Multi-Agent Path Finding for Large Agents Is Intractable

Artem Agafonov[a] **and Konstantin Yakovlev**[b,a,c]

[a]National Research University Higher School of Economics, Moscow, Russia
[b]Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Moscow, Russia
[c]AIRI, Moscow, Russia

Abstract. The multi-agent path finding (MAPF) problem asks to find a set of paths on a graph such that when synchronously following these paths the agents never encounter a conflict. In the most widespread MAPF formulation, the so-called Classical MAPF, the agents' sizes are neglected and two types of conflicts are considered: occupying the same vertex or using the same edge at the same time step. Meanwhile in numerous practical applications, e.g. in robotics, taking into account the agents' sizes is vital to ensure that the MAPF solutions can be safely executed. Introducing large agents yields an additional type of conflict arising when one agent follows an edge and its body overlaps with the body of another agent that is actually not using this same edge (e.g. staying still at some distinct vertex of the graph). Until now it was not clear how harder the problem gets when such conflicts are to be considered while planning. Specifically, it was known that Classical MAPF problem on an undirected graph can be solved in polynomial time, however no complete polynomial-time algorithm was presented to solve MAPF with large agents. In this paper we, for the first time, establish that the latter problem is NP-hard and, thus, if P≠NP no polynomial algorithm for it can, unfortunately, be presented. Our proof is based on the prevalent in the field technique of reducing the seminal 3-SAT problem (which is known to be an NP-complete problem) to the problem at hand. In particular, for an arbitrary 3-SAT formula we procedurally construct a dedicated graph with specific start and goal vertices and show that the given 3-SAT formula is satisfiable iff the corresponding path finding instance has a solution.

1 Introduction

Multi-agent path finding (MAPF) [12] generalizes the task of finding a single path in a graph to the case of finding multiple paths with certain restrictions. In the basic MAPF formulation, the so-called Classical MAPF, we are given a graph and n start and goal vertices, one for each agent. The time is discretized and at each time step an agent can either stay at its current vertex or move to an adjacent one following an edge. Thus the individual path for an agent may encounter a repetition of the same vertex, meaning that the agent waits for several time steps in it. The waiting actions are crucial in MAPF because they are often needed to eliminate *conflicts*. In particular, two paths are said to contain an edge (vertex) conflict if they use the same edge (end up in the same vertex) at the same time step. The MAPF problem asks to find n paths (one for each agent), s.t. any pair of them are conflict-free.

The cost of a MAPF solution is typically measured as either the sum of costs of the individual solutions (SOC or flowitme) or the
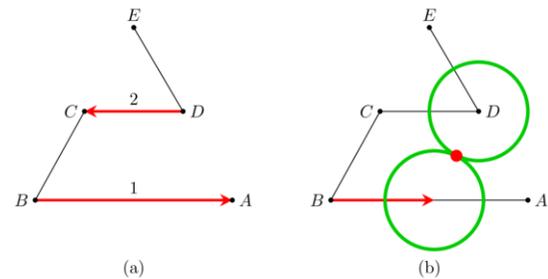

Figure 1. MAPF vs. LA-MAPF.

maximum over them – the makespan. Here, the individual cost is the time step when the agent reaches its goal (and never moves away). It is known that solving MAPF optimally w.r.t. SOC or makespan is NP-Hard [13]. Meanwhile, a decision variant of MAPF, which asks for the existence of a solution without any restrictions on the cost, can be solved in polynomial time [5] (on an undirected graph) and, indeed, a plethora of fast, subtimal MAPF algorithms exists: Push and Rotate [2], PIBT [10], LaCAM [9] to name a few.

Meanwhile, directly transferring MAPF solutions to the real world is complicated as Classical MAPF relies on a range of simplifying assumptions that are not met in practice, e.g. body-less agents, uniform duration of moves, instantaneous acceleration/deceleration etc. To this end a less restrictive and more practically-oriented MAPF formulations were introduced and studied. One of such prominent formulation is the one introduced in [6] and named MAPF for Large Agents, or LA-MAPF for short. In this formulation the graph is naturally embedded into the Euclidean space (workspace) and it is assumed that an agent with body i) occupies a portion of the workspace while residing in the graph vertex; ii) sweeps a certain volume of space when moving along an edge which is explicitly associated with a segment of a straight line in the workspace. Consequently, the number of ways how the agents may get in conflict increases.

Consider for example a problem instance depicted in Figure 1. Here the first agent, a_1, has to move from B to A and the second one, a_2, from D to C. In case of MAPF (Figure 1-a) the trivial solution: $a_1 : B \rightarrow A, a_2 : D \rightarrow C$ is collision-free. However the same solution for the case of LA-MAPF may be not valid (Figure 1-b). The problem is that no matter whether the disk-shaped agent a_2 stays at D or moves from D to C the collision occurs with the same-sized agent a_1 traversing BA edge. To solve the problem a_2 has to move to vertex E first, making enough room for a_1 to move from B to A. After this move a_2 can go back to D and then to C.

Intuitively, LA-MAPF is harder to solve than MAPF, but until now the computational complexity of LA-MAPF was not formally established, contrary to MAPF which is known to be solvable in polynomial time (on undirected graphs) since 1984 [5]. In this work we fill this gap, by proving that the decision variant of LA-MAPF (i.e. giving YES/NO answer to the question "if a solution for a LA-MAPF exists") is NP-Hard. This implies that, if $P \neq NP$, then there is no complete polynomial algorithm for solving LA-MAPF.

