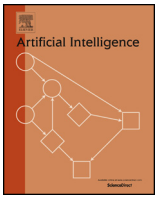
Artificial Intelligence 322 (2023) 103950

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Conflict-tolerant and conflict-free multi-agent meeting

Dor Atzmon [a,*], Ariel Felner [a], Jiaoyang Li [b], Shahaf Shperberg [a], Nathan Sturtevant [d], Sven Koenig [c]

[a] *Ben-Gurion University of the Negev, Israel*
[b] *Carnegie Mellon University, USA*
[c] *University of Southern California, USA*
[d] *University of Alberta, Canada*

ARTICLE INFO

Article history:
Received 11 November 2021
Received in revised form 22 May 2023
Accepted 23 May 2023
Available online 26 May 2023

Keywords:
Multi-agent meeting
Conflict-tolerant
Conflict-free
Multi-agent path finding
Multi-directional heuristic search
MM*
Conflict-based search
Network-flow

ABSTRACT

In the *Multi-Agent Meeting* problem (MAM), the task is to find the optimal meeting location for multiple agents, as well as a path for each agent to that location. Among all possible meeting locations, the optimal meeting location has the minimum cost according to a given cost function. Two cost functions are considered in this research: (1) the sum of all agents paths' costs to the meeting location (SOC) and (2) the cost of the longest path among them (MKSP). MAM has many real-life applications, such as choosing a gathering point for multiple traveling agents (humans, cars, or robots).

In this paper, we divide MAM into two variants. In its basic version, MAM allows multiple agents to occupy the same location, i.e., it is *conflict tolerant*. For MAM, we introduce MM*, a *Multi-Directional Heuristic Search* algorithm, that finds the optimal meeting location under different cost functions. MM* generalizes the *Meet in the Middle* (MM) bidirectional search algorithm to the case of finding an optimal meeting location for multiple agents. Several admissible heuristics are proposed for MM*, and experiments demonstrate the benefits of MM*.

As agents may be embodied in the world, a solution to MAM may contain conflicting paths, where more than one agent occupies the same location at the same time. The second variant of the MAM problem is called *Conflict-Free Multi-Agent Meeting* (CF-MAM), where the task is to find the optimal meeting location for multiple agents (as in MAM) as well as conflict-free paths (in the same manner as the prominent *Multi-Agent Path Finding* problem (MAPF)) to that location. For optimally solving CF-MAM, we introduce two novel algorithms, which combine MAM and MAPF solvers. We prove the optimality of both algorithms and compare them experimentally, showing the pros and cons of each algorithm.[1]

* Corresponding author.
E-mail address: dorat@post.bgu.ac.il (D. Atzmon).

[1] Portions of this work have been previously published [1,2]. This paper ties together all the results, provides more insights, more theoretical understandings, more experimental results, and presents a comprehensive manuscript that summarizes this line of work.

https://doi.org/10.1016/j.artint.2023.103950
0004-3702/

1. Introduction and overview

In the *Multi-Agent Meeting* problem (MAM) [3,1] the input is a graph and a set of k traveling agents, each with its start location. The task is to find a meeting location for the agents, and a path for each agent from its start location to the meeting location. Ideally, one should find an optimal location, which is determined by the cost function. This usually considers the travel effort by each agent. Two common cost functions [4], which we consider in this paper, are the *Sum-of-Costs* (SOC) and *Makespan* (MKSP) cost functions. SOC is the sum of all agents paths' costs to the meeting location, while MKSP is the cost of the longest path among them. Both functions are formally defined in Section 3. MAM is a very practical problem. It is applicable to finding a gathering location for multiple agents (humans, cars, or robots). Additionally, given a set of locations, in many scenarios, one may wish to choose a point that is as close as possible to all these locations. An example would be placing a hospital close to a number of schools. We divide MAM into two variants of the problem, namely, the basic *Conflict-Tolerant* MAM (denoted for simplicity as MAM) and *Conflict-Free* MAM (CF-MAM).

1.1. Basic, conflict-tolerant multi-agent meeting

In the basic variant (MAM) the paths can be conflicting, i.e., more than one agent is allowed to occupy the same location at the same time. This is relevant, for instance, if the locations are much larger than the physical body of the agents. For example, if a number of persons wish to meet in a nearby coffee shop. In addition, conflict-tolerant solutions are relevant for the second scenario described above (of the hospital), as usually there is only one moving agent that evacuates a patient to the hospital. To find the optimal meeting location for MAM, we introduce the *Multi-Directional Meet in the Middle* algorithm (MM*). MM* is a best-first search algorithm that progresses in k directions until a meeting location is found. MM* is a general algorithm that uses a priority function to order nodes in its open list. We provide a unique priority function for each of the *Sum-of-Costs* (SOC) and *Makespan* (MKSP) cost functions. We prove the optimality of MM* for both of these priority functions. MM* is strongly related to the *Meet in the Middle* algorithm (MM) [5], a recently introduced bidirectional heuristic search algorithm that is guaranteed to *meet in the middle*, i.e., the two frontiers of the search meet at the halfway point of the optimal solution (the exact definition is given below). The priority function of MM* for MKSP is a generalization of that of MM, although their halting conditions are different: MM returns a path from the start location to the goal location while MM* returns the actual meeting location (and paths from the start locations to that meeting location).

MM* relies on an admissible heuristic function that estimates the remaining cost to the goal location for each node encountered during the search. We propose a number of such heuristic functions for the SOC and MKSP priority functions, and prove their admissibility. We then provide experimental results that demonstrate the benefits of MM* with these heuristic functions.

1.2. Conflict-free multi-agent meeting

As agents may be embodied in the world, practically, a solution to MAM may cause conflicts (collisions) between the agents on their paths. Indeed, in a MAM solution, agents may be located in the same location at the same time. Therefore, we introduce a second variant of the problem called *Conflict-Free Multi-Agent Meeting* (CF-MAM). In CF-MAM, we seek a meeting location for multiple agents and *conflict-free paths* to that location. In CF-MAM, the agents are prohibited from occupying the same location at the same time, except, of course, for the meeting location, where all agents gather. This scenario is relevant when the physical shape of the agents is large, such that only one agent fits a location.

Several related, but distinctly unique problems have been studied previously. A related problem for which algorithms find conflict-free paths for multiple agents is the *Multi-Agent Path Finding* problem (MAPF) [4,6,7]. MAPF algorithms must find a path for each agent from its start location to its specified goal location, while avoiding conflicts with other agents. A commonly used algorithm for optimally solving MAPF is the *Conflict-Based Search* (CBS) algorithm [8]. Importantly, both variants of MAM are not a special case of MAPF and are significantly different. The main significant difference between MAM and MAPF is that, in MAPF, goal locations are given as input. By contrast, in MAM, goal locations are *not* part of the input and the task is to find a specific meeting location for the agents, as well as a set of paths to that specific meeting location. Therefore, whether conflicts are allowed or not is a parameter setting in MAM and thus different variants of the problem exist. Naturally, in MAPF, if conflicts are allowed, the problem does not exist as agents can solve a set of independent single-agent pathfinding sub-problems to solve the problem.

Indeed, CF-MAM has some connections to MAPF as we now discuss. A unique variant of MAPF is the *Permutation Invariant MAPF* problem (PI-MAPF) [9]. 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. A special case of PI-MAPF is the *Shared-Goal MAPF* problem (SG-MAPF) [10], in which all goal locations are identical, i.e., finding conflict-free paths to a single goal location. CF-MAM is different from SG-MAPF as in CF-MAM the goal location is not given as input and must be calculated.

We present two algorithms for solving CF-MAM and prove their optimality and completeness. The first algorithm, called *CF-MAM CBS* (CFM-CBS), uses the framework of CBS. CFM-CBS has two levels. The low level solves the given problem as MAM (e.g., using MM*) under a given set of constraints. The high level of CFM-CBS repeatedly calls the low level, identifies new 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

Table 1
Related problems overview.

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

location can be determined. We introduce a specifically designed reduction from SG-MAPF to Network Flow. An experimental study shows that each algorithm has circumstances where it performs best.

Table 1 summarizes all related problems. For each problem (each column), the output example in the table (second row) shows a solution for the input example (first row). In these examples, all edges have a unit cost. Optimal solutions to all these problems minimize some cost function. In the case of the presented solutions in the table, they are optimal for both SOC and MKSP. All the problems in the table receive as input a graph and a set of start locations. While MAPF problems (MAPF, PI-MAPF, and SG-MAPF; columns 2-4) also receive, as input, a destination for leading the agents to (goal locations), MAM problems (MAM and CF-MAM; columns 1 and 5) are required to find a destination for the agents (a meeting location). Note that, while PI-MAPF gets the goal locations as input, a solution to the problem determines which goal location will be reached by each agent. Thus, as presented in the table, MAPF and PI-MAPF may get the same input but have different optimal solutions. MAM is the only problem in the table that allows conflicts. It is less restricted than CF-MAM, and hence, a solution to MAM may be invalid for CF-MAM. In the examples for MAM and CF-MAM, both get similar input. In the output example depicted for MAM, both agents a_1 and a_4 that start at locations s_1 and s_4, respectively, conflict at timestep 1 (both located in the same location at the same time). As CF-MAM does not allow 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.

The paper is organized as follows. In Section 2, we provide basic definitions and details of background and related work. Then, we specifically define the MAM problem in Section 3. We present a naive approach and propose the MM* algorithm for optimally solving MAM in Sections 4 and 5. We discuss the different priority functions for MM* and their relation to MM in Section 6. We suggest a number of heuristic functions for MM* and compare them experimentally in Sections 7 and 8, respectively. In Section 9, we formally define the problem of CF-MAM and, in Sections 10 and 11, we propose the CFM-CBS and IMS algorithms, respectively, for optimally solving CF-MAM. We evaluate these algorithms experimentally in Section 12. Finally, we conclude this research and suggest directions for future work in Section 13.

Preliminary versions of this paper were published in IJCAI-2020 [1] for MAM and in ICAPS-2021 [2] for CF-MAM. This paper **(1)** unifies these two papers; **(2)** creates a similar terminology for this area of research; **(3)** extends CF-MAM to support an additional cost function (i.e., MKSP); and **(4)** presents extended experimental and theoretical results.

2. Definitions, background, and related work

This paper is related to three research areas: **(1)** Heuristic and Bidirectional Search; **(2)** Multi-Agent Meeting; and **(3)** Conflict-Free Path Planning. We cover each of them next.

2.1. Heuristic and bidirectional search

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a start location $s \in \mathcal{V}$, and a goal location $g \in \mathcal{V}$, a *Search Algorithm* finds a *path* π from the start location s to the goal location g. Such a path consists of a sequence of locations, i.e., $\pi = (s, \ldots, g)$. Let $\pi(t)$ denote the t-th location in π. Thus, $\pi(0) = s$ and $\pi(|\pi| - 1) = g$. Let $N(v)$ represent the neighbors of v, i.e., $N(v) = \{v' \in \mathcal{V} \mid (v, v') \in \mathcal{E}\}$. Each two consecutive locations $\pi(t), \pi(t+1)$ $(0 \leq t < |\pi| - 1)$ must satisfy $\pi(t+1) \in N(\pi(t))$. We call this edge traversal, performed between two neighboring locations at two consecutive timesteps, a *move action*.

Depending on the problem, graph $\mathcal{G}$ can either be weighted or have unit costs for the edges. The cost of edge $e = (v_1, v_2) \in \mathcal{E}$ is denoted by $c(e) = c(v_1, v_2) > 0$. The cost of path π is denoted by $C(\pi)$ and equals the sum of the cost of all

move actions in π ($C(\pi) = \sum_{0 \leq t < |\pi|-1} c(\pi(t), \pi(t+1))$). We use $d(v_1, v_2)$ to denote the cost of a shortest (optimal) path between the two locations v_1 and v_2. Trivially, if path π is a shortest path, $C(\pi) = d(s, g)$.

Heuristic Search algorithms investigate the given graph by building a search tree of nodes, where each node maintains some location v, a g-value, and an h-value. The g-value ($g(v)$) represents the cost of the path from s to v (along the search tree) and the h-value ($h(v)$) estimates the cost of the shortest path from v to g. h is calculated by a heuristic function. A heuristic function is called *admissible* if it never overestimates, i.e., $\forall v, h(v) \leq d(v, g)$. A heuristic function is called *consistent* if its estimate is less than or equal to the estimate from any neighbor plus the cost of reaching that neighbor, i.e., $\forall v \forall v' \in N(v), h(v) \leq c(v, v') + h(v')$, and its estimate of the goal location g equals 0, i.e., $h(g) = 0$.

Best-First Search (BFS) algorithms organize nodes in two lists: an *open list* (OPEN) and a *closed list* (CLOSED). Iteratively, such algorithms select for expansion the best node v from OPEN (according to some priority function), generate its neighbors $N(v)$ while adding them to OPEN, and move v to CLOSED . If a shorter path to a node on CLOSED is found, it is re-opened by being placed back on OPEN, however, this cannot happen if the heuristic function is consistent. A^* [11] is a BFS algorithm that uses the priority function $f = g + h$ for choosing the next node for expansion. If h is admissible, then A* is guaranteed to return an optimal solution and, under certain circumstances, is optimally efficient [12].

Algorithms, such as A*, initialize OPEN with location s and halt when location g is expanded. Therefore, their search is *unidirectional* (one sided). Other algorithms, called *bidirectional*, start the search both from location s and from location g, and halt when some path is found and the termination conditions are met.

MM [5] is a bidirectional heuristic search algorithm with a unique priority function; nodes are ordered in OPEN according to $\max\{2g, g + h\}$. MM is guaranteed to *meet in the middle*. The practical meaning of this property is that the two search frontiers never venture (nodes are never expanded) further than $C^*/2$ from their start locations, where C^* is the cost of the shortest path. Nevertheless, it is important to note that a meeting location is not returned by MM. In fact, a meeting location is not even defined in MM. As any (bidirectional) search algorithm, MM returns a shortest path from a start location to a goal location. Below, we introduce our new algorithm, MM*, which generalizes MM from bidirectional search to multi-directional search. Unlike MM, MM* returns the, so called, meeting location and paths from the start locations to the meeting location, as fully defined and explained below.

