Adversarial Multi-Agent Path Finding is Intractable

1st Marika Ivanová
Faculty of Mathematics and Physics
Charles University
Prague, Czechia
ivanova@ktiml.mff.cuni.cz

2nd Pavel Surynek
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czechia
0000-0001-7200-0542

***Abstract*—Adversarial Multi-Agent Path Finding (AMAPF) extends the standard discrete Multi-Agent Path Finding with an adversarial element. Agents of two competing teams are deployed in a shared environment represented by an undirected graph. The first team aims to navigate all its agents from their initial locations to given goal locations, while the second team aims to prevent agents of the first team from fulfilling their goal. We prove that the problem of finding a winning strategy is EXPTIME-complete.**

***Index Terms*—multi-agent path finding, adversary, computational complexity, EXPTIME-completeness**

I. Introduction

In the standard *Multi-Agent Path Finding* (MAPF) the problem is to find paths for agents deployed in a shared environment represented by an undirected graph $G = (V, E)$ from their initial locations to individual goal locations, such that the agents do not collide with each other or with obstacles while following their paths [1]–[3]. Often, the MAPF problem is solved with respect to cumulative objectives such as the *sum-of-costs* (the sum of time steps that agents have to perform until they arrive at their goal locations) [4], [5] or the makespan (the number of steps until the last agent reaches its goal location).

We address Adversarial Multi-Agent Path Finding (AMAPF) that generalizes MAPF by dividing agents into two disjoint competing teams. The objective of the first team is to navigate all its agents via non-colliding paths to their goal locations. Like in the standard MAPF, having all agents in their goal locations is the winning configuration for the first team.

The second team aims to prevent the first team from achieving its goal or equivalently protecting the goal locations of the first team. That is, if the first team fails to deliver one or more agents to their goal locations because the second team prevents them from doing so, then the first team loses. We are studying the yes/no question whether there exists a winning strategy for the first team. We address this decision problem from the computational complexity perspective.

We note that an optimization variant of AMAPF can be considered as well. In this variant, which is of practical importance, the first team tries to maximize the number of agents that manage to reach their goal locations.

This work is supported by the Czech Science Foundation, project no. 19-17966S, and by OP RDE project no. CZ.02.2.69/0.0/0.0/18_053/0016976.

A. Related work

MAPF attracts considerable interest driven by the requirements of modern warehouse logistics. Many algorithmic approaches have been developed for the standard MAPF ranging from sub-optimal rule-based algorithms [6] to optimal search-based [7] and compilation-based algorithms [8].

The optimality requirement with respect to any of the two common objectives adds a significant computational challenge to the problem. While finding a feasible solution of MAPF is a polynomial-time problem [3]. Finding an optimal solution to the standard MAPF in undirected graphs has been shown to be NP-hard [9]–[11]. Recently, [12] has shown that NP-hardness also holds for optimal MAPF on directed graphs.

Unique identification of agents can be regarded as another source of complexity. A variant of MAPF in which all agents are anonymous and can therefore reach any of the goal locations in the union of all goal locations across all agents, reduces the complexity to polynomial time even in the optimal setup. Such MAPF problem can be reduced to the minimum cost network flow problem, and is thus solvable in polynomial time. However, the problem with two or more teams of anonymous agents [13]–[15] is NP-hard [11] when optimality is required.

A variant of AMAPF was first introduced in [16]. A more detailed study of computational complexity of this early variant of AMAPF and several solving algorithms can be found in [17]. These works, however, assume a symmetric variant of AMAPF where each of the two teams has assigned goal locations to reach. The variant of AMAPF studied in this paper is asymmetric with different objectives of each of the teams which provides a more natural modeling of real-life situations [18].

B. Motivation

AMAPF is motivated by various multi-agent scenarios with adversaries. Examples include tactical military command and control, security operations, and computer games. The most developed related topic is represented by *multi-agent patrolling* and its variants [19]–[24]. Multi-agent patrolling is usually regarded as a two player game, similarly as AMAPF. In contrast to patrolling problems, the AMAPF problem has important aspect of planning cooperative movements for agents which is often omitted or addressed in a limited way only in patrolling problems.

Understanding the complexity of the MAPF problem turned out to be important for the design of compilation-based solving methods. Hence we assume that understanding that understanding the complexity in the case of AMAPF may also lead to the design of novel solving methods.