2 Related works

Two lines of research are especially relevant for our work: extending MAPF to handle agents that are not body-less (i.e. have shape and size) and analyzing the computational complexity of MAPF.

Extending MAPF to handle large agents   It is known that Classical MAPF (with point agents) is solvable in polynomial time and one of the widespread polynomial MAPF algorithm with strong theoretical guarantees is Push and Rotate [2]. In [3] an adaptation of this algorithm for LA-MAPF was proposed that keeps the core logic of Push and Rotate but adds dedicated moves tailored to clear the way for the large agents in case of obstruction. The resultant algorithm runs in polynomial time, but, unfortunately, it is not complete, i.e. may fail to find a solution for a solvable LA-MAPF instance.

Another widespread approach to solve Classical MAPF is the Conflict-Based Search (CBS) [11]. This solver is originally tailored to obtain optimal MAPF solutions and the expected running time of the algorithm is estimated by an exponential function [4], which limits its applicability in complex scenarios involving large number of agents. Moreover CBS may not correctly terminate on an usolvable MAPF instance. Still, it is widely used in MAPF community and adapted to various modifications of the problem.

The most relevant CBS adaptation to the problem we are interested in is presented in [6], where CBS was specifically modified to handle large agents. Specifically, several new types of CBS constraints were introduced that increase the pruning power of CBS and positively influence its runtime. Additionally, in this paper the authors state without proof that the optimization variant of LA-MAPF is NP-hard, since it generalizes the optimization version of Classical MAPF which is known to be NP-hard. Still, for the decision variant of LA-MAPF no such claim has been made.

In [15] another adaptation of CBS to LA-MAPF problem is introduced. The key idea behind this method is to decompose the problem into smaller subproblems with a smaller number of agents. It does not compromise the guarantees and allows to find solutions faster in practice. Still, the complexity of this method remains exponential.

Overall, several adaptations of seminal MAPF methods to LA-MAPF setting are known but, none of such algorithms are both complete and run in polynomial time. Thus, it is still not clear whether LA-MAPF can be solved in polynomial time (on an undirected graph) like regular MAPF, or not.

MAPF complexity   A range of papers analyzing the computational complexity of different variants of Classical MAPF problems have been published so far. The key results are summarized in Table 1. These works consider both variants of the problem: decision and optimization, and they also investigate this problem under different assumptions about the graph, such as edge orientation and graph layout. P and NP results have been obtained, but all of these papers explore the problem under the assumption that agents are body-less. Therefore, these papers do not directly answer our question, but they leave room for further research on the LA-MAPF problem.

Table 1.   Relevant results of complexity analysis for the MAPF problem.

Publication	Graph	Task type	Complexity
Kornhauser. 1984 [5]	Undirected	Decision	P
Nebel. 2020 [7]	Directed	Decision	NP-complete
Nebel. 2024 [8]	Directed, strongly-connected	Decision	P
Yu. 2015 [16]	Planar	Optimization	NP-complete
Banfi et al. 2017 [1]	2D grid	Optimization	NP-hard
Tan et al. 2023 [14]	Directed acyclic graph	Optimization	NP-complete

3 Problem statement

Let $G = (V, E)$ be an undirected graph that is embedded into the Euclidean plane $\mathbb{R}^2$. That is, each vertex $v \in V$ corresponds to a point on a plane with the coordinates (v_x, v_y) and each edge $e = (u, v) \in E$ corresponds to a segment of a straight line connecting (u_x, u_y) to (v_x, v_y).

Let $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$ be a set of k agents. Each agent is modeled as a disk of radius $r = const > 0$, the agent's reference point is the center of the disk. We say that an agent is located at vertex $v \in V$ if its reference point has coordinates (v_x, v_y). We say that an agent is moving along the edge $(u, v) \in E$ if its reference point is moving along the straight line segment connecting (u_x, u_y) and (v_x, v_y).

The time is discretised into the time steps, i.e. $T = [0, 1, \ldots)$. At each time stamp t the joint configuration of the agents is an ordered sequence of distinct graph vertices, specifying where the agents are located: $\Pi_t = (v_1, \ldots v_k)$. The location of the i-th agent is denoted as $\Pi_t(a_i) = v_i$. Consider now two consecutive configurations Π_t and Π_{t+1}. These configurations are said to form a transition iff:

1. $\exists! i \in [1, k] : (\Pi_t(a_i), \Pi_{t+1}(a_i)) \in E$
2. $\forall j \neq i : \Pi_t(a_j) = \Pi_{t+1}(a_j)$

In other words a transition occurs when exactly one agent moves along an edge while the others stay put.

Consider now a sequence $\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_L)$. It is said to contain a vertex conflict, $\langle a_i, a_j, u, v, t \rangle$, if $\exists i, j : \|\Pi_t(a_i), \Pi_t(a_j)\| < 2 \cdot r$. I.e., the vertex conflict occurs when the agents' bodies overlap. Similarly, the edge conflict is defined as $\langle a_i, a_j, u, v, v', t \rangle$. Here, a_i is the agent that changes its location from $u = \Pi_t(a_i)$ to $v = \Pi_{t+1}(a_i)$ and a_j is the agent residing at v' at Π_t and Π_{t+1}, and the conflict occurs if the distance between the point (v'_x, v'_y) and the segment connecting (u_x, u_y) and (v_x, v_y) is less than $2r$.

