On the Computational Complexity of
Multi-Agent Pathfinding on Directed Graphs

Bernhard Nebel

Albert-Ludwigs-Universität
Freiburg, Germany
nebel@uni-freiburg.de

Abstract

The determination of the computational complexity of multi-agent pathfinding on directed graphs has been an open problem for many years. For undirected graphs, solvability can be decided in polynomial time, as has been shown already in the eighties. Further, recently it has been shown that a special case on directed graphs can be decided in polynomial time. In this paper, we show that the problem is NP-hard in the general case. In addition, some upper bounds are proven.

Introduction

The multi-agent pathfinding (MAPF) problem is the problem of deciding the existence of a movement plan for a set of agents moving on a graph, most often a graph generated from a grid (Ma and Koenig 2017). An example is provided in Figure 1. Here, the circular agent C wants to move to
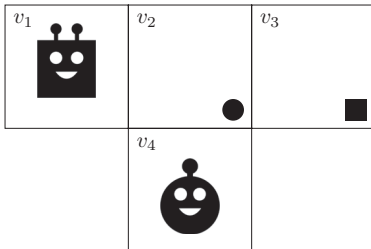

Figure 1: Multi-agent pathfinding example

v_2 and the square agent S wants to move to v_3. Both want to reach their destination and then stay there. So, S could move to v_2 and then to v_3. After that C could move to its destination v_2. So, in this case, a movement plan exists. Note that for this graph, regardless of how we place the agents and the destinations, there is always a movement plan, provided the destinations are on different grid fields. When removing v_4, however, there are situations for which no movement plan is possible.

Kornhauser et al. (1984) have shown in the eighties already that deciding solvability is a polynomial-time prob-

lem. Later on, variations of the problem have been studied, such as using parallel movements and considering optimal movement plans (Surynek 2010; Yu and LaValle 2013; Ma et al. 2016; Felner et al. 2017). However, in almost all cases, the results apply to undirected graphs only. A notable exception is the paper by Botea et al. (2018), which shows polynomial-time decidability for MAPF on directed graphs, provided the graph is strongly biconnected and there are at least two unoccupied vertices. The general case has been, however, open so far.

In a similar vein, Wu and Grumbach (2010) generalized the robot movement problem on an undirected graph as introduced by Papadimitriou et al. (1994) to directed graphs. The robot movement problem is the problem of finding a plan to move a robot from a vertex s to a vertex t, whereby mobile obstacles on vertices can be moved around but are not allowed to collide. Wu and Grumbach showed that solvability can be decided in polynomial time if the graph is either acyclic or strongly connected. In their conclusion they suggested to study the more difficult problem when all mobile obstacles are themselves also agents, which again is the MAPF problem on directed graphs.

We address this open problem by showing that the MAPF problem on directed graphs, which we will call *diMAPF*, is NP-hard. Interestingly, proving completeness for this problem seems to be quite non-trivial and we will only provide a loose upper bound for the general case, a tight upper bound for the special case of acyclic directed graphs, and a conditional result.

Notation and Terminology

A *graph* G is a tuple (V, E) with $E \subseteq \{\{u, v\} \mid u, v \in V\}$. The elements of V are called *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 V are called *vertices*, the element of A *arcs*. Given a digraph D, the *underlying graph* of $D = (V, A)$, in symbols $\mathcal{G}(D)$, is the graph resulting from ignoring the direction of the arcs, i.e., $\mathcal{G}(D) = (V, \{\{u, v\} \mid (u, v) \in A\})$. 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 $D = (V, A)$ (or a graph $G = (V, E)$),

the digraph $D' = (V', A')$ (resp. graph $G' = (V', E')$) is called *sub-digraph* of D (resp. *sub-graph* of G)) if $V \supseteq V'$ and $A \supseteq A'$ (resp. $E \supseteq E'$). Let $D = (V, A)$ again be a directed graph (or $G = (V, E)$ a graph) and let $X \subseteq V$. Then by $D - X$ (resp. $G - X$) we refer to the sub-digraph $(V - X, A - (X \times V) - (V \times X))$ (resp. $(V - X, E - \{\{u, v\} \mid u \in X \vee v \in X\})$).

