Moving Agents in Formation in Congested Environments

Jiaoyang Li
University of Southern California
jiaoyanl@usc.edu

Kexuan Sun
University of Southern California
kexuansu@usc.edu

Hang Ma
Simon Fraser University
hangma@sfu.ca

Ariel Felner
Ben-Gurion University
felner@bgu.ac.il

T. K. Satish Kumar
University of Southern California
tkskwork@gmail.com

Sven Koenig
University of Southern California
skoenig@usc.edu

ABSTRACT

In this paper, we formalize and study the Moving Agents in Formation (MAiF) problem, that combines the tasks of finding short collision-free paths for multiple agents and keeping them in close adherence to a desired formation. Previous work includes controller-based algorithms, swarm-based algorithms, and potential-field-based algorithms. They usually focus on only one or the other of these tasks, solve the problem greedily without systematic search, and thus generate costly solutions or even fail to find solutions in congested environments. In this paper, we develop a two-phase search algorithm, called SWARM-MAPF, whose first phase is inspired by swarm-based algorithms (in open regions) and whose second phase is inspired by multi-agent path-finding (MAPF) algorithms (in congested regions). In the first phase, SWARM-MAPF selects a leader among the agents and finds a path for it that is sufficiently far away from the obstacles so that the other agents can preserve the desired formation around it. It also identifies the critical segments of the leader's path where the other agents cannot preserve the desired formation and the refinement of which has thus to be delegated to the second phase. In the second phase, SWARM-MAPF refines these segments. Theoretically, we prove that SWARM-MAPF is complete. Empirically, we show that SWARM-MAPF scales well and is able to find close-to-optimal solutions.

KEYWORDS

Multi-agent path finding; multi-agent planning; formation control

ACM Reference Format:
Jiaoyang Li, Kexuan Sun, Hang Ma, Ariel Felner, T. K. Satish Kumar, and Sven Koenig. 2020. Moving Agents in Formation in Congested Environments. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

Planning paths for multiple agents in known congested environments is an important problem that arises in many real-world applications of multi-agent systems. Examples include aircraft-towing vehicles [19], warehouse robots [31], and video game characters [18]. In many applications, it is also important for the agents to move in a specific formation while avoiding obstacles. For example, unmanned vehicles have to move in specific formations in order to transport large objects or maintain a communication network. Game characters or army personnel have to move in specific formations in order to protect vulnerable agents. These applications involve two key tasks: (a) planning collision-free paths for multiple agents, and (b) keeping the agents in formation. Task (a) can be addressed with multi-agent path finding (MAPF) algorithms, which typically minimize one of several possible metrics on the path costs. MAPF is NP-hard to solve optimally [17, 33]. It can be solved via reductions to other well-studied combinatorial problems [5, 10, 28, 32] or with dedicated MAPF algorithms [14, 15, 22–26, 29, 30]. Task (b) can be addressed with formation-control algorithms, which try to restore the desired formation in case it is compromised because of obstacles. Examples include behavior-based [1], leader-follower [3], virtual-structure [12], potential-field [11], graph-based [4], and other swarm-based algorithms [9, 13, 21]. However, these algorithms often do not work well in congested environments and provide no completeness guarantees.

We thus study the *Moving Agents in Formation (MAiF)* problem in congested environments to bridge the gap between algorithms that focus on tasks (a) or (b) exclusively. MAiF is a problem related to MAPF where a desired formation is given and the task is to plan collision-free paths for all agents that balance between the minimization of the makespan and a close adherence to the desired formation at all times. To the best of our knowledge, none of the previous work in AI or robotics is directly applicable to MAiF. A first attempt, presented in [18], calls a MAPF algorithm repeatedly to solve MAiF. However, the agents do not always restore the desired formation when it is compromised after they move around obstacles.

We organize the paper as follows. In Section 2, we define MAiF formally and introduce a distance metric that measures the difference between two formations in a Cartesian system. In Section 3, we present a leader-follower controller, adapted from formation-control algorithms, and analyze its drawbacks with respect to its solution quality in congested environments. In Section 4, we present joint-state A*, adapted from MAPF algorithms. Although joint-state A* is able to find Pareto-optimal solutions, it is extremely time-consuming and does not scale to large numbers of agents. In Section 5, we present SWARM-MAPF, a novel two-phase algorithm that combines the ideas of formation-control and MAPF. It uses a loose coupling between a swarm-based first phase and a MAPF-based second phase. A swarm-based algorithm is used in the first phase for open parts of the environment where all agents can move

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, and 1837779 as well as a gift from Amazon. The research was also supported by the United States-Israel Binational Science Foundation (BSF) under grant number 2017692.

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand. © 2020 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

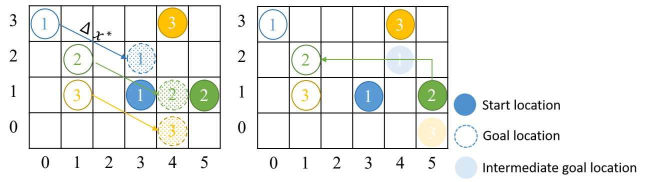

Figure 1: A MAiF instance on a 4-neighbor grid. The first dimension is horizontal, and the second dimension is vertical. In the left figure, the hatched circles show the locations of all agents after applying the optimal translation $\Delta x^* = (3, -1)$ to the goal locations. In the right figure, the green line shows the path of the leader.

in the desired formation. CBS-M, a novel MAPF-based algorithm, is used in the second phase for congested parts of the environment where the desired formation has to be temporarily compromised. CBS-M can also be used as a stand-alone MAiF algorithm to solve MAiF. It produces solutions that minimize the makespan and uses novel combinatorial search algorithms to try to keep the agents in close adherence to the desired formation. Theoretically, we prove that SWARM-MAPF is complete for all MAiF instances, and CBS-M minimizes the makespan. Experimentally, in Section 6, we show that SWARM-MAPF scales well and produces solutions that keep all agents in the desired formation better than CBS-M with only a small loss of optimality in makespan.

2 PROBLEM DEFINITION

In this section, we formalize MAiF in a Cartesian system: We are given an undirected graph $G = (V, E)$ in a d-dimensional Cartesian system. The vertices V correspond to locations, and the edges E correspond to transitions between locations. A location $v_i \in V$ can be recognized by its coordinates $\mathbf{v_i} = (\mathbf{v_{i1}}, \ldots, \mathbf{v_{id}}) \in \mathbb{R}^d$. We are also given a set of M agents $\{a_i | i = 1, \ldots, M\}$, each with a unique start location $s_i \in V$ and a unique goal location $g_i \in V$. Time is discretized into timesteps. Between successive timesteps, every agent can either move to an adjacent location or wait at its current location. A *path* π_i for agent a_i is a sequence of locations, one for each timestep, that moves agent a_i from its start location s_i to its goal location g_i. $\pi_i(t) \in V$ is the location of agent a_i at timestep t. Agents remain at their goal locations forever after their paths end. A *collision* between the paths of agents a_i and a_j is either a vertex collision $\langle a_i, a_j, v, t\rangle$, i.e., agents a_i and a_j are at the same location v at the same timestep t, or an edge collision $\langle a_i, a_j, u, v, t\rangle$, i.e., agent a_i moves from location u to location v and agent a_j moves from location v to location u at the same timestep t. A solution is a set of M paths $\{\pi_i | i = 1, \ldots, M\}$, one for each agent, such that no two paths collide. The *makespan* of a solution is the maximum length of all paths in the solution, i.e., $\max_{1 \le i \le M} |\pi_i|$.