The LA-MAPF problem now is: given the start and the goal configurations find a sequence of transitions connecting them, s.t. it is conflict-free. Please note, that in this work we are interested in getting any solution of LA-MAPF (decision variant), rather than getting a solution that minimizes some cost objective (optimization variant).

4 Reduction of 3-SAT to LA-MAPF

Our ultimate goal is to demonstrate that LA-MAPF is an NP-Hard problem. We do this by following an established approach (Karp-

reduction) of reducing the seminal NP-complete problem, i.e. 3-SAT, to the problem at hand, i.e. LA-MAPF.

3-SAT is a problem that for a (special type of) Boolean formula asks whether there exists an assignment of the variables that renders this formula True. Specifically, the input formula is assumed to be a 3CNF, i.e. to be a conjunction of $m \geq 1$ clauses where each clause is the disjunction of exactly three literals: a variable or its negation. The total number of variables in the formula is denoted as $n \geq 1$.

Our reduction is a careful demonstration that for any 3-SAT instance, i.e. for arbitrary Boolean 3CNF formula, one can construct a graph and define the start and goal locations of the agents (as well as their radius), such that there exists a solution to the resulting LA-MAPF problem iff there is a satisfying assignment for the original 3-SAT formula. Additionally, we ensure that the size of the graph and the number of agents are polynomial in the number of variables and clauses of the initial 3-SAT instance.

The agents' radius in the reduction is set to be equal to the number of clauses in the original problem, i.e. $r = m$.

The reduction itself is made up of the three stages, each adding the so-called *gadgets* – subgraphs (that may contain start and goal locations for the agents) out of which the final LA-MAPF graph is constructed. Each vertex of each gadget has a specific coordinates on a plane. To better visualize them we divide the latter into the several disjoint zones as shown in Figure 2.
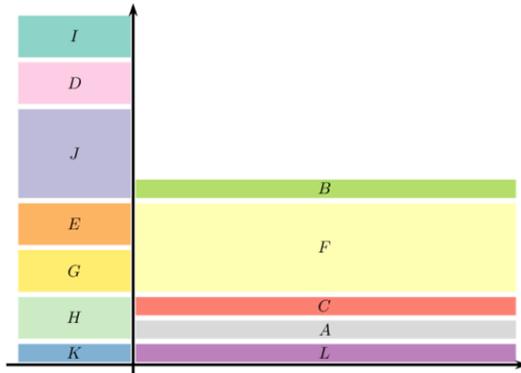

Figure 2. The division of the plane for the reduction.

Please note, that for the sake of clarity, the locations of certain vertices in the subsequent figures will not exactly match their actual locations as per provided coordinates.

In the figures in the following sections, we will use a 3-SAT instance with 3 variables and 1 clause: $x_1 \lor \neg x_2 \lor x_3$, as an illustrative example. However, all formulas are provided for the general case, meaning that our aproach is indeed applicable to any 3-SAT instance.

4.1 Variables

The first type of gadget we introduce is composed of 3 vertices (A, B, C) and an agent. The agent's starting and goal vertices are both A. Even though this agent is already located at the final vertex, we will later show that it will need to move away from and return to this vertex in order to allow other agents to reach their destinations.

We populate this gadget n times, one for each variable in the initial 3-SAT formula. We will refer to the agent associated with the i-th variable as the i-th v-agent. The vertices in the i-th gadget have the following coordinates:

$$\begin{cases} A_i = ((2m+1) \cdot i,\ 1) \\ B_i = ((2m+1) \cdot i,\ 4m^2 n + 2m^2 + 2mn + 5m - 1) \\ C_i = ((2m+1) \cdot i,\ 2m^2 + m) \end{cases}$$

The gadgets of the first type used in the example are shown in Figure 3. To improve clarity, the vertices of each gadget are not aligned along a vertical line.
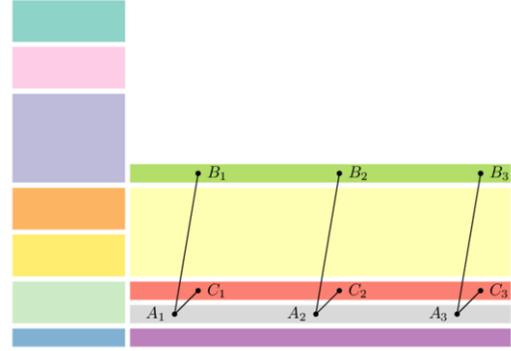

Figure 3. The gadgets of first type.

We assume that the i-th v-agent, located at the B_i vertex, corresponds to the true value of the i-th variable ($x_i =$ True), and at the C_i vertex corresponds to the false value ($x_i =$ False).

4.2 Clauses

The second type of gadget has 8 vertices (D, E, 3 vertices in the F zone, G, H, I) and an agent. The agent's starting vertex is D and goal vertex is I.