A *path* in a digraph $D = (V, A)$ (or a graph $G = (V, E)$) is a non-empty sequence of vertices and arcs (resp. edges) of the form $v_0, e_1, v_1, \ldots, e_k, v_k$ such that $v_i \in V$, for all $0 \le i \le k$, $v_i \ne v_j$ for all $0 \le i < j \le k$, $e_j \in A$ (resp. $e_j \in E$) for all $1 \le j \le k$, and $(v_{j-1}, v_j) = e_j$ for all $1 \le j \le k$. A *cycle* in a digraph $D = (V, A)$ (or a graph $G = (V, E)$) is a non-empty sequence of vertices $v_0, v_1, \ldots, v_k$ such that $v_0 = v_k$, $(v_i, v_{i+1}) \in A$ (resp. $\{v_i, v_{i+1}\} \in E$) for all $0 \le i < k$ and $v_i \ne v_j$ for all $0 \le i < j < k$. If a digraph does not contain any cycle, it is called *directed acyclic graph (DAG)*.

A graph $G = (V, E)$ is said to be *connected* if there is a path between each pair of vertices. It is *biconnected* if $G - \{v\}$ is connected for each $v \in V$. Similarly, 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, there is a path in D from u to v and one from v to u. The smallest strongly connected digraph is the one with one vertex and no arcs. A digraph is called *strongly biconnected* if it is strongly connected and the underlying graph $\mathcal{G}(D)$ is biconnected.

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. The *condensation* of a digraph D is the digraph consisting of its strongly connected components D_i: $C(D) = (\{D_i\}, \{(D_i, D_j) \mid (u, v) \in A, u \in V_i, v \in V_j, D_i \ne D_j\})$. Note that $C(D)$ is a DAG.

A *multi-agent pathfinding (MAPF) instance* is given by a graph $G = (V, E)$, a set of *agents* R with $|R| \le |V|$, an *initial state* that is an injective function $s : R \to V$, and a *goal state* that is another injective function $t : R \to V$. The vertex $t(r)$ is called *destination* of agent r. Given a *state* s, one possible *successor state* s' is the function such that one agent r *moves* from one vertex to an adjacent vertex: If $s(r) = u$, $\{u, v\} \in E$ and there is no $r' \in R$ such that $s(r') = v$, then the successor state s' is identical to s except at the point r, where $s'(r) = v$. The MAPF problem is then to decide whether there exists a sequence of moves that transforms s into t.

Often the MAPF problem is defined in terms of parallel movements (Ryan 2008; Surynek 2010), where one step consists of parallel move and *wait* actions of all agents. However, as long as we are interested only in solution existence, there is no difference between the MAPF problems with parallel and sequential movements. If we allow for *simultaneous cyclic rotations* (Standley and Korf 2011; Yu and LaValle 2013), where one assumes that all agents in a fully occupied cycle can move simultaneously, things are a bit different. For the hardness proof latter on such movements are irrelevant, though.

Multi-agent pathfinding on directed graphs (diMAPF) 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.

We assume that the reader is familiar with basic notions from computational complexity theory (Papadimitriou 1994).

A Lower Bound for *diMAPF*

As mentioned above, Kornhauser et al. (1984) have shown that deciding MAPF (on undirected graphs) is a polynomial-time problem and that movement plans have only cubic length in the number of vertices. Botea et al. (2018) have shown that deciding solvability of diMAPF is again a polynomial-time problem and plans have cubic length, provided the digraph is a strongly biconnected digraph and there are at least two empty vertices. One intuitive reason for these positive results are 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 a digraph without strong connectivity, moves are not necessarily reversible and an agent might paint itself 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 is the case in the reduction from 3SAT that we use in the proof of the following theorem.

