Journal of Artificial Intelligence Research 64 (2019) 497-527 Submitted 7/18; published 3/19

Implicitly Coordinated Multi-Agent Path Finding under Destination Uncertainty: Success Guarantees and Computational Complexity

Bernhard Nebel NEBEL@UNI-FREIBURG.DE
Institut für Informatik, Albert-Ludwigs-Universität Freiburg,
Georges-Köhler-Allee 52, 79110 Freiburg, Germany

Thomas Bolander TOBO@DTU.DK
Department of Applied Mathematics and Computer Science
Technical University of Denmark (DTU)
Richard Petersens Plads, building 324, DK-2800 Lyngby, Denmark

Thorsten Engesser ENGESSER@CS.UNI-FREIBURG.DE
Robert Mattmüller MATTMUEL@CS.UNI-FREIBURG.DE
Institut für Informatik, Albert-Ludwigs-Universität Freiburg,
Georges-Köhler-Allee 52, 79110 Freiburg, Germany

Abstract

In multi-agent path finding (MAPF), it is usually assumed that planning is performed centrally and that the destinations of the agents are common knowledge. We will drop both assumptions and analyze under which conditions it can be guaranteed that the agents reach their respective destinations using *implicitly coordinated plans* without communication. Furthermore, we will analyze what the computational costs associated with such a coordination regime are. As it turns out, guarantees can be given assuming that the agents are of a certain type. However, the implied computational costs are quite severe. In the distributed setting, we either have to solve a sequence of NP-complete problems or have to tolerate exponentially longer executions. In the setting with destination uncertainty, bounded plan existence becomes PSPACE-complete. This clearly demonstrates the value of communicating about plans before execution starts.

1. Introduction

In a spatial multi-agent environment, e.g., a warehouse (Wurman, D'Andrea, & Mountz, 2008), a street intersection (Dresner & Stone, 2008), an airport (Hatzack & Nebel, 2013), or a video game (Lawrence & Bulitko, 2013), agents have to move to different destinations in a collision-free manner. Such scenarios can be formalized as *multi-agent path finding* (MAPF) problems.

In its most basic variant, the problem can be described as follows. Given an undirected, simple graph $G = (V, E)$, a set of agents A, an initial configuration assigning agents to distinct vertices, and a goal configuration with another assignment of agents to distinct vertices, the question is how one can transform the initial configuration into the goal configuration by single movements, where one agent moves from a vertex to an empty adjacent vertex.

Often, the graph is given as a grid map as in Figure 1, where agents can move to orthogonally adjacent empty grid cells. In the displayed situation, the circular agent C wants to go to the cell marked by the solid circle and the square agent S wants to reach the place with the solid square (the empty circle and square will only become important later). One could come up with the following movements:

1. C moves to v_2 and then to v_4,

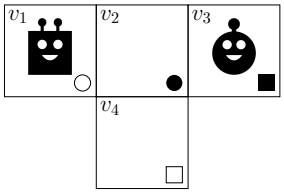


Figure 1: Multi-agent coordination example

2. S moves to v_2 and then to destination field v_3, and
3. C finally moves to destination field v_2.

This problem, and a number of variants, have been studied quite extensively, the computational complexity of these problems has been determined, and a number of optimal and sub-optimal algorithms have been proposed. The assumption has usually been that movements are computed centrally before execution starts. Furthermore, because of this assumption, destinations are considered to be common knowledge. In this paper we drop both assumptions and analyze whether the agents are able to coordinate their movements just by observing the movements of the other agents. Such a scenario is, for instance, plausible in human-robot interactions or when agents do not share a common communication channel.

As a first step, in Section 3 planning as well as execution are assumed to be distributed with no communication between the agents. In order to cope with the problem that the generated movement plans might be incompatible, replanning might be necessary. The question is then whether it is still possible to guarantee successful executions and what the computational price for such implicitly coordinated executions is. As we show, success guarantees can be given if we assume all agents to be *eager*, i.e., not waiting for others to act first, and *acting optimally*. Based on known complexity results concerning solving MAPF optimally, it follows that the planning problem in such an implicitly coordinated regime is *NP-complete*. As an alternative we explore the notion of *conservative eager* agents which start replanning from the initial situation following the already executed partial plan. While these agents do not need to solve NP-complete problems and can avoid infinite executions at the same time, the worst-case execution length can be exponentially longer than the length of a plan by a single agent.

