Artificial Intelligence 328 (2024) 104063

Contents lists available at ScienceDirect

Artificial Intelligence

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

The computational complexity of multi-agent pathfinding on directed graphs

Bernhard Nebel

Department of Computer Science, University of Freiburg, Georges-Koehler-Allee, 79110, Freiburg, Germany

ARTICLE INFO

Keywords:
Multi-agent pathfinding
diMAPF
MAPF
Pebble motion on graphs
Cooperative pathfinding
Motion planning on graphs
Computational complexity
Permutation group theory

ABSTRACT

While the non-optimizing variant of multi-agent pathfinding on undirected graphs is known to be a polynomial-time problem since almost forty years, a similar result has not been established for directed graphs. In this paper, it will be shown that this problem is NP-complete. For strongly connected directed graphs, however, the problem is polynomial. And both of these results hold even if one allows for synchronous rotations on fully occupied cycles. Interestingly, the results apply also to the so-called graph motion planning feasibility problem on directed graphs.

1. Introduction

Multi-agent pathfinding (MAPF), also called *pebble motion on graphs* or *cooperative pathfinding*, is the problem of deciding the existence of or generating a collision-free movement plan for a set of agents moving on a graph [1,2]. Because of its relevance to topics such as warehouse logistics [3], air traffic coordination [4], and video games [5], this problem has received much interest in recent years. An example is provided in Fig. 1.

Is it possible to transform the left configuration into the right one? It turns out that in the example above, this is not possible.

Kornhauser et al. [6] had shown already almost 40 years ago that deciding MAPF is a polynomial-time problem and movement plans have cubic length in the size of the graph, provided only one agent can move at each time step to an adjacent node. However, what happens if we consider directed graphs? Interestingly, for this problem, which we will call diMAPF, the computational complexity was unknown for a long time, although it is a very plausible and relevant variation of the original problem.

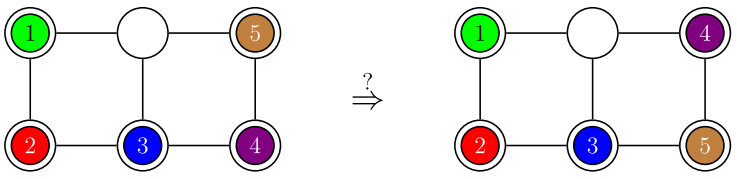

Fig. 1. Multi-agent pathfinding example. Agents are signified by numbered disks occupying nodes in a graph. At each time-point a single agent can move to an adjacent unoccupied node.

E-mail address: nebel@uni-freiburg.de.

https://doi.org/10.1016/j.artint.2023.104063
Received 14 March 2023; Received in revised form 29 October 2023; Accepted 25 December 2023

Available online 9 January 2024

Only recently, NP-completeness of the problem has been established. In two conference publications, which form the basis of this journal article, NP-hardness [7] and membership in NP [8] were proved, even if one allows for synchronous rotations on fully occupied cycles in addition to simple movements. One can explain these results as follows. The possibility of ending up in dead ends in directed acyclic graphs leads to a combinatorial nature of the problem, giving rise to NP-hardness. On strongly connected directed graphs, on the other hand, diMAPF is very similar to MAPF on undirected graphs allowing for polynomial sized solutions. Combining these two results leads more or less directly to the NP-completeness result.

In addition to the above stated results, we will also show that strongly connected directed graphs admit a polynomial decision procedure, regardless of what kind of movements we allow. Further, we will have a look at how the results can help to settle some open questions concerning the motion planning problem on directed graphs [9].

The rest of the paper is structured as follows. In the next section, related work will be surveyed. In Section 3, we will introduce the necessary notation and terminology that is needed for proving the results of the paper. In Section 4, a lower bound for diMAPF will be proven and an NP-completeness result for diMAPF an DAGs will be derived (Theorem 2). The following two sections analyze diMAPF on strongly connected graphs, first for simple moves, and then for synchronous rotations. Based on the results in these two sections, we will show in Section 7 that diMAPF on strongly connected digraphs is a polynomial-time problem (Theorem 19), while diMAPF is NP-complete in general (Theorem 21), regardless of whether synchronous rotations are allowed or not. The connection to a related problem, the motion planning feasibility problem on directed graphs, is then analyzed in Section 8. Finally, in Section 9, we conclude and discuss the results.

2. Related work

A special case of the MAPF problem is the 15-puzzle, a sliding puzzle on a 4×4 grid graph with 15 markers. Johnson and Story [10] showed already in the 19th century that solvability of such a puzzle, even generalized to the $n \times m$-case, can be reduced to the question of whether the corresponding permutation is even, which is a polynomial-time problem. Wilson [11] generalized this result to arbitrary biconnected graphs and proved that solvability is still a polynomial-time problem, the condition is slightly more involved than in the above case, though. In particular, he showed that problem instances on bipartite graphs have a solution if, and only if, the corresponding permutation is even. Applying this to our initial example in Fig. 1 gives immediately a negative result.

The seminal paper by Kornhauser et al. [6] generalized the above results to arbitrary graphs and provided a polynomial-time algorithm for generating movement plans and a polynomial upper bound for such plans. However, it took quite a while until this result was recognized in the community [12]. The optimizing variant of this problem had been shown to be NP-complete soon after the initial result [13,14].

Later on, variations of the problem have been studied [2]. It is obvious that all agents that move to different empty nodes in one time step could move in parallel, which may lead to shorter plans. Assuming tight coordination between the agents, one can also consider train-like movements, where only the first agent moves to an empty node and the others follow in a chain, all in one time step [15–17]. Taking this one step further, synchronous rotations of agents on a cycle without any empty nodes have been considered [18–20].

Concerning plan existence and polynomial plan length, parallel and train-like movements do not make a difference to the case when only simple moves are permitted. Synchronous rotations are a different story altogether, however. There are problem instances which cannot be solved using only simple moves, but which are solvable when synchronous rotations are allowed. The additional degree of freedom does not add to the computational complexity, however. MAPF with synchronous rotations is still a polynomial-time problem [20].

Optimizing wrt. different criteria turned out to be NP-complete [17,19] for all kinds of movements, and this holds even for planar and grid graphs [21–23]. Additionally, it was shown that there are limits to the approximability of the optimal solution for makespan optimizations [24].

The mentioned results all apply to undirected graphs only. However, a couple of years ago, researchers also started to look into the case of directed graphs [25,26], and it has been proved that diMAPF can be decided in polynomial time, provided the directed graph is strongly biconnected and there are at least two unoccupied vertices [27,28]. Similar to the undirected case [6], plan length is bounded by $O(n^3)$. Recently, Ardizzoni at al. [29] generalized this result and presented an algorithm that can check the solvability of instances on strongly connected graphs with two unoccupied nodes. However, they did not provide a bound for the plans the algorithm generates.

Wu and Grumbach [9] generalized the *robot movement problem* on an undirected graph as introduced by Papadimitriou et al. [30] to directed graphs. The robot movement problem is the problem of finding a plan to move one robot from a vertex s to a vertex t, whereby anonymous mobile obstacles on vertices can be moved around but are not allowed to collide. In the conclusion of their paper [9], they suggested to study the more difficult problem when there is more than one robot, which is a problem slightly more general than diMAPF.

3. Notation and terminology

In this section, we will introduce basic concepts from graph theory and permutation groups, and will formally define MAPF and diMAPF. It is assumed that the reader is familiar with basic notions from computational complexity theory, as e.g. presented in the book by Garey and Johnson [31] or any textbook on theoretical computer science.

3.1. Graph theory

A *graph* G is a tuple (V, E) with $E \subseteq \{\{u, v\} \mid u, v \in V\}$. The elements of V are called *nodes* or *vertices* and the elements of E are called *edges*. A *directed graph* or *digraph* D is a tuple (V, A) with $A \subseteq V^2$. The elements of A are called *arcs*. Graphs and digraphs with $|V| = 1$ are called *trivial*. We assume all graphs and digraphs to be *simple*, i.e., not containing any self-loops of the form $\{u\}$, resp. (u, u). Given a digraph $D = (V, A)$, the *underlying graph* of D, in symbols $\mathcal{G}(D)$, is the graph resulting from ignoring the direction of the arcs, i.e., $\mathcal{G}(D) = (V, \{\{u, v\} \mid (u, v) \in A\})$.

Given graphs $G = (V, E)$ and $G' = (V', E')$, G' is called *sub-graph* of G if $V \supseteq V'$ and $E \supseteq E'$. Similarly, for digraphs $D = (V, A)$ and $D' = (V', A')$, D' is a *sub-digraph* if $V \supseteq V'$ and $A \supseteq A'$. $D - v$ denotes a sub-digraph of D where the node v and all its incident arcs have been removed, i.e., $D - v = (V - \{v\}, A - \{(v, x), (x, v) | x \in V\})$. Similarly, $G - v = (V - \{v\}, E - \{\{v, x\} | x \in V\})$. Set union and intersection on graphs and digraphs are defined by taking the component-wise union or intersection, e.g., for digraphs $D = (V, A)$ and $D' = (V', A')$ we have $D \cup D' = (V \cup V', A \cup A')$.

A *path* in a graph $G = (V, E)$ is a non-empty sequence of vertices of the form $v_0, v_1, \ldots, v_k$ such that $v_i \in V$ for all $0 \leq i \leq k$, $v_i \neq v_j$ for all $0 \leq i < j \leq k$, and $\{v_i, v_{i+1}\} \in E$ for all $0 \leq i < k$. A *cycle* in a graph $G = (V, E)$ is a non-empty sequence of vertices $v_0, v_1, \ldots, v_k$ with $k \geq 3$ such that $v_0 = v_k$, $\{v_i, v_{i+1}\} \in E$ for all $0 \leq i < k$ and $v_i \neq v_j$ for all $0 \leq i < j < k$. Sometimes, we view cycles and paths not as sequences of nodes, but as the sub-graphs formed by the nodes in the sequence and the respective edges.

In a digraph $D = (V, A)$, *path* and *cycle* are similarly defined, except that the direction of the arcs has to be respected. This means that $(v_i, v_{i+1}) \in A$ for all $0 \leq i < k$. Further, the smallest cycle in a digraph has 2 nodes instead of 3. A digraph that does not contain any cycle is called *directed acyclic graph (DAG)*. A digraph that consists solely of a cycle is called *cycle digraph*.

