Moving Agents in Formation in Congested Environments

Jiaoyang Li
University of Southern California
jiaoyanl@usc.edu

Kexuan Sun
University of Southern California
kexiansu@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
tkswork@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

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...
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 \( v \in V \) can be recognized by its coordinates \( v = (v_1, \ldots, v_d) \in \mathbb{R}^d \). 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 \( \pi_i \) 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 \( g_i \). \( \pi_i(t) \) 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 \((a_i, a_j, v, t)\), i.e., agents \( a_i \) and \( a_j \) are at the same location \( v \) at the same timestep \( t \), or an edge collision \((a_i, a_j, u, v, t)\), 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., \( \max_{1 \leq i \leq M} |\pi_i| \).

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 \( \ell(t) = (\pi(t), \ldots, \pi_M(t)) \) specified by the coordinates of the locations of all agents at timestep \( t \). The desired formation is an \( M \)-tuple \( \ell_g = (g_1, \ldots, g_M) \) specified by the coordinates of the goal locations of all agents. The formation distance \( F(\ell, \ell') \) between two formations \( \ell \) and \( \ell' \) characterizes the least effort needed to transform formation \( \ell \) to formation \( \ell' \). 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 \( \Delta x \) to \( \ell' \), minimized over all such translations. Formally, let \( \ell = (u_1, \ldots, u_M) \) and \( \ell' = (v_1, \ldots, v_M) \). Then,

\[
F(\ell, \ell') = \min_{\Delta x} \sum_{i=1}^{M} |u_i - (v_i + \Delta x)|_1 = \sum_{j=1}^{d} \min_{\Delta x_j} \sum_{i=1}^{M} |u_{ij} - v_{ij} - \Delta x_j| = \sum_{j=1}^{d} \sum_{i=1}^{M} |u_{ij} - v_{ij} - \Delta x_j|,
\]

where, for each dimension \( j \), \( \Delta x_j \) is the median of all differences \( u_{ij} - v_{ij} \).

The formation deviation \( F(\ell(t), \ell_g(t)) \) at timestep \( t \) is the formation distance between the formation \( \ell(t) \) at timestep \( t \) and the desired formation \( \ell_g \), i.e., \( F(t) = F(\ell(t), \ell_g) \). The total formation deviation is the sum of the formation deviations over all timesteps, i.e., \( \sum_t F(t) \). Figure 1 (left) shows an example: The differences between the coordinates of the start and goal locations for every agent in every dimension \( \{u_{11} - g_{11} | i = 1, 2, 3\} \) and \( \{u_{12} - g_{12} | i = 1, 2, 3\} \) are \( \{3, 4, 3\} \) and \( \{-2, -1, 2, -2\} \), respectively. Therefore, at timestep 0, the optimal translation is \( \Delta x^* = (3, -1) \), and the formation deviation at timestep 0 is thus \( F(0) = 5 \).

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 the leader-follower controller by ignoring the other agents. At each timestep, we try to move toward their desired location \( g_\star \). The leader-follower controller selects the leader \( \pi_\star \) by minimizing \( \ell_\star \), i.e., \( \ell_\star = F(\pi_\star(t), \ell_g(t)) \). The total formation deviation is the sum of the formation deviations over all timesteps, i.e., \( \sum_t F(t) \). Figure 1 (left) shows an example: The differences between the coordinates of the start and goal locations for every agent in every dimension \( \{u_{11} - g_{11} | i = 1, 2, 3\} \) and \( \{u_{12} - g_{12} | i = 1, 2, 3\} \) are \( \{3, 4, 3\} \) and \( \{-2, -1, 2, -2\} \), respectively. Therefore, at timestep 0, the optimal translation is \( \Delta x^* = (3, -1) \), and the formation deviation at timestep 0 is thus \( F(0) = 5 \).

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 the leader \( \pi_\star \) by minimizing \( \ell_\star \), i.e., \( \ell_\star = F(\pi_\star(t), \ell_g(t)) \). The total formation deviation is the sum of the formation deviations over all timesteps, i.e., \( \sum_t F(t) \). Figure 1 (left) shows an example: The differences between the coordinates of the start and goal locations for every agent in every dimension \( \{u_{11} - g_{11} | i = 1, 2, 3\} \) and \( \{u_{12} - g_{12} | i = 1, 2, 3\} \) are \( \{3, 4, 3\} \) and \( \{-2, -1, 2, -2\} \), respectively. Therefore, at timestep 0, the optimal translation is \( \Delta x^* = (3, -1) \), and the formation deviation at timestep 0 is thus \( F(0) = 5 \).