As a second step, in Section 4 we drop the assumption that destinations are common knowledge. We call the resulting path-finding problem *MAPF under destination uncertainty* or simply *MAPF/DU*. In order to illustrate this point, let us again consider the situation in Figure 1, but unlike before, let us assume that each agent knows about its own destination with certainty (the solid circle and square), but there is uncertainty about the destinations of the other agent (the empty circle and square are considered as additional potential destinations in addition to the solid circle and square for C and S, respectively).

Here, we first have to come up with a solution concept. We introduce *i-strong branching plans* that correspond to *implicitly coordinated policies* as they have been proposed in the area of epistemic planning (Engesser, Bolander, Mattmüller, & Nebel, 2017). One interesting property of these plans is that one can reduce them to skeletons that are composed out of *stepping stones*, configurations in which one agent can reach its destination with certainty and success for the rest is guaranteed. Using this result, one can show that the worst-case execution costs of a branching plan for a MAPF/DU instance are polynomially bounded.

As a third step, in Section 5, we analyze joint execution of these branching plans as a generalization of joint executions for the fully observable case. Since the agents now have different perspectives, it can happen that some agents can come up with an i-strong plan while others are clueless, i.e., cannot form a plan. So, we introduce the stronger notion of *objectively strong* plans, which can be executed by any agent.

As in the fully observable setting, conservative eager agents are guaranteed to succeed, however, executions can have a length exponentially longer than a plan by a single agent. Since we also want to guarantee reasonable execution length, optimally eager agents are considered. Unfortunately, for them success cannot be guaranteed. As we demonstrate, it is possible to get caught in infinite executions. In order to address this problem, we combine conservative eagerness with optimality.

In Section 6, we have a look at the computational complexity of deciding the existence of bounded i-strong and objectively strong plans. We show that these problems are PSPACE-complete (in the number of agents). This demonstrates that communication about destinations pays off significantly. For the case that we deal only with few agents, we show that deciding existence and bounded existence can be done in time polynomial in the size of the graph provided the number of agents is fixed.

In Section 7 we summarize our results and discuss open problems and further research directions.

2. Related Work

There is a very rich body of research on the MAPF problem and its variations. The first paper in this area with substantial results is the one by Kornhauser, Miller, and Spirakis (1984). They considered memory contents moving over computer networks, formalized as *pebbles*. However, the results apply, of course, to agents in spatial environments represented as graphs as well. The paper spells out all important ingredients, demonstrates that solvability can be decided in polynomial time, and sketches an algorithm for generating movement plans. All the important details are, however, in Kornhauser's unpublished Master Thesis.

The paper by Ratner and Warmuth (1986) was the first published paper that looked at the problem of generating optimal movement plans. They showed that the problem of finding minimal plans for the generalized 15-puzzle is NP-complete. A simpler NP-completeness proof for the general pebble-movement problem can be found in the paper by Goldreich (2011), which had been written and circulated already in 1984, as cited in the paper by Kornhauser et al. (1984).

Later, variations of this problem were analyzed, for instance, MAPF with simultaneous moves (Surynek, 2010), MAPF on strongly bi-connected, directed graphs (Botea, Bonusi, & Surynek, 2018), and variations concerning payload transfers (Ma, Tovey, Sharon, Kumar, & Koenig, 2016). Furthermore, different metrics were considered (Yu & LaValle, 2013). In all cases, though, the problem of generating shortest plans remains NP-complete.

Complete MAPF solvers (optimal and sub-optimal) were proposed for different variations of the problem (Wang & Botea, 2011; de Wilde, ter Mors, & Witteveen, 2014; Luna & Bekris, 2011; Felner et al., 2017). Taking some of the kinematics of real robots into account, the multi-robot path finding problem was studied as well (Grady, Bekris, & Kavraki, 2010; Bhattacharya, Kumar, & Likhachev, 2010). Finally, also uncertainty of the position or the movements was taken into account (Wagner & Choset, 2017).

If optimality and/or completeness is an issue, then central planners are usually used. However, they have to plan in the product space of the single agent search spaces, which leads to a restriction on how many agents can be handled. In order to scale better, often decoupled planning approaches are used that combine plans generated by/for single agents (Silver, 2005; Jansen & Sturtevant, 2008). However, it is always assumed that the agents can communicate in order to resolve conflicts.

There exist also some truly distributed approaches to the multi-robot path-finding problem. However, they tend to be reactive and cannot guarantee completeness as we do or they need some form of minimal communication. The work by van den van den Berg, Lin, and Manocha (2008), for instance, considers non-communicating robots moving in a plane and uses the notion of *reciprocal velocity obstacle*, which anticipates that also other agents react in the same way. Nevertheless, such an approach will fail, for example in very narrow, complex environments. The work by Cáp, Vokrínek, and Kleiner (2015) gives completeness guarantees for "well-formed environments", but it needs some minimal form of communication.