A graph $G = (V, E)$ is *connected* if there is a path between each pair of vertices. A connected graph that does not contain a cycle is called *tree*. Node v in a connected graph G is called *articulation* if $G - v$ is not connected. A digraph $D = (V, A)$ is *weakly connected*, if the underlying graph $\mathcal{G}(D)$ is connected. It is *strongly connected*, if for every pair of vertices $u, v \in V$, there is a path in D from u to v. A *strong articulation* in a strongly connected digraph D is a node such that $D - v$ is not strongly connected. An *articulation* of a weakly connected digraph D is a node v, such that $D - v$ is not weakly connected, i.e., it is an articulation of the underlying undirected graph $\mathcal{G}(D)$. Note that it is well known that a graph with n nodes cannot have more than $n - 2$ articulations, because the leaves of a maximal sub-tree of the graph cannot separate the graph. A strongly connected digraph that does not have any articulation is also called *strongly biconnected digraph* [9]. The smallest strongly connected digraph is the trivial digraph. The *strongly connected components* of a digraph $D = (V, A)$ are the maximal sub-digraphs $D_i = (V_i, A_i)$ that are strongly connected.

3.2. Permutation groups

In order to be able to establish a polynomial bound for diMAPF movement plans containing synchronous rotations, we introduce some background on permutation groups.[1]

A *permutation* is a bijective function σ over a set X. In what follows, we assume X to be finite.

A permutation has *degree* d if it exchanges d elements and fixes the rest of the elements in X. We say that a permutation is an *m-cycle* if it exchanges elements $x_1, \ldots, x_m$ in a cyclic fashion, i.e., $\sigma(x_i) = x_{i+1}$ for $1 \leq i < m$, $\sigma(x_m) = x_1$ and $\sigma(y) = y$ for all $y \notin \{x_i\}_{i=1}^m$. Such a cyclic permutation is written as a list of elements, i.e., $(x_1\ x_2\ \cdots\ x_m)$. A 2-cycle is also called *transposition*. A permutation can also consist of different disjoint cycles. These are then written in sequence, e.g., $(x_1\ x_2)(x_3\ x_4)$.

The composition of two permutations σ and τ, written as $\sigma \circ \tau$ or simply $\sigma\tau$, is the function mapping x to $\tau(\sigma(x))$.[2] This operation is associative because function composition is. The special permutation ϵ, called *identity*, maps every element to itself. Further, σ^{-1} is the *inverse* of σ, i.e., $\sigma^{-1}(y) = x$ if and only if $\sigma(x) = y$. The k-fold composition of σ with itself is written as σ^k, with $\sigma^0 := \epsilon$. Similarly, $\sigma^{-k} := (\sigma^{-1})^k$. We also consider the *conjugate of σ by τ*, written as σ^τ, which is defined to be $\tau^{-1}\sigma\tau$. Such conjugations are helpful in creating new permutations out of existing ones. We use exponential notation as in the book by Mulholland [32]: $\sigma^{\alpha+\beta} := \sigma^\alpha \sigma^\beta$ and $\sigma^{\alpha\beta} := (\sigma^\alpha)^\beta$.

A set of permutations closed under composition and inverse forms a *permutation group* with $\circ$ as the *product operation* (which is associative), $\cdot^{-1}$ being the *inverse operation*, and ϵ being the *identity element*. Given a set of permutations $\{g_1, \ldots, g_i\}$, we say that $\mathbf{G} = \langle g_1, \ldots, g_i \rangle$ is *the permutation group generated by* $\{g_1, \ldots, g_i\}$ if $\mathbf{G}$ is the group of permutations that results from product operations over the elements of $\{g_1, \ldots, g_i\}$. We say that $\sigma \in \mathbf{G} = \langle g_1, \ldots, g_i \rangle$ is k-expressible if it can be written as a product over the generators using $< k$ product operations. The *diameter* of a group $\mathbf{G} = \langle g_1, \ldots, g_i \rangle$ is the least number k such that every element of $\mathbf{G}$ is k-expressible. Note that this number depends on the generator set.

A group $\mathbf{F}$ is a subgroup of another group $\mathbf{G}$, written $\mathbf{F} \leq \mathbf{G}$, if the elements of $\mathbf{F}$ are a subset of the elements of $\mathbf{G}$. In our context, two permutation groups are of particular interest. One is $\mathbf{S}_n$, the *symmetric group* over n elements, which consists of all permutations over n elements. A permutation in $\mathbf{S}_n$ is *even* if it can be represented as a product of an even number of transpositions, *odd* otherwise. Note that no permutation can be odd and even at the same time [32, Theorem 7.1.1]. The set of even permutations forms another group, the *alternating group* $\mathbf{A}_n \leq \mathbf{S}_n$. Since any m-cycle can be equivalently expressed as the product of $m - 1$ transpositions, m-cycles with even m are not elements of $\mathbf{A}_n$.

A permutation group $\mathbf{G}$ is *k-transitive* if for all pairs of k-tuples $(x_1, \ldots, x_k)$, $(y_1, \ldots, y_k)$, there exists a permutation $\sigma \in \mathbf{G}$ such that $\sigma(x_i) = y_i, 1 \leq i \leq k$. In case of 1-transitivity we simply say that $\mathbf{G}$ is transitive.

[1] A gentle introduction to the topic is the book by Mulholland [32].

[2] Note that this order of function applications, which is used in the context of permutation groups, is different from ordinary function composition.

A *block* is a non-empty subset $B \subseteq X$ such that for each permutation σ of a given permutation group over X, either $\sigma(B) = B$ or $\sigma(B) \cap B = \emptyset$. Singleton sets and the entire set X are *trivial blocks*. Permutation groups that contain only trivial blocks are *primitive*.

3.3. Multi-agent pathfinding

Given a graph $G = (V, E)$ and a set of *agents* R such that $|R| \leq |V|$, we say that the injective function $S : R \to V$ is a *multi-agent pathfinding (MAPF) state* (or simply *state*) over R and G. Any node not occupied by an agent, i.e., a node $v \in V - S(R)$, is called *blank*.

A *multi-agent pathfinding (MAPF) instance* is then a tuple $\langle G, R, I, T\rangle$ with G and R as above and I and T MAPF states. A *simple move* of agent r from node u to node v, in symbols $m = \langle r, u, v\rangle$, transforms a given state S, where we need to have $\{u, v\} \in E$, $S(r) = u$ and v is a blank, into the successor state $S^{[m]}$, which is identical to S except at the point r, where $S^{[m]}(r) = v$.

If there exists a (perhaps empty) sequence of simple moves, a *movement plan*, that transforms S into S', we say that S' is *reachable* from S. The MAPF problem is then to decide whether T is reachable from I.

The MAPF problem is often defined in terms of parallel or train-like movements [15,17], where one step consists of parallel non-interfering moves of many agents. However, as long as we are interested only in solution existence and polynomial bounds, there is no difference between the MAPF problems with parallel and sequential movements. If we allow for *synchronous rotations* [18–20], where one assumes that all agents in a fully occupied cycle can move synchronously, things are a bit different. In this case, even if there are no blanks, agents can move.

A *synchronous rotation on a cycle* $v_0, \ldots v_{k-1}, v_k = v_0$ is a set of simple moves $M = \{\langle r_0, u_0, v_0\rangle, \ldots, \langle r_{k-1}, u_{k-1}, v_{k-1}\rangle\}$, with $v_i = u_{i+1}$ for $0 \leq i < k - 1$, and $v_{k-1} = u_0$. Such a rotation M is executable in S if $S(r_i) = u_i$ for $0 \leq i < k$. The successor state $S^{[M]}$ of a given state S is identical to S except at the points $r_0, \ldots, r_{k-1}$, where we have $S^{[M]}(r_i) = v_i$.

Multi-agent pathfinding on directed graphs is similar to MAPF, except that we have a directed graph and the moves have to follow the direction of an arc, i.e., if there is an arc $(u, v) \in A$ but $(v, u) \notin A$, then an agent can move from u to v but not vice versa. In other words, *multi-agent pathfinding on directed graphs (diMAPF)* is the following decision problem:

Instance: A tuple $\langle D, R, I, T\rangle$ with $D = (V, A)$ a directed graph, R a set of agents, and two states $I : R \to V$ and $T : R \to V$.
Question: Is T reachable from I using only agent movements that respect the directions of the arcs in the directed graph?

And this problem will be analyzed in the remaining part of the paper.

4. Deciding diMAPF on DAGs

As mentioned in the Introduction, deciding MAPF (on undirected graphs) is a polynomial-time problem and movement plans have cubic length [6]. And this holds also for diMAPF on strongly biconnected digraphs if there are at least two empty vertices [28]. One intuitive reason for these positive results is that on undirected graphs and strongly biconnected digraphs one can usually restore earlier sub-configurations. This means that agents can move out of the way and then back to where they were earlier. In particular, there do not exist dead ends in the state space, i.e., if the instance has a solution, then no move can ever lead to unsolvability. All this means that these problems have the flavor of permutation puzzles (as discussed, e.g., by Mulholland [32]), which are often solvable with polynomially many moves.

In a digraph without strong connectivity, moves are not necessarily reversible and an agent might paint itself easily into a corner. Given that in every state there are different possible moves for one agent, it might be hard to decide which is the one that in the end will not block another agent in the future. As a matter of fact, this property will be used in the NP-hardness proof below.

Lemma 1. *Deciding diMAPF is NP-hard, regardless of whether synchronous rotations are allowed or not. And this holds even for DAGs.*

Proof. We prove NP-hardness by a reduction from 3SAT, the problem of deciding satisfiability for a formula in conjunctive normal form with 3 literals in each clause, which is one of the first problems that have been proved to be NP-complete [33]. Let us assume a 3SAT instance, consisting of n variables x_i and k clauses c_j with 3 literals each.

Now we construct a diMAPF instance as follows.[3] The set of agents is:

$$R = \{x_1, \ldots, x_n, x'_1, \ldots, x'_n, c_1, \ldots, c_k, f_1, \ldots, f_{nk}\}.$$

The x_i's are called *variable agents*, the x'_i's are named *shadow agents*, the c_j's are called *clause agents*, and the f_ℓ's are called *filler agents*. The set of vertices of the digraph is constructed as follows:

$$V = \{v_1, \ldots, v_{nk+n+k}\} \cup \bigcup_{i=1}^{n} \{v_i^T, v_i^F, v_{x_i}, v_{x'_i}\} \cup \bigcup_{j=1}^{k} \{v_{c_j}\}.$$

[3] This reduction uses inspirations from a reduction that has been used to show PSPACE-hardness for a generalized version of MAPF [34].