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 the leader \( \pi_\star \) by minimizing \( \ell_\star \), i.e., \( \ell_\star = F(\pi_\star(t), \ell_g(t)) \). The total formation deviation is the sum of the formation deviations over all timesteps, i.e., \( \sum_t F(t) \). Figure 1 (left) shows an example: The differences between the coordinates of the start and goal locations for every agent in every dimension \( \{u_{11} - g_{11} | i = 1, 2, 3\} \) and \( \{u_{12} - g_{12} | i = 1, 2, 3\} \) are \( \{3, 4, 3\} \) and \( \{-2, -1, 2, -2\} \), respectively. Therefore, at timestep 0, the optimal translation is \( \Delta x^* = (3, -1) \), and the formation deviation at timestep 0 is thus \( F(0) = 5 \).

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 \( \Delta x^* \). Therefore, differentiating \( \sum_{i=1}^{M} |u_{ij} - v_{ij} - \Delta x_j| \) with respect to \( \Delta x_j \) yields \( \sum_{i=1}^{M} \text{sign}(u_{ij} - v_{ij}) - \Delta x_j \), which is 0 only if the number of positive items equals the number of negative items. This happens when \( \Delta x_j \) is the median of all differences \( u_{ij} - v_{ij} \).
We 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.,

$$C_{fh}(n) = \sum_{j=1}^{\lceil F(t)/M \rceil} (F(t) - M \cdot j).$$

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 $\epsilon$-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 $\epsilon \geq 1$ is a user-provided parameter. The focal list prioritizes its nodes $n$ in increasing order of their total formation deviation $f$-value $C_f(n)$. 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 $\epsilon$ 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 $\epsilon$.

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.

\section{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

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.
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 planned 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 $a^\ast$ 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 $B = \max_{0 \leq i \leq M} \text{dist}(s_i, g_i)$.

To do so, SWARM-MAPF performs a best-first search for each agent $a_i$ to find a path $\pi_i$ 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 $\pi^\ast$ and the corresponding agent as the leader $a^\ast$. It breaks ties in favor of shorter paths.

5.1.2 Partitioning the Path of the Leader into Segments. Once the leader $a^\ast$ has been chosen, its path $\pi^\ast$ is partitioned into open segments and congested segments alternately. Each open segment is a maximum segment $[\pi^\ast(t_b), \pi^\ast(t_e)]$ ($t_b \leq t_e$) such that, for all $t_b \leq t \leq t_e$, location $\pi^\ast(t)$ 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 $p^1_0, \ldots, p^2_K$ denote the first and last locations of all open segments, i.e., the $k$-th open segment is $[p^2_k-1, p^2_k]$. Let $p^0_0$ denote the start location of the leader. $p^0_0$ and $p^1_0$ are identical if the start locations of all agents are in the desired formation. There are also $K$ congested segments, and the $k$-th congested segment can be represented by $[p^2_{2k-2}, p^2_{2k-1}]$.

Let $\ell^t_k$ denote the $M$-tuple of the start locations of all agents and $\ell^t_k$ ($1 \leq k \leq 2K$) 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 $\ell^t_{2k-1}$ to locations $\ell^t_{2k}$. 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 $[p^2_{2k-1}, p^2_{2k}]$ of the leader. In each congested segment $[p^2_{2k-2}, p^2_{2k-1}]$, any location except for locations $p^2_{2k-2}$ and $p^2_{2k-1}$ is formation-blocking. Each congested segment specifies a sub-MAiF instance where all agents need to move from the intermediate start locations $\ell^t_{2k-2}$ to the intermediate goal locations $\ell^t_{2k}$. 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 $\ell^t_0 = (3(1), 5(1), 4(3))$ to the intermediate goal locations $\ell^t_1 = (4(2), 5(1), 5(0))$ and an open segment from locations $\ell^t_2$ to the goal locations $\ell^t_2 = ((0, 3), (1, 2), (1, 1))$. 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 \((a_i, a_j, u, v, t)\), then CBS adds the edge constraint \((a_i, u, v, t)\) to \(N_i.constrains\) to prohibit agent \(a_i\) from moving from location \(u\) to location \(v\) at timestep \(t\) and similarly adds the edge constraint \((a_j, v, u, t)\) to \(N_j.constrains\).