As mentioned in the Introduction, there is almost always the assumption that all agents know the destinations of the other agents and that they act in a coordinated manner. This means that a plan can be generated centrally. On the other hand, there is a long tradition of analyzing general distributed planning and acting, (desJardins, Durfee, Ortiz, & Wolverton, 1999). Brenner and Nebel (2009), for example, looked at this problem. They also proposed as one of their benchmarks a problem similar to the MAPF/DU problem. However, their solution, which is a simple self-interested, greedy plan, did not include the anticipation of goals or actions by other agents as we consider in this paper. This means that their algorithm is incomplete with respect to our problem specification.

Distributed POMDPs (decPOMPDs) allow for distributed execution (Goldman & Zilberstein, 2004) and might therefore be considered as providing solutions to the problems such as MAPF/DU. However, decPOMDPs are based on a central offline planning process. This problem is overcome by using interactive POMDPs (Gmytrasiewicz & Doshi, 2005). However, it is not immediately clear, how one could use this framework in order to model and solve the MAPF/DU problem.

A completely different approach to model and solve a problem such as MAPF/DU could be to use *general game playing*. Extensions such as the one introducing games with imperfect information (Schiffel & Thielscher, 2014) could be used to model MAPF/DU instances, which could then be solved using an GDL-II game solver, e.g., HyperPlay (Schofield & Thielscher, 2017). Since the solver is sampling based, it is probably incomplete, though.

The approach that comes closest to the one proposed in this paper is epistemic multi-agent planning (Löwe, Pacuit, & Witzel, 2011; Bolander & Andersen, 2011; Andersen, Bolander, & Jensen, 2012; Muise et al., 2015; Engesser et al., 2017; Bolander, Engesser, Mattmüller, & Nebel, 2018). In epistemic multi-agent planning, the epistemic states of the other agents are taken into account when generating a plan. In particular, the notion of *implicitly coordinated plan* or *policy* introduced by Engesser et al. (2017) plays an important role in our paper. Furthermore, we make use of the notion of *joint execution* and *guarantees of success* of the joint execution of epistemic plans as discussed by Bolander et al. (2018). Since the MAPF/DU problem is much more specialized and simpler than the general epistemic planning problem (which is undecidable, as shown by Bolander and Andersen (2011)), it is possible to achieve some positive results. On the other hand, the results in this paper may be useful for giving inspirations to the research on general epistemic planning. Although some of the results in our paper are just instantiations of results from epistemic planning, we will prove them from first principles and point out the parallels to epistemic planning in general.

Finally, one should mention that one of the key concepts in solving the distributed MAPF and MAPF/DU problems is *replanning*. Of course, replanning or continual planning is not new and is used regularly to deal with contingency problems (Ambros-Ingerson & Steel, 1988; Brenner & Nebel, 2009; Brafman & Shani, 2012). Our approach uses replanning in a different way, though. Replanning is only used in order to account for unanticipated actions of the other agents. Since deviations of other agents are nevertheless part of a successful plan, replanning will never choose an action leading into a dead end, which might occur in other approaches. By requiring in addition eagerness and conservatism and/or optimality, we can even guarantee success for some types of agents. This distinguishes our approach from the usual continual planning approaches.

3. Distributed MAPF

As mentioned in the Introduction, the spatial environment is modeled using a graph $G = (V, E)$. Throughout the paper, we consider as usual *undirected, simple graphs*. A ***configuration*** of ***agents*** A on the graph G is an injective function $\alpha\colon A \to V$. For $i \in A$ and $v \in V$, by $\alpha[i/v]$ we refer to the function that has the same values for all $j \neq i$ as α, but for i it has the value v: $\alpha[i/v](i) = v$.

Given a ***movement action*** of agent i from v to v' and a configuration α, a ***successor configuration*** $\alpha' = \alpha[i/v']$ is generated, provided $\alpha(i) = v$, $(\alpha(i), \alpha'(i)) \in E$, and there exists no j with $\alpha(j) = v'$. The ***MAPF problem*** is then to generate for a given ***MAPF instance***

$\mathcal{M} = \langle A, G, \alpha_0, \alpha_* \rangle$ with a given set of agents A, a given graph G, the ***initial configuration*** α_0, and the ***goal configuration*** α_*, a sequence of movements from α_0 to α_*. Such a ***movement plan*** π is written as a sequence of movement triples consisting of the moving agent i and its current and next location: $(i, \alpha(i), \alpha'(i))$. We always assume that such movement plans are ***cycle-free***, i.e., that during the execution of such a plan no configuration is reached twice. We call a plan ***successful*** for a MAPF instance if it transforms α_0 into α_*. Since in the following we only consider successful movement plans, we just call them ***plans***. If there exists such a plan for a given instance, we call the instance ***solvable***.