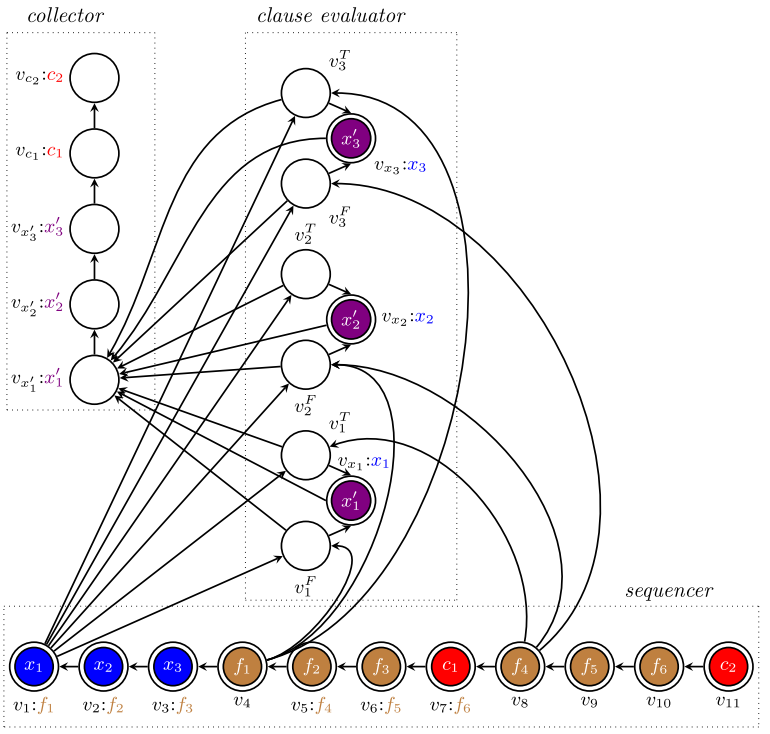

Fig. 2. Example for $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

We proceed by constructing three gadgets, which we call *sequencer*, *clause evaluator*, and *collector*, respectively. We illustrate the construction using the example in Fig. 2. In this visualization, vertices occupied by an agent are filled with a smaller colored circle containing the name of the agent. The color depicts the group the agents belongs to. Blue for variable agents x_i, violet for shadow agents x'_i, red for clause agents c_j, and brown for filler agents f_ℓ. Empty circles symbolize empty vertices. Each vertex is labeled by its identifier, perhaps followed by a colon and the name of an agent, which symbolizes the destination for this agent. For example, the leftmost node in the sequencer gadget is named v_1, it is initially occupied by agent x_1, and it is the destination of agent f_1.

The task of the *sequencer* is to enforce the sequence of truth-value choices of the variable agents x_i. Each of the variable agents x_i has to go to one of the vertices v_i^T or v_i^F—and these are the only vertices x_i can go to. After that the filler and clause agents can move to the left and the clause agents can start to go through the clause evaluator. The *clause evaluator* is created in a way so that a clause agent c_j can move through it from right to left, provided one of the literals of the corresponding clause is true according to the truth-value choices made by the variable agents. Finally, the *collector* contains the destination vertices for all clause agents c_j and for the *shadow agents* x'_i. First the clause agents c_j need to get to their destinations, then the shadow agents x'_i can arrive at their goals, making room for the variable agents x_i to move to their final destinations.

The *sequencer* consists of a sub-graph with $nk + n + k$ vertices, which are named v_1 to v_{nk+n+k}. These vertices are connected linearly, i.e., there is an arc from v_{i+1} to v_i. The vertices v_1 to v_n are occupied by *variable agents* named x_1 to x_n. In addition we have *clause agents* $c_j, 1 \leq j \leq m$ on the vertices $v_{n+j(n+1)}$, respectively. The rest of the vertices are filled with *filler agents* f_p for all the not yet occupied vertices. The destination for each filler agent f_p is the vertex with an index n lower than the one f_p is starting from. These filler agents are necessary to enforce that the clause agents enter the clause evaluator only after the variable agents have made their choices.

The *clause evaluator* contains for each variable x_i one pair of vertices: v_i^F and v_i^T. These vertices represent the truth assignment choices false and true, respectively, for x_i. In addition, there exists an additional vertex v_{x_i}, which can be reached from both v_i^F and v_i^T and which is the destination for agent x_i and initially occupied by the *shadow agent* x'_i. This setup enforces the variable agent x_i to move to v_i^F or v_i^T once it has reached v_1 waiting for the shadow agent x'_i to move towards its destination.

Once all the x_i agents have reached their vertices v_i^T or v_i^F, the remaining agents in the sequencer can move n vertices to the left, i.e., from v_p to v_{p-n} bringing all the filler agents f_p to their respective destinations. Further, all clause agents c_j have to go from $v_{n+j(n+1)}$ to $v_{j(n+1)}$, whereby these latter vertices are connected to the clause evaluator in the following way. The vertex $v_{j(n+1)}$, which will hold clause agent c_j after all agents moved n steps to the left, is connected to v_i^F iff the clause c_j contains x_i positively and it is connected to v_i^T iff c_j contains x_i negated. This means that the clause agent c_j can pass to $v_{x'_1}$ if and only if one of the variable agents x_i participating in the clause c_j made the "right" choice.

Finally, the collector gadget provides the destinations for all the clause agents c_j and the shadow agents x'_i. The vertices v_i^T, v_i^F, and v_{x_i} all lead to the vertex $v_{x'_1}$, which is the destination of the shadow agent x'_1. Starting at this node, we have a linearly connected

path up to vertex $v_{x'_n}$ from which v_{c_1} can be reached, which in turn is a linear path to v_{c_k}. This implies that first all clause agents c_j have to reach their destination vertices, after which the shadow agents x'_i can move to their destinations. Only after all this has happened, the variable agents can move to their destinations v_{x_i}.

By the construction, a successful movement plan will contain the following phases:

1. In the first phase the variable agents x_i will move to the vertices v_i^T or v_i^F. Which vertex x_i moves to can be interpreted as making a choice on the truth value of the variable. Note that no other vertices are possible, because then the final destination would not be reachable any more for x_i.
2. In the second phase, all filler and clause agents move n vertices to the left in the sequencer widget. Note that no other vertices are possible for filler agents because then their goal would not be reachable any more.
3. After phase 2 has finished, all clause agents c_j occupy vertices $v_{j(n+1)}$, from which they can pass through the clause evaluator widget. By construction, they can pass through it if and only if for one of the variables occurring in clause c_j, the variable agent has made a choice in phase 1 corresponding to making the clause true. Note that no other group of agents can move, or otherwise they will no longer be able to reach their destination or block the clause agents. The phase ends when all clause agents have reached their destinations.
4. After the end of phase 3, the shadow agents x'_i move to their respective destinations, enabling the variable agents x_i to go to their destinations.
5. Finally all variable agents can move to their destinations, finalizing the movement plan.

Note that in a successful plan some of the phases could overlap. However, one could easily disentangle them. The critical phases are apparently phase 1 and phase 3. Phase 3 is only successful if in phase 1 the variable agents made the choices in a way, so that all clauses are satisfied. In other words, the existence of a successful movement plan implies that there is a satisfying truth value assignment to the CNF formula. Conversely, if there exists a satisfying truth value assignment, then this could be used to generate a successful movement plan by using it to make the choices in phase 1. Since the construction is clearly polynomial in the size of the 3SAT instance, it is a polynomial many-one reduction, proving that diMAPF is NP-hard.

Finally note that the constructed graph is a DAG. This implies that the result applies already to DAGs and that it holds even if synchronous rotations are allowed, because on DAGs these are impossible.

While this result demonstrates that diMAPF is more difficult than MAPF (provided $NP \neq P$), it leaves open how much more difficulty is introduced by moving from undirected to directed graphs. Although one might suspect that diMAPF is just NP-complete, this is by no means obvious. The main obstacle in proving that diMAPF is NP-complete is the fact that the state space of the diMAPF problem has size $O(n!)$, n being the number of nodes. However, in cases similar to the one used in the proof of Lemma 1, namely if the digraph is acyclic, it is obvious that the number of moves is bounded polynomially.

Theorem 2. *The diMAPF problem on DAGs is NP-complete.*

Proof. In a DAG, each agent can make at most $|V|$ moves, since the agent can never visit a vertex twice. This means that overall no more than $|V|^2$ moves are possible. This implies that all solutions have a length bounded by a polynomial in the input size, implying that the problem is in NP. Together with Lemma 1, this implies the claim.

When looking for the reason that stops us from proving a general NP-completeness result, one notices that strongly connected components are the culprits. They allow agents to reach the same location twice with the other agents in a perhaps slightly different configuration. This may imply that a particular configuration can only be reached when agents walk through super-polynomially many distinct configurations.

On the other hand there are positive results for undirected graphs (wrt. simple moves and synchronous rotations) and for strongly connected directed graphs with two blanks (wrt. simple moves). For this reason, it looks very unlikely that for general strongly connected digraphs plan-length will become all of the sudden super-polynomial. And this is precisely what will be proved in the next three sections.

5. Deciding diMAPF on strongly connected digraphs when synchronous rotations are prohibited

One crucial property for proving NP-hardness in the previous section was the presence of dead ends in the state space. So, the absence of such dead ends makes live probably much easier. Here, the observation that each simple move on a strongly connected component can be undone is very helpful. As we will see, this is indeed the key observation for proving our results.

When considering cycle graphs, it is obvious that it is always possible to restore a previous state.

Proposition 3. *Let S be a diMAPF state over the set of agents R and the cycle digraph $D = (V, A)$, and let S' be a state reachable from S with simple moves, then one can reach S from S' in at most $O(|V|^2)$ simple moves.*

Proof. Since the relative order of agents on a cycle cannot be changed by movements, one can always reach the initial state, regardless of what movements have been made in order to deviate from the initial state. Further, the maximal distance of an agent

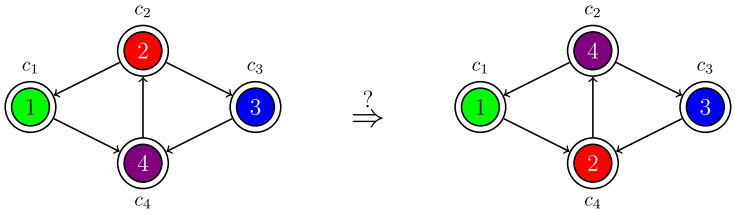

Fig. 3. An unsolvable diMAPF example.

from its position in S is $|V| - 1$, which is the maximal number of moves the agent has to make to reach S. Since there are at most $|V|$ agents, the stated upper bound follows.

If we now consider a simple move on a strongly connected digraph, then it is obvious that we can restore the original state, because all movements in such a graph take place on cycles.

Proposition 4. *Let S be a diMAPF state over R and a strongly connected digraph $D = (V, A)$ and let $m = \langle r, u, v \rangle$ be a simple move to transform S into $S^{[m]}$. Then there exists a plan consisting of simple moves of length $O(|V|^2)$ to reach S from $S^{[m]}$.*

Proof. Each move m in a strongly connected digraph is a move on a cycle in the digraph. Hence, we can apply Proposition 3 and restore the original state S using $O(k^2)$ simple moves, provided k is the number of nodes in the cycle. Note that by restoring the configuration on the cycle, we restore the entire state. Further, since $k \leq |V|$, the claim about the plan length follows.