Formation. The notion of a formation captures the relative locations between agents and can be specified by the coordinates of all agents. The *formation* at timestep t is an M-tuple $\ell(t) = \langle \boldsymbol{\pi}_\mathbf{1}(\mathbf{t}), \ldots, \boldsymbol{\pi}_\mathbf{M}(\mathbf{t})\rangle$ specified by the coordinates of the locations of all agents at timestep t. The *desired formation* is an M-tuple $\ell_g = \langle \mathbf{g_1}, \ldots \mathbf{g_M}\rangle$ specified by the coordinates of the goal locations of all agents. The *formation distance* $\mathscr{F}(\ell, \ell')$ between two formations ℓ and ℓ' characterizes the least effort needed to transform formation ℓ to formation ℓ'. It is defined as the sum of the L_1-distances over all agents between the two locations of the same agent after applying any translation $\mathbf{\Delta x}$ to ℓ', minimized over all such translations. Formally, let $\ell = \langle \mathbf{u_1}, \ldots \mathbf{u_M}\rangle$ and $\ell' = \langle \mathbf{v_1}, \ldots \mathbf{v_M}\rangle$. Then,

$$\begin{aligned}\mathscr{F}(\ell, \ell') &= \min_{\mathbf{\Delta x}} \sum_{i=1}^{M} ||\mathbf{u_i} - (\mathbf{v_i} + \mathbf{\Delta x})||_1 \\ &= \sum_{j=1}^{d} \min_{\mathbf{\Delta x}_j} \sum_{i=1}^{M} |(\mathbf{u_{i}}_j - \mathbf{v_{i}}_j) - \mathbf{\Delta x}_j| \\ &= \sum_{j=1}^{d} \sum_{i=1}^{M} |(\mathbf{u_{i}}_j - \mathbf{v_{i}}_j) - \mathbf{\Delta x}^*_j|, \qquad (1)\end{aligned}$$

where, for each dimension j, $\mathbf{\Delta x}^*_j$ is the median of all differences $\mathbf{u_i}_j - \mathbf{v_i}_j$.[1] The *formation deviation* $\mathscr{F}(t)$ at timestep t is the formation distance between the formation $\ell(t)$ at timestep t and the desired formation ℓ_g, i.e., $\mathscr{F}(t) = \mathscr{F}(\ell(t), \ell_g)$. The *total formation deviation* is the sum of the formation deviations over all timesteps, i.e., $\sum_{t=0}^{T} \mathscr{F}(t)$, where $T = \max_{1 \le i \le M} |\pi_i|$. Figure 1 (left) shows an example: The differences between the coordinates of the start and goal locations for every agent in every dimension $\{\mathbf{s_{i1}} - \mathbf{g_{i1}} | i = 1, 2, 3\}$ and $\{\mathbf{s_{i2}} - \mathbf{g_{i2}} | i = 1, 2, 3\}$ are $\{3, 4, 3\}$ and $\{-2, -1, 2\}$, respectively. Therefore, at timestep 0, the optimal translation is $\mathbf{\Delta x}^* = (3, -1)$, and the formation deviation at timestep 0 is thus $\mathscr{F}(0) = 5$.

Objective. The quality of a MAiF solution $\{\pi_i | i = 1, \ldots, M\}$ is evaluated by both its makespan and its total formation deviation. It is not easy to minimize both metrics simultaneously or any algebraic combination of them. However, minimizing only one of them is insufficient. MAPF algorithms can produce MAiF solutions that minimize the makespan but do not attempt to maintain the desired formation, while the dummy solution of keeping all agents idle in the desired formation minimizes the total formation deviation but prevents the agents from reaching their goal locations.

3 LEADER-FOLLOWER CONTROLLER

The leader-follower controller is a classic class of algorithms in formation control which has been widely studied and applied in robotics. As the name implies, the leader-follower controller selects one of the agents as the leader and treats the other agents as the followers. The leader tries to find a path toward its goal location, while each follower follows the leader by keeping its relative location to the leader in close adherence to the one in the desired formation.

We adapt a simple leader-follower controller to MAiF. It first arbitrarily chooses an agent a^* as the leader and plans a shortest path π^* for it by ignoring the other agents. At each timestep t, the leader a^* moves one step along its planned path π^*. Once the next location $\pi^*(t+1)$ of the leader a^* is determined, the followers (i.e.,

[1]We use the fact that the median minimizes the sum of the absolute differences to M numbers. The subgradient of the L_1-norm is the L_1-norm $\frac{d|x|}{dx} = \text{sign}(x)$. Therefore, differentiating $\sum_{i=1}^{M} |(\mathbf{u_i}_j - \mathbf{v_i}_j) - \mathbf{\Delta x}_j|$ with respect to $\mathbf{\Delta x}_j$ yields $\sum_{i=1}^{M} -\text{sign}((\mathbf{u_i}_j - \mathbf{v_i}_j) - \mathbf{\Delta x}_j)$, which is 0 only if the number of positive items equals the number of negative items. This happens when $\mathbf{\Delta x}_j$ is the median of all differences $\mathbf{u_i}_j - \mathbf{v_i}_j$.

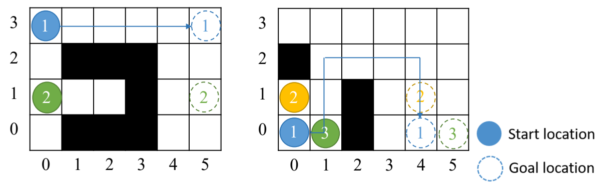

Figure 2: Two MAiF instances on a 4-neighbor grid. The black cells indicate obstacles. The first dimension is horizontal, and the second dimension is vertical. In each figure, the blue line shows the path of the leader.

all agents except for agent a^*) try to move toward their desired locations that are determined relative to the leader in the formation. To be more specific, the next location of follower a_i is

$$\pi_i(t+1) = \arg\min_{v \in S} ||(\mathbf{v} - \boldsymbol{\pi}^*(\mathbf{t}+\mathbf{1})) - (\mathbf{g_i} - \mathbf{g}^*)||_1, \tag{2}$$

where g^* is the goal location of agent a^* and $S \subseteq V$ consists of the current location $\pi_i(t)$ of agent a_i and its adjacent locations.