2.2. Multi-agent meeting

In this paper, we focus on MAM, which is the problem of finding a meeting location and a set of paths to the meeting location. Previous work has solved this problem with a variety of approaches. MAM has been investigated on general graphs by applying variants of *Dijkstra's algorithm* [13] in parallel, one for each agent. Once a path from all start locations to all locations in the graph is known, the best meeting location can be chosen by exhaustively iterating over all relevant meeting locations. Yan et al. [3] suggested an algorithm that progresses in parallel from all start locations, in a BFS manner. When a new location is reached from one of the search frontiers, the Dijkstra's algorithm is executed from that frontier up to the new location. Thus, the algorithm prunes areas of the state space by executing Dijkstra's algorithm only up to a number of potential meeting locations in order to determine the optimal meeting location. More improvements for pruning additional areas of the state space exist, however, they require prepossessing time in which the given graph needs to be explored in advance. Xu et al. [14] repeatedly solved the problem while assuming that the start locations of the agents change between each problem instance. They demonstrated how the history of the movement of agents can be used for future calculations. Geisberger et al. [15] showed that some nodes can be removed from the graph (e.g., nodes with a degree of one) in such a way that shortest paths, in the remaining graph, are preserved. Moreover, the authors showed that nodes can be ordered based on their likelihood to be part of shortest paths, in such a way that finding paths in the graph is enhanced, in terms of runtime.

Izmirlioglu et al. [16] solved a related problem, for which the agents must pass through specific locations on their way to the meeting location. By adding this constraint, forcing agents to pass specific locations, the problem becomes intractable. The authors solve the problem using *Answer Set Programming* (ASP) [17], a logic-based representation and automated reasoning framework.

In the field of computational geometry, MAM is known as the *Weber problem* [18]. Many efficient algorithms exist for MAM in continuous Euclidean spaces [19–22] that calculate a geometric point that satisfies the relevant constraints. Other researches showed how to find the optimal weighted center (*1-Center*), which minimize a weighted euclidean distance, on a plane [23,24], and how to find the smallest circle (*Smallest Enclosing Discs*) that contains all the given agents' locations on a plane [25].

FastMap [26,27] is a near-linear preprocessing algorithm that approximates the cost of a shortest path between any two locations. In Section 7, we provide additional details on FastMap and use it for MAM.

2.3. Conflict-free path planning

Some path finding problems that involve multiple agents have different restrictions on the paths of the agents. We say that two paths *conflict* if they do not comply with some of the restrictions of the problem. We define below two types of such conflicts. A related problem to MAM that finds conflict-free paths for multiple agents is the *Multi-Agent Path Finding* problem (MAPF) [4,6,7]. MAPF gets as input an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a set of start locations $S = \{s_1, \ldots, s_k\} \subset \mathcal{V}$,

and a set of goal locations $G = \{g_1, \ldots, g_k\} \subset \mathcal{V}$ for a set of agents $A = \{a_1, \ldots, a_k\}$. As output, algorithms for MAPF return a set of paths $\Pi = \{\pi_1, \ldots, \pi_k\}$ for the agents, respectively. In conflict-tolerant problems, agents are allowed to conflict. Thus we assume that, in such problems, there are only move actions and, in fact, agents can simply solve a set of single-agent pathfinding sub-problems independently, and the problem is trivially solved. On the other hand, conflict-free problems restrict the returned set of paths Π to be conflict-free, and thus wait actions must also be considered, i.e., an agent can stay at its current location between two consecutive timesteps. Both move actions and wait actions have unit cost.

Two main types of conflicts are defined for MAPF [4]: *vertex conflicts* and *swapping conflicts*. 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., $(\pi_i(t), \pi_i(t+1)) = (\pi_j(t+1), \pi_j(t)) = e$.

The cost of a set of paths is determined by a cost function. There are two commonly used cost functions: **(1)** *Sum-Of-Costs* (SOC) is the sum of the costs of all paths in Π ($C_{SOC}(\Pi) = \sum_{\pi_i \in \Pi} C(\pi_i)$). **(2)** *Makespan* (MKSP), which equals the cost of the path with the maximum cost among all paths in Π ($C_{MKSP}(\Pi) = \max_{\pi_i \in \Pi} C(\pi_i)$). We use C^* to denote the cost of the optimal (minimum cost) solution (either SOC or MKSP). In this paper, we focus on these two cost functions in the content of MAM.

Although solving a MAPF instance optimally is NP-hard [28,29], a number of algorithms have been developed that are capable of solving instances optimally for many agents, e.g., M^* [30], *BIBOX* [31], *ICTS* [32], and *Conflict-Based Search* (CBS) [8]. 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 have been introduced for CBS [33–36]. In general, MAPF has been investigated extensively and has many variants and extensions, including large agents [37], trains [38], convoys [39], heterogeneous agents [40], deadlines [41], and robustness [42,43]. Later in the paper we introduce a new algorithm for solving CF-MAM that uses the framework of CBS.

A unique variant of MAPF is the *Permutation Invariant MAPF* problem (PI-MAPF) [9]. 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, i.e., each goal location must be reached by one of the agents. This problem is also known as *Anonymous MAPF* [44] or *Unlabeled MAPF* [45]. While optimally solving MAPF instances is NP-hard, PI-MAPF instances can be optimally solved in polynomial time by reducing the problem to Network-Flow [10] and using a Network-Flow solver.

A special case of PI-MAPF is the *Shared-Goal MAPF* problem (SG-MAPF; also known as *Evacuation*) [10], in which all goal locations are identical, i.e., finding conflict-free paths to a single goal location. There are a number of possible assumptions for how agents behave at their goal locations [4] for the classical MAPF problem. As all agents share a single goal, SG-MAPF must either assume that when an agent reaches the goal it immediately disappears or that agents are allowed to conflict at the goal location. SG-MAPF can also be optimally solved by the same reduction used for PI-MAPF [10]. As mentioned, SG-MAPF is different from CF-MAM as, in CF-MAM, the goal is not given in advance. We modify their reduction specifically from the SG-MAPF to Network Flow and use it to solve CF-MAM with a number of calls for a Network-Flow solver, as described in Section 11.

3. Multi-agent meeting (MAM)

Next, we formally define the basic version of the *Multi-Agent Meeting* problem (MAM), which is tolerant to conflicts. MAM receives as input the tuple $\langle \mathcal{G}, S \rangle$, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a weighted undirected graph and S is a set of k start locations for k agents $A = \{a_1, \ldots, a_k\}$. A solution is a location $m \in \mathcal{V}$, indicating a meeting location for the agents, plus a set of shortest paths Π from each start location s_i to location m. In its basic form, MAM is conflict-tolerant and more than one agent can occupy any component of the graph at any time. Under these assumptions, the cost functions for SOC and MKSP for a meeting location m can be calculated as follows. For SOC:

$$C_{SOC}(m) = \sum_{a_i \in A} d(s_i, m). \tag{1}$$

This corresponds to the sum of the costs of the paths to the meeting location. For MKSP:

$$C_{MKSP}(m) = \max_{a_i \in A} d(s_i, m). \tag{2}$$

This is the cost of the longest path to the meeting location. An *optimal* solution has the lowest cost C^* among all possible solutions. The meeting location of the optimal solution is denoted by m^*.

Fig. 1 illustrates a MAM problem instance with three agents a_1, a_2, and a_3 with start locations s_1, s_2, and s_3, respectively. Edges are labeled with their costs. Consider location v_1 as a meeting location. Since $d(s_1, v_1) = 5$, $d(s_2, v_1) = 5$, and $d(s_3, v_1) = 5$, $C_{SOC}(v_1) = 15$ while $C_{MKSP}(v_1) = 5$. Now, consider v_2. Since $d(s_1, v_2) = 8$, $d(s_2, v_2) = 2$, and $d(s_3, v_2) = 2$, $C_{SOC}(v_2) = 12$ and $C_{MKSP}(v_1) = 8$. v_1 is an optimal meeting location for minimizing MKSP and v_2 is an optimal meeting location for minimizing SOC.

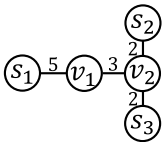


Fig. 1. MAM cost functions example.

4. Naïve approach

A common straightforward approach for finding optimal paths for multiple agents defines a search space in which each state is a vector $(v_1, \ldots, v_k)$ of the location of all agents, where v_i represents the location of agent a_i. For a given state $(v_1, \ldots, v_k)$, a successor is each item in the Cartesian product $N(v_1) \times \cdots \times N(v_k)$, where $N(v)$ represents the neighbors of location v. In MAM, a goal state is a state $(v_1, \ldots, v_k)$ in which all locations are equal $(v_1 = \cdots = v_k)$. Searching an optimal solution for MAM in this search space can be done, for example, using Dijkstra's algorithm or A*. As each state in this approach contains the locations of all agents, the size of the search space is exponential in the number of agents. This can also be seen in our experiments below. Next, we present our efficient MM* algorithm for MAM.

5. Multi-directional MM (MM*) for MAM

In this section, we introduce our novel algorithm MM*. MM* is a multi-directional best-first search algorithm that is guaranteed to return an optimal solution for MAM and the cost for both the SOC and MKSP cost functions. As will be shown below, MM* has a general structure, but each cost function (SOC and MKSP) has a different unique priority function.

A node in MM* 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).[2] 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 (possibly overwriting if the cost is improved) a node (a_i, v') for each $v' \in N(v)$, setting $g(a_i, v') \leftarrow g(a_i, v) + c(v, v')$, and inserting it into OPEN.
2. Moving (a_i, v) to CLOSED.

In heuristic search, given a node n in the search tree, $f^*(n)$ is defined to be the cost of the optimal solution that passes through n, and $f(n)$ is defined to be a lower bound on $f^*(n)$. This terminology is migrated to MM*. Let $f^*(a_i, v)$ be the cost of the optimal MAM solution (for either SOC or MKSP) such that a_i passes through v (via a path of cost $g(a_i, v)$ along the search tree[3]) on its way to the meeting location. $f(a_i, v)$ is a lower bound on $f^*(a_i, v)$ $(f(a_i, v) \leq f^*(a_i, v))$. Note that $f(a_i, v)$ (and $f^*(a_i, v)$), besides agent a_i, also depends on the other agents meeting agent a_i after it visits location v. In Section 6, we define $f(a_i, v)$ for either SOC or MKSP by exploiting admissible heuristic functions that estimate the remaining cost of all agents (including that of a_i) that can be added to $g(a_i, v)$ (the cost of the path from s_i to v along the search tree). Each of these f-values can be plugged into MM*.

There is no notion of a goal node in MM* but instead we have a goal condition on each location v. We say that location v becomes a *possible goal* when it has been generated from all directions, i.e., $\forall a_i \in A$, $(a_i, v) \in \text{OPEN} \cup \text{CLOSED}$. To manage this in practice, for each location v, we keep a k bit-vector, where bit i is set when node (a_i, v) is generated. When location v becomes a possible goal, the cost of meeting at location v, denoted by $C(v)$, can be calculated depending on the cost function as follows[4]:

$$C_{SOC}(v) = \sum_{a_i \in A} g(a_i, v) \text{ and}$$

$$C_{MKSP}(v) = \max_{a_i \in A} g(a_i, v).$$

Let U be the cost of the *incumbent solution*, i.e., U is the minimum $C(v)$ among all possible goals that have been identified thus far (initially $U = \infty$). U is an upper bound on C^*. The halting condition for MM* is to halt if $fmin \geq U$, where $fmin$ is the minimum f-value in OPEN.[5] This guarantees that U cannot be further improved.

[2] Bidirectional searches usually maintain two open-lists, one for each search direction, but the priority function can choose a node from either one of them. This is logically equivalent to a single open-list that contains nodes from both directions. MM* uses a single open-list, which is equivalent to k open-lists, one for each agent, that share the same priority function. That is, the priority function can choose a node from any of these open-lists.

[3] Note that in our definition of $f^*(a_i, v)$, the current g-value is assumed even if shorter paths from s_i to v might be found later.

[4] In Equations 1 and 2, we used $d(s_i, v)$. Here, we use $g(a_i, v)$ because the path is the path from location s_i to location v along the search tree.

[5] f-values, for SOC and MKSP, are fully defined below in Section 6.

Algorithm 1: The MM* Algorithm.

```
1  Main(MAM problem instance I = {A, S})
2      Init OPEN, CLOSED; U ← ∞
3      foreach (a_i, s_i) ∈ {A, S} do
4          Insert (a_i, s_i) into OPEN
5      while OPEN is not empty do
6          Extract (a_i, v) from OPEN // with lowest f(a_i, v)
7          if f(a_i, v) ≥ U then
8              return U // including the meeting location and the corresponding paths
9          foreach v' ∈ N(v) do
10             if CLOSED contains (a_i, v') then
11                 if g(a_i, v') ≤ g(a_i, v) + c(v, v') then
12                     continue
13                 Remove (a_i, v') from CLOSED
14             else if OPEN contains (a_i, v') then
15                 if g(a_i, v') ≤ g(a_i, v) + c(v, v') then
16                     continue
17             Insert (a_i, v') into OPEN // possibly overwriting if cost improved
18             U ← min{U, C(v')} // if C(v') < U, v' becomes the incumbent meeting location
19         Insert (a_i, v) into CLOSED
20     return U // including the meeting location and the corresponding paths
```

Algorithm 1 gives the pseudo-code of MM*. First, MM* initializes OPEN and CLOSED, and sets $U \leftarrow \infty$ (Line 2). Then, the initial nodes (a_i, s_i) are inserted into OPEN (Lines 3-4). MM* performs a best-first search as follows. While OPEN is not empty (Line 5), it extracts (a_i, v), the best node (with the lowest f-value) from OPEN (Line 6). Then, it checks the halting condition on location v, i.e., whether $fmin = f(a_i, v) \geq U$ (Lines 7-8). Otherwise, it performs the expansion cycle on (a_i, v) (Lines 9-18). MM* performs *duplicate detection and pruning* on CLOSED (Lines 10-13) and OPEN (Lines 14-16). As a result, MM* always keeps the lowest seen g-value for each generated node (a_i, v'). In general, MM* allows nodes in CLOSED to be re-opened (Line 13). But, this will never happen for consistent heuristics (including all heuristics that we propose and experiment with below). If (a_i, v') is not a duplicate node, then (a_i, v') is inserted into OPEN (Line 17). If location v' is a possible goal and its solution is better than U ($C(v') < U$) then U is decremented accordingly (Line 18). If location v' is not a possible goal (v' has not been generated from all directions), its cost is $C(v') = \infty$, and U is not updated. After its expansion, (a_i, v) is inserted into CLOSED (Line 19). When U is returned (Lines 8 and 20), it also includes the meeting location m as well as the paths to m, that can be constructed by parent-pointers (not included in the pseudo-code).