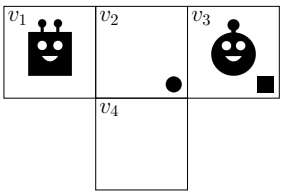
As mentioned in Section 2, solvability on undirected graphs can be decided in polynomial time; and if a plan exists, then its length can be bounded by a polynomial and it can be generated in polynomial time. If we are interested in shortest plans, then the corresponding problem of bounded existence for undirected graphs becomes NP-complete.

3.1 Distributed Planning and Execution

While in MAPF one usually considers the generation of a plan by a central instance and leaves the distributed execution to the agents, we now consider the setting where each agent generates a plan—consisting of its own movements and the movements of the other agents, leading to the goal configuration. We call this setting ***distributed MAPF*** We call such a plan ***implicitly coordinated*** since the planning agent presupposes that the other agents behave in a cooperative way. The underlying basic assumption is, of course, that all agents want to reach the goal configuration. But it would be a coincidence when all of them came up with the same plan. Nevertheless, if one agent acts, this will follow a plan towards the goal configuration—and so will never end up in a ***dead end***, i.e., a state from which the goal state cannot be reached.[1] And if an action by another agent was not anticipated, then one can replan in order to account for this unanticipated move.

After all agents have planned, we have a ***family of plans*** $(\pi_i)_{i \in A}$. ***Joint execution*** of this family of plans is then performed in an asynchronous, interleaved fashion. From all the agents i that have as their first action one of their own moves, one agent is chosen and its movement is executed. This is very similar to what happens in real-time board games, such as *Magic Maze*. The player who acts fastest carries out the action. For all the other agents the following happens: Either the movement was anticipated and then the movement is removed from the plan or the agent has to replan from the new situation. The interesting question is, whether such an asynchronous, distributed execution is guaranteed to eventually lead to the desired goal configuration and how many steps it takes to reach the common goal. The ***execution length*** of a joint execution is defined to be the number of steps that are used to reach the goal configuration.

In the example from the Introduction, sometimes joint execution leads to success. Assume that



$\pi_C = \langle (C, v_3, v_2), (C, v_2, v_4), (S, v_1, v_2), (S, v_2, v_3), (C, v_4, v_2) \rangle$
$\pi_S = \langle (S, v_1, v_2), (S, v_2, v_4), (C, v_3, v_2), (C, v_2, v_1), (S, v_4, v_2), (S, v_2, v_3), (C, v_1, v_2) \rangle$

Figure 2: Plans for the multi-agent coordination example

1. Note that in the basic and the distributed MAPF setting, there are no dead ends, because all actions are reversible. For directed graphs or in the setting with destination uncertainty that we introduce later, dead ends are possible.

the initial plan by agent C is the sketched one, i.e., the plan π_C in Figure 2. Now agent S might have generated the same plan, in which case both agents will reach their destinations. What will happen, however, if S had chosen a different plan? If agent S had generated a different plan, say π_S in Figure 2, then it may happen that C will be chosen to execute the first action and S is forced to replan. In this case, it might come up with π_C with the first action removed. If, on the other hand, S is chosen to act first, C needs to replan and might come up with π_S with the first action removed, and joint execution will be successful.

3.2 Success Guarantees for Joint Execution

The interesting question is, whether we can find conditions that guarantee success for such joint executions with replanning in the general case. In order to demonstrate one of the issues, let us assume that S comes up with the plan π_C (expecting C to act first), and that C comes up with π_S (expecting S to act first). In this case, both agents would wait for each other to act first forever.