II. Formal Model

The *Adversarial Multi-Agent Path-Finding* problem is a 4-tuple $\Pi = (G, R, \alpha_0, \alpha_{A+})$, where $G = (V, E)$ is a connected graph that models the environment in which agents are deployed, $R = A \cup B$ is a set of agents partitioned into two disjoint sets A and B referred to as Team-A and Team-B, $\alpha_0 : R \mapsto V$ assigns a unique initial vertex to each agent (initial configuration), and $\alpha_{A+} : A \mapsto V$ assigns a unique goal vertex to each agent in Team-A (goal configuration). Agents in Team-B do not have predefined goal vertices.

Teams in AMAPF alternate in making their moves in discretized fashion; Team-A opens the game.

A team can perform a move corresponding to a single time step according to MAPF movement rules in each turn. That is, each agent from the team on the turn can either move to an adjacent vertex or stay in its current vertex. We consider the variant in which agents can move to a vacant adjacent vertex or to an adjacent vertex that is being vacated by another agent from the same team (train-like movement) provided that the two agents do not move along the same edge (edge collisions must be avoided). All agents from the team that is not on the turn must stay in their vertices. That is, Team-B's agents are regarded as static obstacles during Team-A's turn and vice-versa.

Definition 1. *An AMAPF* game-play *for input* Π *is a sequence* $\left([\alpha_i^A, \alpha_i^B]\right)_{i \in \mathbb{N}_0}$. α_i^B *is a result of Team-A's turn in configuration* α_i^A, α_{i+1}^A *is a result of Team-B's turn in configuration* α_i^B *for each* $i \in \mathbb{N}_0$. *As Team-A starts, we have* $\alpha_0^A = \alpha_0$. *A finite AMAPF* game-play *in which Team-A makes the last turn is a game-play such that* $\exists \mu \in \mathbb{N}_0$ *such that for all* $i \geq \mu$ $\alpha_i^A = \alpha_i^B = \alpha_\mu^A$; μ *is the length of the game.*

As soon as Team-A reaches the goal vertices, the game finishes and agents stop to move. After reaching the goals, configurations are identical for all following steps. Formally, finishing the game by reaching the goal is as follows:

Definition 2. *A winning AMAPF game-play* $\left([\alpha_i^A, \alpha_i^B]\right)_{i=0}^{\mu}$ *of length* μ *for input* Π *is a finite game-play such that* $\alpha_\mu^B(a) = \alpha_{A+}(a)$ *for all* $a \in A$. *We also say that the last configuration is the* winning configuration *for Team-A.*

The *winning configuration* for Team-B occurs whenever one or more of its agents reach a goal location of some agent from Team-A, i.e., if there exists $a \in A$ and $b \in B$ such that $\alpha_i(b) = \alpha_{A+}(a)$ at some time step i.

Definition 3. *An* AMAPF state-space *for input* Π *is a directed graph* $\mathcal{D}(\Pi)$ *whose nodes are configurations, the source of* $\mathcal{D}(\Pi)$ *corresponds to* α_0. *Sinks of* $\mathcal{D}(\Pi)$ *correspond to winning configurations. Each path from the source towards any sink is a prefix of a game-play.*

Definition 4. *An AMAPF winning strategy for Team-A for input* Π *is a sub-graph* $\mathcal{W}(\Pi)$ *of* $\mathcal{D}(\Pi)$ *such that each path from the source to any sink in* $\mathcal{W}(\Pi)$ *is a winning game-play for Team-A.*

III. Computational Complexity

AMAPF can be regarded as a two-player game formally defined as the following decision problem:

Definition 5. *Given an AMAPF instance* Π*, does there exist a winning strategy for Team-A?*

Our hypothesis was that it belongs to the same complexity class as generalized versions of other two-player games. While single-player games/puzzles are often PSPACE-complete, such as hex [25], [26], Sokoban [27], or Rush Hour [28], two-player alternation causes an increase of the complexity towards EXPTIME-completeness, as in the case of chess [29] or go [30]. It is shown in [17] that the symmetric variant of AMAPF is PSPACE-hard and in EXPTIME, but does not prove whether it belongs to PSPACE.

A. Reduction from Formula Game

For showing that AMAPF is EXPTIME-hard, we use the following two player Formula Game (FG) known to be EXPTIME-complete from [31] (denoted G6 in the reference).