5.1. Theoretical analysis

Theorem 1 *(Completeness). MM* is guaranteed to return a solution if one exists, and it returns $U = \infty$ otherwise.*

Proof. For each agent a_i, MM* performs a best-first search from location s_i. In the worst case, MM* explores every reachable location for each agent. If a solution exists, a location reachable for all agents will be generated from all directions and U will be updated (Line 18). At some point, either $fmin$ will reach U or the entire graph will have been explored for all agents (OPEN will be empty), and a solution will be returned. If no solution exists (because some of the agents are in different connected components), there is no location that is reachable for all agents and U will never be updated and thus remains ∞. □

Observation 1. If MM* has not yet generated node (a_i, m^*) for agent a_i from the optimal path of agent a_i to m^*, there exists a node (a_i, v_i) in OPEN such that v_i is a location on the optimal path of a_i to location m^*, and every node before v_i on the optimal path has already been expanded.

Proof. Let π_i^* be the optimal path of agent a_i from location s_i to the optimal meeting location m^*. We prove the observation by induction on the nodes of the optimal path of each agent along the search.

Base case: At the beginning, OPEN is initialized with (a_i, s_i) for each agent a_i, which is the first location on the optimal path of agent a_i to location m^*, i.e., $(a_i, \pi_i^*(0))$ is in OPEN.

Inductive step: Let us assume that exactly x ($|\pi_i^*| > x \geq 0$) nodes have been expanded on the optimal path of agent a_i to location m^* and one node (a_i, v_i) such that $v_i = \pi_i^*(x)$ is generated and not yet expanded. When node (a_i, v_i) is expanded, it generates a new node (a_i, v_i') in OPEN for each neighbor $v_i' \in N(v_i)$. By definition, a path consists of adjacent locations. Therefore, one such neighbor $v_i'' \in N(v_i)$ must be on the optimal path to m^* ($v_i'' = \pi_i^*(x+1)$) and $(a_i, \pi_i^*(x+1))$ is inserted into OPEN. Note that the duplicate detection and pruning mechanism of MM* will never prune the node (a_i, v_i'') as v_i'' is on the optimal path and there cannot exist a different path to v_i'' with a lower cost.

Conclusion: Since the base case and the inductive step are both true, by induction we conclude that, for agent a_i that has not yet reached m^* from the optimal path, there must be a node on the optimal path of agent a_i in OPEN. □

Theorem 2 *(Optimality). Given an admissible f (i.e., $f(n) \leq f^*(n)$ for all nodes n), MM* is guaranteed to return the optimal location m^* with cost C^*, if one exists.*

Proof. Assume, by contradiction, that MM* returned a suboptimal location $m \neq m^*$ with cost $C > C^*$. Since MM* has terminated and returned a solution, $fmin \geq C > C^*$. Since MM* terminated without returning an optimal solution, Observation 1 holds and there exists a node $n' = (a_i, v_i)$ in OPEN such that v_i is a location on the optimal path of a_i to location m^*, and every node before n' on the path has already been expanded. Since n' is the first node on the optimal path that was not expanded, it was generated by a node on the optimal 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'. Therefore, since $g(n') = d(s_i, v_i)$, $f^*(n') = C^*$. Since f is admissible then $f(n') \leq f^*(n') = C^*$. As $n' \in$ OPEN, $fmin \leq f(n') \leq f^*(n') = C^*$, which contradicts the fact that $fmin \geq C > C^*$. □

6. MM* priority functions

We next define the priority function f for MM* for both SOC and MKSP to be used on the basic MAM problem. Recall that in A*, given a node n, a perfect heuristic function is $h^*(n) = d(n, g)$ and a perfect priority function is $f^*(n) = g(n) + h^*(n)$. Similarly, if $h(n)$ is a lower bound on $h^*(n) = d(n, g)$, then $f(n) = g(n) + h(n)$ is a lower bound on $f^*(n)$. Next, we generalize this to MM* and define all these functions (h^*, h, f^*, and f) for both the SOC and MKSP cost functions.

6.1. The functions for SOC

Consider node (a_i, v) in OPEN. $f^*_{SOC}(a_i, v)$ is the cost of the optimal solution such that:

Item 1. Agent a_i passes through location v (via the path of cost $g(a_i, v)$ along the search tree).
Item 2. Agent a_i continues from location v along a shortest path to meet the other agents at some location m (it might be that $m = v$).
Item 3. Each of the other agents a_j travels from its start location s_j along a shortest path to location m.

We note that the meeting location m in $f^*_{SOC}(a_i, v)$ may not be the optimal meeting location for our problem; it is the optimal meeting location assuming agent a_i reaches location v via a path of cost $g(a_i, v)$.

Now, $f^*_{SOC}(a_i, v) = g(a_i, v) + h^*_{SOC}(a_i, v)$, where $h^*_{SOC}(a_i, v)$ is defined to be the sum of the cost of agent a_i to get from location v to location m along a shortest path (item 2), plus the cost of the other agents to get from their start locations to location m along shortest paths (item 3). Formally:

$$h^*_{SOC}(a_i, v) = \min_{m \in V}\{d(v, m) + \sum_{a_j \in A \setminus \{a_i\}} d(s_j, m)\}. \tag{3}$$

So, f^* is the sum of all three items while h^* is the sum of items 2 and 3. We next move to discuss f and h.

We denote the best meeting location m w.r.t. node (a_i, v) by $m^*(a_i, v)$. Let $h_{SOC}(a_i, v)$ be an admissible estimate (lower bound) of $h^*_{SOC}(a_i, v)$, i.e., $h_{SOC}(a_i, v) \leq h^*_{SOC}(a_i, v)$. We propose a number of admissible h-functions for SOC in Section 7.

For SOC, naturally,

$$f_{SOC}(a_i, v) = g(a_i, v) + h_{SOC}(a_i, v). \tag{4}$$

6.2. The functions for MKSP

The MKSP case is more complicated. Since, in MKSP, we take the maximum among agents (not the sum), we do not know which agent has the path with the highest cost. We begin by defining $f^*_{MKSP}(a_i, v)$, which is the cost of the optimal solution given that a_i passes through v, via a path of cost $g(a_i, v)$:

$$f^*_{MKSP}(a_i, v) = \min_{m \in V}\left[\max\left\{\begin{array}{l} g(a_i, v) + d(v, m), \\ \max_{a_j \in A \setminus \{a_i\}} d(s_j, m) \end{array}\right\}\right]. \tag{5}$$

For a given possible meeting location m, we want the path of one of the agents with the highest cost. If this is our current agent a_i, this is given by $g(a_i, v) + d(v, m)$ (top line of the max term in Equation 5). If it is some other agent a_j, it is given by $d(s_j, m)$ (bottom line).

Next, we need to define f_{MKSP} as a lower bound on f^*_{MKSP}. Here, we do not define h^*_{MKSP} and h_{MKSP} but define $f_{MKSP}(a_i, v)$ in terms of $h_{SOC}(a_i, v)$ as follows:

$$f_{MKSP}(a_i, v) = \max\left\{g(a_i, v), \frac{g(a_i, v) + h_{SOC}(a_i, v)}{k}\right\}, \tag{6}$$

where k is the number of agents. $g(a_i, v)$ is a lower bound on $f^*_{MKSP}(a_i, v)$ because a_i has already traveled along a path of cost $g(a_i, v)$. Thus, $f^*_{MKSP}(a_i, v) \geq g(a_i, v)$. Now, let m^*_x be the optimal meeting location for objective function x (x is either

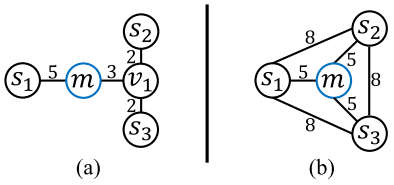


Fig. 2. f_{MKSP} examples.

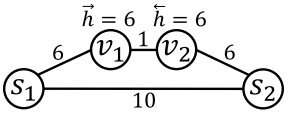


Fig. 3. MM and MM* difference example.

SOC or MKSP). Observe that $C_{SOC}(m^*_{SOC}) \leq C_{SOC}(m^*_{MKSP})$ and $C_{MKSP}(m^*_{MKSP}) \leq C_{MKSP}(m^*_{SOC})$ (because the cost of an optimal solution is always $\leq$ than the cost of any other given solution). Moreover, for any meeting location m, it holds that $\frac{C_{SOC}(m)}{k} \leq C_{MKSP}(m)$ because one of the agents must travel at least $\frac{C_{SOC}(m)}{k}$ (average is $\leq$ than max). Thus, we get that $\frac{C_{SOC}(m^*_{SOC})}{k} \leq \frac{C_{SOC}(m^*_{MKSP})}{k} \leq C_{MKSP}(m^*_{MKSP})$. Along the same reasoning, for agent a_i that passes through v, $\frac{f^*_{SOC}(a_i,v)}{k} \leq f^*_{MKSP}(a_i, v)$. Since $f_{SOC}(a_i, v)$ is a lower bound on $f^*_{SOC}(a_i, v)$, dividing it by k will yield a lower bound on $f^*_{MKSP}(a_i, v)$, i.e., $\frac{g(a_i,v)+h_{SOC}(a_i,v)}{k} \leq f^*_{MKSP}(a_i, v)$.

6.2.1. Costs of subsets

f^*_{MKSP} for k agents is determined by the path of one of the agents with the highest cost. Therefore, f^*_{MKSP} and f_{MKSP} for any subset of these k agents are also lower bounds on $f^*_{MKSP}(a_i, v)$ for *all* k agents. Thus, for any subset of $k' < k$ agents, we can compute f_{MKSP} and use it as a lower bound on f^*_{MKSP} for the entire set of k agents. Therefore, while the right-hand side of the max function in $f_{MKSP}(a_i, v)$ (Equation 6) contains all k agents, it can also contain any subset of agents. This can be done by calculating $h_{SOC}(a_i, v)$ for the selected subset of $k' < k$ agents and dividing it by k' instead of k.

Fig. 2 shows examples of MAM problem instances with three agents. In both cases, the optimal MKSP is 5 and the agents meet at location m. For Fig. 2(a), the optimal SOC is 12 where the agents meet at location v_1. Assume a perfect heuristic for SOC. Thus, for each agent a_i, $f_{SOC}(a_i, s_i) = 12$, and, by computing MKSP for all agents, we get $f_{MKSP}(a_i, s_i) = 12/3 = 4$. Now, consider the subset of agents $\{a_1, a_2\}$. Their SOC is 10, and hence $f_{MKSP}(a_i, s_i) = 10/2 = 5$. For Fig. 2(b), the optimal SOC is 15 at location m, and thus (assuming a perfect heuristic h_{SOC}) $f_{SOC}(a_i, s_i) = 15$. By computing MKSP for all agents, we get $f_{MKSP}(a_i, s_i) = 15/3 = 5$ but the SOC of the subset of agents $\{a_1, a_2\}$ is 8, and hence $f_{MKSP}(a_i, s_i) = 8/2 = 4$. This shows that there is no best subset for all cases. In our experiments, we used all combinations of pairs of agents, in addition to the set of all agents. It is future work to investigate additional subset selection policies.

6.3. MM versus MM* – similarities and differences

MM* is named after the MM algorithm. We discuss their relationship here. It is very interesting to see that f_{MKSP} is a generalization of the priority function of the MM algorithm: $pr(n) = \max(2g(n), g(n)+h(n))$ [5]. If we divide this expression by two, we get $pr(n) = \max(g(n), \frac{g(n)+h(n)}{2})$. This is a special case of the proposed f_{MKSP} for $k = 2$ ($h(n)$ is equivalent to an estimate of the cost from the start location of the backward agent – the goal in MM – to the current location of the forward agent). MM prioritizes nodes based on MKSP to keep MM *restrained* [46], i.e., to never expand nodes with $g(n) > C^*/2$.

While the priority function is similar, the halting condition is different. In Fig. 3, we illustrate the difference between MM and MM* (for MKSP with 2 agents) with regard to halting. Both algorithms start by inserting s_1 and s_2 into OPEN. Next, both algorithms expand s_1 and generate v_1 and s_2. MM sets $pr(v_1) = 12$ and $pr(s_2) = 10$ (from the forward side; $h_F(v_1) = 6$ and $h_F(s_2) = 0$), and $U = 10$ because a path of cost 10 has been found via the direct edge between locations s_1 and s_2. At this point, MM halts, as $U \leq fmin = 10$, so a path of a smaller cost cannot be found. By contrast, MM* sets $f_{MKSP}(a_1, v_1) = \max\{g(a_1, v_1), \frac{g(a_1,v_1)+h(a_1,v_1)}{2}\} = 6$ and $f_{MKSP}(a_1, s_2) = \max\{g(a_1, s_2), \frac{g(a_1,s_2)+h(a_1,s_2)}{2}\} = 10$, and $U = 10$ because a meeting location with cost 10 has been found. Unlike MM, MM* continues to search because a better meeting location might be found as $fmin = f_{MKSP}(a_1, v_1) = 6$. While U is similar for both algorithms, the priorities of MM* are half of the ones of MM. So, MM* continues and returns either v_1 or v_2 as meeting location for MKSP. Note that for SOC, MM* returns s_2 as meeting location with cost 10, as it is on a path of the minimum cost.

7. Heuristics for MM*

We now introduce a number of heuristics for SOC and prove their admissibility. They compute $h_{SOC}(a_i, v)$ in $f_{SOC}(a_i, v) = g(a_i, v) + h_{SOC}(a_i, v)$ and are used indirectly for f_{MKSP}, as shown in Equation 6. Recall that $m^*(a_i, v)$ is the optimal meeting

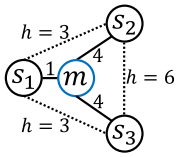


Fig. 4. Clique heuristic example.

location for SOC where a_i passes through v.[6] For simplicity, we use $h(a_i, v)$ to denote $h_{SOC}(a_i, v)$. Recall that S is the set of all start locations. 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, $h^*_{SOC}(a_i, v) = \sum_{v' \in S_i(v)} d(v', m^*(a_i, v))$ (from Equation 3); we want to compute a lower bound on h^*_{SOC}.[7]

7.1. h_1: clique heuristic

We assume that, for every pair of locations (v_x, v_y), there exists a classic admissible heuristic h (e.g., straight-line distance or Manhattan distance), such that $h(v_x, v_y) \leq d(v_x, v_y)$.