Theorem 1. *The diMAPF problem is NP-hard, even when simultaneous cyclic rotations are allowed.*

Proof. We prove NP-hardness by a reduction from the 3SAT problem, the problem of deciding satisfiability for a formula in conjunctive normal form with 3 literals in each clause. 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.[1] 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:

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

We proceed by constructing three gadgets, which we call *sequencer*, *clause evaluator*, and *collector*, respectively. We illustrate the construction using the example in Figure 2. In this visualization, vertices occupied by an agent are shown as squares containing the name of the occupying agent. Black circles symbolize empty vertices. Each vertex is labelled by its identifier, perhaps followed by a colon and the name of an agent in order to symbolize the destination for this agent. For example, v_1 is the destination for agent f_1.
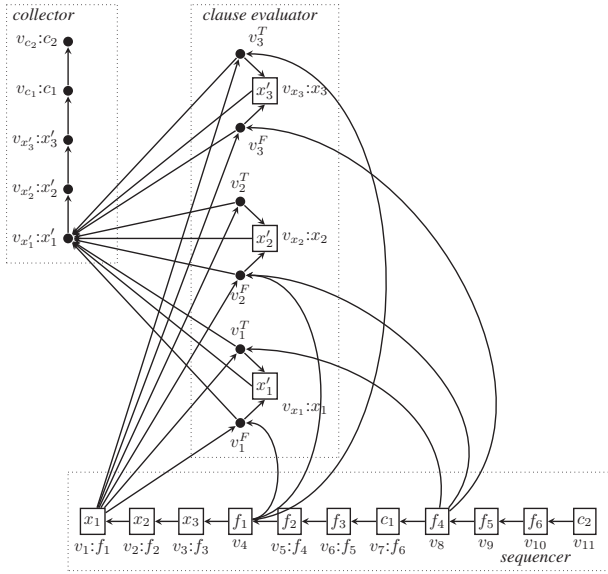

Figure 2: Example for $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

The task of the *sequencer* is to enforce first 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* $c_j, 1 \le j \le m$ 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 $v_{x_i'}$ 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, i.e., simultaneous cyclic rotations are impossible. This implies that the problem is NP-hard even if such movements were allowed. □

Upper Bounds for *diMAPF*

While the result of the previous section 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 way obvious. The main obstacle in proving this is the fact that the state space of the diMAPF problem is exponential. Nevertheless, it cannot be more complex than the propositional STRIPS planning problem, which has a similar state space (Bylander 1994). Indeed, the proof below uses exactly the arguments as Bylander's (1994, Theorem 3.1) PSPACE membership proof.

Proposition 2. *The diMAPF problem is in PSPACE.*

Proof. A movement sequence from the initial state to a goal state, if one exists, can be generated non-deterministically using for each movement only polynomial space (the representation of the two states). One can verify that this non-deterministically generated sequence is indeed a successful movement sequence by checking during the generation process that each movement is legal and that the final state is the goal state using only polynomial space. In other words, the problem is in NPSPACE, which is identical to PSPACE (Savitch 1980). □

However, it is by no means obvious that one has to go through a significant part of the state space in order to arrive at the goal configuration, if this is possible at all. In particular, in cases similar to the one used in the proof of Theorem 1, it seems obvious that the number of moves is bounded polynomially.

Proposition 3. *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 Theorem 1, this implies the claim. □

When looking at what stops us from proving a general NP-completeness result, we notice that strongly connected components are the culprits. They allow agents to reach the same location twice with the other agents in a perhaps different configuration. This may imply that a particular configuration can only be reached when agents walk through exponentially many distinct configurations. We know from Botea et al. (2018) that for all strongly biconnected digraphs with at least two empty vertices, all configurations can be reached using only cubic many moves. If we allow for only one empty vertex, solution existence cannot be any longer guaranteed (Botea, Bonusi, and Surynek 2018) and it is not any longer clear whether a polynomial long sequence suffices, if the instance is solvable at all. If we further weaken the requirement to only strongly connected graphs, it is neither clear whether solvability can be decided in polynomial time nor whether movement sequences can be bounded polynomially, although the latter sounds very plausible. For this reason, we will assume it for now and call it the *short solution hypothesis for strongly connected digraphs*: "For each solvable diMAPF instance on strongly connected digraphs, there exists a movement plan of polynomial length."