The leader-follower controller is not complete. Figure 2 shows two examples. Assume that agent a_1 is selected as the leader in both examples. In Figure 2 (left), agent a_2 moves one step toward the right following agent a_1 at every timestep but gets stuck at location $(2, 1)$ after agent a_1 arrives at location $(2, 3)$ at timestep 2. While agent a_1 continues moving toward its goal location, agent a_2 always waits at location $(2, 1)$ because this location is closest to its desired location. In Figure 2 (right), agent a_1 moves from location $(0, 0)$ to location $(1, 0)$ at timestep 0, following its planned path. If we assign a move to agent a_2 before agent a_3, agent a_2 moves to location $(1, 1)$ in order to stay in formation. Then, agent a_3 has nowhere to move.

4 JOINT-STATE A*

Joint-state A* [6] is a straightforward algorithm for solving MAPF optimally. It applies A* in an M-agent joint state space. The states assign each of the M agents a different location. Agent a_i is assigned its start location s_i and it goal location g_i, respectively, in the start and goal states. The operators assign each of the agents a non-colliding move or wait action.

We adapt joint-state A* to MAiF by considering both the makespan and the total formation deviation when computing the g-value and admissible h-value of each node in the A* search tree. For a node n at timestep t, the g-value for the makespan is

$$C_{mg}(n) = t, \tag{3}$$

while the g-value for the total formation deviation is

$$C_{fg}(n) = \sum_{k=0}^{t} \mathscr{F}(k). \tag{4}$$

We design an admissible h-value $C_{mh}(n)$ for the makespan as the minimum number of timesteps needed for all agents to reach their goal locations, which can be estimated as the maximum of the distances from the current location $\boldsymbol{\pi}_i(t)$ of each agent a_i to its goal location g_i, i.e.,

$$C_{mh}(n) = \max_{1 \le i \le M} \text{dist}(\pi_i(t), g_i), \tag{5}$$

where dist(u, v) is the distance between locations u and v on graph G. We also design an admissible h-value $C_{fh}(n)$ for the total formation deviation as the sum of the formation deviations after timestep t under the best-case assumption that all agents move one step toward the desired formation at every timestep. For example, if graph G is a 4-neighbor grid with unit-sized cells, the best circumstances are that the formation deviation decreases by M with every timestep, i.e.,

$$C_{fh}(n) = \sum_{j=1}^{\lfloor \mathscr{F}(t)/M \rfloor} (\mathscr{F}(t) - M \cdot j). \tag{6}$$

Hence, there are two f-values for each node n, namely a makespan f-value $C_m(n) = C_{mg}(n) + C_{mh}(n)$ and a total formation deviation f-value $C_f(n) = C_{fg}(n) + C_{fh}(n)$.

We use the ϵ-constraint algorithm [8] to obtain the Pareto frontier of this bi-objective optimization problem. In particular, we replace A* with focal search [20]. Like A*, the focal search uses an OPEN list to prioritize its nodes n in increasing order of their makespan f-values $C_m(n)$. Unlike A*, the focal search also uses a FOCAL list that consists of all nodes currently in the OPEN list whose makespan f-values are no larger than $\epsilon \min_{n \in \text{OPEN}} C_m(n)$, where $\epsilon \ge 1$ is a user-provided parameter. The focal list prioritizes its nodes n in increasing order of their total formation deviation f-values $C_f(n)$. At each iteration, the focal search expands the node in the FOCAL list (rather than the OPEN list) with the minimum total formation deviation f-value. The focal search is guaranteed to find a solution whose makespan C_m is at most ϵ times larger than the minimum makespan and whose total formation deviation is the minimum total formation deviation among all solutions with the same makespan C_m. We can obtain some points on the Pareto frontier by varying ϵ.

Joint-state A* is complete and Pareto optimal. However, due to the exponential state space, joint-state A* is extremely inefficient. Many techniques exist to speed up joint-state A* for MAPF, such as independence detection [25], operator decomposition [25], partial expansions [7], and sub-dimensional expansions [29]. However, most of these techniques are based on the assumption that the objective can be decomposed into M independent cost functions, one for each agent. But agents in MAiF have a joint cost function where the total formation deviation relies on the relative locations of all agents. Thus, most of these techniques do not apply to MAiF, and we add only operator decomposition to joint-state A*. But, even after adding this technique, joint-state A* still scales poorly in the number of agents, as shown in the experimental section.

5 SWARM-MAPF

In this section, we present our novel two-phase MAiF algorithm, called SWARM-MAPF, which leverages ideas from both formation control and MAPF, and is complete and efficient. It tackles the combined task of MAiF by dividing the problem into sub-problems and the planning into two phases. In the first phase, the *leader* is chosen to be the agent with the smallest number of timesteps when the desired formation has to be compromised. The other agents are the *followers*. Then, the path of the leader is divided into segments of two types. For segments where all agents can move in the desired formation (e.g., in open regions), SWARM-MAPF uses a swarm-based algorithm. For segments where the desired formation has to be compromised (e.g., in regions with obstacles), SWARM-MAPF uses a novel hierarchical MAPF-based algorithm, called CBS-M, to move the agents around the obstacles as quickly as possible while still trying to keep the total formation deviation small. CBS-M can also be used as a stand-alone MAiF algorithm that minimizes the makespan only but biases its searches toward solutions with small total formation deviation. SWARM-MAPF is not guaranteed to provide optimal solutions for either of the two objectives. But we demonstrate experimentally that it often produces solutions that keep all agents in close adherence to the desired formation with only a small loss of optimality in the makespan.

5.1 Phase I: Leader Path Generation

In Phase I, SWARM-MAPF chooses a leader among all agents and partitions its path into open and congested segments. In each open segment, the agents form the desired formation and follow the leader, while, in each congested segment, their paths are planed in Phase II.

5.1.1 Choosing the Leader and Its Path. A *formation-blocking* location for an agent is one where the desired formation cannot be kept by the remaining agents when the agent is at it. The *total formation-blocking value* of a path is the number of formation-blocking locations on it. In Phase I, SWARM-MAPF chooses the agent a^* as the leader whose path minimizes the total formation-blocking value among all paths for all agents of lengths no larger than wB, where $w \geq 1$ is a user-provided parameter and B is a lower bound on the makespan. In this paper, SWARM-MAPF uses $B = \max_{1 \leq i \leq M} \text{dist}(s_i, g_i)$.

To do so, SWARM-MAPF performs a best-first search for each agent a_i to find a path π_i with the minimum total formation-blocking value subject to the constraint that the path length is no larger than wB. It breaks ties in favor of shorter paths. Then, SWARM-MAPF chooses the path with the minimum total formation-blocking value among the paths of all agents as the leader's path π^* and the corresponding agent as the leader a^*. It breaks ties in favor of shorter paths.

5.1.2 Partitioning the Path of the Leader into Segments. Once the leader a^* has been chosen, its path π^* is partitioned into *open segments* and *congested segments* alternately. Each open segment is a maximum segment $[\pi^*(t_b), \pi^*(t_e)]$ $(t_b \leq t_e)$ such that, for all $t_b \leq t \leq t_e$, location $\pi^*(t)$ is not formation-blocking. Each remaining segment is a congested segment.

