

Proof outline. IMS starts the search by calling the low level for the selected start location s_i. The low level returns a valid solution for meeting at s_i. IMS either returns this solution or a solution of lower cost. In the worst case, IMS explores every reachable location (OPEN will be empty), the search will halt and a solution will be returned. □

[3] In the pseudo code we only keep U, but IMS also returns the paths of the optimal solution and the optimal meeting location.

#Agents	Solver	10 × 10 Cost	10 × 10 Time	50 × 50 Succ.	50 × 50 Cost	50 × 50 Time
3	CFM-CBS	11	**0.0**	**50**	59	**0.0**
	IMS		**0.0**	**50**		0.3
5	CFM-CBS	23	**0.0**	**50**	106	**0.5**
	IMS		**0.0**	**50**		8.3
7	CFM-CBS	34	**0.1**	**50**	155	**3.9**
	IMS		**0.1**	49		40.9
9	CFM-CBS	45	**0.3**	**49**	204	**26.3**
	IMS		0.4	46		126.4
11	CFM-CBS	56	9.5	**39**	245	**50.5**
	IMS		**0.8**	23		200.2
13	CFM-CBS	67	30.2	**29**	-	-
	IMS		**1.2**	3		
15	CFM-CBS	79	57.3	**21**	-	-
	IMS		**1.7**	1		

Table 2: Results for 10x10 and 50x50 grids with 20% Obs.

Lemma 5 (Optimality). *IMS is guaranteed to return the optimal meeting location m^* with cost C^*.*

Proof outline. Let $s_i \in S$ be the start location of agent a_i from which IMS start searching. Assume, by contradiction, that IMS returned a sub-optimal location $m \neq m^*$ with cost $C > C^*$. Since IMS has terminated and returned a solution, $fmin \geq C > C^*$. Since IMS terminated without returning an optimal solution, there exists a node $N' = (a_i, v_i) \in$ OPEN such that v_i is a location on the path of agent a_i to location m^* in the optimal solution, and every node before N' on that path has already been expanded. Since N' is the first node on that path that was not expanded, it was generated by a node on that path, and thus $g(N') = d(s_i, v_i)$. By definition, $f^*(N')$ is the cost of the optimal solution that passes through N', assuming conflicts are ignored (MAM). Hence, $f^*(N')$ is a lower bound on the optimal cost C^*, considering conflicts, i.e., $f^*(N') \leq C^*$. f is admissible, and therefore, $f(N') \leq f^*(N') \leq C^*$. As $N' \in$ OPEN, $fmin \leq f(N') \leq f^*(N') \leq C^*$, which contradicts the fact that $fmin \geq C > C^*$. □

Experiments

We experimented on an Intel® Xeon E5-2660 v4 @2.00GHz processor with 16GB of RAM. For CFM-CBS, we used CF-MM* as a low-level solver. For IMS, we used for solving the Minimum-Cost Flow problem (MCFP) an efficient implementation (Goldberg 1997) of the cost–scaling algorithm of Goldberg and Tarjan (1990), which runs in polynomial time. For both, we used the clique heuristic as an admissible heuristic for a meeting location and Manhattan distance as admissible heuristic between two locations.

Random Grids

We compared CFM-CBS and IMS on 10x10 and 50x50 grids with 20% randomly placed obstacles, and $3, 5, 7, 9, 11, 13$, and 15 randomly allocated agents. We created 50 problem instances for each combination and mea-

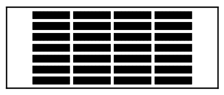
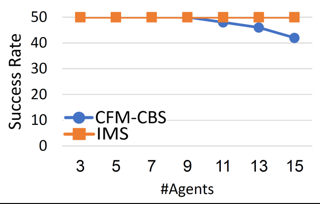
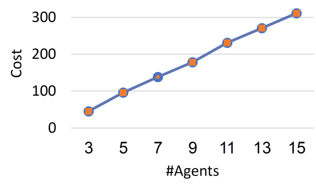
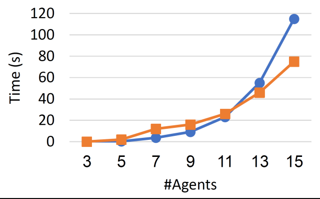
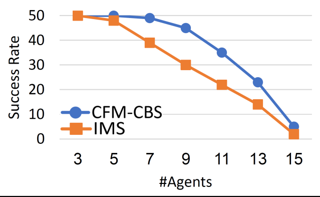
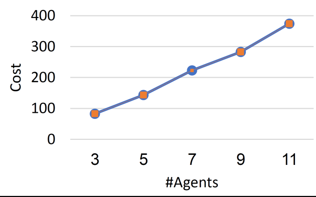
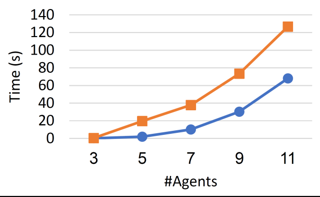
Map	Success Rate	Cost	Time (s)
			