Based on the triangle inequality, for every pair of locations $v_x, v_y \in S_i(v)$ (with $v_x \neq v_y$), we have that:

$$d(v_x, v_y) \leq d(v_x, m^*(a_i, v)) + d(v_y, m^*(a_i, v)). \tag{7}$$

By summing over all such pairs in $S_i(v)$, we get:

$$\sum_{\substack{\{v_x, v_y\} \in S_i(v)^2 \\ x<y}} d(v_x, v_y) \leq \sum_{\substack{\{v_x, v_y\} \in S_i(v)^2 \\ x<y}} \left[d(v_x, m^*(a_i, v)) + d(v_y, m^*(a_i, v)) \right]. \tag{8}$$

As each $v' \in S_i(v)$ is paired with $k-1$ other locations in $S_i(v)$, we can rewrite the right-hand side of Equation 8 as $(k-1) \cdot \sum_{v' \in S_i(v)} d(v', m^*(a_i, v))$. Therefore:

$$\sum_{\substack{\{v_x, v_y\} \in S_i(v)^2 \\ x<y}} \frac{d(v_x, v_y)}{k-1} \leq \sum_{v' \in S_i(v)} d(v', m^*(a_i, v)) = h^*(a_i, v). \tag{9}$$

Now, since $h(v_x, v_y) \leq d(v_x, v_y)$, we get that:

$$h_1(a_i, v) = \sum_{\substack{\{v_x, v_y\} \in S_i(v)^2 \\ x<y}} \frac{h(v_x, v_y)}{k-1} \leq h^*(a_i, v). \tag{10}$$

This heuristic h_1 is called the *Clique heuristic*, as it combines the heuristic values of *every* (unordered) pair of locations in $S_i(v)$. Fig. 4 presents an example of the clique heuristic for three agents. For node (a_1, s_1), $S_1(s_1) = \{s_1, s_2, s_3\}$. Therefore, $h(a_1, s_1) = \frac{h(s_1,s_2)+h(s_1,s_3)+h(s_2,s_3)}{2} = \frac{3+3+6}{2} = 6$.

For each start location $s_i \in S$, $S_i(s_i) = S$ and hence h_1 can be calculated once for all start locations. For each location v that is not a start location, all locations in $S_i(v)$ except for v remain the same. So, h_1 can be calculated incrementally in time that is linear in the number of agents. Assume we generate location m from location s_1 for agent a_1 in the above example. As $h(a_1, m) = \frac{h(m,s_2)+h(m,s_3)+h(s_2,s_3)}{2}$, it can also be calculated as follows: $h(a_1, m) = h(a_1, s_1) - \frac{h(s_1,s_2)+h(s_1,s_3)}{2} + \frac{h(m,s_2)+h(m,s_3)}{2}$. It is easy to see that if $h(v_x, v_y)$ is consistent, then h_1 is also consistent.

7.2. h_2: median heuristic

For a set of numbers $B \subset \mathbb{R}$, the median of B provably minimizes the sum of the absolute deviations, i.e.,

$$median(B) = \operatorname*{argmin}_{r \in \mathbb{R}} \sum_{b \in B} |b - r|. \tag{11}$$

[6] We do not use m^* in order to avoid overloading this term, which is already used to denote the optimal meeting location in the general context, unlike the current context of (a_i, v).

[7] This is a form of a front-to-end heuristic as we assume that all other agents (except for a_i which is located at v) are located at their start states. A front-to-front heuristic needs to estimate the remaining costs when all other agents are in their current locations, but these locations thus need to be specified for a given node (a_i, v). This is left for future work.

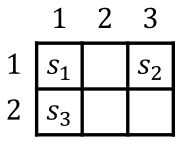


Fig. 5. Median heuristic example.

Inspired by this property, we design the *Median heuristic* (h_2) for 4-neighbor 2D grids. On such a grid, each location has two coordinates – x and y. For a set of locations, we can find the median over the x-coordinates (dimension 1) of all locations and the median over the y-coordinates (dimension 2) of all locations. Let tm_d be the median of dimension d. This creates a potential meeting location $tm = (tm_1, tm_2)$, that minimizes the sum of the absolute deviations over both dimensions. Namely, if there are no obstacles on the grid, tm will be the optimal meeting location for minimizing SOC. This is due to the fact that the distance between any two locations is their L_1-*distance* (also known as Manhattan distance on 2D grids).

Assume that the input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a 4-neighbor 2D grid where every location $v \in \mathcal{V}$ is represented by its coordinates $\vec{v} = (v_1, v_2)$. The L_1-distance for any two locations $u, v \in \mathcal{V}$ is defined as $\vec{u} - \vec{v} = |u_1 - v_1| + |u_2 - v_2|$. Due to the existence of obstacles, for any pair of locations $u, v \in \mathcal{V}$, $\vec{u} - \vec{v} \leq d(u, v)$. So, for a given node (a_i, v),

$$\sum_{v' \in S_i(v)} \vec{v'} - m^*(\vec{a_i}, v) \leq \sum_{v' \in S_i(v)} d(v', m^*(a_i, v)) = h^*(a_i, v). \tag{12}$$

Therefore, by modeling the problem in an empty 2D L_1-space (i.e., without obstacles), we introduce a new admissible (and consistent) heuristic, called the *Median heuristic*:

$$h_2(a_i, v) = \min_{\vec{m} \in \mathbb{R}^2} \{ \sum_{v' \in S_i(v)} \vec{v'} - \vec{m} \} \tag{13}$$

Following the property of the median, defined in Equation 11,

$$h_2(a_i, v) = \sum_{v' \in S_i(v)} \vec{v'} - \vec{tm} \tag{14}$$

$$= \sum_{v' \in S_i(v)} \left[|v'_1 - tm_1| + |v'_2 - tm_2| \right]. \tag{15}$$

This is admissible because the right-hand side of Equation 13 is no larger than the left-hand side of Equation 12.

We use the Quick-select algorithm for finding medians [47], which runs in $\Theta(k)$ time, to compute $h_2(a_i, s_i)$ for all root nodes. Then, for every non-root node (a_i, v) (i.e., $v \neq s_i$), $k - 1$ locations have not changed (as in the clique heuristic h_1) and we only need to update the median based on the single location that has changed. This can be done in $O(1)$ time.

Fig. 5 shows an example of a MAM problem instance with three agents (s_1, s_2, and s_3) that are located in $(1, 1)$, $(3, 1)$, and $(1, 2)$. The x-coordinates are {1,3,1} and the y-coordinates are {1,1,2}. Thus, the median location is $(1, 1)$. By computing the Manhattan distances from $(1, 1)$ to each start location, we get $h_2(a_i, s_i) = 3$ for each agent a_i.

h_2 can be generalized easily for 6-neighbor 3D grids and similar graphs of higher dimensions.

7.3. h_3: FastMap heuristic

The Median heuristic (h_2) only works for graphs that are embedded in 4-neighbor 2D grids. In addition, h_2 ignores obstacles, which may introduce inaccuracies to the heuristic. The *FastMap heuristic* (h_3) handles this and can work for any general graph. *FastMap* [26,27] is a near-linear preprocessing algorithm that embeds the locations of a given edge-weighted undirected connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ into a D-dimensional L_1-space $\mathbb{R}^D$. The dimension D of the L_1-space is user-specified. Each location $v_i \in \mathcal{V}$ is mapped to a D-dimensional point $\vec{p}_i \in \mathbb{R}^D$. The length of a shortest path $d(v_i, v_j)$ between any two locations $v_i, v_j \in \mathcal{V}$ is approximated by the L_1-distance $\vec{p}_i - \vec{p}_j$ between the corresponding two points $\vec{p}_i, \vec{p}_j \in \mathbb{R}^D$ in this space. The way that the embedding is calculated in FastMap ensures that the L_1-distance in $\mathbb{R}^D$ can be used as an admissible and consistent heuristic for the shortest path computation in $\mathcal{G}$. To compute h-values for MAM, h_3 applies the Median heuristic on the generated embedding $\mathbb{R}^D$. For any node (a_i, v), its FastMap heuristic is defined as:

$$h_3(a_i, v) = \min_{\vec{m} \in \mathbb{R}^D} \{ \sum_{v' \in S_i(v)} \vec{p'} - \vec{m} \}, \tag{16}$$

where $\vec{p'} \in \mathbb{R}^D$ is the point of the embedding of location v' generated by FastMap. By the same analysis as for h_2, it can be shown that the FastMap heuristic (h_3) is admissible (and consistent), and it can be computed for every root node in $\Theta(kD)$ time and for any non-root node in $O(D)$ time.

Table 2
Results on 6×6 open grids, for SOC and MKSP.

	SOC		MKSP	
#Agents	Naive	MM*	Naive	MM*
2	146	33	125	12
3	936	78	1,464	27
4	16,677	137	28,259	50

Table 3
Results on 500×500 grids with varying obstacles, for SOC.

		SOC											
#Obs.	Cost	Initial h-value				#Expansions (thousands)				Time (sec)			
		h_0	h_1	h_2	h_3	h_0	h_1	h_2	h_3	h_0	h_1	h_2	h_3
0%	995	0	828	**995**	643	1,244	330	**34**	634	22.05	4.50	**0.28**	9.48
10%	1,008	0	828	**995**	680	1,120	322	**58**	561	19.04	4.29	**0.54**	8.66
20%	1,033	0	828	**995**	736	994	320	**83**	465	16.10	4.18	**0.77**	6.81
30%	1,103	0	828	**995**	778	856	318	**143**	402	13.04	4.01	**1.40**	5.60

There are many other approaches for embedding a graph in a continuous space [48–50]. However, most of these use L_2-distances, which are not computationally feasible here because the L_2-distance version of Equation 16 is NP-hard to solve optimally [47].

We next provide experiments for solving MAM with MM* for both SOC and MKSP using all our heuristics.

8. Experimental results for MAM

We experimented with MM* on an Intel® Xeon E5-2660 v4 @2.00GHz processor with 16GB of RAM. We compared all our new heuristics to the Dijkstra version of MM*, i.e., where $h = 0$ (denoted by h_0), on different grids while minimizing both SOC and MKSP. For h_1, we used the Manhattan Distance (MD) as a classic admissible heuristic between any two locations. The number of dimensions D for h_3 was always set to 10, as suggested by Li et al. [27].[8]

8.1. Naïve approach vs. MM*

First, we experimentally compare the non-heuristic versions of the naive approach and our proposed MM* algorithm presented above, for minimizing SOC and MKSP. Table 2 shows the average number of expansions of 50 problem instances of 2, 3, and 4 agents on a small 6×6 open grid. As can be seen, for the naive approach, increasing the number of agents causes exponential growth in the number of expansions. Consequently, while each problem instance tested for 2, 3, or 4 agents was solved in less than 5 minutes, no instance (out of 50) was solved by the naive approach within this time limit. Therefore, we only consider MM* in our next experiments below.

8.2. SOC

We experimented on 500×500 grids with 0% – 30% randomly allocated obstacles. Table 3 shows the average over 50 instances of: the solution cost, the initial h-value, the number of expansions, and the CPU time for 5 randomly placed agents. Each instance was randomly created with the selected number of agents and percentage of obstacles. The best results are highlighted in bold. h_2 had the best initial h-value, had the lowest number of expansions, and was the fastest. The initial h-values of h_1 (MD) and h_2 remain constant as more obstacles are added since they both ignore obstacles. By contrast, h_3 increases as more obstacles are added. So, when adding more obstacles h_2 degrades while h_3 improves, in terms of number of expansions and CPU time. h_3 incurred a preprocessing time of $\approx$ 30s, which is incurred only once per grid and thus was amortized over multiple problem instances. Thus, this was not included in the numbers in the table.

We also fixed the number of obstacles at 10% while varying the number of agents from 3 to 9. Table 4 shows the CPU time for SOC. Here, too, h_2 was the best heuristic with only 1.27s for 9 agents because h_2 is suitable for grids with small numbers of obstacles.

We also experimented on the 768×768 Enigma grid (presented in Fig. 6(a)) from the Starcraft video game, available in the `movingai` repository [51]. Figs. 6(b) and 6(c) show the average number of expansions and CPU time, respectively, for 3 to 9 agents for SOC. Here, h_3 was the best heuristic in both expansions and time. Since this grid has many obstacles (about 57%), h_1 and h_2 were less effective than h_3, which uses real distances (albeit in the embedded graph). Nevertheless, h_3 required a preprocessing time of 39s for this grid (done once). h_2 was the second best heuristic but h_2 is only suited for

[8] Our implementation is publicly available at https://github.com/doratzmon/MAM.

Table 4
Average time (sec) on 500 × 500 grids with 10% obstacles, for SOC.