Assume that there are K open segments. K should be at least 1 because the goal location of the leader is not formation-blocking. Let $p^*_1, \ldots, p^*_{2K}$ denote the first and last locations of all open segments, i.e., the k-th open segment is $[p^*_{2k-1}, p^*_{2k}]$. Let p^*_0 denote the start location of the leader. p^*_0 and p^*_1 are identical iff the start locations of all agents are in the desired formation. There are also K congested segments, and the k-th congested segment is $[p^*_{2k-2}, p^*_{2k-1}]$.

Let ℓ^*_0 denote the M-tuple of the start locations of all agents and ℓ^*_k $(1 \leq k \leq 2K)$ denote the M-tuple of the locations of all agents that form the desired formation around location p^*_k of the leader. Each open segment specifies a sub-MAiF instance where all agents need to move from locations ℓ^*_{2k-1} to locations ℓ^*_{2k}. SWARM-MAPF obtains a sub-solution for each such sub-MAiF instance for free since all agents move in the desired formation along the path segment $[p^*_{2k-1}, p^*_{2k}]$ of the leader. In each congested segment $[p^*_{2k-2}, p^*_{2k-1}]$, any location except for locations p^*_{2k-2} and p^*_{2k-1} is formation-blocking. Each congested segment specifies a sub-MAiF instance where all agents need to move from the *intermediate start locations* ℓ^*_{2k-2} to the *intermediate goal locations* ℓ^*_{2k-1}. SWARM-MAPF obtains a sub-solution for each such sub-MAiF instance in Phase II. Finally, SWARM-MAPF concatenates the sub-solutions for all sub-MAiF instances of both types of segments to obtain a solution for the overall MAiF instance.

Figure 1 (right) demonstrates the generation of the leader's path for the running example. SWARM-MAPF determines the lower bound B on the makespan to be 5. No matter what the user-provided parameter w is, SWARM-MAPF always finds a path of length 5 for agent a_2 and chooses it to be the leader because agent a_2 has a path of length 5 with total formation-blocking value 0. The green line in the figure is such a path. There are two segments, namely a congested segment from the (intermediate) start locations $\ell^*_0 = \langle(3,1),(5,1),(4,3)\rangle$ to the intermediate goal locations $\ell^*_1 = \langle(4,2),(5,1),(5,0)\rangle$ and an open segment from locations ℓ^*_1 to the goal locations $\ell^*_2 = \langle(0,3),(1,2),(1,1)$. The solution for the overall MAiF instance is a concatenation of the sub-solutions to the sub-MAiF instances resulting from both segments.

5.2 Phase II: CBS-M for Congested Segments

In Phase II, SWARM-MAPF uses a MAPF algorithm to compute paths for the sub-MAiF instances resulting from each congested segment that move the agents from the given intermediate start locations to the given intermediate goal locations. For ease of exposition, in the description of Phase II, we let s_i and g_i denote the given (intermediate) start and goal locations of agent a_i, respectively. We develop a new variant of Conflict-Based Search (CBS) [22], called CBS-M, that minimizes the makespan and incentivizes the agents to keep close adherence to the desired formation.

5.2.1 CBS. The original CBS [22] is a two-level MAPF algorithm that minimizes the *flowtime*, i.e., the sum of the lengths of the paths for all agents. On the high level, CBS performs a best-first search on a *constraint tree (CT)* to resolve collisions among the agents. Each CT node N contains a set of spatio-temporal constraints ($N.constraints$), paths for all agents ($N.paths$) that obey $N.constraints$ but might result in collisions, and a cost ($N.cost$) that equals the flowtime of the paths in $N.paths$. CBS always expands the CT node with the smallest cost. The root CT node has no constraints and contains shortest paths on graph G (not considering the other agents) for all agents. When CBS expands a CT node N, it checks whether the paths in $N.paths$ are collision-free. If so, then N is a goal CT node, and CBS terminates successfully. Otherwise, CBS chooses a collision to resolve and generates two child CT nodes N_1 and N_2 of N that inherit $N.constraints$ and $N.paths$. If the collision to resolve is a vertex collision $\langle a_i, a_j, v, t\rangle$, then CBS adds the vertex constraint $\langle a_i, v, t\rangle$ to $N_1.constraints$ to prohibit agent a_i from being at location v at timestep t and similarly adds the vertex constraint $\langle a_j, v, t\rangle$ to $N_2.constraints$. If the collision to resolve is an edge collision $\langle a_i, a_j, u, v, t\rangle$, then CBS adds the edge constraint $\langle a_i, u, v, t\rangle$ to N_1.*constraints* to prohibit agent a_i from moving from location u to location v at timestep t and similarly adds the edge constraint $\langle a_j, v, u, t\rangle$ to N_2.*constraints*.

For each child CT node, say N_1, CBS performs a low-level A* search to compute a new shortest path for agent a_i that obeys the constraints in N_1.*constraints* relevant to agent a_i. Each low-level node (in the search tree of A*) contains a location v and a timestep t, representing the agent being at location v at timestep t. The A* search terminates when the expanded node satisfies the following two conditions: (a) its location is location g_i, and (b) its timestep is greater than or equal to the latest timestep of any constraint relevant to agent a_i (to ensure that agent a_i can wait at its goal location forever).

5.2.2 *CBS-M.*

CBS-M is similar to CBS except that its objective is to minimize the makespan. In fact, Ma and Koenig [16] have already developed a variant of CBS that minimizes the makespan by simply using the makespan, instead of the flowtime, as the cost of each CT node. Since, in general, there are many combinations of paths for all agents that result in the same makespan, CBS-M exploits this leeway to speed up both its high- and low-level searches while trying to maintain the desired formation.

On the high level, CBS-M uses the makespan of the paths in N.*paths* as the cost N.*cost* of a CT node N. Each CT node N also keeps track of the total formation deviation of these paths. The high-level search always expands the CT node with the smallest cost. The primary tie-breaking criterion is to favor the CT node N with the smallest number of pairs of colliding paths in N.*paths*, which has empirically been shown to speed up the high-level search [2]. The secondary tie-breaking criterion is to favor the CT node with the smallest total formation deviation, which incentivizes the agents to adhere closely to the desired formation.