Theorem 4. *If the short solution hypothesis for strongly connected digraphs is true, then diMAPF is NP-complete.*

Proof. NP-hardness follows from Theorem 1.

Assume a diMAPF instance on a digraph $D = (V, A)$ that is solvable, which implies that there exists a movement plan π for the agents on D. This plan may be arbitrarily long. Consider now each strongly component in isolation and focus on the events when an agent enters the component, leaves the component, or moves to its final destination in the component without moving afterwards. In each component there can only be $2|R| \leq 2|V|$ such events because the condensation of D is a DAG. Between two such events, arbitrarily many movements of agents in this component may occur in the original plan π. However, since we assumed the short solution hypothesis to be true, there must also be a plan of polynomial length $p(|V|)$. 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 implies that the problem is in NP. □

Conclusion and Outlook

We gave a first answer to a long-standing open problem, namely, what the computational complexity of MAPF on digraphs is. In contrast to solvability on undirected graphs, which is a polynomial time problem, solvability on digraphs turns out to be NP-hard in the general case. While we also provide an NP upper bound for DAGs and a PSPACE upper bound in general, we were only able to show a conditional upper bound of NP for the general problem, provided the short solution hypothesis for strongly connected digraphs is true.

While the result in itself may not have a high relevance for practical purposes, it still is significant in ruling out the possibility of a polynomial-time algorithm similar to the one developed by Kornhauser et al. (1984). Furthermore, the short solution hypothesis could be taken as a suggestion that the result by Botea et al. (2018) could be strengthened to general strongly connected digraphs.

References

Botea, A.; Bonusi, D.; and Surynek, P. 2018. Solving multi-agent path finding on strongly biconnected digraphs. *Journal of Artificial Intelligence Research* 62:273–314.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1–2):165–204.

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N. R.; Wagner, G.; and Surynek, P. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Proceedings of the Tenth International Symposium on Combinatorial Search (SOCS-17)*, 29–37.

Kornhauser, D.; Miller, G. L.; and Spirakis, P. G. 1984. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium on Foundations of Computer Science (FOCS-84)*, 241–250.

Ma, H., and Koenig, S. 2017. AI buzzwords explained: multi-agent path finding (MAPF). *AI Matters* 3(3):15–19.

Ma, H.; Tovey, C. A.; Sharon, G.; Kumar, T. K. S.; and Koenig, S. 2016. 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-16)*, 3166–3173.

Nebel, B.; Bolander, T.; Engesser, T.; and Mattmüller, R. 2019. Implicitly coordinated multi-agent path finding under destination uncertainty: Success guarantees and computational complexity. *Journal of Artificial Intelligence Research* 64:497–527.

Papadimitriou, C. H.; Raghavan, P.; Sudan, M.; and Tamaki, H. 1994. Motion planning on a graph (extended abstract). In *35th Annual Symposium on Foundations of Computer Science (FOCS-94)*, 511–520.

Papadimitriou, C. H. 1994. *Computational complexity*. Addison-Wesley.

Ryan, M. R. K. 2008. Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research* 31:497–542.

Savitch, W. J. 1980. Relations between nondeterministic and deterministic tape complexity. *Journal of Computer and System Sciences* 4:177–192.

Standley, T. S., and Korf, R. E. 2011. Complete algorithms for cooperative pathfinding problems. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, 668–673.

Surynek, P. 2010. An optimization variant of multi-robot path planning is intractable. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI-10)*.

Wu, Z., and Grumbach, S. 2010. Feasibility of motion planning on acyclic and strongly connected directed graphs. *Discrete Applied Mathematics* 158(9):1017–1028.

Yu, J., and LaValle, S. M. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the Twenty-Seventh Conference on Artificial Intelligence (AAAI-13)*.