The plan of restoring the state before move m is executed can actually be seen as the "inverse move" m^{-1}, which moves an agent against the direction of an arc! Although such a "synthesized" move against the arc direction is costly—it may involve $O(|V|^2)$ simple moves—this opens up the possibility to view a diMAPF instance on the digraph D as a MAPF instance on the underlying undirected graph $\mathcal{G}(D)$.[4] Because of the existence of inverse moves for each possible simple move with only polynomial overhead, the next corollary is immediate.

Corollary 5. *Let $\langle D, R, I, T \rangle$ be a diMAPF instance with D a strongly connected digraph. Then the MAPF instance $\langle \mathcal{G}(D), R, I, T \rangle$ has a polynomial movement plan consisting of simple moves if and only if $\langle D, R, I, T \rangle$ has a polynomial movement plan consisting of simple moves.*

Using Corollary 5 together with the result about polynomial decidability and the upper bound for the plan length on undirected graphs from the paper by Kornhauser et al. [6, Theorem 1 and 2] gives us the first partial result for diMAPF on strongly connected digraphs.

Lemma 6. *Provided that only simple moves are allowed, diMAPF on strongly connected digraphs can be decided in polynomial time, and plans are polynomially bounded.*

Note that because of the $O(|V|^2)$ overhead for inverse moves and the cubic length for plans on undirected graphs, we have an $O(|V|^5)$ upper bound for plan length, and one certainly would hope to do better than that. However, we are satisfied since it is a polynomial.

Using Lemma 6, we could now prove NP-completeness of diMAPF when only simple moves are allowed. However, we will defer that to Section 7, when we are in a position to prove a more general result.

6. Deciding diMAPF on strongly connected digraphs with only synchronous rotations permitted

As a next step, we consider the case that the only kind of movements are synchronous rotations. In this context, we assume w.l.g. that all nodes are occupied by agents. Unfortunately, it is not possible to reduce the reachability on a strongly connected digraph to reachability on the underlying undirected graph. In order to see the problem, consider the example in Fig. 3.

The cycle in the underlying undirected graph formed by the nodes c_1, c_2, c_3, c_4, c_1 could be used for a rotation on the underlying undirected graph, but there is no obvious way to emulate such a rotation on the directed graph. In fact, the diMAPF instance in this example is unsolvable while the corresponding MAPF instance on the underlying undirected graph is solvable, as we will show later on. For this reason, we will employ permutation group theory in order to derive a polynomial upper bound for plan length in this case. For this purpose, we will first have a look at the structure of the digraph.

[4] This had already been noted by Ardizzoni et al. [29, Theorem 4.3]. However, the proof appears to be incomplete, and the implication for plan length had not been stated.

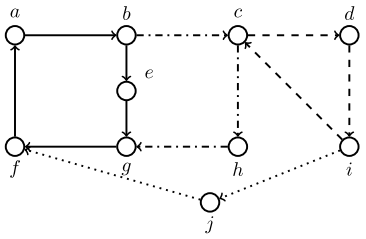

Fig. 4. Example of an ear decomposition: The basic cycle is solid, the first ear is dash-dotted, the second ear is dashed and the third ear is dotted.

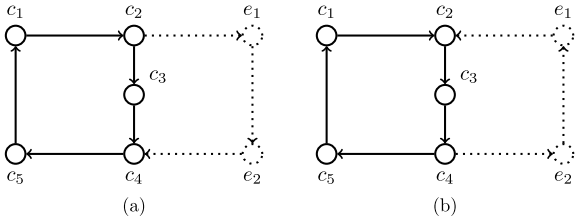

Fig. 5. Strongly connected digraphs consisting of a cycle and an ear.

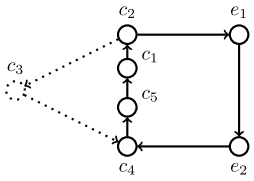

Fig. 6. Different perspective on graph from Fig. 5(a).

6.1. Cycle pairs

It is well known that for every non-trivial strongly connected digraph, there exists an *ear decomposition* [35, Theorem 5.3.2] (see Fig. 4 for an example of such a decomposition).

This is a decomposition of the digraph into a *basic cycle* P_0 and a sequence of so-called *ears* or *handles*, which are cycles or paths $P_1, \ldots, P_n$, where each ear P_j is attached at one node (if the ear is a cycle) or at two nodes (if the ear is a path) to the union over the P_i's with an index less than j: $\bigcup_{i=0}^{i=j-1} P_i$. Furthermore, each cycle of the digraph can be the basic cycle of such an ear decomposition.

As one can see in this example, one could also start with a different basic cycle, e.g., with $P_0 = c, d, i, c$. Then it is possible to add the (cycle) ear $P_1 = c, h, g, f, a, b, c$, then the (path) ear $P_2 = b, e, g$, and finally the ear $P_3 = i, j, f$. If we talk about the *size of an ear*, we mean the number of nodes that are belonging exclusively to the ear. For example, the size of P_3 is 1.

Based on the fact that each digraph can be decomposed into a basic cycle and ears, each strongly connected digraph that is not a cycle digraph contains at least a sub-digraph consisting of a directed cycle (such as, e.g., c_1, c_2, c_3, c_4, c_5, c_1 in Fig. 5, which is drawn solidly), and an ear (such as, e.g., c_2, e_1, e_2, c_4 drawn in a dotted way). The ear could either be oriented in the same direction as the basic cycle, providing a detour or short-cut as in Fig. 5(a), or it points back, as in Fig. 5(b).

In order to be able to deal with only one kind of cycle pair, it is always possible to view a graph as in Fig. 5(a) as one in Fig. 5(b). This can be accomplished by considering the outer, larger cycle as the basic cycle and the path with c_2, c_3, c_4 as an ear, as illustrated in Fig. 6. Note that in the extreme one may have an ear with no additional nodes.

This means that in a strongly connected digraph which is not a cycle digraph, one can always find *cycle pairs* of a particular form as stated in the following proposition.

Proposition 7. *Any strongly connected non-trivial digraph that is not a cycle digraph contains two directed cycles that share at least one node, where at least one cycle contains more nodes than the shared ones. Further, if one cycle contains only shared nodes, then there are at least two shared nodes.*

Proof. An ear decomposition leads to a basic cycle and an ear, which are, in fact, two connected cycles. One of these cycles needs to contain more than the shared nodes because otherwise we would not have two cycles. Finally, if one cycle contains only the shared nodes, then there needs to be at least two shared nodes, because otherwise the structure would be a non-simple digraph, which we excluded above.

6.2. Synchronous rotations as permutations

Synchronous rotations on cycles in the digraph will now be viewed as permutations. Note that such permutations are cyclic permutations. If we refer to such a cyclic permutation of degree m, we call them *m-cycle* (as introduced in Section 3.2). If we refer to a graph cycle consisting of n nodes, we will write *cycle of size n*.

Given a diMAPF instance $\langle D, R, I, T\rangle$ on a strongly connected digraph D with no blanks, we will view the sequence of rotations that transforms I into T as a sequence of permutations on V that when composed permutes I into T. The *permutation group rotation-induced by such an instance* will also often be called *permutation group rotation-induced by* D, since the concrete sets R, I, and T are not essential for our purposes. By the one-to-one relationship between rotations and permutations, it is obvious that a polynomial diameter of the rotation-induced group implies that the length of movement plans can be polynomially bounded. In the following, we will also often use arguments about possible movements of agents in order to prove that a particular permutation exists.

One obvious property of such rotation-induced permutation groups is their transitivity.

Proposition 8. *Every permutation group that is rotation-induced by a strongly connected digraph is 1-transitive.*

Proof. This property is equivalent to the fact that each agent in such a digraph can reach each node in the digraph (perhaps moving other agents around). This can be accomplished by rotating agents on the appropriate cycles for moving the agent from the source to its target node.

In order to show that the rotation-induced permutation groups have polynomial diameter, we will use the following result by Driscoll and Furst [36, Theorem 3.2].

Theorem 9 *(Driscoll & Furst). If* $\mathbf{G}$ *is a primitive group containing a polynomially expressible 3-cycle, then the diameter of* $\mathbf{G}$ *is polynomially bounded.*

Incidentally, if the conditions of the Theorem are satisfied, then $\mathbf{G} = \mathbf{A}_n$ or $\mathbf{G} = \mathbf{S}_n$, as follows from a Lemma that is used in Driscoll and Furst's paper [36, Lemma 3.4], which is cited from Wielandt's book on finite permutation groups [37, Theorem 13.3] and attributed to Camille Jordan.

Lemma 10 *(Jordan). A primitive group that contains a 3-cycle is either alternating or symmetric.*

It should be noted that primitiveness is implied by 2-transitivity. Assuming otherwise, i.e., that a group is 2-transitive but not primitive, leads to a contradiction, because for a non-trivial block Y there would exist one permutation that fixes one element (staying in the block) and moves another element out of the block, which contradicts that Y is a block.

Proposition 11. *Every 2-transitive permutation group is primitive.*

In other words, it is enough to show 2-transitivity and the polynomial expressibility of a 3-cycle in order to be able to apply Theorem 9, which enables us to derive a polynomial bound for the diameter.

Further, for demonstrating that a rotation-induced group is the symmetric group, given 2-transitivity and a 3-cycle, it suffices to show that the rotation-induced group contains an odd permutation. This follows from the fact that $\mathbf{A}_n$ contains only even permutations and Lemma 10. Note that in case the permutation group is the symmetric group, it means that the corresponding diMAPF instance is always solvable, a fact we will use later on.

6.3. 2-transitivity

Almost all permutation groups rotation-induced by diMAPF instances on strongly connected digraphs are 2-transitive, as shown next. Intuitively, it means that we can move any two agents to any two places in the digraph—moving perhaps other agents around as well.

Lemma 12. *Permutation groups rotation-induced by strongly connected non-trivial digraphs that are not cycle digraphs are 2-transitive.*

Proof. In order to prove this lemma, we will show that for two fixed nodes x and y, it is possible to move any pair of agents a_u and a_v from the node pair (u, v) to the node pair (x, y). This implies that we also could move any pair $(a_{u'}, a_{v'})$ from (u', v') to (x, y). Composing the first plan with the inverse of the second plan means that we can move agents from any pair of nodes (u, v) to any other pair of nodes (u', v'), as illustrated in Fig. 7. This means that the group is 2-transitive.