When a child CT node N_1 is generated from its parent CT node N with a new constraint for agent a_i, CBS-M performs a low-level search to compute a path for agent a_i that obeys the constraints in N_1.*constraints* relevant to agent a_i and closely adheres to the desired formation. On the low level, CBS-M uses a focal search, instead of an A* search, to find paths for agents, which allows it to find (non-shortest) paths that result in smaller total formation deviations than the shortest paths. The focal search uses an OPEN list that prioritizes all low-level nodes in increasing order of their f-values (which correspond to the lengths of the paths from the start location s_i of agent a_i to the locations of the low-level nodes) and a FOCAL list of all low-level nodes currently in the OPEN list whose f-values are no larger than the focal bound $\max\{N.cost, \min_{n\in\text{OPEN}} f(n)\}$. For the root CT node, that has no parent CT node, the low-level search simply uses an A* search. Besides the location and the timestep, each low-level node also keeps track of the sum of the formation deviations (with respect to the paths of the other agents in N_1.*paths*) along the path from location s_i to the location of the low-level node. The low-level search always expands the low-level node n from the FOCAL list whose path has the fewest collisions with the paths of the other agents in N_1.*paths*, which has empirically been shown to speed up the high-level search [2]. The tie-breaking criterion is to favor the low-level node with the smallest sum of the formation deviations. Compared to the low-level A* search described in Section 5.2.1, the termination criterion of the low-level focal search has a third condition, namely that the timestep of the expanded low-level node is at least N.*cost*. This condition is important because it allows the low-level search to take into account the collisions and formation deviations for all timesteps, including the timesteps after the agent reaches its goal location g_i.

Lemma 1. *The cost* N.cost *of each CT node* N *is at most the makespan of any solution whose paths obey the constraints in* N.constraints.

Proof. We prove by induction the statement that N.*cost* of each CT node N is at most the makespan of **any set of paths** (possibly with collisions) of all agents that obey N.*constraints*. The lemma then holds because any solution whose paths obey N.*constraints* is a set of paths of all agents that obey N.*constraints*. The root CT node R contains a shortest path for each agent, and the statement therefore holds for R. Assume that the statement holds for a parent CT node N of any child CT node N'. Compared to N.*paths*, CBS-M changes the path for one agent only, say agent a_i, in N'.*paths* by performing a low-level search with the constraints N'.*constraints*.

(1) If the length of the path returned by the low-level search is at most N.*cost*, then the resulting N'.*cost* is at most N.*cost*, which is at most the makespan of any set of paths of all agents that obey N.*constraints* according to the induction assumption. In turn, this makespan is at most the makespan of any set of paths of all agents that obey N'.*constraints* since N'.*constraints* is a super set of N.*constraints*. (2) Otherwise, the length of the path returned by the low-level search is greater than N.*cost*, and this path is a shortest path for agent a_i that obeys N'.*constraints*. The resulting N'.*cost* is equal to the length of this shortest path, which is at most the makespan of any set of paths of all agents that obey N'.*constraints*. Therefore, the statement holds for CT node N' in both cases. □

In order to guarantee completeness, CBS-M runs an efficient MAPF algorithm, such as the one in [34], as a first step to determine the solvability of the sub-MAiF instance and return an upper bound on the makespan. If the instance is solvable, CBS-M always returns a solution. Otherwise, it reports that no solution exists once it expands a CT node with cost larger than the upper bound. The proof of completeness of CBS-M is exactly the same as the one in [22]. The proof of optimality of CBS-M is also the same as the one in [22], except that it is adapted to makespan minimization and uses Lemma 1. Therefore, the following theorem holds:

Theorem 2. *CBS-M is complete for all MAiF instances and minimizes the makespan.*

5.3 Iterative Goal Updating for Completeness

In Phase I, SWARM-MAPF does not determine the solvability of each sub-MAiF instance, so the completeness of SWARM-MAPF requires an additional technique, called *iterative goal updating*: In Phase II, SWARM-MAPF calls CBS-M to solve the sub-MAiF instances resulting from the congested segments, one by one, from the first congested segment to the last congested segment. If CBS-M reports that no sub-solution exists for the k-th sub-MAiF instance with the intermediate start and goal locations ℓ^*_{2k-2} and ℓ^*_{2k-1}, then location p^*_{2k-1} is regarded as a formation-blocking location.

SWARM-MAPF returns to Phase I, re-partitions the path π^* after location p^*_{2k-2} and updates ℓ^*_k accordingly. SWARM-MAPF iteratively calls CBS-M to solve the new sub-MAiF instance with the intermediate start locations ℓ^*_{2k-2} and the updated intermediate goal locations $(\ell^*_{2k-1})'$, until (1) CBS-M returns a sub-solution or (2) the locations $(\ell^*_{2k-1})'$ are the goal locations and CBS-M reports that no sub-solution exists. In case (1), SWARM-MAPF saves the computed sub-solution from ℓ^*_{2k-2} to $(\ell^*_{2k-1})'$ and proceeds to the next congested segment of the re-partitioned path. In case (2), SWARM-MAPF reports that no solution exists for the overall MAiF instance.

Theorem 3. *SWARM-MAPF with iterative goal updating is complete for all MAiF instances.*

Proof. After CBS-M is called to solve a sub-MAiF instance, it either (1) returns a solution, and SWARM-MAPF thus proceeds to the next congested segment, or (2) does not return a solution, and SWARM-MAPF thus updates the intermediate goal locations and attempts to solve the resulting new sub-MAiF instance. In both cases, SWARM-MAPF does not terminate until it attempts to solve the sub-MAiF instance resulting from the last congested segment that moves the agents from some intermediate start locations ℓ^*_{2k-2} to the goal locations ℓ_g. We now argue that SWARM-MAPF either (1) returns a solution if there exists a solution to the overall MAiF instance or (2) correctly reports that no solution exists.

Consider (1), where there exists a solution to the overall MAiF instance that moves all agents from their start locations ℓ_s to their goal locations ℓ_g. We first argue that there must exist a sub-solution to the sub-MAiF instance resulting from the last congested segment with the intermediate start locations ℓ^*_{2k-2} and the goal locations ℓ_g. Concatenating the computed sub-solutions up to the intermediate start locations ℓ^*_{2k-2} produces collision-free paths that move all agents from the locations ℓ_s to the locations ℓ^*_{2k-2}. Reversing all actions produces collision-free paths that move them from the locations ℓ^*_{2k-2} to the locations ℓ_s. By assumption, we also know that there exist collision-free paths that move all agents from the locations ℓ_s to the locations ℓ_g. Therefore, there exists a solution that moves all agents from the locations ℓ^*_{2k-2} to the locations ℓ_g (namely, via the locations ℓ_s). Since CBS-M is complete, it must return a sub-solution to the last sub-MAiF instance. This sub-solution appended to the concatenation of all computed sub-solutions up to ℓ^*_{2k-2} produces a solution to the overall MAiF instance.