Definition 6. *A position in* Formula Game *is a tuple* $(t, \varphi(X, Y), \epsilon)$*, where* $t \in \{I, II\}$ *indicates a player on the turn,* φ *is a formula in CNF, and* ϵ *is a* (X, Y)*- truth evaluation.*

Player I (II) in FG moves by changing the truth value of at most one variable in X (Y). Player I wins if φ ever becomes *true* under ϵ. Specifically, player II cannot move from $(\text{II}, \varphi, \epsilon)$ to (I, φ, ϵ') if φ is satisfied under ϵ, but player I can always move from $(\text{I}, \varphi, \epsilon)$ to $(\text{II}, \varphi, \epsilon')$ provided that ϵ and ϵ' differ in truth evaluation to at most one variable in X. Note that not changing anything at all is also a valid move.

The following instance Ψ of FG is an example illustrating the proof: $\Psi = (\text{I}, \varphi(\{X_1, X_2, X_3\}, \{Y_1, Y_2, Y_3\}), \epsilon)$, where

$$\varphi = (X_1 \vee X_2 \vee \bar{Y}_3) \& (\bar{X}_1, \vee \bar{X}_3 \vee Y_1) \\ \& (\bar{X}_1 \vee X_3 \vee Y_3) \& (\bar{X}_2 \vee \bar{Y}_2), \quad (1)$$

$\epsilon(X_1) = \epsilon(X_3) = true$ and $\epsilon(X_2) = \epsilon(Y_1) = \epsilon(Y_2) = \epsilon(Y_3) = false$.

We now present a polynomial-time reduction from FG to AMAPF. Given an instance $(t, \varphi(X, Y), \epsilon)$ of FG, we construct a corresponding instance of AMAPF. Let $|X| = n_1$, $|Y| = n_2$ and let m denote the number of clauses in φ.

Figure 1 depicts a simplified illustration of the AMAPF instance that matches Ψ. For brevity, agents are often identified by their initial node.

1) The big picture: The two players in FG are represented by Team-A and Team-B in AMAPF denoted by red and green nodes, respectively, in all the illustrations. Given the variables in X and Y in φ, there are two star subgraphs with grey background in Figure 1. The leaves of these two subgraphs are denoted as variables and their negations, and

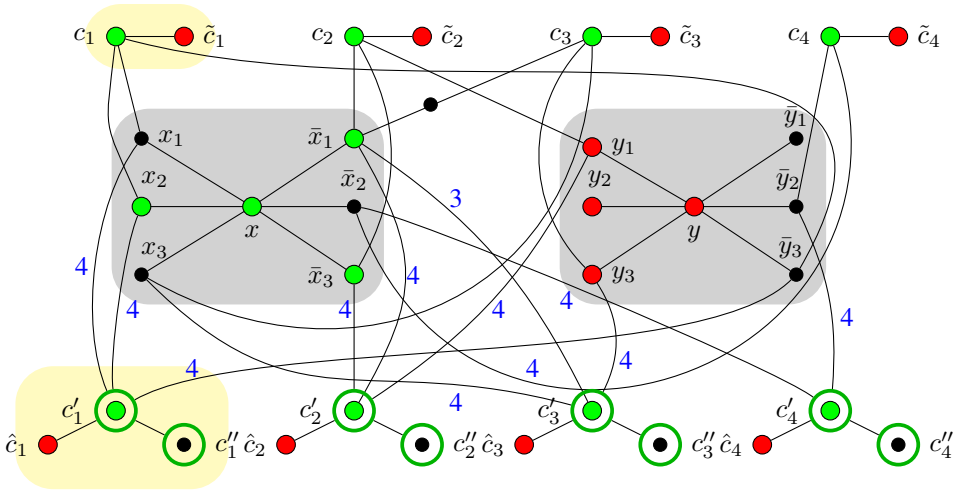

Fig. 1. AMAPF instance constructed from formula (1). Green nodes represent agents of Team-A with goals at green circles. Agents of Team-B are the red nodes. Unoccupied nodes are black. Blue numbers along some arcs indicate how many nodes between the endpoints are not displayed. If there is no such number, it means that the endpoints form a regular edge.

the placement of the agents in these leaves models the truth value of corresponding variables. It is therefore important that exactly one of nodes x_i and $\bar{x}_i$ is occupied at any time (Section III-A3 explains how is this property achieved). As the players in FG take turns, the agents in the corresponding star graph also move so that the agent currently occupying x moves to say x_i, and the agent occupying $\bar{x}_i$ moves to x. Such a movement represents switching the truth value of X_i from true to false.