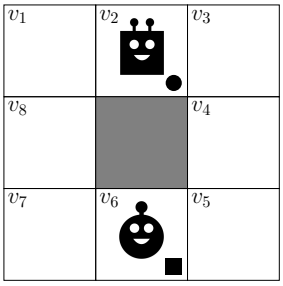
Intuitively, we would call these plan *lazy*, because they put the burden on an agent other than oneself. Let π be a movement plan. We say that π is a ***lazy plan*** *with respect to agent i and a given set Π of plans*, if there exists another movement plan $\pi' \in \Pi$ that has an identical prefix of $k \geq 0$ actions and the $k+1$th in π is a movement by agent $j \neq i$, while the $k+1$th action in π' is by agent i. We say that i is a ***lazy agent*** (wrt. Π) if it sometimes generates lazy plans. Clearly, lazy agents can produce plans that can lead to ***deadlock*** situations, i.e., situations in which all agents are waiting for each other to act first, as in the scenario described above. Note that a *deadlock* can occur although none of the plans contain a *dead end*.

We call i an ***eager agent*** if it never generates lazy plans (with respect to itself and the given set of plans). Note that if a plan is not lazy, then all of its suffixes started in the situation reached by the prefix are also not lazy (provided the suffixes also belong to the set of plans). The reason is that there does not exist *any* prefix such that after the prefix the agent chooses an action of another agent if it does not have to in order to reach the goal.

With eager agents, we avoid deadlock situations, since there is always at least one agent that can act. In our case, if both S and C from our initial example are eager wrt. to all plans, then C is bound to generate π_C and S will necessarily generate π_S, leading to a successful execution, since there are no other possible movement plans.

However, simple eager agents (being eager with respect to the set of all plans) have serious problems. The first problem is that they may plan to make moves that increase the distance to the goal configuration, e.g., by moving away from the destination node, as shown in Figure 3. The second problem, resulting from the first one, is that joint execution with replanning can easily lead to infinite executions, as demonstrated in Figure 3. Initially, S and C come up with the two plans π_1 (going around clockwise) and π_2 (going around couter-clockwise), respectively. As one can easily verify, both plans are not lazy wrt. the respective agents. Now S starts to execute (moving from v_2 to v_3). After that C has to replan (since C did not anticipate S's move). It comes up with π_3 inserting an action into the original plan π_2 that undoes S's action (without leading to a cycle in the revised plan). Note that the resulting plan is again not lazy for C (since C acts first until S has to move out of the way). Now, assume that C executes the first action from π_3. Since C is not following S's plan, S comes up with a new plan π_4 going around counter-clockwise. After executing the first action of π_4, C needs again to replan because the executed action deviates from C's plan π_3. So it comes up with π_5 (going around clockwise), which leads to the original configuration, if the first action of it is executed. From this situation on, they can repeat this forever. Note that by revising the plans, we have created a cycle in the actual execution, although every plan itself is cycle-free.

The problem can be addressed by restricting the set of plans Π to *shortest plans*. We measure the length of a plan by the number of actions it contains. In principle, we could also allow for general cost measures, as long as they are strictly positive.



π_1 (S initially): $\langle(\mathbf{S}, \mathbf{v_2}, \mathbf{v_3}), (S, v_3, v_4), (S, v_4, v_5), (C, v_6, v_7), (S, v_5, v_6), (C, v_7, v_8),$
$(S, v_6, v_7), (C, v_8, v_1), (S, v_7, v_8), (C, v_1, v_2), (S, v_8, v_7), (S, v_7, v_6)\rangle$

π_2 (C initially): $\langle(C, v_6, v_5), (C, v_5, v_4), (C, v_4, v_3), (S, v_2, v_1), (C, v_3, v_2), (S, v_1, v_8),$
$(C, v_2, v_1), (S, v_8, v_7), (C, v_1, v_8), (S, v_7, v_6)(C, v_8, v_1), (C, v_1, v_2)\rangle$

π_3 (C after (S, v_2, v_3)): $\langle(\mathbf{C}, \mathbf{v_6}, \mathbf{v_5}), (C, v_5, v_4), \underline{(S, v_3, v_2)}, (C, v_4, v_3), (S, v_2, v_1), (C, v_3, v_2),$
$(S, v_1, v_8), (C, v_2, v_1), \overline{(S, v_8, v_7)}, (C, v_1, v_8), (S, v_7, v_6), (C, v_8, v_1),$
$(C, v_1, v_2)\rangle$

π_4 (S after (C, v_6, v_5)): $\langle(\mathbf{S}, \mathbf{v_3}, \mathbf{v_2}), (S, v_2, v_1), (S, v_1, v_8), (S, v_8, v_7), (S, v_7, v_6), (C, v_5, v_4),$
$(S, v_6, v_5), (C, v_4, v_3), (S, v_5, v_4), (C, v_3, v_2)(S, v_4, v_5), (S, v_5, v_6)\rangle$