Now consider (2), where no solution exists to the overall MAiF instance. There exists no sub-solution to the sub-MAiF instance resulting from the last congested segment with the intermediate start locations ℓ^*_{2k-2} and the goal locations ℓ_g either, since, otherwise, this solution appended to the concatenation of the computed sub-solutions up to ℓ^*_{2k-2} would produce a contradictory solution to the overall MAiF instance. Since CBS-M is complete, it must report that no solution exists to the this sub-MAiF instance, and, consequently, SWARM-MAPF must correctly report that no solution exists to the MAiF instance. □

5.4 Limitation of SWARM-MAPF

We now show that, in extreme cases, SWARM-MAPF computes solutions with large makespans. Figure 3 shows a conceptual example

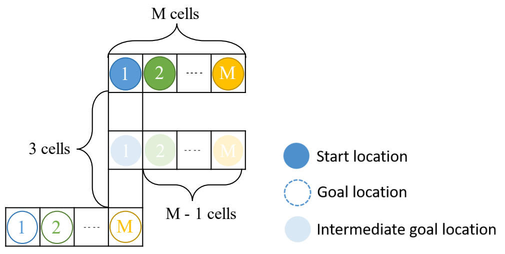

Figure 3: A MAiF instance on a 4-neighbor grid.

where M agents need to travel through a narrow vertical passageway of length 3 with one horizontal passageway of length $M-1$. The minimum makespan is $M+3$, resulting from the agents moving through the vertical passageway one after another. SWARM-MAPF chooses agent a_1 as the leader because its location m_1 in the middle of the vertical passageway (indicated by the blue circle labeled "1" in a lighter shade) is not formation-blocking and all agents are planned to restore the desired formation along the horizontal passageway in the middle of the figure. Therefore, the leader's path is partitioned into 5 segments, namely an open segment $[s_1, s_1]$, a congested segment $[s_1, m_1]$, an open segment $[m_1, m_1]$, a congested segment $[m_1, g_1]$ and an open segment $[g_1, g_1]$. The makespans of the sub-solutions for all open segments are 0, while the makespans of the sub-solutions for both congested segments are determined by the lengths of the shortest paths of agent a_M from its (intermediate) start locations to its (intermediate) goal locations. That is, the makespans of the sub-solutions for the congested segments are $2M$ and $M+1$, respectively. Therefore, the makespan of the solution for the overall MAiF instance is $3M+1$.

We now generalize the example to M agents moving through a narrow vertical passageway of length $2C+1$ with C horizontal passageways of length $M-1$ each. The minimum makespan is $M+2C+1$, resulting from the agents moving through the vertical passageway one after another. SWARM-MAPF still chooses agent a_1 as the leader and partitions its path into $C+2$ open segments $[m_k, m_k]$ $(0 \le k \le C+1)$ and $C+1$ congested segments $[m_k, m_{k+1}]$ $(0 \le k \le C)$, where m_0 is the start location s_1 of agent a_1, m_{C+1} is the goal location g_1 of agent a_1, and m_k $(1 \le k \le C)$ is the location of agent a_1 in the k-th horizontal passageway. The makespans of the sub-solutions for all open segments are 0, while the makespans of the sub-solutions for all congested segments are determined by the lengths of the shortest paths of agent a_M from its (intermediate) start location to its (intermediate) goal location, i.e., $2M$ for each of the first $C+1$ congested segments and $M+1$ for the last congested segment. Therefore, the makespan of the solution for the overall MAiF instance is $2M(C+1)+M+1$, which is much larger than the minimum makespan $M+2C+1$ for large C and M.

6 EXPERIMENTS

In this section, we describe our experimental results on a 2.2 GHz Intel Core i5-5200 laptop with 4 GB RAM. We tested two algorithms. The first one (labeled **CBS**) calls CBS-M to solve the entire MAiF instance, which produces the optimal makespan and uses the total formation deviation to break ties between search nodes. The second

Table 1: Results for SWARM-MAPF for different user-provided parameters w and CBS-M for MAiF instances with 10 agents. "Leader path length" represents the length of the leader's path computed in Phase I. "Leader path subopt." represents the suboptimality ratio of the leader's path compared to the makespan lower bound B, i.e., the largest distance between all pairs of start and goal locations. "Leader form. block." represents the total formation-blocking value of the leader's path. "CBS calls" represents the number of calls to CBS-M. "Makespan subopt." represents the suboptimality ratio of the makespan compared to the minimum makespan produced by CBS-M. "Total form. dev." represents the total formation deviation.

w (SW only)	leader path length	leader path subopt.	leader form. block.	CBS calls	make-span	makespan subopt.	total form. dev.	run-time (s)
CBS	-	-	-	1.00	54.00	1.000	102.48	0.03
1.00	44.00	1.00	8.42	6.65	56.46	1.046	57.46	0.16
1.05	44.58	1.01	8.10	6.39	56.55	1.047	55.38	0.20
1.10	44.94	1.02	7.99	6.32	56.78	1.051	55.32	0.24
1.15	45.02	1.02	7.97	6.24	56.57	1.048	54.26	0.29
1.20	45.10	1.03	7.96	6.29	56.60	1.048	54.24	0.34
1.25	45.10	1.03	7.96	6.24	56.71	1.050	54.00	0.41
1.30	45.10	1.03	7.96	6.30	56.91	1.054	54.87	0.46
1.35	45.10	1.03	7.96	6.25	56.81	1.052	54.82	0.52
1.40	45.10	1.03	7.96	6.30	56.70	1.050	54.52	0.58
1.45	45.10	1.03	7.96	6.30	56.69	1.05	54.57	0.63
1.50	45.10	1.03	7.96	6.29	56.91	1.05	55.24	0.73

one is SWARM-MAPF (labeled **SW**), which better balances between the two objectives. Each CBS-M call is given a runtime limit of 5 minutes. We also compare them with two baseline algorithms, namely the leader-follower controller (described in Section 3) and joint-state A* (described in Section 4).

6.1 Experiment 1: User-Provided Parameter

We use 10 30×30 4-neighbor grids. For each grid, the top-left 8×8 cells are possible start locations, and the bottom-right 8×8 cells are possible goal locations. These 128 cells are unblocked. The other cells are blocked with 10% probability uniformly at random. We generate 10 different formations for 10 agents by placing them uniformly at random in an 8×8 boundary box of cells. We use them as the desired formations to generate MAiF instances. For each of the 10 grids, we put each of the 10 8×8 boxes both in the top-left corner as the start locations of the agents and the bottom-right corner as their goal locations, which results in 100 MAiF instances where the start locations are in the desired formations. A visualization of such an instance is shown in Figure 5 (left).

We vary the user-provided parameter w from 1.00 to 1.50 and report the average results in Table 1. Both algorithms solve all instances within the runtime limit. For SWARM-MAPF, as the user-provided parameter w increases, the length and the suboptimality ratio of the leader's path tend to increase, and its total formation-blocking value tends to decrease. The tendency stops for w larger than 1.20, where SWARM-MAPF always finds a leader's path with the smallest total formation-blocking value for each instance. For large w, the number of calls to CBS-M and the total formation deviation tend to be small, and the makespan tends to be large. But the tendency is not consistent for w larger than 1.20. As w increases, the runtime of SWARM-MAPF increases. Overall, running CBS-M directly on the MAiF instances produces minimum makespans and results in the smallest runtimes. SWARM-MAPF, on the other hand, does a better job at keeping the agents in close adherence to the desired formation than CBS-M while only slight increasing the makespan.