Each clause is represented by two subgraphs denoted by yellow background in Figure 1. Agents in c_j aim to reach c'_j, while the agent initially at c'_j holds the target for the agent in c_j. Once φ is satisfied, all the agents from c_j may simultaneously start approaching their targets via vacant nodes in the two star graphs. This is the most crucial aspect of the construction. As we show further, if the agents in c_j start their advancement prematurely, the adversarial red agents from Team-B can prevent them from reaching the targets.

2) Gadget construction: We now describe individual parts of the construction in detail.

a) X-variable gadget: Let us create a star graph whose number of leaves equals two times the number of variables in X, and its central node is occupied by an agent from Team-A. The leaves are denoted x_i and $\bar{x}_i$ for each variable X_i, $i = 1 \dots n_1$ in X. If $\epsilon(X_i) = \mathit{true}$, a Team-A's agent is placed in node $\bar{x}_i$, otherwise the agent is placed in node x_i. The distribution of agents in this gadget represents a current evaluation of individual variables in X. We assume that the initial positions of these agents are also their targets, but they can be distributed arbitrarily in the star graph. The targets are not explicitly marked for simplicity of the illustrations. Each pair of nodes x_i and $\bar{x}_i$ is connected to u_i and v_i by an edge. The gadget further contains agents a_{i1} and a_{i2} from Team-A with goals at v_i and u_i, and agents b_i, b_{i1} and b_{i2} from Team-B distributed in the gadget as illustrated in Fig. 3. The grey area in the figure marks the part of the X-variable gadget that represents variable X_i.

b) Y-variable gadget: For variables in Y, we create a similar star graph gadget with nodes y, $y_1, \dots, y_{n_2}$, $\bar{y}_1, \dots, \bar{y}_{n_2}$ with a distribution of agents that corresponds to initial truth values of variables in Y. Additionally, there is an edge $\{w_j, w'_j\}$ such that w'_j is adjacent to both y_i and $\bar{y}_i$, $i = 1, \dots, n_2$. Nodes y, w_j and w'_j are so called *shortcut nodes* whose function will be clear in Sect. III-A3. The grey area in Fig. 4 highlights the part of the gadget representing variable Y_i.

For example, the instance in Figure 1 depicts the evaluation of variables $\epsilon(X_1) = \epsilon(X_3) = \mathit{true}$, while the remaining variables are set to *false*. It is expected that the central nodes x and y are always occupied and that exactly one node in each pair x_i and $\bar{x}_i$ (y_i and $\bar{y}_i$) is occupied. If the agent located at x moves to $\bar{x}_2$ and at the same time, the agent located at x_2 moves to x, it indicates that the truth value of variable X_2 was switched to *true*.

c) Clause gadgets: For each clause C_j, there are nodes c_j, c'_j, c''_j, $\tilde{c}_j$, $\hat{c}_j$, and edges $\{c_j, \tilde{c}_j\}$, $\{c'_j, c''_j\}$, and $\{c'_j, \hat{c}_j\}$. An agent from Team-A is initially placed at each c_j and c'_j with goals at c'_j and c''_j, respectively. An agent from Team-B is placed at $\tilde{c}_j$ and $\hat{c}_j$. If variable X_i (Y_i) occurs in clause C_j, nodes c_j and c'_j are connected by a path of length $m+2$ passing through x_i (y_i) in case the variable appears as a positive literal, and through $\bar{x}_i$ ($\bar{y}_i$), if the variable appears as a negative literal. This path is referred to as the *transit path*. It is essential that all transit paths have equal lengths, which ensures that agents c_j must initiate the movement towards their targets at the same time, as explained later. If a literal X_i occurs in only one

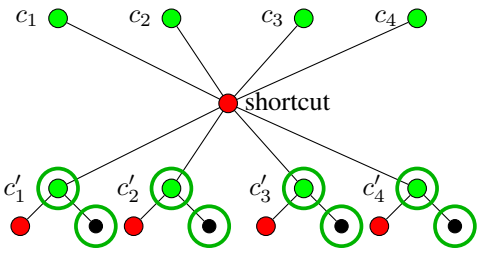

Fig. 2. A shortcut node

clause C_j, c_j is adjacent to x_i, and the length of the path from x_i to c'_j is $m+1$. If the literal X_i occurs in 2 clauses C_k and C_ℓ, c_k is again adjacent to x_i, but there is a path of length 2 between c_ℓ and x_i, and of length m between x_i and c'_ℓ. There is also another path from each c_j to c''_k ($k \neq j$) of length $m+1$ referred to as the *chasing path*, which does not contain any agents, and is not displayed in Figure 1 in order to keep the illustration simpler. Transit and chasing paths are disjoint except for the first node.