π_5 (C after (S, v_3, v_2)): $\langle(\mathbf{C}, \mathbf{v_5}, \mathbf{v_6}), (C, v_6, v_7), (C, v_7, v_8), (C, v_8, v_1), (S, v_2, v_3), (C, v_1, v_2),$
$(S, v_3, v_4), (C, v_2, v_3), (S, v_4, v_5), (C, v_3, v_4), (S, v_5, v_6), (C, v_4, v_3),$
$(C, v_3, v_2)\rangle$

Figure 3: Example for infinite execution

We call a plan ***optimally eager*** with respect to agent i if it is not lazy with respect to agent i and the set of shortest plans. Agents producing only such plans are called ***optimally eager agents*** (Bolander et al., 2018). As can be easily seen, such agents are always successful, provided the instance is solvable at all, as spelled out in the next Proposition.[2] The reason is that all agents produce plans of the same length, which in each execution step are shortened.

Proposition 1 *For MAPF instances, joint execution and replanning of movement plans generated by optimally eager agents is always successful, provided the instance is solvable.*

Proof: We use induction over the length of the shortest plan. For the base case 0, the proposition is obviously true. Now assume the proposition is true for all cases with length n or less. We show that it is true for length $n+1$ as well. If there exists a shortest plan of length $n+1$, then all agents will generate optimally eager plans with length $n+1$ and therefore there is at least one agent who has generated a plan, where the first movement can be executed by itself. After executing this action, some of the agents have to replan, coming up with new optimally eager plans of length n. The other agents just delete the first action of their plan. Since the action was anticipated, the remaining plans must still be optimally eager from the perspective of those other agents. So all agents have now optimally eager plans of length n and so the induction hypothesis applies. ■

This means that a form of implicit coordination can be achieved by observation and replanning under the assumption that everybody acts rationally in the sense that only shortest plans are considered. In our example from Figure 3, for instance, C will replan to move clockwise after the first move of S moving clockwise.

While this is good news, the bad news is that this implies that the agents have to solve an NP-hard problem (generating a shortest eager plan) not only once, but potentially after each action execution. Note that eagerness does not simplify the problem because any shortest plan for n agents

2. Note that this a special case of Proposition 9 in the paper by Bolander et al. (2018).

could be transformed into an eager plan for an additional agent that sits on an isolated destination node. On the other hand, the problem of deciding whether there exists an *eager plan of length k or less* (the decision problem corresponding to the optimization problem) is not harder than deciding whether there exists *any plan of length k or less*. The reason is that eagerness is defined on a set of plans, e.g. the ones with length k or less, i.e., if there are plans of length k, then there also eager ones.

Now the question may come up whether a computationally simpler version would be possible, using the fact that sub-optimal plans for the MAPF problem can be generated in polynomial time as pointed out in Section 2. Instead of trying to find the shortest, eager plan in order to avoid infinite executions, one can restrict replanning in a way such that already executed actions have to be part of an updated plan. In other words, when an agent has to replan, it creates the plan from the initial configuration α_0 and starts with the already executed actions.[3] Agents that replan in this way are called ***conservative agents***. An agent is a ***conservative eager agent*** if it is eager with respect to the agent and the set of plans containing the already executed plan as a prefix.

This way, one never will create a cyclic execution (since we assumed that plans are never cyclic), hence, executions are always finite. Note that this condition will never lead to a dead end, because an acting agent has always a valid plan to reach the goal configuration. Furthermore, this positive result does not depend on generating shortest plans, so we are not forced to solve NP-hard problems. Unfortunately, however, the polynomial upper bound for the number of movements goes out of the window. In fact, it is easy to construct an example where the entire (exponentially sized) state space is visited. This means, agents also have to remember exponentially many steps. Such an example is shown in Figure 4, where dots are empty nodes, boxes denote agents occupying a node, and destinations for the agents are specified by providing their identifiers after node names separated by a colon.

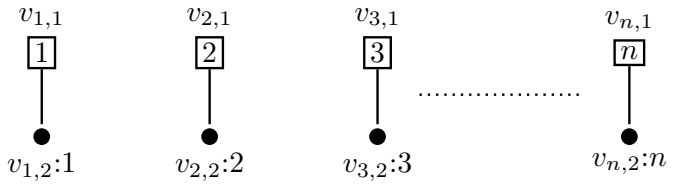


Figure 4: Example, which may result in an exponentially long execution

In this example, each agent i wants to move from its initial location $v_{i,1}$ to its destination $v_{i,2}$. An eager plan for agent 1 could look like as follows. Agent 1 moves down to $v_{1,2}$. Since cycles are not allowed, now another agent has to move, say 2. After that, 1 needs to move again (because it is eager), and then 3 could move to its destination. All in all, the shortest eager plan will have a length of $2n-1$ if n is odd and $2n-2$ if n is even. In general, each agent will generate a plan in which it moves first, provided the entire configuration is not the goal configuration.