We proceed now by showing how to construct a plan moving agents a_u and a_v from the node pair (u, v) to the node pair (x, y). For this purpose, we first fix two nodes x and y. By Proposition 7, the digraph must contain at least two cycles, both containing at least two nodes, where the left cycle may consist of only shared nodes, as depicted in Fig. 8.

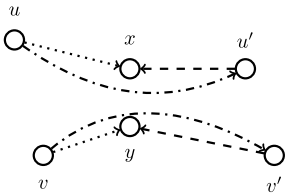

Fig. 7. Constructing a plan demonstrating 2-transitivity. The dotted movement plan moves the agents from (u, v) to (x, y), the dashed movement plan from (u', v') to (x, y). The dash-dotted plan is the composition of the first and the inverse of the second plan.

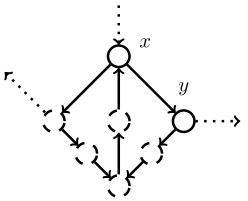

Fig. 8. Strongly connected component containing two cycles with the two target nodes x and y. The dashed nodes signify possible additional nodes. The dotted arcs exemplify potential connections to other nodes in the digraph.

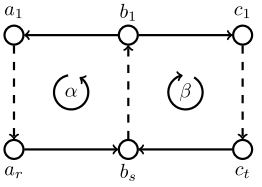

Fig. 9. Strongly connected component consisting of at least two cycles inducing two permutations.

By transitivity (Proposition 8), we can move any agent a_u from node u to node x in Fig. 8. After that, we can move any agent a_v from node v to node y in Fig. 8. This may lead to rotating the agent a_u out of the left cycle. In order to prohibit that, we modify the movement plan as follows. As long as a_v has not entered one of the two cycles yet, every time a_u is threatened to be rotated out of the left cycle in the next move, we rotate the entire left cycle so that a_u is moved to a node that will not lead to rotating a_u out of the left cycle.

If a_v arrives in the right cycle (including the shared nodes) in Fig. 8, we rotate the right cycle iteratively. Whenever a_u is placed on x and a_v has not yet arrived at y, we rotate on the left cycle in Fig. 8. Otherwise, we stop and are done. When a_v is placed on x, then in the next move, we rotate the right cycle and a_v is placed on y. After that, we can rotate the left cycle in Fig. 8 until a_u arrives at x. This is possible, because a_u never leaves the left cycle.

If a_v arrives on nodes not belonging to the right cycle, we rotate the left cycle. When a_v arrives at x, we rotate on the right cycle and then we rotate on the left cycle until a_u arrives again at x.

This means that we can always generate a plan to move two agents from arbitrary positions in the graph to (x, y). Using the idea of composing two plans as described in the beginning, we can move the two agents to any pair of nodes, demonstrating 2-transitivity.

6.4. 3-cycles

The construction of 3-cycles will be shown by a case analysis over the possible forms of two connected cycles. By Proposition 7, we know that every strongly connected non-trivial digraph that is not a cycle digraph contains a subgraph as shown in Fig. 9.

We characterize such connected cycles by the three parameters (r, s, t) and will talk about *cycle pairs of type* (r, s, t), assuming w.l.g. $r \leq t$. Below, we will show that for almost all cycle pairs one can construct a 3-cycle, save for cycle pairs of type $(2, 2, 2)$ and $(1, 3, 2)$. These are the counterparts to Kornhauser et al.'s [6] T_0- and Wilson's [11] θ_0-graph (see Fig. 10), which are the outliers in the case of simple moves on undirected graphs. We will therefore call such cycle pairs T_0*-pairs*.

The natural question coming up is: Why does the T_0-graph has seven nodes, while the T_0-pairs have only six? The reason is that Kornhauser et al. and Wilson considered simple moves, where you need a blank in order to rotate the agents on a cycle. If this blank is on any of the b_i nodes, then the resulting permutations correspond to the permutations induced by the T_0-pair of type $(2, 2, 2)$. If the blank is one of the a_i nodes, then the resulting permutations correspond to those induced by the T_0-pair of type $(1, 3, 2)$.

Lemma 13. *Each permutation group rotation-induced by a cycle pair that is not a T_0-pair contains a polynomially expressible 3-cycle.*

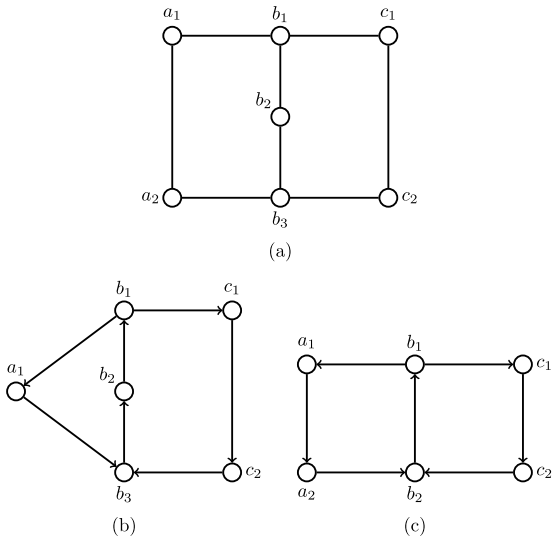

Fig. 10. (a) T_0/θ_0-graph, (b) T_0-pair of type $(1,3,2)$, and (c) T_0-pair of type $(2,2,2)$.

Proof. The 3-cycles will be constructed from $\alpha = (a_1 \dots a_r b_s \dots b_1)$ and $\beta = (c_1 \dots c_t b_s \dots b_1)$. We prove the claim by case analysis over the parameters (r,s,t) (see Fig. 9). We visit all cases by ordering the values (r,s,t) lexicographically. If a construction also applies to cases with a higher component value, we use the $\geq$ symbol. For example, $(0,1,\geq 2)$ covers the case $r=0$, $s=1$, and $t=2$ as well as all cases with $r=0, s=1$ and $t>2$.[5]

$(0,\geq 2,\geq 1)$: By Proposition 7, we know that $r=0$ implies $s\geq 2$ and $t\geq 1$, and we have $\beta\ \alpha^{-1}\ \beta^{-1}\ \alpha = (b_s\ c_t\ b_1)$ as a 3-cycle.
$(\geq 1,1,\geq 1)$: In this case, the same expression delivers a slightly different 3-cycle: $\beta\ \alpha^{-1}\ \beta^{-1}\ \alpha = (b_1\ a_1\ c_t)$.
$(1,2,\geq 1)$: α is a 3-cycle.
$(1,\geq 3,1)$: $\alpha^{-1}\beta = (b_s\ a_1\ c_1)$ is the desired 3-cycle.
$(1,3,2)$: This is a T_0-pair, so there is nothing to prove here. It can be shown by exhaustive enumeration that the rotation-induced permutation group does not contain 3-cycles.
$(1,3,\geq 3)$: $\beta\ \alpha^{-1}\ \beta^{-1}\ \alpha = (b_s\ c_t)(b_1\ a_1)$. Consider now $\chi = \beta^2\ (\alpha^{-1}\ \beta)^2\ \beta^{-2}$. This permutation fixes b_s and c_t and moves c_{t-2} to a_1 and a_1 to b_1 while also moving other things around. This means:

$$\begin{aligned}(\beta\ \alpha^{-1}\ \beta^{-1}\ \alpha)^{\chi^{-1}} &= \chi\ \beta\ \alpha^{-1}\ \beta^{-1}\ \alpha\ \chi^{-1}\\ &= \chi(b_s\ c_t)(b_1\ a_1)\chi^{-1}\\ &= (b_s\ c_t)(a_1\ c_{t-2}).\end{aligned}$$

Composing the result with the original permutation is now what results in a 3-cycle.[6] That is, $\lambda = (\beta\ \alpha^{-1}\ \beta^{-1}\ \alpha)^{\epsilon+\chi^{-1}}$ is the permutation, we looked for:

$$\begin{aligned}\lambda &= (\beta\ \alpha^{-1}\ \beta^{-1}\ \alpha)^{\epsilon+\chi^{-1}}\\ &= ((b_s\ c_t)(b_1\ a_1))\circ((b_s\ c_t)(b_1\ a_1))^{\chi^{-1}}\\ &= ((b_s\ c_t)(b_1\ a_1))\circ((b_s\ c_t)(a_1\ c_{t-2}))\\ &= (b_1\ c_{t-2}\ a_1).\end{aligned}$$

$(1,\geq 4,1)$: This case is already covered under $(1,\geq 3,1)$ above.
$(1,\geq 4,\geq 2)$: In this case, $\xi = (\alpha\ \beta^{-1}\ \alpha^{-1}\ \beta)^{\beta(\epsilon+\alpha^{-2})}$ is the claimed 3-cycle[7]:

$$\xi = (\alpha\ \beta^{-1}\ \alpha^{-1}\ \beta)^{\beta(\epsilon+\alpha^{-2})}$$

[5] The case analysis and the construction of the 3-cycles can be manually checked. As an alternative, one could use the SageMath script `lemma13-case-analysis.sage` in the GitHub repository https://github.com/BernhardNebel/diMAPF.

[6] This construction is similar to one used by Kornhauser [38] in the proof of Theorem 1 for T_2-graphs. Note that ϵ denotes the identity permutation as introduced in Section 3.2.

[7] This construction of a permutation as well as the ones for the cases further down are similar to one that has been used in a similar context by Bachor et al. [39].

$$= \left((b_s\ a_r)(b_1\ c_1)\right)^{\beta(\epsilon+\alpha^{-2})}$$
$$= \left((b_{s-1}\ a_r)(c_1\ c_2)\right)^{\epsilon+\alpha^{-2}}$$
$$= ((b_{s-1}\ a_r)(c_1\ c_2))\circ((b_{s-1}\ a_r)(c_1\ c_2))^{\alpha^{-2}}$$
$$= ((b_{s-1}\ a_r)(c_1\ c_2))\circ((b_2\ a_r)(c_1\ c_2))$$
$$= (b_{s-1}\ b_2\ a_r).$$

$(2,1,\geq 2)$: This case is already covered by $(\geq 1,1,\geq 1)$ above.
$(2,2,2)$: This is the other case that is excluded in the claim and the same comment as in case $(1,3,2)$ applies.
$(\geq 2,\geq 2,\geq 3)$: This case is the one for all cycle pairs that are large enough:

$$\zeta = (\beta\ \alpha^{-1}\ \beta^{-1}\ \alpha)^{\alpha(\epsilon+\beta^{-2})}$$
$$= ((b_s\ c_t)(a_1\ b_1))^{\alpha(\epsilon+\beta^{-2})}$$
$$= ((b_{s-1}\ c_t)(a_1\ a_2))^{\epsilon+\beta^{-2}}$$
$$= ((b_{s-1}\ c_t)(a_1\ a_2))\circ((b_{s-1}\ c_t)(a_1\ a_2))^{\beta^{-2}}$$
$$= ((b_{s-1}\ c_t)(a_1\ a_2))\circ(c_{t-2}\ c_t)(a_1\ a_2))$$
$$= (b_{s-1}\ c_{t-2}\ c_t).$$

$(2,\geq 3,2)$: For this case, the same construction as for the previous case can be used. However, one gets a slightly different result because the structure of the cycle pair is different (the difference is underlined):

$$\zeta = (\beta\ \alpha^{-1}\ \beta^{-1}\ \alpha)^{\alpha(\epsilon+\beta^{-2})}$$
$$= ((b_s\ c_t)(a_1\ b_1))^{\alpha(\epsilon+\beta^{-2})}$$
$$= ((b_{s-1}\ c_t)(a_1\ a_2))^{\epsilon+\beta^{-2}}$$
$$= ((b_{s-1}\ c_t)(a_1\ a_2))\circ((b_{s-1}\ c_t)(a_1\ a_2))^{\beta^{-2}}$$
$$= ((b_{s-1}\ c_t)(a_1\ a_2))\circ((b_1\ c_t)(a_1\ a_2))$$
$$= (b_{s-1}\ \underline{b_1}\ c_t).$$

$(\geq 2,\geq 3,\geq 3)$: This general case is covered already under $(\geq 2,\geq 2,\geq 3)$ above.