The idea is that once the arrangement of agents in variable gadgets represents a satisfying evaluation of φ, all Team-A's agents initially placed at nodes c_j are able to pass through nodes in variable gadgets, and eventually reach their goals at c'_j. All these agents must commence their movement along suitable transit paths together at the same time without a delay. If an agent from c_k does not reach its goal at the earliest possible time along with the remaining corresponding agents from other clause gadgets, a Team-B's agent initiated at some $\tilde{c}_\ell$ manages to reach node c''_k via a chasing path. As c''_k is a goal node, its occupation of an agent from Team-B leads to an immediate loss of Team-A. Thus, as long as all agents c_j remain at their initial positions, all chasing paths are blocked, and $\tilde{c}_j$ cannot move. Once some c_j is vacated, its chasing path is clear and all nodes c''_k can be approached by $\tilde{c}_j$ via the chasing path. That is the reason why all the agents c_j must start moving at the same time and cannot be delayed at any point during their advancement. As soon as the agent from c_j is arriving at c'_j, the agent originally at c'_j can safely move to its goal at c''_j. If Team-A agents from clause gadgets start moving once there is a satisfying evaluation, no matter how Team-B moves, it is impossible for them to prevent Team-A from reaching their targets and win. Note that the agent from c'_j cannot leave its initial node except for the last step. If they vacate c'_j earlier, the red agent that lurks at $\hat{c}_j$ will take c'_j, and Team-B instantly wins.

3) Dealing with inconsistency: In order to guarantee correct movement in the constructed AMAPF instance, we introduce an auxiliary nodes called *shortcut* nodes (see Figure 2).

Definition 7. *A* shortcut *is a node adjacent to every c_j and c'_j, $j = 1, \ldots, m$, initially occupied by an agent from Team-B.*

The shortcut nodes prevent Team-B's agents in the variable gadgets to move illegally. If a shortcut is clear, all agents c_j can gradually pass through it and reach their targets earlier than via transit paths. Team-B thus keeps the shortcuts occupied.

The behavior of agents described in the previous section is ideal, but it may seem that nothing forces the agents to relocate in accordance with the turns of the FG instance. We analyze potential inconsistent moves and clarify the technical features of the construction that aim to avert them.

- *Node x becomes empty.* If x ever becomes vacant, Team-A immediately loses as x is a goal node of one of the agents in the variable gadget and is adjacent to a node with an agent from Team-B.
- *Node y becomes empty.* As y is also a shortcut, its clearing causes that all Team-A's agents from c_j can advance through it towards their goals.
- *An agent from c_j enters a variable gadget when the arrangement of agents does not correspond to satisfying evaluation of φ.* In such a case, there must be a clause C_k that is not satisfied, and thus the agent from c_k cannot proceed towards its goal. When this happens, Team-B's agent from $\tilde{c}_j$ moves to c_j and rushes towards c''_k via the suitable chasing path, and c''_j will be vacant when the agent arrives there, causing an immediate loss of Team-A.
- *Both x_i and $\bar{x}_i$ for some $i = 1, \ldots, n_1$ become vacant.* This models evaluating both X_i and $\bar{X}_i$ to true. It occurs when two agents from the variable gadget occupy another x_j and $\bar{x}_j$, or when an agent flees to an available transit path. Such a behavior is avoided due to agents b, b_1 and b_2 depicted in the detailed view in Fig. 3. Node b_i would step towards say u'_i, and there is nobody to block him from entering target u_i except for a_{i2}, but if a_{i2} leaves its position, b_{i2} has a clear path to one of c''_j.
- *Both y_i and $\bar{y}_i$ for some $i = 1, \ldots, n_2$ is vacant.* This corresponds to $\epsilon(Y_i) = \epsilon(\bar{Y}_i) = true$, and happens when agents from Y-variable gadget attempt to seal some y_j as well as $\bar{y}_j$. Figure 4 explains how Team-B is defeated.
- *Agents in variable gadgets move even after agents from c_j started passing through them.* This can be an issue only when some literal is present in two clauses. The two auxiliary gadgets below forbid such a movement.