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 \(N_i.constrains\) 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 \(N_i.constrains\) 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 \(N_i.paths\)) 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 \(N_i.paths\), 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 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 \(N.constrains\).

**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 \(N.constrains\). The lemma then holds because any solution whose paths obey \(N.constrains\) is a set of paths of all agents that obey \(N.constrains\). 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 of \(N'\). (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 \(N'.constrains\) according to the induction assumption. In turn, this makespan is at most the makespan of any set of paths of all agents that obey \(N'.constrains\) since \(N'.constrains\) is a super set of \(N.constrains\). (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 \(N'.constrains\). 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 \(N'.constrains\). 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 \(\ell_{2k-2}^*\) and \(\ell_{2k-1}^*\), then it adds the edge constraint \((a_k, u, v, t)\) to \(N_i.constrains\) to prohibit agent \(a_k\) from moving from location \(u\) to location \(v\) at timestep \(t\).
We now show that, in extreme cases, SWARM-MAPF computes so-

\[ \ell \]

SWARM-MAPF with iterative goal updating is complete (namely, via the locations \[ \ell \]).

5.4 Limitation of SWARM-MAPF

Solutions with large makespans. Figure 3 shows a conceptual example

MAiF instance. SWARM-MAPF must correctly report that no solution exists to the

no solution exists to the this sub-MAiF instance, and, consequently,

overall MAiF instance. Since CBS-M is complete, it must report that

solutions up to

this solution appended to the concatenation of the computed sub-

append the concatenation of all computed sub-solutions up to

turn a sub-solution to the last sub-MAiF instance. This sub-solution

that there exist collision-free paths that move all agents from the

\[ \ell \]

all actions produces collision-free paths that move them from the

\[ \ell \]

intermediate start locations \[ \ell \] to the goal locations \[ \ell \]. 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 \[ \ell \] to their goal locations \[ \ell \]. 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 \[ \ell \] and the goal locations \[ \ell \]. Concatenating the computed sub-solutions up to the inter-

mediate start locations \[ \ell \] produces collision-free paths that move all agents from the locations \[ \ell \] to the locations \[ \ell \]. Reversing

all actions produces collision-free paths that move them from the locations \[ \ell \] to the locations \[ \ell \]. By assumption, we also know

that there exist collision-free paths that move all agents from the locations \[ \ell \] to the goal locations \[ \ell \]. Therefore, there exists a solution that moves all agents from the locations \[ \ell \] to the locations \[ \ell \] (namely, via the locations \[ \ell \]). Since CBS-M is complete, it must re-

turn a sub-solution to the last sub-MAiF instance. This sub-solution appended to the concatenation of all computed sub-solutions up to

\[ \ell \]

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 \[ \ell \] and the goal locations \[ \ell \] either, since, otherwise, this solution appended to the concatenation of the computed sub-

solutions up to \[ \ell \] 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. 

\[ \Box \]

\[ \Box \]

5.4 Limitation of SWARM-MAPF

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

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 pas-
sageway 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 (interme-

diate) 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 \([m_k, m_{k+1}] (0 \leq k \leq C + 1) \) and \( C + 1 \) congested segments \([m_k, g_1] (0 \leq k \leq C) \), where \( m_0 \) is the start location \( s_1 \) of agent \( a_1 \), \( m_{C+1} \) is the goal location of agent \( a_1 \), and \( m_k (1 \leq k \leq C) \) 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
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 $30 \times 30$ 4-neighbor grids. For each grid, the top-left $8 \times 8$ cells are possible start locations, and the bottom-right $8 \times 8$ 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 $8 \times 8$ 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 $8 \times 8$ 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 slightly increasing the makespan.

### 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 $\leq 30$ agents, while the success rate of CBS-M is larger than 90% for $\leq 35$ agents. Overall, CBS-M scales to slightly larger numbers of agents than SWARM-MAPF. Both algorithms can compute solutions for $\leq 20$ agents in real-time ($< 1$ s) 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: $3 \times 3$, $5 \times 5$, and $7 \times 7$. 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).
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 $20 \times 20$ 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 $\epsilon$ 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 $\epsilon$ 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.

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). We visualize them on one of the $30 \times 30$ 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 $30 \times 30$ 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.
REFERENCES