We populate this gadget m times, one for each clause in the initial 3-SAT formula. We will refer to the agent associated with the j-th clause as the j-th c-agent. The vertices in the j-th gadget have the following coordinates:

$$\begin{cases} D_j = (0,\ (2m+1) \cdot j + 4m^2 n + 2m^2 + 4mn + 3m + n - 2) \\ E_j = (0,\ j + 2m^2 n + 2m^2 + mn + 3m - 1) \\ G_j = (0,\ j + 2m^2 n + 2m^2 + mn + 2m - 1) \\ H_j = (0,\ (2m+1) \cdot j) \\ I_j = (0,\ (2m+1) \cdot j + 4m^2 n + 4m^2 + 4mn + 5m + n - 2) \end{cases}$$

The coordinates of the remaining three vertices are calculated based on the literals in the j-th clause. If the j-th clause has a positive literal x_i, then we add a vertex to the graph with the following coordinates:

$$F_j^{x_i} = ((2m+1) \cdot i,\ j + 2m^2 + 2m - 1)$$

If the j-th clause has a negative literal $\neg x_i$, then we add a vertex to the graph with the following coordinates:

$$F_j^{\neg x_i} = ((2m+1) \cdot i,\ j + 4m^2 n + 2m^2 + 2mn + 3m - 1)$$

The gadget of the second type that is used in the example is shown in Figure 4. To improve clarity, the vertices D_1, E_1, G_1, H_1 and I_1 are not aligned along a vertical line.

Lemma 1. *If there are agents in the C_i and $F_j^{x_i}$ vertices (similarly for the B_i and $F_j^{\neg x_i}$ vertices), then there is a conflict between them.*
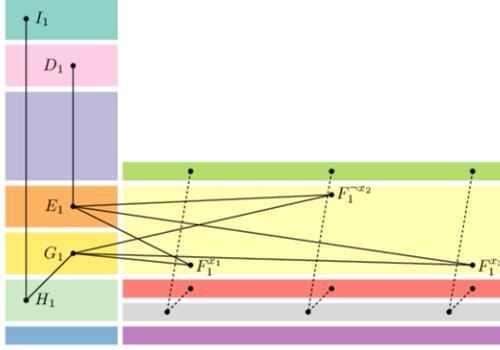
Figure 4. The subgraph of the clause-agent.

Proof. The distance between the C_i and $F_j^{x_i}$ vertices is:

$$\sqrt{(x_{F_j^{x_i}} - x_{C_i})^2 + (y_{F_j^{x_i}} - y_{C_i})^2} =$$

$$= \sqrt{(((2m+1) \cdot i) - ((2m+1) \cdot i))^2 + (y_{F_j^{x_i}} - y_{C_i})^2} =$$

$$= |(j + 2m^2 + 2m - 1) - (2m^2 + m)| =$$

$$= j + m - 1 < 2r$$

Similarly, for the B_i and $F_j^{\neg x_i}$ vertices:

$$\sqrt{(x_{B_i} - x_{F_j^{\neg x_i}})^2 + (y_{B_i} - y_{F_j^{\neg x_i}})^2} =$$

$$= \sqrt{(((2m+1) \cdot i) - ((2m+1) \cdot i))^2 + (y_{B_i} - y_{F_j^{\neg x_i}})^2} =$$

$$= |y_{B_i} - y_{F_j^{\neg x_i}}| = 2m - j < 2r$$

Since the distance between the vertices is less than $2r$, the simultaneous presence of agents at these vertices will lead to a conflict between them. □

Lemma 1 states that if the i-th v-agent is located at the C_i vertex, then the j-th c-agent cannot move to the $F_j^{x_i}$ vertex, as this would cause a collision between the two agents. A similar property holds for the vertices B_i and $F_j^{\neg x_i}$. This restriction creates a connection between the two problems. An inappropriate assignment of variables leads to the unsatisfiability of a clause. Similarly, an inappropriate placement of v-agents may prevent a c-agent from reaching its destination.

4.3   Blocking

The final gadget we discuss has $2n + 1$ vertices (J_1-J_n, K, L_1-L_n) and $n + 1$ agents. The starting and goal vertices for the agents are J_1-J_n and K. Although these agents are already at their final vertices, they will need to move away from and return to these vertices in order for c-agents to reach their destinations. To accomplish this, they must force v-agents to leave their own final vertices, as we mentioned earlier.

We will refer to the agent in this gadget as the b-agent. The vertices in this gadget have the following coordinates:

$$\begin{cases} J_i = (0, \ (2m+1) \cdot i + 4m^2n + 2m^2 + 2mn + 3m - 2) \\ K = (0, \ 0) \\ L_i = ((2m+1) \cdot i, \ 0) \end{cases}$$

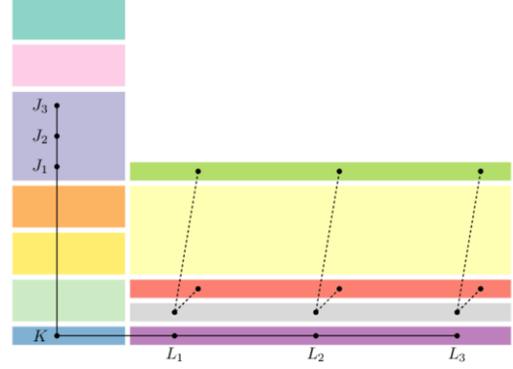The gadget of the third type used in the example is shown in Figure 5.