The following gadgets are not necessary when it comes to understanding the logic behind the construction. They are not displayed as their presence would make the figures less comprehensible. The reason why they are present in the construction is merely to prevent movement of agents in the star subgraphs of variable gadgets once the agents from clause gadget started approaching their targets.

a) X-auxiliary gadgets: Let X_i be a literal that appears in clauses C_j and C_k, and $\bar{X}$ appears in C_ℓ. Without a loss of generality assume that c_j is adjacent to x_i, and that c_k is connected to x_i by a path of length 2. Next, let u and v be arbitrary inner nodes on paths from x_i to c'_j and from $\bar{x}_i$ to c'_ℓ, respectively. There is a node p, which is an initial as well as a goal node of an agent $a \in A$, and edges $\{u, p\}$ and $\{v, p\}$. Further, we create a shortcut node p' initially occupied by an agent $b \in B$ also adjacent to u and v. Assume that $\bar{x}_i$ is vacant and was entered by the agent from c_ℓ, and the agent from c_k

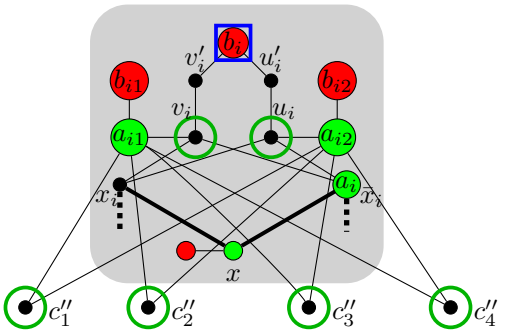

Fig. 3. A part of the X-variable gadget representing variable X_i. Agent b_i is located in a shortcut node (blue square), and makes sure that either x_i or $\bar{x}_i$ is occupied by an agent from Team-A at every time step. If b_i moves say to v'_i, agent a_i takes v_i, and Team-A wins because the shortcut node is vacant. If both x_i and $\bar{x}_i$ are vacant and b_i moves to v'_i, v_i must be blocked by a_{i1}, but then b_{i1} is free to capture some c''_j, and Team-B wins.

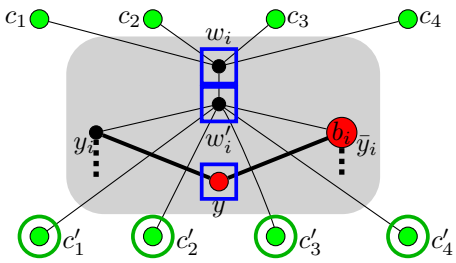

Fig. 4. A detailed view of one part of the Y-variable gadget. The edge $\{w_i, w'_i\}$ has the same function as a shortcut node. If both y_i and $\bar{y}_i$ become vacant (illegally), Team-A wins, because agents c_j start flowing through $\{w_i, w'_i\}$ to their goals. Otherwise, the agent b_i has an opportunity to stop them by entering w'_i, and Team-B wins, because c_js do not arrive on time.

made a step towards x_i, intending to wrongfully pass via x_i, which could be emptied in the next step (x_j is passing via another path). Agent a takes node v, otherwise b could block it, and thereby prevent c_ℓ from reaching its goal. Agent b is now able to block u and thwart the plans of c_k to enter x_i in the next step, if the agents in X-variable gadget empty x_i at the same time. If b leaves p' before the movement of clause agents begins, Team-A wins since p' is a shortcut.

b) Y-auxiliary gadgets: Let Y_i be a literal appearing in clauses C_j and C_k, and c_j is adjacent to y_i. Assume that y_i is vacant, and c_k makes a step towards it, but c_j does not pass y_i. In the next step, agents in the Y-variable gadget could obstruct y_i. In order to avert such a move, there is a node q with an agent $a \in A$ adjacent to y_i. If needed, a can reserve y_i for the agent from c_k, and consequently go towards its goal via a separate path. If such a reservation is not needed, a has an alternative path to its goal. For the sake of disabling a from entering y_i earlier, we put a neighbor q' of q with an agent $b \in B$, and a path from q to c''_k of length $m+1$. A premature clearing of q by a causes that b captures c''_k and Team-A loses.

It follows from the discussion above that agents are forced to move in accordance with variable evaluation in FG, and deviations lead to immediate loss of their team.

B. EXPTIME-completeness

It can be verified that the size of the resulting AMAPF instance is polynomial in the size of the original instance of FG. For establishing the proof of EXPTIME-completeness of AMAPF, we need the following proposition.

Proposition 1. *The decision whether there exists a winning strategy for Team-A in a given instance* Π *is in EXPTIME.*

Proof. A state in AMAPF is defined by the locations of agents and the information about which team is on the turn. As the agents are not anonymous, the size of the state space is $2\binom{n}{k}k!$, and since there is no more than n agents, we get the state space of size $2n!$. All languages recognizable by deterministic algorithms in $\mathcal{O}(n!)$ are in EXPTIME since $\mathcal{O}(n!) \subseteq \mathcal{O}(2^{n^2})$.