	SOC			
#Agents	h_0	h_1	h_2	h_3
3	6.87	**0.16**	**0.16**	1.92
5	18.98	4.66	**0.81**	8.50
7	29.46	16.15	**0.85**	18.66
9	44.17	36.29	**1.27**	33.20

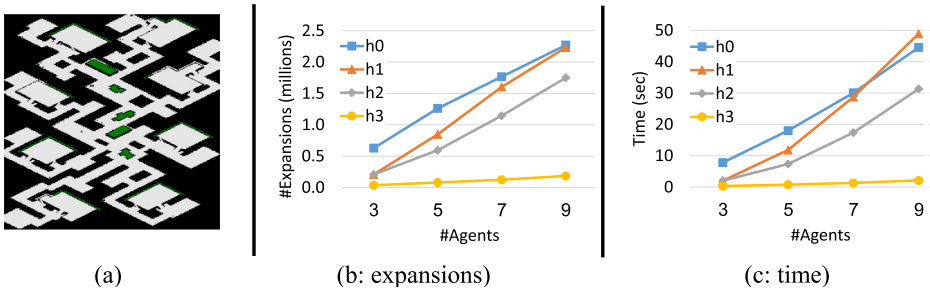
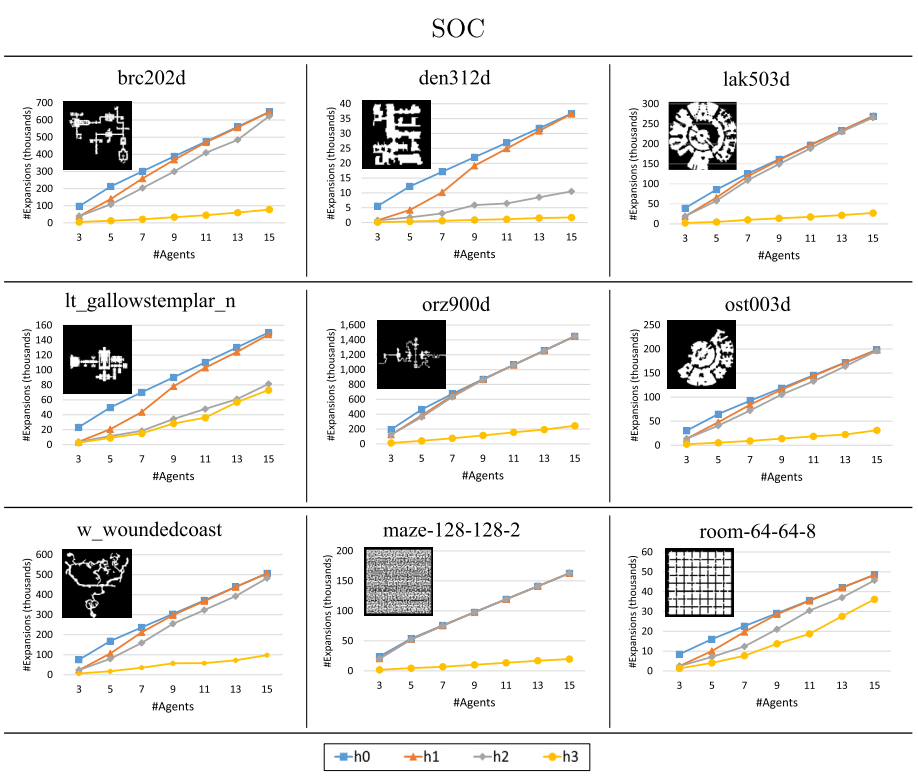


Fig. 6. (a) Enigma grid. (b) SOC expansions. (c) SOC time.

Table 5
Results on nine benchmark maps, for minimizing SOC.



grids while h_1 can be used on any graph. For 9 agents, h_1 expanded slightly fewer nodes than h_0 but, since it consumes time for computing the heuristic, was a little slower than h_0.

Finally, besides the Enigma map (which is the largest in the suit), we tested MM* with our heuristics on another set of nine publicly available benchmark maps [51]: brc202d, den312d, lak503d, lt_gallowstemplar_n, orz900d, ost003d, orz101d, maze-128-128-2, and room-64-64-8, and measured the average number of expansions. The results are presented in Table 5. Here too, as in the results performed on the Enigma map, h_3 was the best heuristic with the fewer number of expansions.

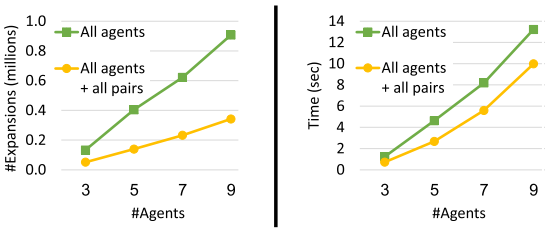


Fig. 7. MKSP results for h_3 on the Enigma grid.

Table 6
Results on 500 × 500 grids with varying obstacles, for MKSP.

		MKSP											
#Obs.	Cost	Initial h-value				#Expansions (thousands)				Time (sec)			
		h_0	h_1	h_2	h_3	h_0	h_1	h_2	h_3	h_0	h_1	h_2	h_3
0%	292	0	166	**199**	127	542	180	**179**	308	7.81	2.83	**2.79**	6.99
10%	293	0	166	**199**	137	485	159	**158**	299	6.69	2.43	**2.40**	5.44
20%	295	0	166	**199**	140	420	133	**132**	239	5.47	1.94	**1.93**	4.12
30%	305	0	166	**199**	148	341	121	**119**	197	4.10	1.66	**1.64**	4.00

Table 7
Average time (sec) on 500 × 500 grids with 10% obstacles, for MKSP.

	MKSP			
#Agents	h_0	h_1	h_2	h_3
3	1.91	0.47	**0.46**	1.48
5	6.73	2.62	**2.60**	6.57
7	11.03	3.57	**3.52**	9.59
9	18.07	6.53	**6.38**	16.46

To summarize, based on our experiments, for SOC, we believe that, on sparse maps, h_2 is best while, on dense maps with structured obstacles, h_3 is best.

8.3. MKSP

As described in Section 6, there are different policies for choosing subsets of agents for computing heuristics for MKSP. We compared two of these policies: **(1)** selecting all agents and **(2)** selecting all agents plus all pairs of agents and taking the maximum over all of those. Fig. 7 shows the number of expansions and CPU time averaged over 50 instances for h_3 on the Enigma grid with 3 to 9 agents. As expected, adding all pairs of agents produces a better heuristic and fewer expansions. It was also better in terms of the CPU time despite the fact that its computational overhead is larger. Therefore, below, we use this subset selection policy (2) for computing the heuristics when minimizing MKSP. We do not consider all subsets of agents for more than two agents as the number of such possible subsets grows exponentially with the size of the subsets. It is future work to investigate different policies for choosing specific subsets, out of the many possible subsets.

Next, for MKSP, we repeat the entire set of experiments on all domains that were used above for SOC. Table 6 presents the results on the 500 × 500 grids. Here, too, h_2 was the best heuristic but here, unlike SOC, h_1 was very close to h_2. This is probably because the clique heuristic (h_1) for MKSP also guides the agents to the median. In addition, as the number of obstacles increases, the initial h-value of h_3 increases while all other heuristics remain the same; at 0% obstacles, the initial h-value of h_3 was 127, while, at 30% obstacles, the initial h-value of h_3 was 148. This is reasonable as h_3 is the only heuristic function that considers obstacles when calculating its estimate.

The same trends were observed when we varied the number of agents with 10% obstacles and are presented in Table 7. Again, for all heuristics, a larger number of agents results in a longer average time. Here, as in Table 6, h_2 was the fastest heuristic where h_1 was a close second.

In the Enigma grid for MKSP (Figs. 8(b,c)), h_3 was again the best heuristic. h_1 and h_2 (the curves cover each other) had fewer expansions, but were slower than h_0. As mentioned, in maps with many obstacles h_1 and h_2 become less accurate, while h_3 has an advantage over the other heuristics as it is the only heuristic function that considers obstacles.

Finally, for MKSP too, we experimented on the additional nine benchmark maps. Table 8 presents the results. Here too, h_3 resulted in the fewest number of expansions while h_0 resulted in the highest number of expansions.

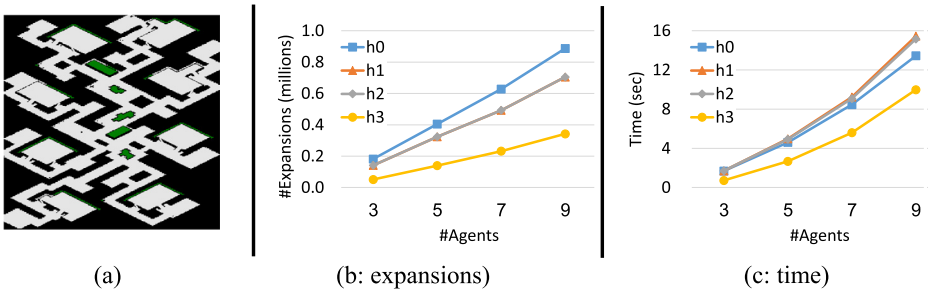


Fig. 8. (a) Enigma grid. (b) MKSP expansions. (c) MKSP time.

Table 8
Results on nine benchmark maps, for minimizing MKSP.

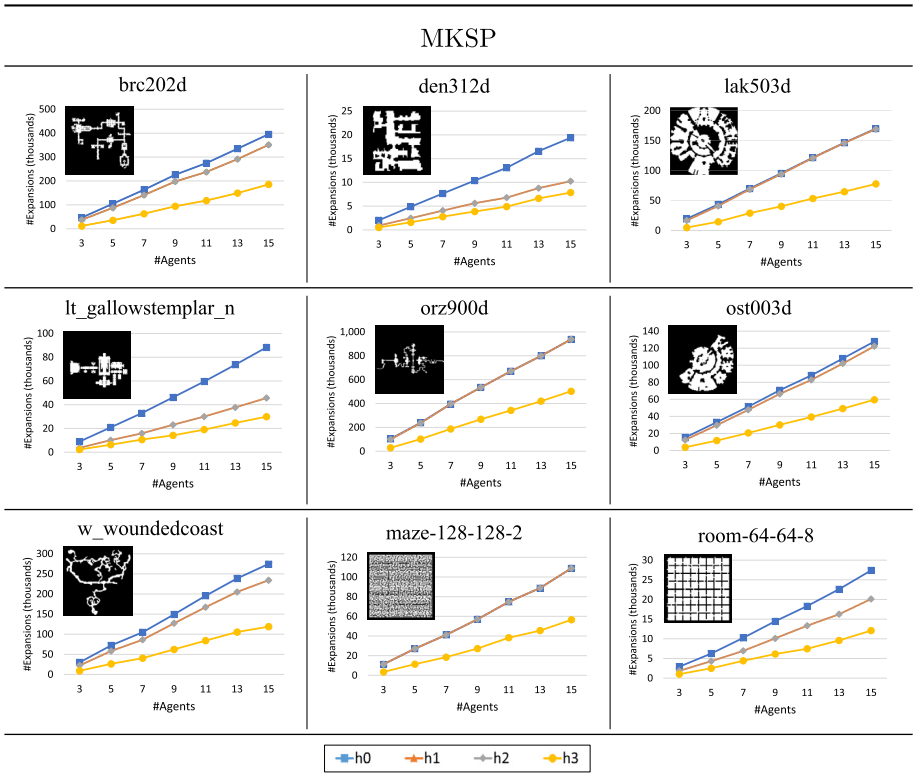


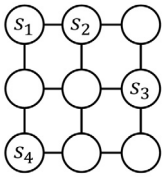
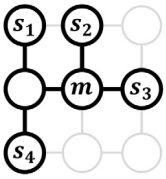
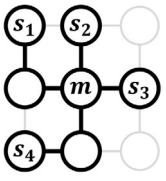
Table 9
Results on 200 × 200 grids with %10 obstacles, for SOC and MKSP.

	SOC				MKSP			
#Agents	h_0	h_1	h_2	h_3	h_0	h_1	h_2	h_3
20	20.2	22.9	**0.7**	19.4	15.8	7.3	**7.0**	10.9
40	51.8	62.2	**2.2**	52.6	54.1	23.9	**23.3**	40.1
60	103.6	124.8	**4.9**	104.8	112.8	57.9	**53.6**	92.4
80	176.4	204.2	**9.1**	173.0	155.3	99.7	**86.1**	152.0

8.4. Increasing the number of agents

To examine the impact of more agents on MM*, we experimented with 20, 40, 60, and 80 agents on 200×200 grids with 10% randomly allocated obstacles. Table 9 shows the average time (in seconds; 50 problem instances) for h_0, h_1, h_2, and h_3, for SOC and MKSP. Similarly to the experiments above on the 500×500 grids, here too, h_2 performs best, which is more prominent when minimizing SOC than MKSP. This is again a result of the fact that, when MKSP is minimized, our heuristic for MKSP is calculated indirectly, using a heuristic for SOC. Here, we can see that as the number of agents increases, the

Table 10
Input for both MAM and CF-MAM, and output for each, separately.

Input	MAM output	CF-MAM output
· Graph · Set of start locations	Meeting location and paths (possibly conflicting) to the meeting location	Meeting location and conflict-free paths to the meeting location
		

grid becomes more crowded, and the advantage of the heuristic decreases. In fact, we can see that, in SOC, h_0 performs similarly or even better than h_1 and h_3. The results here again show that h_2 performs best when the given map is sparse and not many obstacles are present. While h_3 achieved better performance in structured grids, h_2 is simpler and performs better on grids with random obstacles.

9. Conflict-free multi-agent meeting (CF-MAM)

While the basic version of MAM discussed thus far seeks a meeting location and paths for multiple agents to that meeting location, it is conflict-tolerant and ignores conflicts between the agents. Path-finding problems for multiple agents, such as MAPF, are often restricted to return conflict-free paths. We next define the conflict-free version of MAM as finding both a meeting location and conflict-free paths to that meeting location.

The *Conflict-Free Multi-Agent Meeting* problem (CF-MAM) receives 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. For simplicity, in this paper, we assume that for CF-MAM all edges have unit cost, which is common in other conflict-free MAPF problems [4]. Extending the CF-MAM algorithms presented below to general weighted graphs is left for future work [52,53].

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 cannot conflict at the goal location. Otherwise, the problem cannot be defined. We also define that agents can arrive at location m at the same timestep. Even if agents cannot physically conflict at the goal location, this is practical, for example, in the case that agents disappear at goal [4] (e.g., robots entering a charging station or autonomous vehicles entering a garage).

9.1. Solutions to MAM and CF-MAM

In some cases, a solution to MAM is invalid for CF-MAM. In the examples for MAM and CF-MAM in Table 10 (taken from Table 1) both get identical input (column 1). In the output example depicted for MAM (column 2), both agents a_1 and a_4 conflict at timestep 1. As defined in Section 2, we focus on two types of conflicts, i.e., vertex conflicts and swapping conflicts. As CF-MAM does not allow either vertex conflicts or swapping conflicts, this solution is invalid for CF-MAM. Thus, in the example output for CF-MAM (column 3), the path of agent a_4 is modified to be a non-conflicting path.

Lemma 1. *There exists a solution to CF-MAM iff there exists a solution to MAM.*

Proof. Direction 1: A solution to CF-MAM is naturally also a (not necessarily optimal) solution to MAM. **Direction 2:** Let $\Pi = \{\pi_1, \ldots, \pi_k\}$ be a set of shortest paths from a set of start locations S to a meeting location m, namely, a solution to MAM. Based on Π we can construct (reschedule) a new set of conflict-free paths from the given set of start locations S to m, namely, a solution to CF-MAM, as follows. We order on the agents in increasing order of their shortest paths to the meeting location. In its turn, only agent a_i follows its path π_i. Meanwhile, the other agents a_j where $j > i$ (those that had not yet moved) wait at their start locations. As only the agent with the shortest path moves among agents that are not yet at the meeting location, it must not conflict with a start location of one of the other agents a_j. Otherwise, the path of a_j is shorter than that of a_i. Therefore, this process results in a new set of paths without conflicts, which is a (not necessarily optimal) solution to CF-MAM. □

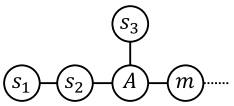


Fig. 9. Example of eliminating a swapping conflict.