			

Table 3: Results for the warehouse domain (first row) and the den312d map from DAO (second row).

sured the success rate (for timeout of 5min for each instance), average cost, and average time (seconds). Table 2 presents the results for this experiment. Each row shows the number of agents. The results for the 10x10 grids and for the 50x50 grids are in columns 3-4 and 5-7, respectively.

For the 10x10 grids, both CFM-CBS and IMS solve all problem instances and hence we do not present the success rate in the table. As expected, larger number of agents increases the average cost and the average time for both solvers. However, the influence of this increase is greater for CFM-CBS than for IMS. For example, for 7 agents, both solvers ran for $\sim 0.1s$, and for 15 agents, CFM-CBS ran $\sim 57.3s$ while IMS ran only $\sim 1.7s$. The runtime of CBS-based solutions is exponential in the number of conflicts it resolves. Thus, CFM-CBS does not perform well in dense environments, such as small grids with many agents.

For the 50x50 grids, not all instances were solved by both solvers within the 5min timeout. As the number of agents increased, both solvers solved fewer instances. However, CFM-CBS solved more instances than IMS. For 13 agents, CFM-CBS solved 29 problem instances while IMS only solved 3. The average cost and average time in the table were calculated from instances that were solved by both solvers. The same trend that was observed for the success rate can be seen for the time: CFM-CBS was faster than IMS. For 11 agents, CFM-CBS and IMS ran $\sim 50.5s$ and $\sim 200.2s$, respectively. Here, the environment is sparser and fewer conflicts occur. Thus, CFM-CBS can perform better than observed above.

Structured Maps

We also tested CFM-CBS and IMS on (1) a warehouse map, used by Ma et al. (2017) and Atzmon et al. (2020c); and on (2) the den312d map from the Dragon Age Origins (DAO) video game, which is publicly available (Sturtevant 2012). The leftmost column in Table 3 shows figures of both maps, respectively. We created 50 problem instances with $3, 5, 7, 9, 11, 13$, and 15 randomly allocated agents, and measured the success rate, average cost, and average time in seconds (columns 2-4 in Table 3, respectively). Here also, the average cost and average time were calculated only from instances that were solved by both solvers.

For the warehouse map, while IMS solved all instances, a few instances were not solved by CFM-CBS. This is similar to the trend that was observed in the 10x10 grids above; the environment becomes denser, more conflicts occur, and the problem becomes harder for CFM-CBS to solve. This trend can also be seen in the time figure for 13 and 15 agents: the runtime of CFM-CBS exceeds the runtime of IMS.

The den312d map is larger than the warehouse and fewer instances were solved by both solvers. Similarly to the 50x50 grids, CFM-CBS solved more instances than IMS and ran faster in instances that were solved by both solvers because the environment was sparse.

Our experiments provide the following general rule. CFM-CBS should be used in sparse environments while IMS should be used in dense environments.

Conclusions and Future Work

In this paper, we explored the problem of *Conflict-Free Multi-Agent Meeting* (CF-MAM), in which a meeting location is required for multiple agents as well as conflict-free paths to that meeting location. For solving CF-MAM, we introduced two algorithms: CFM-CBS and IMS. We proved that both algorithms are complete and optimal and compared them experimentally. Our experiments showed that IMS performs better in denser domains while CFM-CBS performs better in sparser domains. Choosing a solver in environments that are not clearly sparse or dense is left for future work.

Moreover, future work may improve both algorithms. For CFM-CBS, one may adjust many of the improvements that were done for CBS (such as prioritizing conflicts (Boyarski et al. 2015)). For IMS, a number of improvements, e.g., more sophisticated rules for calling the low level, are in place. Finally, other MAPF solvers, such as ICTS (Sharon et al. 2013) may be adjusted for CF-MAM.

Acknowledgements

This research was supported by ISF grant 844/17 and BSF grant 2017692 to Ariel Felner.

References

Atzmon, D.; Diei, A.; and Rave, D. 2019. Multi-Train Path Finding. In *SoCS*, 125–129.