A classic graph search algorithm searches the state space and uses a global list of visited states to avoid visiting one state more than once. Hence, it determines whether there exists a winning strategy for the instance Π in exponential time. □

We show that the construction described in Section III-A is a polynomial-time reduction from FG to AMAPF.

Proposition 2. *The decision whether there exists a winning strategy for Team-A in an instance* Π *is EXPTIME-hard.*

Proof. First, we show that if a given instance of FG has a winning strategy for player I, then there exists a winning strategy for Team-A in the corresponding AMAPF instance. Each evaluation corresponds to a distribution of agents in variable gadgets. Switching from $\epsilon(X_i) = true$ to $\epsilon(X_i) = false$ corresponds to moving the agent currently located at x to x_i, and a simultaneous transition of the agent currently at $\bar{x}_i$ to node x. Let us consider a sequence of evaluations of variables that is consistent with turns in FG. There is an analogous movement of agents in the variable gadgets. Once the formula becomes true, all agents from c_j, $j = 1, \ldots, m$ can start moving towards their goals via nodes corresponding to literals causing the clauses to be satisfied. At this point, if agents originally located in variable gadgets move, such a movement is not relevant as it cannot influence the agents passing through the gadgets. The evaluation of variables based on the placement of agents thus remains constant after the clause agents start passing. If all the clause agents advance towards their goals via the shortest paths, they all arrive in the goals at the same time, and the agents originally located at c'_j can move to their goal at c''_j in the last time step. Finally, the remaining Team-A's agents rearrange their positions to their goals within the variable gadget.

Let us consider that there exists a winning strategy for Team-A in the constructed AMAPF instance. The only possibility to reach the destinations is via variable gadgets. Agents in variable gadgets rearrange themselves as Team-A and Team-B take turns, which models the truth evaluation changes made by players I and II in FG. Once there is a path from every c_j to c'_j via a vacant node in one of the variable gadgets, and so the agents in c_j can start moving, the formula φ is satisfied. □

Propositions 1 and 2 imply that AMAPF is EXPTIME-complete.

IV. Concluding Remarks

We studied Adversarial Multi-Agent Path Finding (AMAPF), a generalization of MAPF that adds an adversary, and analyzed its computational complexity. It turned out that adding the adversarial element greatly increases the theoretical computational complexity of the problem. We have shown that AMAPF is EXPTIME-complete.

Our proof technique is based on the reduction of a two-player game, where players compete in satisfying a propositional formula by changing the truth value assignments of variables, to AMAPF. The proof also relies on the common MAPF movement rule that allows train-like movement of agents where an agent can enter a vertex being simultaneously vacated by another agent of the same team (without edge collisions).

Whether the result holds for the AMAPF variant without the train-like movements remains an open question.

A possible generalization of AMAPF may consider capacities of vertices and edges where avoidance of low level collisions are granted if the capacity is not exceeded by agents occupying a given vertex or edge. Such generalization is particularly relevant in military applications of AMAPF.

Another interesting area of future research is an investigation of instances on special graphs, such as grid graphs or bi-connected graphs. From the practical perspective, there is a promising research potential in the development and analysis of specific algorithmic strategies for both teams in such special graphs.

References

[1] D. Silver, "Cooperative pathfinding," in *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*. AAAI Press, 2005, pp. 117–122.

[2] M. R. K. Ryan, "Exploiting subgraph structure in multi-robot path planning," *J. Artif. Intell. Res.*, vol. 31, pp. 497–542, 2008.

[3] G. Röger and M. Helmert, "Non-optimal multi-agent pathfinding is solved (since 1984)," in *Multiagent Pathfinding, Papers from the 2012 AAAI Workshop, MAPF@AAAI 2012*, vol. WS-12-10. AAAI Press, 2012.

[4] T. S. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press, 2010.

[5] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," in *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. IJCAI/AAAI, 2011, pp. 662–667.

[6] R. Luna and K. E. Bekris, "Push and swap: Fast cooperative pathfinding with completeness guarantees," in *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, 2011*. IJCAI/AAAI, 2011, pp. 294–300.

[7] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 219, pp. 40–66, 2015.

[8] P. Surynek, A. Felner, R. Stern, and E. Boyarski, "Efficient SAT approach to multi-agent path finding under the sum of costs objective," in *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, ser. Frontiers in Artificial Intelligence and Applications, vol. 285. IOS Press, 2016, pp. 810–818.