Table 2: Results for SWARM-MAPF with user-provided parameter $w = 1$ and CBS-M for MAiF instances with different numbers of agents M (top) and different desired formations (bottom). "Success" represents the success rates. All numbers, except for the ones in the "success" columns, are averaged over the solved instances and reported in the same way as in Table 1.

M	leader path length	leader form. block.	CBS calls	success		makespan		make-span subopt.	total form. dev.		runtime (s)	
	SW only			SW	CBS	SW	CBS	SW	SW	CBS	SW	CBS
5	44	2.19	2.08	1.00	1.00	48.23	44	1.10	8.62	63.96	0.03	0.02
10	44	8.42	6.65	1.00	1.00	56.46	44	1.28	57.46	161.84	0.16	0.04
15	44	14.18	8.02	1.00	1.00	58.53	44	1.33	144.04	522.64	0.29	0.16
20	44	19.79	8.61	1.00	1.00	59.13	44	1.34	266.81	680.96	0.42	0.44
25	44	24.96	7.94	0.99	1.00	57.71	44	1.31	448.08	1,189.70	0.69	1.96
30	44	27.60	6.75	0.91	0.95	55.08	44	1.25	696.48	1,797.81	0.75	0.50
35	44	30.43	5.69	0.83	0.93	53.01	44	1.20	1,044.20	2,212.32	2.30	2.84
40	44	32.77	4.95	0.65	0.86	51.54	44	1.17	1,571.62	3,330.79	10.43	10.62

form. box	start in form.?	leader path length	leader form. block.	CBS calls	makespan		makespan subopt.	total form. dev.		runtime (s)	
		SW only			SW	CBS	SW	SW	CBS	SW	CBS
3×3	yes	54.00	0.90	0.88	55.35	54.00	1.03	4.36	102.48	0.02	0.02
3×3	no	52.26	0.92	1.92	58.08	55.70	1.04	24.68	131.84	0.04	0.05
5×5	yes	50.00	1.39	1.31	52.28	50.00	1.05	6.01	81.26	0.02	0.02
5×5	no	48.16	1.29	2.13	57.12	53.00	1.08	45.32	119.72	0.05	0.05
7×7	yes	46.00	1.64	1.56	49.20	46.00	1.07	6.74	57.86	0.02	0.02
7×7	no	44.91	1.74	2.55	57.36	50.20	1.14	73.42	144.89	0.06	0.06

6.2 Experiment 2: Number of Agents

We repeat Experiment 1 but vary the number of agents from 5 to 40 for user-provided parameter $w = 1.00$. For each number of agents, we generate 100 instances in the same way as above.

Since not all instances are solved within the runtime limit, Table 2 (top) reports the *success rates* (i.e., the percentages of solved instances within the runtime limit) and the same statistics as Table 1 but over the solved instances only. As the number of agents M increases, the total formation-blocking value, the total formation deviation, and the runtime tend to increase. The number of calls to CBS-M and the makespan reach their peak values for 20 agents. The success rate of SWARM-MAPF is larger than 90% for ≤ 30 agents, while the success rate of CBS-M is larger than 90% for ≤ 35 agents. Overall, CBS-M scales to slightly larger numbers of agents than SWARM-MAPF. Both algorithms can compute solutions for ≤ 20 agents in real-time (< 1s) on average.

6.3 Experiment 3: Desired Formations

We repeat Experiment 2 but vary the desired formations of 5 agents, which are randomly generated from boundary boxes of 3 different sizes: 3×3, 5×5, and 7×7. For each box size, we generate two groups of instances: 100 instances with the start locations being in the desired formation, and 100 instances with the start locations being generated randomly (i.e., not in the desired formation).

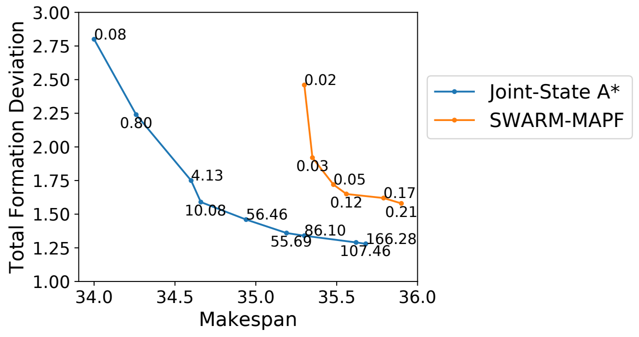

Figure 4: Trade-off between the makespan and the total formation deviation for MAiF instances with 3 agents on 20×20 4-neighbor grids. The number next to each point on the graphs represents the corresponding runtime (in seconds). The graph of joint-state A^* represents the Pareto frontier.

Table 2 (bottom) reports the results. Both algorithms solve all instances within the runtime limit. Larger box sizes allow for formations that are more spread out, which are harder to keep around obstacles. Therefore, large box sizes result in more calls to CBS-M and larger total formation deviations for SWARM-MAPF. Compared to the first group of instances, the second group results in larger total formation deviations, larger makespans, and larger runtimes, because it takes both algorithms extra effort to move the agents into the desired formation if they do not start in it.

6.4 Experiment 4: Baseline Algorithms

Experiment 4 compares SWARM-MAPF with the two baseline algorithms. Due to the limited scalability of joint-state A*, we reduce the size of the grids to 20×20 and the number of agents to 3. We randomly generate 10 grids and 10 formations in the same way as in Experiment 1, which results in 100 MAiF instances.

We first compare SWARM-MAPF with joint-state A* and vary the user-provided parameter ϵ of joint-state A* from 1.00 to 1.80 and the user-provided parameter w of SWARM-MAPF from 1.00 to 1.60. We use infinite runtime limits for both algorithms, and thus they both solve all MAiF instances. As the user-provided parameters ϵ and w increase, the makespan and runtime increase while the total formation deviation decreases. Figure 4 shows this trade-off. The graph of joint-state A* represents the Pareto frontier. Although not optimal, the solution quality of SWARM-MAPF is often similar to that of joint-state A*. For example, when the total formation deviation is around 1.58, the makespans of the solutions produced by SWARM-MAPF and joint-state A* are 35.90 and 34.66, respectively, which is a difference of only about 3.6%. However, SWARM-MAPF runs substantially faster than joint-state A*. For example, joint-state A* needs 10.08 seconds while SWARM-MAPF needs only 0.21 seconds to obtain a solution with a total formation deviation of around 1.58, which is a significant difference of about a factor of 50.

We also run the leader-follower controller on the same 100 MAiF instances. It does not solve 16 instances due to deadlocks. Although its average runtime is only about 0.6 milliseconds, the average makespan and average total formation deviation over the solutions of all solved instances are 38.65 and 97.36, respectively, which are substantially worse than those of SWARM-MAPF.