9.2. Eliminating swapping conflicts

Lemma 2. *Let Π be a set of paths from a set of start locations S to a meeting location m with a cost of $C(\Pi)$ (either for SOC or MKSP), such that Π only contains swapping conflicts (and no vertex conflicts). Then, 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 with the same cost and the swapping conflict is eliminated, 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 $\pi'_j(t') \leftarrow \pi_i(t')$. After performing this mechanism, $C(\pi'_i) = C(\pi_j)$ and $C(\pi'_j) = C(\pi_i)$. Thus, it maintains both $C(\pi'_i) + C(\pi'_j) = C(\pi_i) + C(\pi_j)$ (same SOC) and $\max\{C(\pi'_i), C(\pi'_j)\} = \max\{C(\pi_i), C(\pi_j)\}$ (same MKSP). This can be performed repeatedly for each swapping conflict until Π' is conflict-free and with the same cost of $C(\Pi)$. To eliminate swapping conflicts, the agents are forced to switch paths and then each of the two smoothly follows the other agent's path. In a swapping conflict, the agents are at some locations v_1 and v_2 at some timestep t and at v_2 and v_1 at $t+1$, respectively. Therefore, it is identical to the case where the agents are at locations v_1 and v_2 at timestep t and $t+1$, which are wait actions. This wait action may be required in order to avoid other conflicts with other agents (other than these two agents). If the optimal solution contains such swapping conflicts, it means that such wait actions are mandatory. □

Fig. 9 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 Lemma 2, 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 Sections 10 and 11 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 2), both algorithms ignore possible swapping conflicts and are only designed to avoid vertex conflicts.

We now introduce our algorithms.

10. CBS-based solution for CF-MAM

Conflict-Based Search (CBS) [8] is a prominent, state-of-the-art MAPF solver. It plans a set of paths that may contain conflicts and iteratively resolves them by imposing constraints on the agents and replanning new paths for the constrained agents. Here, we introduce the *CF-MAM-CBS* (CFM-CBS) algorithm for solving CF-MAM, which uses the framework of the CBS algorithm. CFM-CBS works with either SOC or MKSP.

A *constraint* is a tuple $\langle a_i, v, t\rangle$ that prohibits agent a_i from occupying location v at timestep t. As in CBS, we use such constraints in CFM-CBS for resolving conflicts, as explained below.

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 possibly conflicting paths ($N.\Pi$) from start locations S to the optimal meeting location m such that the paths satisfy all constraints in $N.constraints$ (but may conflict otherwise).
3. The cost of $N.\Pi$ ($N.cost$).

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*, which was presented in Section 3.[9] However, to support constraints, as well as wait actions, MM* needs to be slightly modified

[9] Originally, to solve MAPF, the low level of CBS finds a path for each individual agent, e.g., using A*. Here, we jointly search for all agents with CF-MM*.

Algorithm 2: High level of CFM-CBS.

```
1  Main(CF-MAM problem instance)
2      Root.constraints ← {}
3      Root.Π ← low-level(instance, Root.constraints)
4      Root.cost ← C(Root.Π) // either SOC or MKSP
5      Insert Root into OPEN
6      while OPEN is not empty do
7          N ← Pop the node with the lowest cost in OPEN
8          if N.Π is conflict-free then
9              return N.Π // N is goal
10         ⟨a1, a2, v, t⟩ ← get-conflict(N)
11         N1 ← GenChild(N,⟨a1, v, t⟩)
12         N2 ← GenChild(N,⟨a2, v, t⟩)
13         Insert N1 and N2 into OPEN
14     return No Solution
15 GenChild(Node N, Constraint NewCons)
16     N′.constraints ← N.constraints ∪ {NewCons}
17     N′.Π ← low-level(instance, N′.constraints)
18     N′.cost ← C(N′.Π) // either SOC or MKSP
19     return N′
```

to become CF-MM*. Thus, instead of a 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* only moves N to CLOSED and does not expand it. Note that, as the root node of CT does not contain constraints, it is the only node that executes MM* instead of CF-MM*. Besides the difference described above, CF-MM* is identical to MM*; CF-MM* performs a best-first search using OPEN, but also performs duplicate detection and pruning on CLOSED.

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 $\langle a_j, v, t\rangle$.

Pseudo code. Algorithm 2 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. Fig. 10 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 MM* with 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.

Theorem 3 *(Completeness). CFM-CBS is guaranteed to return a solution if one exists, and it returns No Solution otherwise.*

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

As the root node of the CT does not contain any constraints, a standard version of MM* can be executed at the root, instead of CF-MM*. Following Lemma 1, if there is no solution to MAM, there is no solution to CF-MAM. Thus, if MM* returns No Solution, we can say that there is no solution to the given problem for CF-MAM. □

Theorem 4 *(Optimality). When CFM-CBS returns a solution, the solution has the lowest cost among all solutions.*

Proof. 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. □

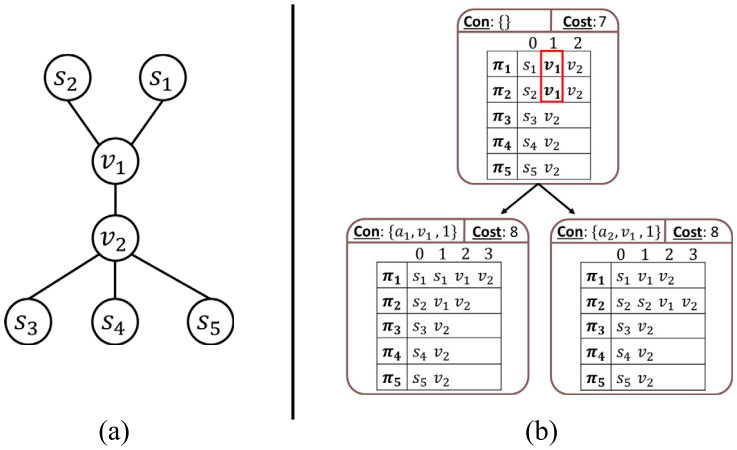


Fig. 10. CF-MAM problem instance and CFM-CBS's CT.

11. Iterative meeting solution for CF-MAM

As described in Section 2, PI-MAPF [9] 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, a solution for PI-MAPF must include an assignment of a goal location $g_i \in G$ and a proper path for that agent to that goal location. Yu and LaValle [10] defined SG-MAPF as a special case of PI-MAPF in which for each agent a_i we set $g_i \leftarrow g$ (the same goal location g). As PI-MAPF can be optimally solved in polynomial time using a reduction to Network Flow [10], SG-MAPF can also be optimally solved in polynomial time using the same reduction [10].

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 [10] 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 and solves it with a Network Flow algorithm.

11.1. Network flow problems

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

A network $N = \left\langle \overrightarrow{\mathcal{G}}, u, c, source, sink, R \right\rangle$ consists of a directed graph $\overrightarrow{\mathcal{G}} = (\mathcal{V}, \mathcal{E})$ with capacities u and costs c on the edges, i.e., $\forall e \in \mathcal{E}$, $u(e), c(e) \in \mathbb{Z}^+$, $source, sink \in \mathcal{V}$, and a required flow $R \in \mathbb{Z}^+$. 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) Required flow (supply/demand) at sink and source vertices,

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

Definition 1 *(*Minimum-Cost Flow problem *(MCFP)).* Given a network N, let F be the set of all feasible flows. The MCFP problem returns a minimum-cost feasible flow f, i.e., $\min_{f \in F}(\sum_{e \in \mathcal{E}} c(e) \cdot f(e))$.

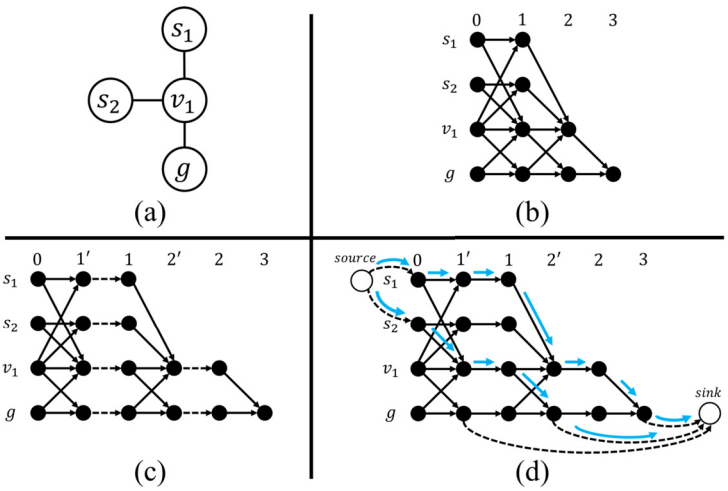


Fig. 11. Reducing SG-MAPF to Network Flow. Numbers on the x-axis are timesteps and letters on the y-axis are locations. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

The cost–scaling algorithm [54,55] is a commonly used polynomial-time solver for MCFP. We used it in our experiments below.

11.2. Reducing SG-MAPF to network flow

As defined above, MCFP minimizes the total cost of the flow. Thus, naturally, this reduction is more suitable for SOC than for MKSP because costs are being summed up. We first describe it in terms of SOC and later we show how it can also be used for MKSP.

Recall that the input to SG-MAPF is $\langle \mathcal{G}, S, g \rangle$. Let l' be the latest timestep (the cost of the longest path) among all paths of all optimal solutions to a SG-MAPF problem instance. For this reduction, we need to first calculate the depth of the network T, which must be an upper bound on l' to allow all optimal solutions. 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 with 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 the reduction of Yu and LaValle [10], $T = l + k - 1$ is a tight upper bound on l', i.e., $T \geq l'$ (for more details, see [10]). Note that, by using T as an upper bound on a possible solution to SG-MAPF, the agents will have paths of different lengths and can no longer conflict, even if in the set of shortest individual paths Π two agents conflict multiple times.

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 using 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 $t > 0$, 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 neighbor 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 has cost 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 $t = 0$. Fig. 11(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. Fig. 11(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 taking a wait action and moving diagonally corresponds to a move action.

Step 2. Fig. 11(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 $0 < t < T$ into 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 enforce that only one agent may enter the N nodes (nodes without an apostrophe) due to their capacity.

Step 3. Finally, as depicted in Fig. 11(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. We set a required flow $R = |A|$, for all agents. We need a capacity of k for any goal node $N' = (g, t')$, including $N' = (g, T)$, to allow all agents

Algorithm 3: Independence-detection enhancement.

```
1  ID(Set of shortest paths Π)
2      Init A' ← {}
3      C ← get-all-conflicts(Π)
4      foreach Conflict ⟨a1, a2, v, t⟩ ∈ C do
5          Δ ← 0
6          l̂ ← get-path-length(Π, a1)
7          do
8              Â ← get-all-agents-with-path-length(Π, l̂)
9              A' ← A' ∪ Â
10             l̂ ← l̂ + 1
11             Δ ← Δ + |Â| − 1
12         while Δ > 0;
13     return A'
```

to arrive at the goal at the same time. The bold blue arrows show the solution returned by executing an MCFP solver with $R = 2$. 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.

11.2.1. Independence detection enhancement

To determine T, as described above, we need to calculate all shortest individual paths Π, e.g., using a breadth-first search from location g. As the runtime of an MCFP solver is mainly influenced by T [10], before reducing SG-MAPF to Network Flow we execute an *independence-detection* (ID) mechanism [56]. Such mechanism detects paths in Π that can be preserved unchanged in the returned solution and not be passed to the reduction. We now identify a set of agents $A' \subseteq A$ that will be reduced to Network Flow. We only reduce agents that either conflict in their paths in Π or may conflict after conflicts are resolved as we now describe (Algorithm 3). First, we initialize an empty set A' (Line 2) and get all conflicts in Π to set C (Line 3). Then, for each conflict $\langle a_1, a_2, v, t\rangle$ we perform the following steps (Lines 4-12). Let Δ represent the amount by which any path can be further increased after conflicts are resolved. Based on this definition, after the length of a path of an agent is extended by Δ, more agents may conflict with the agent and these agents must be also considered in the reduction. We set $\Delta \leftarrow 0$ and $\hat{l}$ with the path length of the conflicting agents a_1 and a_2 (Lines 5-6). The two agents a_1 and a_2 conflict on their individual shortest paths (in Π) to location g. Thus, both agents must have paths of similar length and $\hat{l}$ is the length of the paths of both agents. Then, we start adding agents to A' and stop when $\Delta = 0$ (Lines 7-12). We get all agents $\hat{A}$ with a path length of $\hat{l}$ and add them to A' (Lines 8-9). These agents might conflict, after the conflict is resolved, since they have paths of a similar length. We then increment $\hat{l}$ and update $\Delta \leftarrow \Delta + |\hat{A}| - 1$ (Lines 10-11). This specific update of Δ comes from the fact that each agent can extend, in the worst case, its path by 1 to resolve each conflict. This occurs when, in the optimal solution, each agent must have a different length. Finally, we return A' (Line 13) as the set of agents to be reduced. The other agents can safely use their optimal individual path because they will never conflict with the other agents. T is then calculated on A' and might be smaller than if it had been calculated on the entire set of agents A. Note that, this mechanism does not depend on the order by which conflicts are considered by the algorithm as any order must result in the same set of agents A'.

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. a_1 and a_2 are added to the set A'. We know that, in the optimal solution, one of agents a_1 or a_2 may have a path of length 5. Thus, we also add a_3 (because it also has a path of cost to 5) to A'. 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 [10] and they will be at least one step ahead on their way to location g.

11.2.2. Minimizing MKSP

While the reduction described above is for minimizing SOC, it can also be used for minimizing MKSP, as follows. We first calculated an upper bound T on the depth, for which a solution surely exists. Then, we created a network with depth T for an MCFP solver, which minimizes SOC. For SOC we set $T \leftarrow l + k - 1$ and looked for the lowest SOC solution for which the longest individual paths is *smaller than or equal to* this depth T. By contrast, for minimizing MKSP the depth of the network *is* the cost of the solution. Thus, we set $T \leftarrow l$ as l is the lowest depth for which a solution may exist. We perform the reduction with this bound and execute an MCFP solver. If no solution is found, we repeatedly increment T and execute an MCFP solver until we find a solution or reach $T = l + k - 1$ with no solution, in which case a failure is returned.