Figure 5. The subgraph of the block-agents.

Lemma 2. *If there are agents in the A_i and L_i vertices, then there is a conflict between them.*

Proof. The distance between the A_i and L_i vertices is:

$$\sqrt{(x_{A_i} - x_{L_i})^2 + (y_{A_i} - y_{L_i})^2} =$$

$$= \sqrt{(((2m+1) \cdot i) - ((2m+1) \cdot i))^2 + (1 - 0)^2} =$$

$$= 1 < 2r$$

Since the distance between the vertices is less than $2r$, the simultaneous presence of agents at these vertices will lead to a conflict between them. □

Lemma 2 states that if the b-agent is located at the L_i vertex, then the i-th v-agent cannot move to the A_i vertex, as this would result in a collision between these two agents. This restriction prevents the i-th v-agent from moving between B_i and C_i vertices. It other words, the values of all variables remain fixed whenever the b-agents are located at the L vertices.

5   Correctness of reduction

In this section, we will demonstrate the equivalence between the original 3-SAT problem and the LA-MAPF problem that we have constructed. We will prove that if one of the problems has a solution, then so does another problem. The following subsections will present algorithms for constructing the solution to one problem from the given solution to the other.

5.1   3-SAT to LA-MAPF

Given a solution to the 3-SAT problem, which is an assignment of variables that makes a formula satisfiable, we can use Algorithm 1 to create a conflict-free plan that moves all agents to their destinations in the constructed reduction.

The solution constructed by Algorithm 1 consists of six steps. Figure 6 and Figure 7 illustrate these steps using the example of a 3-SAT formula with 3 variables $x_1 = x_2 = True$, $x_3 = False$, and 1 clause: $x_1 \lor \neg x_2 \lor x_3$.

The idea behind this algorithm is to move v-agents to vertices corresponding to a given 3-SAT solution, move c-agents to terminal vertices, and then return v-agents back. To do this, it adds necessary moves by b-agents. More specifically:

Algorithm 1 The construction of a reduction solution based on the 3-SAT solution.

Input: $\{x_i \in \{True, \ False\} : \ i = 1, \ \ldots, \ n\}$ - Solution of 3-SAT

Result: $\{(u_1 \to v_1), \ \ldots, \ (u_N \to v_N)\}$ - Solution of LA-MAPF

1: Solution $\leftarrow \{\}$
2: **for** $i \leftarrow 1$ **to** n **do**
3:     **if** x_i is True **then**
4:         Solution.Append($A_i \to B_i$)
5:     **else**
6:         Solution.Append($A_i \to C_i$)
7:     **end if**
8: **end for**
9: Solution.Append($K \to L_1 \to \ldots \to L_n$)
10: **for** $i \leftarrow 1$ **to** $n - 1$ **do**
11:     Solution.Append($J_i \ \to \ \ldots \ \to \ J_1 \ \to \ K \ \to \ L_1 \ldots \ \to$ L_{n-i})
12: **end for**
13: Solution.Append($J_n \to \ldots \to J_1 \to K$)
14: **for** $j \leftarrow 1$ **to** m **do**
15:     Solution.Append($D_j \to E_j$)
16:     **for** $(id, \ neg)$ **in** arguments of j-th clause **do**
17:         **if** neg is True **then**
18:             $to \leftarrow F_j^{\neg x_{id}}$
19:         **else**
20:             $to \leftarrow F_j^{x_{id}}$
21:         **end if**
22:         **if** x_{id} is not neg **then**
23:             Solution.Append($E_j \to to \to G_j$)
24:             **break**
25:         **end if**
26:     **end for**
27:     Solution.Append($G_j \to H_j$)
28: **end for**
29: **for** $j \leftarrow m$ **to** 1 **do**
30:     Solution.Append($H_j \to I_j$)
31: **end for**
32: Solution.Append($K \to J_1 \to \ldots \to J_n$)
33: **for** $i \leftarrow n - 1$ **to** 1 **do**
34:     Solution.Append($L_{n-i} \ \to \ \ldots \ \to \ L_1 \ \to \ K \ \to \ J_1 \ldots \ \to$ J_i)
35: **end for**
36: Solution.Append($L_n \to \ldots \to L_1 \to K$)
37: **for** $i \leftarrow n$ **to** 1 **do**
38:     **if** x_i is True **then**
39:         Solution.Append($B_i \to A_i$)
40:     **else**
41:         Solution.Append($C_i \to A_i$)
42:     **end if**
43: **end for**
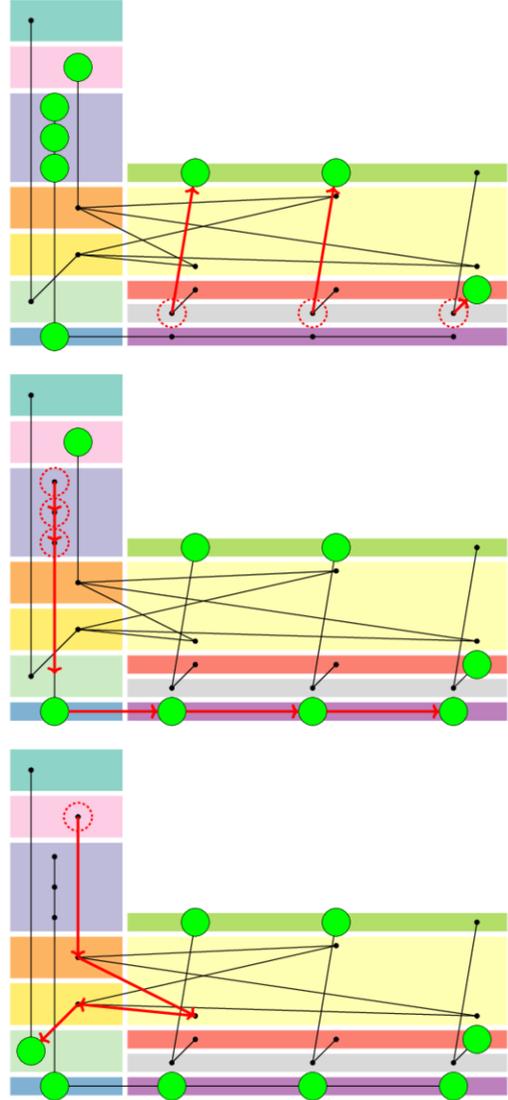44: **return** Solution