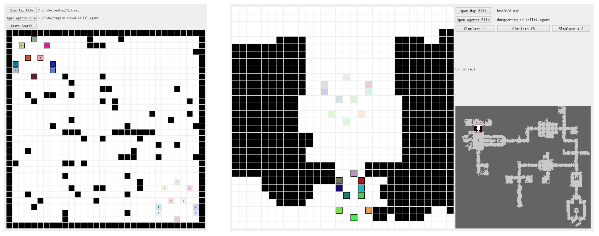

Figure 5: Screenshots of the videos.

6.5 Experiment 5: Visualizations

Videos of the execution of the solutions of SWARM-MAPF with user-provided parameter $w = 1$ are available here (**clickable**).[2] We visualize them on one of the 30×30 grids used in Experiment 1 (Figure 5 (left)) and also in a simulated framework developed in [18] based on the video game environment `brc202d` [27] from *Dragon Age: Origins* (Figure 5 (right)). For the 30×30 grid, the length of the leader's path is 46, the total formation-blocking value of the leader's path is 11, the makespan is 76, the total formation deviation is 137, and the runtime is 0.34 seconds. For the video game environment, a player moves 10 agents by occasionally specifying goal locations for them, for example, after observing new parts of the environment. Once the new goal locations are determined, SWARM-MAPF is called to solve the MAiF instance from the current locations of the agents to their newly specified goal locations. The following statistics sum over all calls to SWARM-MAPF and are reported for the narrow and wide desired formations, respectively. The makespans are 168 and 212, the total formation deviations are 312 and 2309, and the runtimes are 0.28 and 1.13 seconds.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we formalized and studied the Moving in Formation (MAiF) problem, that combines the tasks of finding short collision-free paths for multiple agents and keeping them in close adherence to a desired formation. We developed a two-phase search algorithm, called SWARM-MAPF, that combines swarm-based and MAPF-based algorithms. We developed a variant of a MAPF solver, called CBS-M, that minimizes the makespan, conducts a focal search for the secondary objective of minimizing the deviation from the desired formation, and can also be used as a stand-alone MAiF algorithm. We proved that SWARM-MAPF is complete and showed that it does a better job in trading off the minimization of the makespan against the minimization of the deviation from the desired formation than using CBS-M for the entire MAiF instance.

We suggest the following directions for future work: (1) generalize the formation distance metric to other problems, e.g., multi-robot motion planning; and (2) study MAiF in a fully-distributed setting, i.e., where agents need to communicate and coordinate their actions toward a global objective.

[2]http://idm-lab.org/formation

REFERENCES

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

[2] M. Barer, G. Sharon, R. Stern, and A. Felner. 2014. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *SoCS*. 19–27.

[3] T. D. Barfoot and C. M. Clark. 2004. Motion planning for formations of mobile robots. *Robotics and Autonomous Systems* 46, 2 (2004), 65–78.

[4] J. P. Desai, J. P. Ostrowski, and V. Kumar. 2001. Modeling and control of formations of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation* 17, 6 (2001), 905–908.

[5] E. Erdem, D. G. Kisa, U. Oztok, and P. Schueller. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI*. 290–296.

[6] A. Felner, R. Stern, S. E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. R. Sturtevant, G. Wagner, and P. Surynek. 2017. Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges. In *SoCS*. 29–37.

[7] M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. R. Sturtevant, R. C. Holte, and J. Schaeffer. 2014. Enhanced Partial Expansion A*. *Journal of Artificial Intelligence Research* 50 (2014), 141–187.

[8] Y. Haimes, L. Lasdon, and D. Wismer. 1971. On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-1, 3 (1971), 296–297.

[9] A. Jain, D. Ghose, and P P Menon. 2016. Achieving a desired collective centroid by a formation of agents moving in a controllable force field. In *Indian Control Conference*. 182–187.

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

[11] N. E. Leonard and E. Fiorelli. 2001. Virtual leaders, artificial potentials and coordinated control of groups. In *Decision and Control*, Vol. 3. 2968–2973.

[12] M. A. Lewis and K.-H. Tan. 1997. High Precision Formation Control of Mobile Robots Using Virtual Structures. *Autonomous Robots* 4, 4 (1997), 387–403.

[13] J.-M. Lien, O B. Bayazit, R. T Sowell, S. Rodriguez, and N. M Amato. 2004. Shepherding Behaviors. In *ICRA*, Vol. 4. 4159–4164.

[14] R. Luna and K. E. Bekris. 2011. Push and Swap: Fast Cooperative Path-finding with Completeness Guarantees. In *IJCAI*. 294–300.

[15] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig. 2019. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *AAAI*. 7643–7650.

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

[17] H. Ma, C. Tovey, G. Sharon, T. K. S. Kumar, and S. Koenig. 2016. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In *AAAI*. 3166–3173.

[18] H. Ma, J. Yang, L. Cohen, T. K. S. Kumar, and S. Koenig. 2017. Feasibility Study: Moving Non-Homogeneous Teams in Congested Video Game Environments. In *AIIDE*. 270–272.

[19] R. Morris, C. Pasareanu, K. Luckow, W. Malik, H. Ma, S. Kumar, and S. Koenig. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations. In *AAAI-16 Workshop on Planning for Hybrid Systems*.

[20] J. Pearl and J. H. Kim. 1982. Studies in Semi-Admissible Heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4, 4 (1982), 392–399.

[21] C. W. Reynolds. 1987. Flocks, Herds and Schools: A Distributed Behavioral Model. In *SIGGRAPH*. 25–34.

[22] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66.

[23] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence* 195 (2013), 470–495.

[24] D. Silver. 2005. Cooperative Pathfinding. In *AIIDE*. 117–122.

[25] T. S. Standley. 2010. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *AAAI*. 173–178.

[26] T. S. Standley and R. E. Korf. 2011. Complete Algorithms for Cooperative Pathfinding Problems. In *IJCAI*. 668–673.

[27] N. R. Sturtevant. 2012. Benchmarks for Grid-Based Pathfinding. *Transactions on Computational Intelligence and AI in Games* 4, 2 (2012), 144–148.

[28] P. Surynek. 2015. Reduced Time-Expansion Graphs and Goal Decomposition for Solving Cooperative Path Finding Sub-Optimally. In *IJCAI*. 1916–1922.

[29] G. Wagner and H. Choset. 2015. Subdimensional expansion for multirobot path planning. *Artificial Intelligence* 219 (2015), 1–24.

[30] K. Wang and A. Botea. 2011. MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *Journal of Artificial Intelligence Research* 42 (2011), 55–90.

[31] P. R. Wurman, R. D'Andrea, and M. Mountz. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine* 29, 1 (2008), 9–20.

[32] J. Yu and S. M. LaValle. 2013. Planning Optimal Paths for Multiple Robots on Graphs. In *ICRA*. 3612–3617.

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

[34] J. Yu and D. Rus. 2015. Pebble Motion on Graphs with Rotations: Efficient Feasibility Tests and Planning Algorithms. In *Algorithmic Foundations of Robotics XI, Springer Tracts in Advanced Robotics*. Vol. 107. Springer, 729–746.