11.2.3. Differences between the reductions

There are a number of differences between our reduction and the reduction from PI-MAPF, introduced by Yu and LaValle [10].

1. In our reduction the graph $\mathcal{G}$ is constructed by performing a single breadth-first search from location g, instead of k breadth-first searches (from the k start locations) in their reduction.

Algorithm 4: High level of IMS.

```
1  Main(CF-MAM problem instance)
2      Init OPEN, CLOSED; U ← ∞
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 ← min{U, low-level(instance, v)}
7          if f(a_i, v) ≥ U then
8              return U
9          foreach v' ∈ 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') ≤ 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
```

2. We apply independence detection to construct a smaller network flow.
3. Following Lemma 2, there is no need to avoid swapping conflicts, which requires a special step by Yu and LaValle [10].

Therefore, optimality follows.

11.3. Iterative meeting search

We now present the *Iterative Meeting Search* algorithm (IMS) for solving CF-MAM. IMS 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 (passed by the high level) as a goal location of SG-MAPF and applies a Network-Flow solver to solve it using our reduction.

Algorithm 4 describes the pseudo code of the high level of IMS. First, it initializes OPEN and CLOSED, and initializes 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 the start location 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^* of the optimal solution for CF-MAM, i.e., $C' \leq C^*$. Thus, for each node of the optimal solution, 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 preforming the above reduction with v as the goal location and then executing an MCFP solver on it). Then, U is updated with the lowest cost found (Line 6).[10] 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.[11] The closeness centrality of a start location s_i is 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 performs better than random. This is reasonable as such a location is usually closer to the optimal meeting location. Future work may investigate different start locations for IMS.

Theorem 5 *(Completeness). IMS is guaranteed to either return a solution or return $U = \infty$.*

Proof. 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.

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

[11] In a connected graph, *closeness centrality* of a node is a measure of centrality in a network, calculated as the reciprocal ($1/x$ is the reciprocal for x) of the sum of the length of the shortest paths between the node and all other nodes in the graph [57]. Thus, the more central a node is, the closer it is to all other nodes.

Table 11
Results for 10x10 and 50x50 grids with 20% Obs. for minimizing SOC.

		SOC				
#Agents	Solver	10×10		50×50		
		Cost	Time	Succ.	Cost	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		

In case no solution exists, the MCFP solver will fail at $T = l + k - 1$ (either for SOC or MKSP) and $U = \infty$ will be returned.[12] □

Theorem 6 *(Optimality). IMS is guaranteed to return the optimal meeting location m^* (with cost C^*).*

Proof. Let $s_i \in S$ be the start location of agent a_i from which IMS started to search. 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 $fmin \geq C > C^*$. □

12. Experimental results for CF-MAM

We performed experiments comparing our two algorithms for CF-MAM, again 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 For solving the Minimum-Cost Flow problem (MCFP) we used for an efficient implementation [55] of the cost–scaling algorithm of Goldberg and Tarjan [54], which runs in polynomial time. For both, we used the clique heuristic as an admissible heuristic for a meeting location, which balances well between simplicity and efficiency.

12.1. 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 measured the success rate (for timeout of 5min for each instance), average cost, and average time (seconds). In all experiments, the average cost and time were calculated only from problem instances that were solved by both solvers. Table 11 presents the results for this experiment for minimizing SOC, and Table 12 presents the results for this experiment for minimizing MKSP. Each row shows the number of agents.

For minimizing SOC (Table 11), for the 10x10 grids (columns 3-4), both CFM-CBS and IMS solve all problem instances and hence we do not present the success rate in the table. As expected, a 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 $\approx 0.1s$, and for 15 agents, CFM-CBS ran for $\approx 57.3s$ while IMS ran for only

[12] To simplify the algorithm, this is not presented in Algorithm 4. It can be easily added by returning $U = \infty$ if the low level in Line 6 does not find a solution.

Table 12
Results for 10x10 and 50x50 grids with 20% Obs. for minimizing MKSP.

		MKSP					
#Agents	Solver	10×10 Succ.	Cost	Time	50×50 Succ.	Cost	Time
3	CFM-CBS	**50**	5	**0.0**	**50**	25	**0.0**
	IMS	**50**		**0.0**	49		14.8
5	CFM-CBS	**50**	7	**0.0**	**50**	30	**0.2**
	IMS	**50**		0.1	47		62.9
7	CFM-CBS	**50**	7	**0.0**	**50**	33	**0.7**
	IMS	46		0.2	40		105.7
9	CFM-CBS	**50**	8	2.0	**50**	35	**2.2**
	IMS	**50**		**0.3**	39		178.0
11	CFM-CBS	**49**	8	5.9	**49**	37	**3.0**
	IMS	**50**		**0.6**	32		210.7
13	CFM-CBS	**47**	8	7.7	**49**	-	-
	IMS	**50**		**0.8**	9		
15	CFM-CBS	**39**	9	13.7	**47**	-	-
	IMS	**50**		**1.2**	1		

$\approx 1.7s$. 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. While CF-MM* (the low level of CFM-CBS) is polynomial, many agents are led to the same meeting location which may result in many conflicts that are needed to be resolved. Each such conflict splits a high-level node into two new nodes, which causes an exponential growth in the number of high-level nodes. These conflicts are more likely to occur in dense environments as they require multiple agents to be at the exact same location at the same timestep.

For the 50x50 grids (columns 5-7), 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 for approximately 50.5s and 200.2s, respectively. Here, the environment is sparser and fewer conflicts occur. Thus, CFM-CBS can perform better than observed above for the dense 10x10 grid.

For minimizing MKSP (Table 12), the trends were similar to those observed for minimizing SOC: IMS performed better in the 10x10 grids (columns 3-5) while CFM-CBS performed better in the 50x50 grids (columns 6-8). For 11 agents, the average times of CFM-CBS and IMS for 10x10 grids were approximately 5.9s and 0.6s, respectively, and for 50x50 grids the average times were approximately 3.0s and 210.7s, respectively.

12.2. Structured grids

We also tested CFM-CBS and IMS on a set of benchmark maps. Here, we performed two experiments, as follows.

12.2.1. Experiment 1

We tested CFM-CBS and IMS on:

- A warehouse map, used by Ma et al. [58] and Atzmon et al. [43].
- The den312d map from the Dragon Age Origins (DAO) video game, which is publicly available [51].

The leftmost column in Table 13 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 13, respectively). Here also, the average cost and average time were calculated only from instances that were solved by both solvers. This table presents the results for minimizing SOC.

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.

Table 13
Results for the warehouse domain (first row) and the den312d map from DAO (second row), for minimizing SOC.

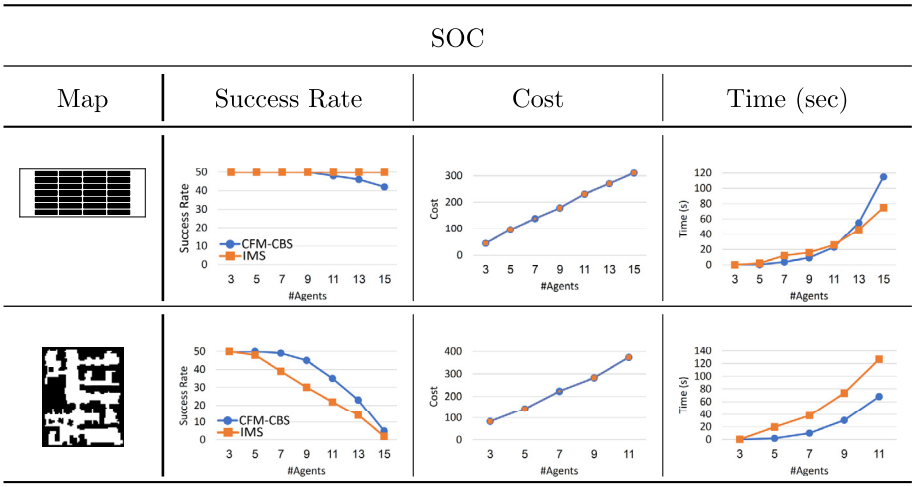
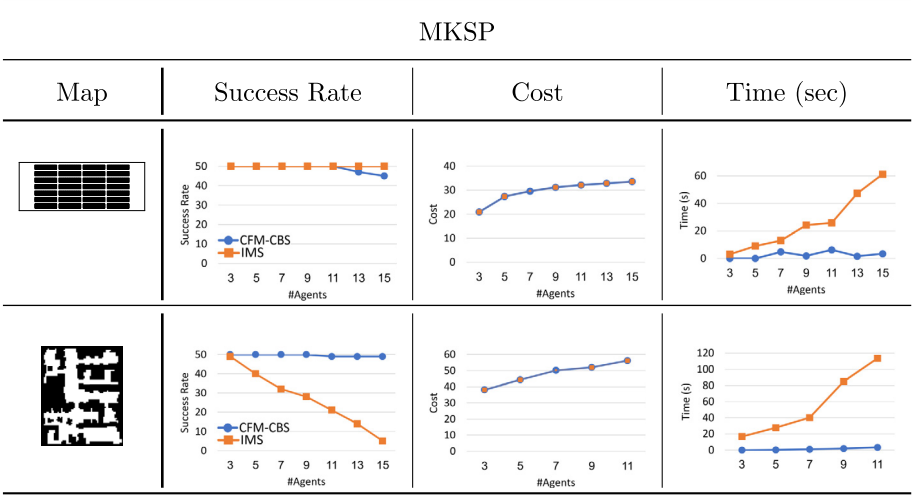


Table 14
Results for the warehouse domain (first row) and the den312d map from DAO (second row), for minimizing MKSP.



We perform a similar experiment for minimizing MKSP (Table 14). In terms of success rate, in the warehouse domain, IMS solved a few more instances than CFM-CBS. However, in the den312d map, CFM-CBS solved many more instances than IMS. Moreover, in terms of runtime, CFM-CBS performed better than IMS in both domains. It is interesting to notice that CFM-CBS, in this experiment, performed better when minimizing MKSP than when minimizing SOC. In MKSP, the cost can only increase when we resolve a conflict which involves the agent with the longest path. Thus, when resolving a conflict in SOC, the solution cost increases more often than when minimizing MKSP.

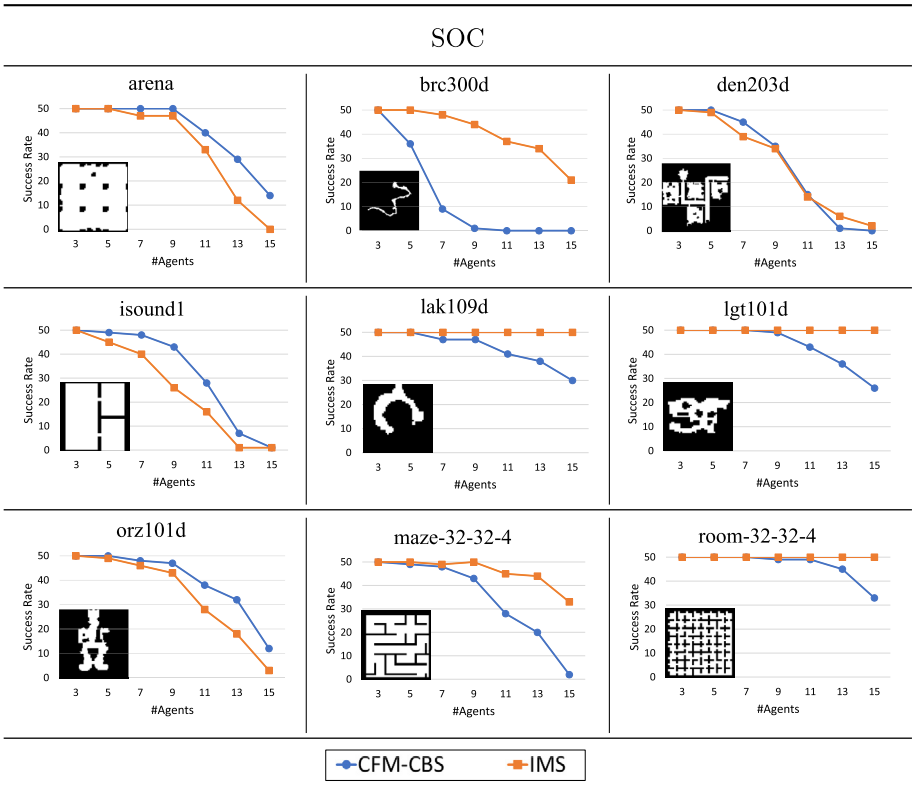
12.2.2. Experiment 2

We tested CFM-CBS and IMS on nine publicly available benchmark maps [51]: arena, brc300d, den203d, isound1, lak109d, lgt101d, orz101d, maze-32-32-4, and room-32-32-4.

We created 50 problem instances for each map with $3, 5, 7, 9, 11, 13$, and 15 randomly allocated agents. Table 15 shows the success rate of CFM-CBS and IMS for a 5min time limit for each instance, for minimizing SOC.

In some maps, CFM-CBS performs better while, in others, IMS performs better. Again, an important factor that influences the success of the algorithm is the size of the map. For example, CFM-CBS performs better in arena and isound1 where there are more than 2,000 states. In contrast, IMS performs better in lak109d, lgt101d, maze-32-32-4, and room-32-32-4 where there are less than 1,000 states. In brc300d, IMS performed better than CFM-CBS although more than 5,000 states exist. This is due to the special structure of the map being a long snake-shaped map, which caused many conflicts for CFM-CBS and thus poor performance. We performed a similar experiment for minimizing MKSP (Table 16) and observe similar trends.

Table 15
Results on nine benchmark maps, for minimizing SOC.



Our experiments provide the following general rule: CFM-CBS should be used in sparse environments while IMS should be used in dense environments. Future work will define and examine different features of problem instances for choosing a preferred solver for cases that are not clearly sparse or dense.

13. Conclusions and future work

In this paper, we explored the problems of *Conflict-Tolerant Multi Agent Meeting* (MAM) and *Conflict-Free Multi-Agent Meeting* (CF-MAM).