Figure 6. The 1-3 steps of the solution constructed by Algorithm 1.

1. In lines 2-8, the v-agents are moved to the vertices that correspond to the given solution. This means that if the i-th variable has a true value, then the i-th v-agent is moved to the B_i vertex, and to the C_i vertex otherwise. In the example, the v-agents are moved to vertices B_1, B_2 and C_3, respectively.

2. In lines 9-13, the b-agents are moved to the L vertices. This action empties J vertices and makes it possible to move the c-agents.

3. In lines 14-28, the c-agents are moved from the D vertex to the H vertex. The cycle in line 16 chooses the literal of the clause that makes it satisfied. In line 23, the c-agent is moved through the selected vertex. In the example, the c-agent moves through vertex $F_1^{x_1}$, as the true value of the first variable satisfies this clause.

4. In lines 29-31, the c-agents are moved to their respective destinations I.

5. In lines 32-36, the b-agents are returned to their starting vertices J.

6. In lines 37-43, the v-agents are returned to their starting vertices A.

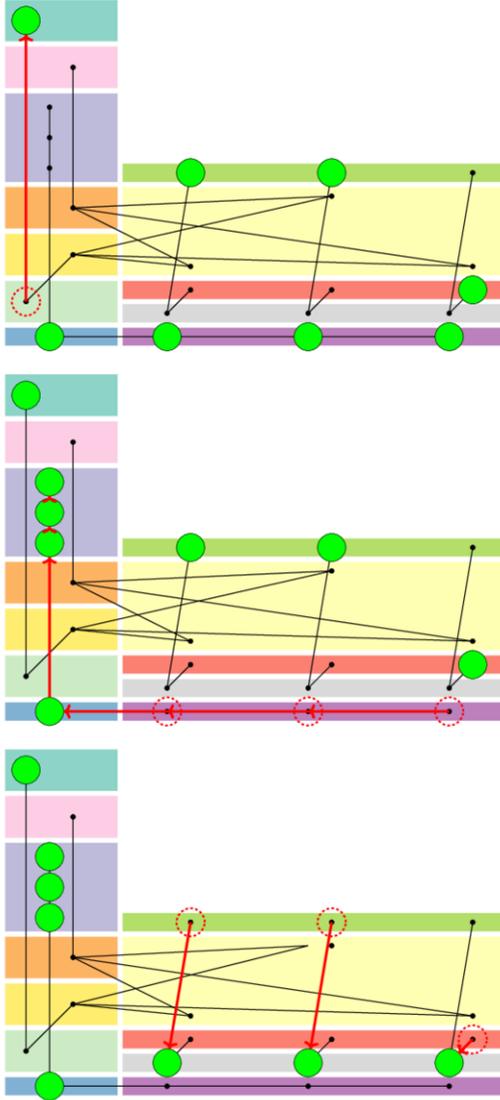Lemma 3 proves that Algorithm 1 constructs conflict-free paths.

Figure 7. The 4-6 steps of the solution constructed by Algorithm 1.

Lemma 3. *The Algorithm 1 provides a correct solution to the constructed reduction based on the given 3-SAT solution.*

Sketch of proof. In lines 2-8, the moves are made only by the v-agents. Therefore, there are no agents in the F and L zones. This allows all v-agents to move freely to the vertices in the B and C zones. After that, the A zone becomes free of agents.

In lines 9-13, the moves are made by the b-agents. Since there are no agents in the A, E, G, and H zones, they are able to move to the vertices of the K and L zones. After that, the J zone becomes empty.

In lines 14-28, the c-agents are moved through the D, E, F, G and H zones. Note that in line 23, the choice of a conflict-free vertex F_j^x is always possible. This is because in the initial 3-SAT solution, there was a literal that satisfied this clause, so the c-agent will be able to move through F_j^x vertex corresponding to that literal.

The moves in lines 29-31 are conflict-free.

In lines 32-43, the moves of the v-agents and the b-agents are the same as those in lines 2-13, but reversed. Therefore they are also conflict-free.