[9] D. Ratner and M. K. Warmuth, "Nxn puzzle and related relocation problem," *J. Symb. Comput.*, vol. 10, no. 2, pp. 111–138, 1990.

[10] P. Surynek, "An optimization variant of multi-robot path planning is intractable," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press, 2010, pp. 1261–1263.

[11] J. Yu and S. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," *Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI 2013*, pp. 1443–1449, 01 2013.

[12] B. Nebel, "On the computational complexity of multi-agent pathfinding on directed graphs," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, no. 1, pp. 212–216, Jun. 2020.

[13] K. Solovey and D. Halperin, "k-color multi-robot motion planning," *The International Journal of Robotics Research*, vol. 33, pp. 82 – 97, 2014.

[14] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," *CoRR*, vol. abs/1612.05693, 2016. [Online]. Available: http://arxiv.org/abs/1612.05693

[15] R. Barták, M. Ivanová, and J. Švancara, "Colored multi-agent path finding: Solving approaches," *The International FLAIRS Conference Proceedings*, vol. 34, Apr. 2021. [Online]. Available: https://journals.flvc.org/FLAIRS/article/view/128535

[16] M. Ivanová and P. Surynek, "Adversarial cooperative path-finding: A first view," in *Late-Breaking Developments in the Field of Artificial Intelligence, 2013*. AAAI, 2013.

[17] ——, "Adversarial cooperative path-finding: Complexity and algorithms," in *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014*. IEEE Computer Society, 2014, pp. 75–82.

[18] M. Ivanová, P. Surynek, and K. Hirayama, "Area protection in adversarial path-finding scenarios with multiple mobile agents on graphs - A theoretical and experimental study of strategies for defense coordination," in *Proceedings of the 10th International Conference on Agents and Artificial Intelligence, ICAART 2018, Volume 1, 2018*. SciTePress, 2018, pp. 184–191.

[19] Y. Elmaliach, N. Agmon, and G. Kaminka, "Multi-robot area patrol under frequency constraints," *Annals of Mathematics and Artificial Intelligence*, vol. 57, pp. 293–320, 04 2007.

[20] B. Bosanský, O. Vanek, and M. Pechoucek, "Strategy representation analysis for patrolling games," in *Game Theory for Security, Sustainability, and Health, Papers from the 2012 AAAI Spring Symposium*, ser. AAAI Technical Report, vol. SS-12-03. AAAI, 2012.

[21] P. J. Villacorta and D. A. Pelta, "Exploiting adversarial uncertainty in robotic patrolling: A simulation-based analysis," in *14th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU 2012*, vol. 300. Springer, 2012, pp. 529–538.

[22] N. Basilico, N. Gatti, and F. Amigoni, "Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder," *Artificial Intelligence*, vol. 184–185, p. 78–123, 06 2012.

[23] M. Abaffy, T. Brázdil, V. Řehák, B. Bošanský, A. Kučera, and J. Krčál, "Solving adversarial patrolling games with bounded error: (extended abstract)," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, ser. AAMAS '14. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2014, p. 1617–1618.

[24] N. Agmon, G. Kaminka, and S. Kraus, "Multi-robot adversarial patrolling: Facing a full-knowledge opponent," *J. Artif. Intell. Res. (JAIR)*, vol. 42, 01 2014.

[25] S. Even and R. E. Tarjan, "A combinatorial problem which is complete in polynomial space," *J. ACM*, vol. 23, no. 4, p. 710–719, Oct. 1976.

[26] S. Reisch, "Hex ist pspace-vollständig," *Acta Informatica*, vol. 15, pp. 167–191, 1981.

[27] J. Culberson, "Sokoban is pspace-complete," 1997.

[28] G. W. Flake and E. B. Baum, "Rush hour is pspace-complete, or "why you should generously tip parking lot attendants"," *Theoretical Computer Science*, vol. 270, no. 1-2, pp. 895–911, 2002.

[29] A. S. Fraenkel and D. Lichtenstein, "Computing a perfect strategy for $n \times n$ chess requires time exponential in n," *Journal of Combinatorial Theory, Series A*, vol. 31, no. 2, pp. 199 – 214, 1981.

[30] J. Robson, "The complexity of go." in *IFIP Congress Series*, vol. 9, 01 1983, pp. 413–417.

[31] L. Stockmeyer and A. Chandra, "Provably difficult combinatorial games," *SIAM J. Comput.*, vol. 8, pp. 151–174, 1979.