This covers all possible cases. Note that when taking α and β as generators, then the inverses α^{-1} and β^{-1} can be expressed by linearly many products. Since the expressions for all cases have constant length, in all cases the 3-cycles are linearly expressible. So, the claim holds.

In order to be able to do away with T_0-pairs, we will assume that our digraphs contain at least seven nodes. For all smaller digraphs, the diameter of the rotation-induced permutation group is constant. One only has then to show that strongly connected digraphs with seven or more nodes admit for the generation of a 3-cycle.

Lemma 14. *Each permutation group rotation-induced by a strongly connected digraph with at least 7 nodes that is not a cycle digraph contains a polynomially expressible 3-cycle.*

Proof. By Lemma 13, it is enough to prove the claim for digraphs that contain a T_0-pair. In order to do so, all possible extensions of T_0-pairs have to be analyzed. It is sufficient to consider all extensions with one additional ear. For each extension, we first check whether a new cycle pair is created that is not a T_0-pair, in which case Lemma 13 is applicable. If the newly created cycle pairs are all T_0-pairs, one has to demonstrate that by the addition of the ear a new permutation is added that can be used to create a 3-cycle.

Because the longest ear in T_0-pairs has a length of 2, we consider ears up to length 2. Adding a longer ear would result in a pair of type $(\_,\_,\geq 3)$, which admits a 3-cycle according to Lemma 13. Since the T_0-pairs contain six nodes each, there are two different T_0 pairs, and we consider ears of length one and two, we need to analyze $6^2\times 2\times 2 = 144$ cases. This has been done using a *SageMath* [40] script, which is listed in the appendix. This script identified three non-isomorphic extensions of the $(1,3,2)$- and $(2,2,2)$-type cycle pairs that contain only T_0-pairs as cycle pairs. These are shown in Fig. 11.

It is now an easy exercise to identify 3-cycles for these cases:

(a): $\beta\ \alpha\ \beta^{-1}\ \gamma^{-1}$,
(b): $\beta^{-1}\ \delta^{-1}\ \alpha^{-1}\ \beta^2\ \delta^{-1}\ \alpha^{-1}\ \delta^{-1}$,
(c): $\alpha\ \beta\ \eta\ \alpha^{-1}\ \beta^{-1}\ \eta^{-1}$.

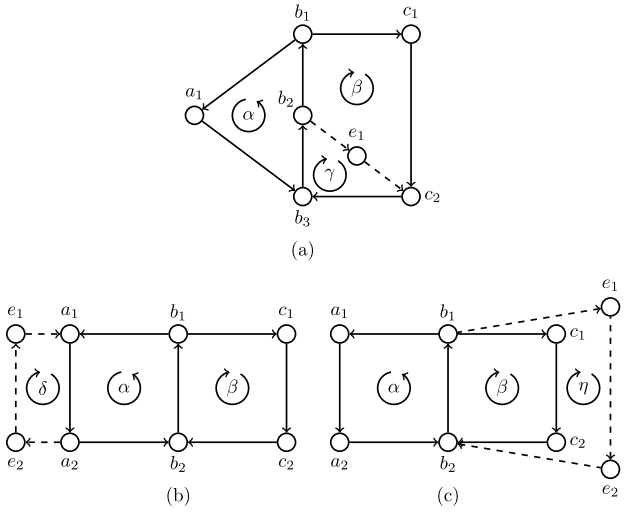

Fig. 11. Extensions of T_0-pairs containing only T_0-pairs.

So, the claim holds for all cases.

With these results, we are now in a position to easily verify the claim from the beginning of the section: The diMAPF instance in Fig. 3 is unsolvable, but the instance on the underlying undirected graph is solvable.

The permutation group rotation-induced by the diMAPF instance, which we call $\mathbf{D}$, must be either $\mathbf{A}_4$ or $\mathbf{S}_4$ (by Lemma 12, Lemma 13, Proposition 11, and Lemma 10). Since $\mathbf{D}$ is generated by two 3-cycles, which are even permutations, all elements of $\mathbf{D}$ are even permutations, so $\mathbf{D} = \mathbf{A}_4$. The permutation required by the diMAPF instance is, however, a simple transposition, i.e., an odd permutation. In other words, the permutation is not an element of $\mathbf{D}$, i.e., the diMAPF instance is unsolvable.

The MAPF instance on the underlying undirected graph contains, however, also the permutation corresponding to the outer cycle in the graph, which is a 4-cycle, i.e., an odd permutation. For this reason, the resulting group must be $\mathbf{S}_4$, i.e., the instance is solvable.

6.5. Polynomial diameter of rotation-induced permutation groups

The above results enable us now to prove the claim that the rotation-induced permutation groups have a polynomial diameter.

Lemma 15. *Each permutation group rotation-induced by a strongly connected digraph has a polynomial diameter.*

Proof. We prove the claim by case analysis. Let n be the number of nodes.

1. $n < 7$*:* There exist only finitely many permutation groups rotation-induced by such graphs. The diameter is therefore $O(1)$ in this case.
2. $n \geq 7$*:*
   (a) *The digraph is a cycle of n nodes:* Each possible permutation can be expressed by at most $O(n)$ compositions of the permutation corresponding to the rotation of all agents by one place.
   (b) *The digraph is a strongly connected digraph that is not a cycle*: For this case, we use Theorem 9, i.e., it is enough to show that a permutation group is 2-transitive and contains a polynomially expressible 3-cycle. By Proposition 7, we know that the digraph contains a cycle pair. Now, 2-transitivity follows from Lemma 12. The existence of a polynomially expressible 3-cycle follows from Lemma 14. So, in this case, the claim holds as well.

This covers all possible cases, so the claim holds.

With that, we can establish that diMAPF on strongly connected digraphs with synchronous rotations is a polynomial problem and admits polynomial plans, parallel to Lemma 6.

Lemma 16. *Provided that only synchronous rotations are allowed, diMAPF on strongly connected digraphs can be decided in polynomial time, and plans are polynomially bounded.*

Proof. The upper bound for the plan length follows immediately from Lemma 15.

For the cases that the digraph is a cycle or it contains less than 7 nodes, polynomial decidability is immediate. Otherwise, note that by Lemmas 12, 13, and 10, it follows that the induced group is either $\mathbf{A}_n$ or $\mathbf{S}_n$. If the digraph contains only cycles of odd size,

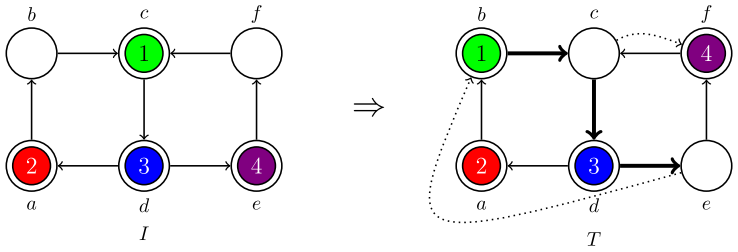

Fig. 12. Example for the construction of a plan to transform T into T'.

the generating set of permutations are all even permutations, i.e., the induced group is $\mathbf{A}_n$, and the instance is only solvable if the permutation induced by the initial and goal configuration is also even, which can be checked in polynomial time. Otherwise the group must be $\mathbf{S}_n$, in which case all instances are solvable. This means, it is a polynomial problem in all cases.

7. The general case: simple moves and synchronous rotations combined

Finally, we will consider the case that simple moves as well as rotations are permitted. In order to be able to apply permutation group theory, we will initially restrict ourselves to diMAPF instances on strongly connected digraphs $\langle D, R, I, T\rangle$ such that the set of occupied nodes is identical in the initial and the goal state, i.e., $I(R) = T(R)$, which can be viewed as permutations on the set of occupied nodes. This restriction is non-essential since one can polynomially transform a general diMAPF instance to such a restricted instance, as shown in Corollary 18 below.

Lemma 17. *Given a diMAPF instance $\langle D, R, I, T\rangle$, with D a strongly connected digraph, an instance $\langle D, R, I, T'\rangle$ can be computed in polynomial time such that $I(R) = T'(R)$, and $\langle D, R, T, T'\rangle$ and $\langle D, R, T', T\rangle$ are both solvable using plans of polynomial length.*

Proof. In order to construct $\langle D, R, I, T'\rangle$, generate a mapping from unoccupied nodes in T, the *source nodes*, to nodes that are unoccupied in I, the *target nodes*. In the example in Fig. 12, c and e are source nodes, and b and f are target nodes. Let (s, t) be such a pair of source and target nodes. We will now construct a movement plan consisting of simple moves that will transfer the blank from node s to node t.

Since D is strongly connected, there must exist a path from t to s. Now move each agent that occupies a node on the path towards s, starting with the agent closest to s. Move this first agent to s. Continuing with the remaining agents on the path, move them one by one to the place the previously moved agent had occupied. This will lead to a configuration where the blank has been transfered to the target node, and all nodes that were unoccupied previously are still unoccupied.

Using the example from Fig. 12 again, let as transfer the blank from the source node e to the target node b. The path between b and e is marked by thick arrows. First, we move agent 3 to e, then agent 1 via c to d, which has transferred the blank from e to b. Finally, we can move 4 to c and have reached a configuration which has the same set of occupied nodes as the initial configuration.

The new configuration is the sought T' and clearly $T'(R) = I(R)$. Further, T' is obviously reachable from T in at most $O(n^2)$ simple moves, n being the number of nodes.