It means that the algorithm creates a plan for the agents to move

towards their destinations without interfering with each other. This proves the correctness of this algorithm. □

5.2 LA-MAPF to 3-SAT

Given a solution to the reduction, which is a plan that moves all agents to their destinations without conflicts, we can use Algorithm 2 to assign variables in such way that satisfies the original 3-SAT formula.

Algorithm 2 The construction of a 3-SAT solution based on the solution of the reduction.

Input: $\{(u_1 \to v_1), \ldots, (u_N \to v_N)\}$ - Solution of LA-MAPF
Result: $\{x_i \in \{True, False\} : \quad i = 1, \ldots, n\}$ - Solution of 3-SAT

1: State ← state where all agents are at their destinations
2: **for** $i \leftarrow N$ **to** 1 **do**
3:     State.Move(v_i, u_i)
4:     **if** State.IsOccupied(G_1) **then**
5:         **break**
6:     **end if**
7: **end for**
8: Solution ← {}
9: **for** $i \leftarrow 1$ **to** n **do**
10:     **if** State.IsOccupied(B_i) **then**
11:         Solution ← Solution ∪ $\{x_i = True\}$
12:     **else**
13:         Solution ← Solution ∪ $\{x_i = False\}$
14:     **end if**
15: **end for**
16: **return** Solution

In the first line, the LA-MAPF state is initialized with the final state, where all agents are in their final positions. In lines 2-7, the agents' moves are reversed until the G_1 vertex is occupied. Finally, in lines 9-15, variables are set: if the B_i vertex is occupied, then x_i is set to true, otherwise is set to false.

Lemma 4 proves that Algorithm 2 sets variables in such a way that the initial 3-SAT formula is satisfied.

Lemma 4. *The Algorithm 2 provides a correct solution for the initial 3-SAT problem, based on the given solution to the reduction.*

Proof. Consider the last time the first c-agent was in the G_1 vertex. Because in the end it must be in the I_1 vertex, the next move must lead it to the H_1 vertex.

Note that there is no b-agent in the J zone. Assume that it is there. Then, it will not be able to move out of the J zone, as the first c-agent will block its movement through the edge between the J_1 and K vertices. This means that the first c-agent will not be able to reach the I_1 vertex, as this would lead to a collision with the b-agent in the J zone.

This means that all b-agents are located in the K and L zones, and therefore, all v-agents must be located in the B and C zones, since they cannot be located in the A zone according to Lemma 2.

Consider the next timestamp, when the first c-agent will move to the I_1 vertex. Since it has stayed in the H_1 vertex, no b-agent can leave the L zone. Consequently, none of the v-agents have changed their positions. Also, this move means that no c-agent is in the D or E zones, so every c-agent was able to make a conflict-free move to a vertex in the F zone, based on the current positions of the v-agents.

Assume that there is a clause in the initial 3-SAT problem that is not satisfied by the constructed variable assignment. This implies that if we move the corresponding c-agent to any vertex of its gadget in the F zone, it would lead to a conflict with the corresponding v-agent according to Lemma 1. However, this would contradict the last statement in the previous paragraph. Therefore, all clauses must be satisfied, and the algorithm has constructed a correct solution to the initial 3-SAT problem. This proves the correctness of the algorithm. $\square$

6 On the intractability of LA-MAPF problem

We have shown that by using the proposed reduction, we can create a LA-MAPF problem that is solvable iff the initial 3-SAT formula has a solution. This means that the LA-MAPF problem is an NP-hard problem.

Theorem 5. *The LA-MAPF problem is an NP-hard problem.*

Proof. The number of vertices in the graph that was constructed is:

$$n \cdot 3 + m \cdot 8 + (2n + 1) = 5n + 8m + 1 \in \text{Poly}(n, m)$$

The number of agents involved in the reduction is:

$$n + m + (n + 1) = 2n + m + 1 \in \text{Poly}(n, m)$$

Therefore, the size of the reduction depends on the size of the original 3-SAT formula in a polynomial way. Lemma 3 states that if the 3-SAT formula has a solution, then the reduction also has the solution. Lemma 4 states that if the reduction has a solution, then the initial 3-SAT formula has a solution as well. This means that we have constructed a polynomial reduction from 3-SAT to LA-MAPF. Since 3-SAT is an NP-complete problem, this proofs that LA-MAPF is NP-hard. $\square$

7 Discussion

Generality of the results We have formally proved that the specific LA-MAPF instances constructed by our reduction are NP-hard to solve, and therefore, we conclude that the general LA-MAPF problem is also NP-hard. However, this does not mean that every LA-MAPF instance cannot be solved in polynomial time. Consider the following analogy – 3-SAT is known to be NP-Hard but this does not imply that all instances of 3-SAT cannot be solved in polynomial time. E.g. for any formula with no negative literals in the clauses the solution is clear: we can assign all variables to true. So these particular instances are in P, but 3-SAT has other instances that make the problem to be out of P (assuming P != NP). Similarly, our statement on NP-Hardness of LA-MAPF allows the existence of certain subclasses of LA-MAPF that can be solved in polynomial time. Identifying such classes (e.g. sparse graphs, specific layouts etc.) is a promising direction for future research.