When it comes to joint execution, it could happen that the order of execution is chosen in a way that corresponds to the bit change in a *Gray counter* (Gray, 1953). A Gray counter visits all possible bit configurations but in each transition only one bit is changed. In our case, this would amount to a move by the corresponding eager agent for each such transition. In other words, the agents will explore the entire state space of 2^n states before they stop in the goal configuration.

Proposition 2 *For MAPF instances, joint execution and replanning of movement plans generated by conservative eager agents is always successful, provided the instance is solvable. However, executions can have a length exponentially longer than the plan by a single agent.*

3. This is somewhat similar to tabu search (Glover, 1986).

Thus, it appears to be the case that when communication between the agents in the form of publishing a common plan is not possible, one has to tolerate high worst-case computational costs, either in form of solving NP-hard problems or in producing and remembering potentially exponentially long plans.

4. MAPF with Destination Uncertainty

Let us generalize the MAPF problem to a setting where the agents are only partially informed about the destinations of the other agents. This means that the goal configuration α_* is not common knowledge any longer, but only the agent itself knows its own destination. Common knowledge are the possible destinations for each agent, formalized by a ***destination function*** $\beta\colon A \to 2^V$, with the constraint that for all $i \in A$ either the real destination is among the possible ones, i.e., $\alpha_*(i) \in \beta(i)$, or $\beta(i) = \emptyset$, because agent i already arrived (and is not allowed to move anymore). We require further that all combinations of possible destinations are consistent, i.e., $\beta(i) \cap \beta(j) = \emptyset$ for all $i \neq j \in A$.[4] We, of course, still assume that all agents are cooperative, i.e., that they want to reach the goal configuration α_*.

Furthermore, we add a *success announcement* action for each agent. This action can be executed when the agent has reached its destination. Only by using such an action, the agents can establish common knowledge that they all have reached their respective destinations. We require that after the announcement the agent is not allowed to move anymore. However, an agent might visit its true destination without revealing it. We call this variation of the MAPF problem the ***multi-agent path-finding with destination uncertainty*** or ***MAPF/DU problem***.

One may question whether the condition of requiring the agent not to move after its success announcement is reasonable from a practical point of view and one may want to know whether other options are possible. First, when considering robots moving in an open space where they can observe each other, it is often very obvious when they have arrived at their destination. For example, logistics robots may load or unload packages, which is observable by the other robots. Airplanes moving on the ground stop moving either when they have arrived at a gate or after they leave the system via a runway.

Second, one could, of course, consider other options for the rules around a success announcement. For example, one could impose that an agent has to stop and announce success whenever it is on a node that is the actual destination of the agent. This would render many problem instances unsolvable, e.g. the one in Figure 1. A more liberal version would be to allow an agent to announce their destination when they are on the node without being forced to stay put. While it is not obvious whether the results in this section all apply to these two variations of the problem, it is obvious from the proof of Theorem 11 that the PSPACE-hardness result applies.

4.1 State Space and Solution Concept

In the original MAPF problem, the state space for the planning process is simply the space of all configurations α of the agents in the graph. For the MAPF problem with destination uncertainty we also have to take into account the possible belief states of all the agents. For this reason, we have to make the possible destination function part of the state space as well, i.e., an ***objective state*** is now the tuple $s = (\alpha, \beta)$, which captures the *common knowledge* of all agents.

A ***MAPF/DU instance*** $\mathcal{M}_{DU} = \langle A, G, s_0, \alpha_* \rangle$ is given by the set of agents A, the graph $G = (V, E)$, the initial objective state $s_0 = (\alpha_0, \beta_0)$, and the goal configuration α_*. Movement actions change the configuration α, while success announcements change the destination function

4. Without this constraint, we either would allow for inconsistent potential goal configurations or, excluding them, we would introduce a form of disjunction over goal configurations concerning more than one agent. By that, the analysis of the problem would become much more involved, leading to complications with perspective shifting and the reduction to stepping stones as spelled out in Lemma 4.