Reaching T from T' is possible by undoing each transfer of a blank in the opposite order. Undoing such a transfer can be done by applying Proposition 4 iteratively resulting in $O(n^4)$ simple moves.

Since T is reachable from T' and vice versa using only polynomial many steps, the next corollary follows immediately.

Corollary 18. *Let $\langle D, R, I, T\rangle$ and $\langle D, R, I, T'\rangle$ be as in Lemma 17. Then $\langle D, R, I, T\rangle$ is solvable with a polynomial plan if and only if $\langle D, R, I, T'\rangle$ is solvable with a polynomial plan.*

As in the previous section, when only rotations were permitted, we will again talk about *induced permutation groups*, however, now they are induced by simple moves as well as rotations. Combining the results from the previous two sections, we can now state one of the main results of this paper.

Theorem 19. *Regardless of whether simple moves or synchronous rotations are allowed, diMAPF on strongly connected digraphs can be decided in polynomial time, and plans are polynomially bounded.*

Proof. If only simple moves are allowed, Lemma 6 applies. If only synchronous rotations are allowed or are possible because there is no blank, the claim follows from Lemma 16.

In case, simple moves and rotations are possible, we proceed by case analysis over the number of nodes n:

1. $n < 8$: The instance can be decided in constant time and the plans have $O(1)$ length.

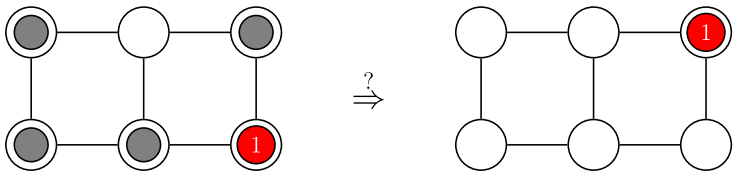

Fig. 13. A motion planning example.

2. $n \geq 8$: We proceed by case analysis over the structure of the digraph:
   (a) *Cycle digraph*: Similar to the proof of Lemma 15, the upper bound for plan length is polynomial and solvability is obviously polynomial-time.
   (b) The case of digraphs with at least 8 nodes that are not cycle digraphs will be reduced to the case where only rotations are allowed.
   If all cycles have odd length, then using Lemma 17, move one blank to a node that is a non-articulation node. Such a node must exist. Make sure that this blank will always be blank after emulating rotations on not fully occupied cycles. This will not destroy transitivity, and it effectively introduces at least one rotation of even length (corresponding to an odd permutation). Consider all remaining blanks as "virtual agents." Now each possible synchronous rotation on a cycle containing such virtual agents or the fixed blank can be emulated by a sequence of simple moves on this cycle. Applying Lemmas 14 and 12 gives us together with Theorem 9 a polynomial diameter. With the presence of an odd permutation and Lemma 10, the induced group must be symmetric, i.e., the instance is solvable for all $\langle I, T\rangle$ pairs. This means, it does not matter how the goal configuration for the "virtual agents" looks like and all instances are solvable.

This means the claim follows in all possible cases.

It is now straight-forward to generalize the result that diMAPF plans have polynomial length on strongly connected components to digraphs in general.

Lemma 20. *Regardless of whether simple moves or rotations are allowed, diMAPF plans can be polynomially bounded.*

Proof. Consider all strongly connected components of the digraph D of the solvable instance $\langle D, R, I, T\rangle$. For a given plan, focus on the events when an agent enters the strongly connected component, leaves the component, or moves to its final destination in the component without moving afterwards. In each strongly connected component there can only be $2|R| \leq 2|V|$ such events, because each agent can only enter and leave a component once. Between two such events, arbitrarily many movements of agents in this component may occur. As stated in Theorem 19, if there is a plan on a strongly connected graph, then there exist also one of polynomial length $p(|V|)$, regardless of what kind of movements are allowed. Since there are at most $|V|$ strongly connected components, there must a plan with no more than $2|V|^2 \times p(|V|)$ moves, i.e., a plan of polynomial length.

This result enables us to finally settle the question for the computational complexity of diMAPF in general.

Theorem 21. *The diMAPF problem is NP-complete, even when synchronous rotations are possible.*

Proof. As established earlier, diMAPF is NP-hard, regardless of whether synchronous rotations are allowed or not (Lemma 1). NP membership follows from Lemma 20.

8. The graph motion planning problem

The *graph motion planning problem with one robot (GMP1R)* as introduced by Papadimitriou et al. [30] is very similar to the MAPF problem with simple moves, but only one agent needs to reach a goal, while the other agents (called movable obstacles) can move, but do not have a goal. Fig. 13 provides an example, where the gray unnumbered disks are movable obstacles.

Wu and Grumbach [9] studied solvability for this problem on directed graphs and proved that the problem can be solved in polynomial time for strongly connected digraphs and DAGs. Combining the two results, they showed that the problem is polynomial on digraphs in general [41]. In the conclusion of their journal paper [9], they suggested to study the more difficult problem when k robots are allowed, called GMPkR. Wu and Grumbach [9] and Papadimitriou et al. [30] state that Kornhauser et al. [6] have already solved a special case of this problem, namely, GMPkR without any obstacles for $k \leq n-1$. Since n is not constant, k is neither. So, this appears to be the problem with arbitrarily many robots and *GMPR* might be the better abbreviation for it, which I will use in the following.

With the results achieved in this paper, we can give a complete characterization of the computational complexity of GMPR. The main observation is that the additional anonymous movable objects neither help nor hinder. The key for solving the problem is to "de-anonymize" the obstacles in an arbitrary way. Although the original formulation of the problem allows only for simple movements, we will cover also the cases when rotations are allowed.

Theorem 22. *GMPR on digraphs is NP-complete, even on DAGs. And this holds regardless of whether synchronous rotations are permitted or not.*

Proof. Since diMAPF is the special case of GMPR with no obstacles, NP-hardness follows from Lemma 1 for DAGs, and therefore also for the general case.

NP membership follows because GMPR plans can be polynomially bounded. If there exists a successful GMPR plan, then there exists obviously also a successful diMAPF plan where the movable obstacles have been assigned names and goals. Because of Lemma 20, we know that such a plan can be polynomially bounded, which obviously applies also to the original GMPR plan.

So, for one robot, the problem is polynomial, for arbitrarily many, it is NP-complete. It is not obvious, what the complexity of GMPkR for any fixed $k > 1$ is, however.

In parallel to Theorem 19, one would expect that the decision problem GMPR on strongly connected digraphs is a polynomial-time problem. However, the non-deterministic assignment of names and goals to anonymous obstacles as used in the proof of the previous theorem is obviously not enough. It is nevertheless possible to show polynomiality also in this case.

Theorem 23. *GMPR on strongly connected digraphs can be decided in polynomial time. And this holds regardless of whether synchronous rotations are permitted or not.*

Proof. The claim holds obviously for small digraphs with less than 8 nodes. It also holds trivially for cycle digraphs. For the remaining digraphs we proceed by case analysis over the allowed movements.

If synchronous rotations are allowed, there is no blank and there is at most one obstacle, then the problem reduces to diMAPF, and is by Theorem 19 polynomial. If there is no blank, but more than one obstacle, then the instance is always solvable. In order to see that assume that each obstacle has a unique name and has as its goal any position that is not a goal position for any other agent. If there exists a cycle of even length, then the instance is clearly solvable. If all cycles are of odd size, then the induced group is $\mathbf{A}_n$ with the same arguments as in the proof of Lemma 16. If the permutation induced by the initial and goal configuration is even, then the diMAPF instance and, hence, the original GMPR instance is solvable. Otherwise, exchange the goal positions of two obstacles, which is an odd permutation. This implies that this modified instance is solvable if the original one was not. Finally, if there is at least blank, then we can assume arbitrary names and goal positions for the obstacles. Using the same arguments as in the proof of Theorem 19 it follows that the induced group is symmetric, i.e., all instances are solvable.

For the case that only simple moves are allowed, we will make use of the reduction to the original MAPF problem on the underlying undirected graph (Corollary 5). We will again assume that each obstacle has a unique name. As in the paper by Kornhauser et al. [6], we assume a decomposition of the graph into transitive components. Since no agent can move from one transitive component to another one, these can be solved in isolation. The goal position of each obstacle is any position inside its transitive components no other agent wants to go to. Applying Theorem 1 from Kornhauser et al.'s [6] paper, we know that if the (sub-)graph is either separable or biconnected with two blanks, then it is solvable. If it is biconnected with only one blank, then it is unconditionally solvable, if it is not bipartite. If it is bipartite and the permutation induced by the initial and goal configuration is even, then it is solvable. If the GMPR instance contains at least two obstacles, then it is also always solvable, using the same arguments as above.

So, in all cases, there exist polynomial-time criteria for deciding the solvability of the GMPR instance.

An upper bound for the general problem is clearly also an upper bound for the problem with a fixed k, regardless of how large k is.

Corollary 24. *GMPkR for any fixed k on strongly connected digraphs can be decided in polynomial time. And this holds regardless of whether synchronous rotations are permitted or not.*

9. Conclusion and discussion

This paper provides an answer to a long-standing open question about the computational complexity of *diMAPF*, the multi-agent pathfinding problem on directed graphs. Kornhauser et al. [6] had shown already almost 40 years ago that deciding MAPF on undirected graphs is a polynomial-time problem. However, the problem for directed graphs had been ignored. Only in recent years [27,28,26,29] MAPF on directed was considered, but a result parallel to the one by Kornhauser et al. [6] was missing.

We showed that diMAPF is NP-complete in the general case. For the important special case of strongly connected digraphs it was shown that the problem can be decided in polynomial time. And these results hold regardless of whether one allows synchronous rotations in addition to simple moves.

At the same time, this answers a question about the generalization of the graph robot movement problem with one robot (GMP1R) [30] to directed graphs with arbitrarily many robots [9], where one has to deal with movable obstacles in addition to agents. Interestingly, though, one achieves exactly the same results as in the case without obstacles. However, the computational complexity of this problem for k robots, called GMPkR, is still open for all $k > 1$.

The results might have only a limited impact for the development of practical algorithms. They nevertheless demonstrate that DAGs are harmful and that even on strongly connected graphs things can get hairy when a partial state should be recovered.

From a theoretical point of view, there are also a number of points worth to mention. First of all, it shows that group theory is applicable for the analysis of diMAPF, something that was not obvious previously [28]. Second, in proving the result, some unforeseen obstacles popped up, such as that Lemma 1 from the paper by Kornhauser et al. [6] is not applicable to directed graphs and that there are two different directed counterparts to the T_0-graph. Third, the result could be taken as a suggestion to extend the diBOX algorithm [28] or the algorithm for solving diMAPF on strongly connected digraphs [29] in order to deal with the one-blank case and/or perhaps with synchronous rotation. Although the proofs are all constructive, for the sake of simplicity in the presentation, no effort was invested in coming up with tight bounds or optimal algorithms. So, at this point some work appears to be necessary.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

Acknowledgements

I would like to thank all reviewers of this paper and of the underlying ICAPS papers for their feedback. Their feedback has definitely improved the readability of the paper, I believe. Any remaining flaws should only be attributed to me, though.