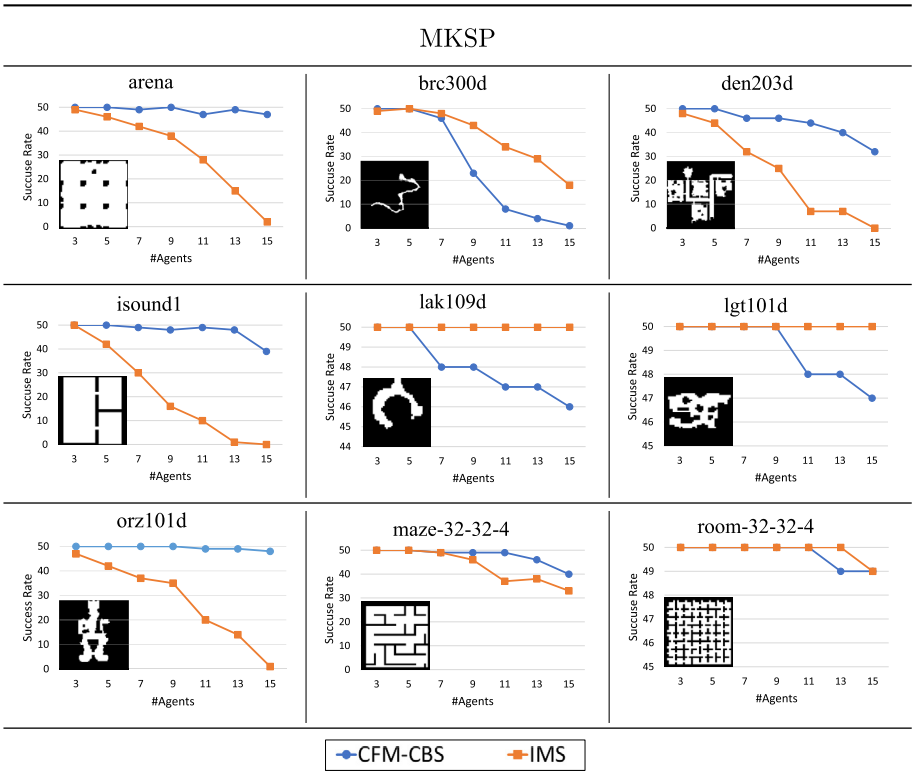
We introduced the multi-directional search algorithm MM* that optimally solves MAM instances. We proved that MM* is complete and optimal and proposed three admissible heuristics: the Clique heuristic (h_1), Median heuristic (h_2), and FastMap heuristic (h_3). Experimentally, we showed that MM* performs better with heuristics. For grids with few obstacles, h_2 is best. For grids with many obstacles, h_3 is best but requires preprocessing. The advantage of h_1 is that it is applicable to all domains without the need for preprocessing.

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. In fact, there is no exact definition of sparse and dense within the context of MAPF and this, too, is left for future work.

Moreover, future work will:

1. Develop a version of MM* that can find bounded-suboptimal solutions.
2. Extend MM* for solving MAM on a continuous space.
3. Further investigate subset selection for MKSP.
4. Enhance CFM-CBS with many of the improvements that were proposed for CBS (such as prioritizing conflicts [33]).
5. For IMS, suggest more sophisticated rules for calling the low level.
6. Adjust other MAPF solvers for solving CF-MAM, such as ICTS [32].

Table 16
Results on nine benchmark maps, for minimizing MKSP.



Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The implemntaion of the algorithms presented in this paper can be reached using the link provided in the text.

Acknowledgements

This research was supported by ISF grant 844/17 and BSF grants 017692 and 2021643 to Ariel Felner, NSF grant 1815660 to Nathan R. Sturtevant and NSF grants 1409987, 1724392, 1817189, 1837779 and 1935712 to Sven Koenig.

References

[1] D. Atzmon, J. Li, A. Felner, E. Nachmani, S.S. Shperberg, N. Sturtevant, S. Koenig, Multi-directional heuristic search, in: The International Joint Conference on Artificial Intelligence (IJCAI), 2020, pp. 4062–4068.
[2] D. Atzmon, S.I. Freiman, O. Epshtein, O. Shichman, A. Felner, Conflict-free multi-agent meeting, in: The International Conference on Automated Planning and Scheduling (ICASP), 2021, pp. 16–24.
[3] D. Yan, Z. Zhao, W. Ng, Efficient processing of optimal meeting point queries in Euclidean space and road networks, Knowl. Inf. Syst. 42 (2) (2015) 319–351.
[4] R. Stern, N.R. Sturtevant, A. Felner, S. Koenig, H. Ma, T.T. Walker, J. Li, D. Atzmon, L. Cohen, T.K.S. Kumar, R. Barták, E. Boyarski, Multi-agent pathfinding: definitions, variants, and benchmarks, in: The International Symposium on Combinatorial Search (SoCS), 2019, pp. 151–159.
[5] R.C. Holte, A. Felner, G. Sharon, N.R. Sturtevant, Bidirectional search that is guaranteed to meet in the middle, in: The AAAI Conference on Artificial Intelligence (AAAI), 2016, pp. 3411–3417.
[6] H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. Hönig, T. Kumar, T. Uras, H. Xu, C. Tovey, G. Sharon, Overview: Generalizations of multi-agent path finding to real-world scenarios, the IJCAI-16 Workshop on Multi-Agent Path Finding.
[7] A. Felner, R. Stern, S.E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N.R. Sturtevant, G. Wagner, P. Surynek, Search-based optimal solvers for the multi-agent pathfinding problem: summary and challenges, in: The International Symposium on Combinatorial Search (SoCS), 2017, pp. 29–37.
[8] G. Sharon, R. Stern, A. Felner, N.R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding, Artif. Intell. 219 (2015) 40–66.
[9] S. Kloder, S. Hutchinson, Path planning for permutation-invariant multirobot formations, IEEE Trans. Robot. 22 (4) (2006) 650–665.
[10] J. Yu, S.M. LaValle, Multi-agent path planning and network flow, in: Algorithmic Foundations of Robotics X, Springer, 2013, pp. 157–173.

[11] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. SSC-4 (2) (1968) 100–107.
[12] R. Dechter, J. Pearl, Generalized best-first search strategies and the optimality of a*, J. ACM 32 (3) (1985) 505–536.
[13] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1) (1959) 269–271.
[14] Z. Xu, H.-A. Jacobsen, Processing proximity relations in road networks, in: The ACM SIGMOD International Conference on Management of Data (SIGMOD), 2010, pp. 243–254.
[15] R. Geisberger, P. Sanders, D. Schultes, D. Delling, Contraction hierarchies: faster and simpler hierarchical routing in road networks, in: The International Workshop on Experimental and Efficient Algorithms, 2008, pp. 319–333.
[16] Y. Izmirlioglu, B.A. Pehlivan, M. Turp, E. Erdem, A general formal framework for multi-agent meeting problems, in: The IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 1299–1306.
[17] G. Brewka, T. Eiter, M. Truszczyński, Answer set programming at a glance, Commun. ACM 54 (12) (2011) 92–103.
[18] L. Cooper, An extension of the generalized Weber problem, J. Reg. Sci. 8 (2) (1968) 181–197.
[19] L.M. Ostresh Jr, The multifacility location problem: applications and descent theorems, J. Reg. Sci. 17 (3) (1977) 409–419.
[20] R. Chen, Location problems with costs being sums of powers of Euclidean distances, Comput. Oper. Res. 11 (3) (1984) 285–294.
[21] F. Radó, The Euclidean multifacility location problem, Oper. Res. 36 (3) (1988) 485–492.
[22] K.E. Rosing, An optimal method for solving the (generalized) multi-Weber problem, Eur. J. Oper. Res. 58 (3) (1992) 414–426.
[23] N. Megiddo, The weighted Euclidean 1-center problem, Math. Oper. Res. 8 (4) (1983) 498–504.
[24] M.A. Lanthier, D. Nussbaum, T.-J. Wang, Calculating the meeting point of scattered robots on weighted terrain surfaces, in: The Australasian Theory Symposium, vol. 41, 2005, pp. 107–118.
[25] E. Welzl, Smallest enclosing disks (balls and ellipsoids), in: New Results and New Trends in Computer Science, Springer, 1991, pp. 359–370.
[26] L. Cohen, T. Uras, S. Jahangiri, A. Arunasalam, S. Koenig, T.K.S. Kumar, The FastMap algorithm for shortest path computations, in: The International Joint Conference on Artificial Intelligence (IJCAI), 2018, pp. 1427–1433.
[27] J. Li, A. Felner, S. Koenig, T.K.S. Kumar, Using FastMap to solve graph problems in a Euclidean space, in: The International Conference on Automated Planning and Scheduling (ICAPS), 2019, pp. 273–278.
[28] J. Yu, S.M. LaValle, Structure and intractability of optimal multi-robot path planning on graphs, in: The AAAI Conference on Artificial Intelligence (AAAI), 2013, pp. 1444–1449.
[29] P. Surynek, An optimization variant of multi-robot path planning is intractable, in: The AAAI Conference on Artificial Intelligence (AAAI), 2010, pp. 1261–1263.
[30] G. Wagner, H. Choset, Subdimensional expansion for multirobot path planning, Artif. Intell. 219 (2015) 1–24.
[31] P. Surynek, Towards optimal cooperative path planning in hard setups through satisfiability solving, in: The Pacific Rim International Conference on Artificial Intelligence (PRICAI), 2012, pp. 564–576.
[32] G. Sharon, R. Stern, M. Goldenberg, A. Felner, The increasing cost tree search for optimal multi-agent pathfinding, Artif. Intell. 195 (2013) 470–495.
[33] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, S.E. Shimony, ICBS: improved conflict-based search algorithm for multi-agent pathfinding, in: The International Joint Conference on Artificial Intelligence (IJCAI), 2015, pp. 740–746.
[34] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T.K.S. Kumar, S. Koenig, Adding heuristics to conflict-based search for multi-agent path finding, in: The International Conference on Automated Planning and Scheduling (ICAPS), 2018, pp. 83–87.
[35] J. Li, D. Harabor, P.J. Stuckey, H. Ma, S. Koenig, Symmetry-breaking constraints for grid-based multi-agent path finding, in: The AAAI Conference on Artificial Intelligence (AAAI), 2019, pp. 6087–6095.
[36] E. Lam, P.L. Bodic, D. Harabor, P.J. Stuckey, Branch-and-cut-and-price for multi-agent pathfinding, in: The International Joint Conference on Artificial Intelligence (IJCAI), 2019, pp. 1289–1296.
[37] J. Li, P. Surynek, A. Felner, H. Ma, T.K.S. Kumar, S. Koenig, Multi-agent path finding for large agents, in: The AAAI Conference on Artificial Intelligence (AAAI), 2019, pp. 7627–7634.
[38] D. Atzmon, A. Diei, D. Rave, Multi-train path finding, in: The International Symposium on Combinatorial Search (SoCS), 2019, pp. 125–129.
[39] S. Thomas, D. Deodhare, M.N. Murty, Extended conflict-based search for the convoy movement problem, IEEE Intell. Syst. 30 (2015) 60–70.
[40] D. Atzmon, Y. Zax, E. Kivity, L. Avitan, J. Morag, A. Felner, Generalizing multi-agent path finding for heterogeneous agents, in: The International Symposium on Combinatorial Search (SoCS), 2020, pp. 101–105.
[41] H. Ma, G. Wagner, A. Felner, J. Li, T.S. Kumar, S. Koenig, Multi-agent path finding with deadlines, in: The International Joint Conference on Artificial Intelligence (IJCAI), 2018, pp. 417–423.
[42] D. Atzmon, R. Stern, A. Felner, N.R. Sturtevant, S. Koenig, Probabilistic robust multi-agent path finding, in: ICAPS, 2020, pp. 29–37.
[43] D. Atzmon, R. Stern, A. Felner, G. Wagner, R. Barták, N.-F. Zhou, Robust multi-agent path finding and executing, J. Artif. Intell. Res. 67 (2020) 549–579.
[44] H. Ma, S. Koenig, Optimal target assignment and path finding for teams of agents, in: The International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2016, pp. 1144–1152.
[45] K. Solovey, D. Halperin, On the hardness of unlabeled multi-robot motion planning, Int. J. Robot. Res. 35 (14) (2016) 1750–1759.
[46] E. Shaham, A. Felner, J. Chen, N.R. Sturtevant, The minimal set of states that must be expanded in a front-to-end bidirectional search, in: The International Symposium on Combinatorial Search (SoCS), 2017, pp. 82–90.
[47] C.A. Hoare, Algorithm 65: find, Commun. ACM 4 (7) (1961) 321–322.
[48] T.S.E. Ng, H. Zhang, Predicting internet network distance with coordinates-based approaches, in: The Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 2002, pp. 170–179.
[49] Y. Shavitt, T. Tankel, Big-bang simulation for embedding network distances in Euclidean space, IEEE/ACM Trans. Netw. 12 (6) (2004) 993–1006.
[50] C. Rayner, M. Bowling, N.R. Sturtevant, Euclidean heuristic optimization, in: The AAAI Conference on Artificial Intelligence (AAAI), 2011, pp. 81–86.
[51] N.R. Sturtevant, Benchmarks for grid-based pathfinding, IEEE Trans. Comput. Intell. AI Games 4 (2) (2012) 144–148.
[52] A. Andreychuk, K. Yakovlev, D. Atzmon, R. Stern, Multi-agent pathfinding with continuous time, in: The International Joint Conference on Artificial Intelligence (IJCAI), 2019, pp. 39–45.
[53] T.T. Walker, N.R. Sturtevant, A. Felner, Extended increasing cost tree search for non-unit cost domains, in: The International Joint Conference on Artificial Intelligence (IJCAI), 2018, pp. 534–540.
[54] A.V. Goldberg, R.E. Tarjan, Finding minimum-cost circulations by successive approximation, Math. Oper. Res. 15 (3) (1990) 430–466.
[55] A.V. Goldberg, An efficient implementation of a scaling minimum-cost flow algorithm, J. Algorithms 22 (1) (1997) 1–29.
[56] T.S. Standley, Finding optimal solutions to cooperative pathfinding problems, in: The AAAI Conference on Artificial Intelligence (AAAI), 2010, pp. 28–29.
[57] G. Sabidussi, The centrality index of a graph, Psychometrika 31 (4) (1966) 581–603.
[58] H. Ma, T.S. Kumar, S. Koenig, Multi-agent path finding with delay probabilities, in: The AAAI Conference on Artificial Intelligence (AAAI), 2017, pp. 3605–3612.