Assumptions used In our reduction we use the value of r that is equal to the number of clauses in a given 3-SAT formula as such value allows us to assign integer coordinates to all vertices in the resulting graph, which makes the graph layout easy to visualize and understand. Meanwhile, for any value of r we can scale the coordinates in our reduction by a factor of r/m (where m is the number of clauses) and all the statements will hold. Thus, for any value of r one can construct instances of LA-MAPF that are NP-Hard to solve.

Another assumption that we have adopted is that only one agent can move at a time. Meanwhile, synchronous simultaneous moves of several agents may be allowed in MAPF as long as they do not result in collision. Our reduction is valid for simultaneous moves assumption as long as a transition from one joint configuration to a consecutive joint configuration can be decomposed into the sequence of atomic, i.e. single-agent, moves. Noteworthy, this is not always the case. For example, consider a "closed-chain" move, when at a time step a group of agents move in such way that the first agent moves to the vertex that was occupied by the second agent, the second agent moves to the vertex that was occupied by the third agent and so on, while the last agent moves to the vertex occupied by the first one. Estimating LA-MAPF tractability in this case is a possible direction of future work.

We would like to point out that the main source of hardness in our reduction stems from the uncertainty about the proper locations of the v-agents, which corresponds to the ambiguity in assigning 3-SAT variables. This is not due to a restriction on moving only one agent at a time, but rather to the complexity of determining the locations for each agent. Therefore, we believe a similar result could be achieved for the setting allowing multiple agents to move simultaneously. However, it would require careful consideration of the properties of the chosen conflict-checking function and the addition of extra gadgets.

8 Conclusions and future work

In this paper, we have investigated the decision version of the LA-MAPF problem, which is a natural extension of the MAPF problem where size of the agents is to be taken into account. So far the computational complexity of this problem had not yet been determined, in contrast to regular MAPF. To this end we have proposed a reduction from the seminal 3-SAT problem to the LA-MAPF problem in order to demonstrate that the latter is intractable, i.e. NP-hard. This means that (unlike regular MAPF) LA-MAPF can not solved in polynomial time (even on undirected graphs) unless P=NP. Our research establishes an important theoretical result and justifies the development of new heuristics for existing algorithms and faster exponential-time algorithms for practical applications.

The main reason why LA-MAPF is hard to solve is the need to consider closeness of the agents in the metric space to which the MAPF graph is embebedded. A prosperous direction of future research is to focus on specific graph layouts that possibly make the problem simpler (while still adequately reflecting the constraints of the real-world MAPF applications). Examples of such graphs are planar graphs or grids or sparse graphs, where the vertices are located far enough from each other. The proposed reduction method is generic and not tailored to these cases, so there may still exist a polynomial-time algorithm that could solve the problem by taking advantage of a specific graph layout.

Another fruitful direction to extend this work, as mentioned previously, is considering the setting where several agents are allowed to move simultaneously.

Finally, we have demonstrated that LA-MAPF is NP-hard, but we still do not know whether it belongs to NP. If one shows that LA-MAPF is in NP, this would also imply that it is NP-complete. One possible way to prove this would be to demonstrate that for any instance of LA-MAPF, a solution of polynomial length can be found.

Overall our study not only establishes an important milestone in MAPF research but opens a range of avenues for future research.

References

[1] J. Banfi, N. Basilico, and F. Amigoni. Intractability of time-optimal multirobot path planning on 2d grid graphs with holes. *IEEE Robotics and Automation Letters*, 2(4):1941–1947, 2017.

[2] B. De Wilde, A. W. Ter Mors, and C. Witteveen. Push and rotate: a complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research*, 51:443–492, 2014.

[3] S. Dergachev and K. Yakovlev. Towards a complete multi-agent pathfinding algorithm for large agents. In *Mexican International Conference on Artificial Intelligence*, pages 355–367. Springer, 2022.

[4] O. Gordon, Y. Filmus, and O. Salzman. Revisiting the complexity analysis of conflict-based search: New computational techniques and improved bounds. In *Proceedings of the International Symposium on Combinatorial Search*, volume 12, pages 64–72, 2021.

[5] D. M. Kornhauser. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. 1984.

[6] J. Li, P. Surynek, A. Felner, H. Ma, T. S. Kumar, and S. Koenig. Multi-agent path finding for large agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7627–7634, 2019.

[7] B. Nebel. On the computational complexity of multi-agent pathfinding on directed graphs. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 212–216, 2020.

[8] B. Nebel. The computational complexity of multi-agent pathfinding on directed graphs. *Artificial Intelligence*, 328:104063, 2024.

[9] K. Okumura. Lacam: Search-based algorithm for quick multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 11655–11662, 2023.

[10] K. Okumura, M. Machida, X. Défago, and Y. Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310:103752, 2022.

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

[12] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, pages 151–158, 2019.

[13] P. Surynek. An optimization variant of multi-robot path planning is intractable. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24, pages 1261–1263, 2010.

[14] X. Tan and P. Bercher. Intractability of optimal multi-agent pathfinding on directed graphs. In *ECAI*, pages 2315–2321, 2023.

[15] Z. Yao. Layered la-mapf: a decomposition of large agent mapf instance to accelerate solving without compromising solvability. *arXiv preprint arXiv:2410.17160*, 2024.

[16] J. Yu. Intractability of optimal multirobot path planning on planar graphs. *IEEE Robotics and Automation Letters*, 1(1):33–40, 2015.