Appendix A. SageMath script for the proof of Lemma 14

You find this SageMath script and others related to this paper at https://github.com/BernhardNebel/diMAPF.

```
def shared(c1, c2):
  return len(set(c1) & set(c2))

def ptype(c1, c2):
  if len(c1) > len(c2): c1,c2 = c2,c1
  if c1 != c2 and shared(c1,c2) > 0:
    return (len(c1)-shared(c1,c2)-1,
              shared(c1,c2),
              len(c2)-shared(c1,c2)-1)

def t0pairs(dig):
  for c1 in dig.all_simple_cycles():
    for c2 in dig.all_simple_cycles():
      if ptype(c1,c2) not in \
        [(2,2,2),(1,3,2),None]: return
  return True

t0a={"a1":["b3"],"b3":["b2"],"b2":["b1"],
   "b1":["a1","c1"],"c1":["c2"],"c2":["b3"]}
t0b={"a1":["a2"],"a2":["b2" ],"b2":[ "b1"],
   "b1":["a1","c1"],"c1":["c2"],"c2":["b2"]}
ears = [["e1"],["e1","e2"]]
tested = []

for g, p in ((t0a, (1,3,2)), (t0b,(2,2,2))):
  for h in g.keys():
    for t in g.keys():
      for e in ears:
        t0 = DiGraph(g)
        t0.add_path([h]+e+[t])
        if all([not t0.is_isomorphic(d) for d in tested]):
          tested += [t0]
          if t0pairs(t0): print(p,[h]+e+[t])
```

References

[1] H. Ma, S. Koenig, AI buzzwords explained: multi-agent path finding (MAPF), AI Matters 3 (3) (2017) 15–19, https://doi.org/10.1145/3137574.3137579.

[2] R. Stern, N.R. Sturtevant, A. Felner, S. Koenig, H. Ma, T.T. Walker, J. Li, D. Atzmon, L. Cohen, T.K.S. Kumar, R. Barták, E. Boyarski, Multi-agent pathfinding: definitions, variants, and benchmarks, in: Proceedings of the Twelfth International Symposium on Combinatorial Search (SOCS 2019), AAAI Press, 2019, pp. 151–159, https://ojs.aaai.org/index.php/SOCS/article/download/18510/18301/22026.

[3] P.R. Wurman, R. D'Andrea, M. Mountz, Coordinating hundreds of cooperative, autonomous vehicles in warehouses, AI Mag. 29 (1) (2008) 9–20, https://doi.org/10.1609/aimag.v29i1.2082.

[4] D. Sislák, P. Volf, M. Pechoucek, Agent-based cooperative decentralized airplane-collision avoidance, IEEE Trans. Intell. Transp. Syst. 12 (1) (2011) 36–46, https://doi.org/10.1109/TITS.2010.2057246.

[5] D. Silver, Cooperative pathfinding, in: R.M. Young, J.E. Laird (Eds.), Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005), AAAI Press, 2005, pp. 117–122, https://www.davidsilver.uk/wp-content/uploads/2020/01/coop-path-AIIDE.pdf.

[6] D.M. Kornhauser, G.L. Miller, P.G. Spirakis, Coordinating pebble motion on graphs, the diameter of permutation groups, and applications, in: 25th Annual Symposium on Foundations of Computer Science (FOCS-84), 1984, pp. 241–250.

[7] B. Nebel, On the computational complexity of multi-agent pathfinding on directed graphs, in: Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS-20), AAAI Press, 2020, pp. 212–216.

[8] B. Nebel, The small solution hypothesis for MAPF on strongly connected directed graphs is true, in: Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS-23), AAAI Press, 2023, pp. 304–313.

[9] Z. Wu, S. Grumbach, Feasibility of motion planning on acyclic and strongly connected directed graphs, Discrete Appl. Math. 158 (9) (2010) 1017–1028, https://doi.org/10.1016/j.dam.2010.02.001.

[10] W.W. Johnson, W.E. Story, Notes on the "15" puzzle, Am. J. Math. 2 (4) (1879) 397–404, https://doi.org/10.2307/2369492, https://www.jstor.org/stable/2369492.

[11] R.M. Wilson, Graph puzzles, homotopy, and the alternating group, J. Comb. Theory, Ser. B 16 (1) (1974) 86–96, https://doi.org/10.1016/0095-8956(74)90098-7.

[12] G. Röger, M. Helmert, Non-optimal multi-agent pathfinding is solved (since 1984), in: Proceedings of the Fifth Annual Symposium on Combinatorial Search (SOCS 2012), AAAI Press, 2012, pp. 173–174.

[13] O. Goldreich, Shortest move-sequence in the graph-generalized 15-puzzle is NPH, this manuscript cited in [6] has been published in 2011, in: O. Goldreich (Ed.), Studies in Complexity and Cryptography, Miscellanea on the Interplay Between Randomness and Computation, 1984.

[14] D. Ratner, M.K. Warmuth, Finding a shortest solution for the N × N extension of the 15-puzzle is intractable, in: Proceedings of the 5th National Conference on Artificial Intelligence (AAAI 1986), 1986, pp. 168–172, https://www.aaaipress.org/Papers/AAAI/1986/AAAI86-027.pdf.

[15] M.R.K. Ryan, Exploiting subgraph structure in multi-robot path planning, J. Artif. Intell. Res. 31 (2008) 497–542, https://doi.org/10.1613/jair.2408.

[16] P. Surynek, An application of pebble motion on graphs to abstract multi-robot path planning, in: 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2009), IEEE Computer Society, 2009, pp. 151–158.

[17] 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), 2010, pp. 1261–1263.

[18] 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, pp. 173–178.

[19] J. Yu, S.M. LaValle, Structure and intractability of optimal multi-robot path planning on graphs, in: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013), 2013, pp. 1443–1449.

[20] J. Yu, D. Rus, Pebble motion on graphs with rotations: efficient feasibility tests and planning algorithms, in: Algorithmic Foundations of Robotics XI (WAFR-14), in: Springer Tracts in Advanced Robotics, Springer, 2014, pp. 729–746.

[21] J. Yu, Intractability of optimal multirobot path planning on planar graphs, IEEE Robot. Autom. Lett. 1 (1) (2016) 33–40, https://doi.org/10.1109/LRA.2015.2503143.

[22] J. Banfi, N. Basilico, F. Amigoni, Intractability of time-optimal multirobot path planning on 2D grid graphs with holes, IEEE Robot. Autom. Lett. 2 (4) (2017) 1941–1947, https://doi.org/10.1109/LRA.2017.2715406.

[23] T. Geft, D. Halperin, Refined hardness of distance-optimal multi-agent path finding, in: 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2022, pp. 481–488.

[24] H. Ma, C.A. Tovey, G. Sharon, T.K.S. Kumar, S. Koenig, Multi-agent path finding with payload transfers and the package-exchange robot-routing problem, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016), AAAI Press, 2016, pp. 3166–3173.

[25] K.C. Wang, A. Botea, Fast and memory-efficient multi-agent pathfinding, in: Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008), AAAI Press, 2008, pp. 380–387, https://cdn.aaai.org/ICAPS/2008/ICAPS08-047.pdf.

[26] J. Li, A. Tinka, S. Kiesel, J.W. Durham, T.K.S. Kumar, S. Koenig, Lifelong multi-agent path finding in large-scale warehouses, in: Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021), AAAI Press, 2021, pp. 11272–11281.

[27] A. Botea, P. Surynek, Multi-agent path finding on strongly biconnected digraphs, in: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015), AAAI Press, 2015, pp. 2024–2030.

[28] A. Botea, D. Bonusi, P. Surynek, Solving multi-agent path finding on strongly biconnected digraphs, J. Artif. Intell. Res. 62 (2018) 273–314, https://doi.org/10.1613/jair.1.11212.

[29] S. Ardizzoni, I. Saccani, L. Consolini, M. Locatelli, Multi-agent path finding on strongly connected digraphs, in: 61st IEEE Conference on Decision and Control (CDC 2022), IEEE, 2022, pp. 7194–7199.

[30] C.H. Papadimitriou, P. Raghavan, M. Sudan, H. Tamaki, Motion planning on a graph (extended abstract), in: 35th Annual Symposium on Foundations of Computer Science (FOCS-94), 1994, pp. 511–520.

[31] M.R. Garey, D.S. Johnson, Computers and Intractability—A Guide to the Theory of NP-Completeness, Freeman, San Francisco, CA, 1979.

[32] J. Mulholland, Permutation Puzzles: A Mathematical Perspective, Self Published, Burnaby, B.C., Canada, 2021, http://www.sfu.ca/~jtmulhol/permutationpuzzles.

[33] R.M. Karp, Reducibility among combinatorial problems, in: Proceedings of a Symposium on the Complexity of Computer Computations, in: The IBM Research Symposia Series, Plenum Press, New York, 1972, pp. 85–103.

[34] B. Nebel, T. Bolander, T. Engesser, R. Mattmüller, Implicitly coordinated multi-agent path finding under destination uncertainty: success guarantees and computational complexity, J. Artif. Intell. Res. 64 (2019) 497–527, https://doi.org/10.1613/jair.1.11376.

[35] J. Bang-Jensen, G.Z. Gutin, Digraphs - Theory, Algorithms and Applications, second edition, Springer Monographs in Mathematics, Springer, 2009.

[36] J.R. Driscoll, M.L. Furst, On the diameter of permutation groups, in: Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC-83), ACM, 1983, pp. 152–160.

[37] H. Wielandt, Finite Permutation Groups, Academic Press, New York, NY, 1964.

[38] D.M. Kornhauser, Coordinating pebble motion on graphs, the diameter of permutation groups and applications, Master's thesis, MIT, Cambridge, 1984, https://www.cs.cmu.edu/~glmiller/Publications/Papers/KornhauserMaster84.pdf.
[39] P. Bachor, R.-D. Bergdoll, B. Nebel, The multi-agent transportation problem, in: Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2023), AAAI Press, 2023, pp. 11525–11532.
[40] The Sage Developers, SageMath, the Sage mathematics software system (version 9.6), https://www.sagemath.org, 2022.
[41] Z. Wu, S. Grumbach, Feasibility of motion planning on directed graphs, in: Theory and Applications of Models of Computation, 6th Annual Conference (TAMC 2009), Springer, 2009, pp. 430–439.