Atzmon, D.; Li, J.; Felner, A.; Nachmani, E.; Shperberg, S. S.; Sturtevant, N.; and Koenig, S. 2020a. Multi-Directional Heuristic Search. In *IJCAI*, 4062–4068.

Atzmon, D.; Stern, R.; Felner, A.; Sturtevant, N. R.; and Koenig, S. 2020b. Probabilistic robust multi-agent path finding. In *ICAPS*, 29–37.

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2020c. Robust multi-agent path finding and executing. *JAIR* 67: 549–579.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. E. 2015. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *IJCAI*, 740–746.

Chen, R. 1984. Location problems with costs being sums of powers of Euclidean distances. *COR* 11(3): 285–294.

Cooper, L. 1968. An extension of the generalized Weber problem. *JRS* 8(2): 181–197.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2018. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. In *ICAPS*, 83–87.

Goldberg, A. V. 1997. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of algorithms* 22(1): 1–29.

Goldberg, A. V.; and Tarjan, R. E. 1990. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research* 15(3): 430–466.

Holte, R. C.; Felner, A.; Sharon, G.; and Sturtevant, N. R. 2016. Bidirectional Search That Is Guaranteed to Meet in the Middle. In *AAAI*, 3411–3417.

Kloder, S.; and Hutchinson, S. 2006. Path planning for permutation-invariant multirobot formations. *IEEE Transactions on Robotics* 22(4): 650–665.

Lam, E.; Bodic, P. L.; Harabor, D.; and Stuckey, P. J. 2019. Branch-and-Cut-and-Price for Multi-Agent Pathfinding. In *IJCAI*, 1289–1296.

Lanthier, M. A.; Nussbaum, D.; and Wang, T.-J. 2005. Calculating the meeting point of scattered robots on weighted terrain surfaces. In *CATS*, volume 41, 107–118.

Li, J.; Felner, A.; Koenig, S.; and Kumar, T. K. S. 2019a. Using FastMap to Solve Graph Problems in a Euclidean Space. In *ICAPS*, 273–278.

Li, J.; Surynek, P.; Felner, A.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2019b. Multi-Agent Path Finding for Large Agents. In *AAAI*, 7627–7634.

Ma, H.; and Koenig, S. 2016. Optimal Target Assignment and Path Finding for Teams of Agents. In *AAMAS*, 1144–1152.

Ma, H.; Kumar, T. S.; and Koenig, S. 2017. Multi-Agent Path Finding with Delay Probabilities. In *AAAI*, 3605–3612.

Ma, H.; Wagner, G.; Felner, A.; Li, J.; Kumar, T. S.; and Koenig, S. 2018. Multi-agent path finding with deadlines. In *IJCAI*, 417–423.

Megiddo, N. 1983. The weighted Euclidean 1-center problem. *Math. Oper. Res* 8(4): 498–504.

Ostresh Jr, L. M. 1977. The multifacility location problem: Applications and descent theorems. *JRS* 17(3): 409–419.

Rosing, K. E. 1992. An optimal method for solving the (generalized) multi-Weber problem. *EJOR* 58(3): 414–426.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *AIJ* 219: 40–66.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *AIJ* 195: 470–495.

Solovey, K.; and Halperin, D. 2016. On the hardness of unlabeled multi-robot motion planning. *IJRR* 35(14): 1750–1759.

Standley, T. S. 2010. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *AAAI*, 28–29.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *SoCS*, 151–159.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *TCIAIG* 4(2): 144–148.

Surynek, P. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *AAAI*, 1261–1263.

Surynek, P. 2012. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *PRICAI*, 564–576.

Thomas, S.; Deodhare, D.; and Murty, M. N. 2015. Extended Conflict-Based Search for the Convoy Movement Problem. *IEEE Intelligent Systems* 30: 60–70.

Wagner, G.; and Choset, H. 2015. Subdimensional expansion for multirobot path planning. *AIJ* 219: 1–24.

Welzl, E. 1991. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, 359–370. Springer.

Yan, D.; Zhao, Z.; and Ng, W. 2015. Efficient processing of optimal meeting point queries in Euclidean space and road networks. *KAIS* 42(2): 319–351.

Yu, J.; and LaValle, S. M. 2013a. Multi-agent path planning and network flow. In *Algorithmic foundations of robotics X*, 157–173. Springer.

Yu, J.; and LaValle, S. M. 2013b. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI*, 1444–1449.