β. If an agent i makes a success announcement while being in location v, we change the destination function to $\beta[i/\emptyset]$, signaling that the agent has reached its destination and is not allowed to move anymore. The goal state is reached if for all agents i, $\alpha(i) = \alpha_*(i)$ (the destination has been reached) and $\beta(i) = \emptyset$ (success has been announced).

When an agent i is starting to generate a plan, the agent knows, of course, its true destination $\alpha_*(i)$. The subjective view of the world is captured by the tuple $(\alpha, \beta, i, \alpha_*(i))$, which we call ***subjective state of agent*** i. Given a subjective state $(\alpha, \beta, i, \alpha_*(i))$, we call (α, β) the ***corresponding objective state***. Using its subjective state, agent i can plan to make movements that eventually will lead to a goal state. Most probably, it will be necessary to plan for other agents to move out of the way or to move to their destination. So, the planning agent has to put itself into the shoes of another agent j: i must make a *perspective shift* taking j's view. Since i does not know the true destination of j, i must take all possibilities into account and plan for all of them. In other words, i must plan for j using all possible subjective states of j: $s^j_v = (\alpha, \beta, j, v)$ for $v \in \beta(j)$. When planning for each possible destination of j, the planning agent i must pretend not to know the true destination of itself because it plans with the knowledge of agent j, which is uncertain about i's destinations.

All in all, a plan in the context of MAPF with destination uncertainty is no longer a linear sequence, but a *branching plan*. Furthermore, it is not enough to reach the true goal state, but the plan has to be successful for all possible destinations of all the agents (except for the starting agent i, who knows its own destination). Such a ***branching plan*** is formally defined as follows. Let a_k be an ***basic action***. This can be a ***movement action***, as before, i.e., (i, x, y) for agent i moving from x to y. In addition, there are ***success announcement actions*** by agent i, denoted by $(i, \mathcal{S})$. Using basic actions a_k, one can form a (perhaps empty) ***action sequence***:[5]

$$\sigma ::= a_1, \ldots, a_n \text{ (with } n \geq 0)$$

A *branching plan* π is now such a sequence σ, followed perhaps by a ***perspective shift*** δ to another agent j:

$$\pi ::= \sigma \mid \sigma\delta,$$

with

$$\delta ::= [j : (?v_{j_1} : \pi_1, \ldots, ?v_{j_m} : \pi_m)] \mid [j : \pi].$$

If a perspective shift has the first form, then one branches over all possible destinations of j: $\beta(j) = \{v_{j_1}, \ldots, v_{j_m}\}$, i.e., agent i considers *all destination possibilities for agent* j. We call such a perspective shift ***branching point*** of the plan. If a perspective shift has the second form, then the perspective is shifted to agent j, but no branching on the possible destinations of j is performed.

Such branching plans correspond roughly to what has been termed *policy* in the more general context of *implicitly coordinated epistemic planing* (Bolander et al., 2018; Engesser et al., 2017). In order to illustrate the concept of a branching plan, let us consider a simplification of our example from the Introduction (see Figure 5). Let us assume that S moves first to v_4. Now S puts itself

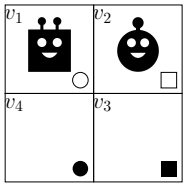


Figure 5: Small example

$$\begin{aligned}(S, v_1, v_4), [C\colon\ & (?v_1 : (C, v_2, v_1), (C, \mathcal{S}), [S : (?v_2 : (S, v_4, v_3), (S, v_3, v_2), (S, \mathcal{S}))\\ & (?v_3 : (S, v_4, v_3), (S, \mathcal{S}))])\\ (?v_4 : (C, v_2, v_1), [S : (?v_2 : & (S, v_4, v_3), (S, v_3, v_2), (S, \mathcal{S}),\\ & [C\colon\ (?v_4 : (C, v_1, v_4), (C, \mathcal{S}))\\ & (?v_1 : (C, \mathcal{S}))])\\ (?v_3 : & (S, v_4, v_3), (S, \mathcal{S}),\\ & [C\colon\ (?v_4 : (C, v_1, v_4), (C, \mathcal{S}))\\ & (?v_1 : (C, \mathcal{S}))])])]\end{aligned}$$

Figure 6: Branching plan for this example

5. We use BNF-like rules to introduce the language of branching plans, using "::=" as the meta-symbol for a syntactic definition and "|" as a meta-symbol for disjunction.

into the shoes of C and reasons about what C would do, if v_1 is C's destination, and how C would continue if v_4 is C's destination. In the former case, C moves to v_1 and announces that it has reached its destination. In the other case, it will also move to v_1, offering S the possibility to move to its destination, whether it is v_2 or v_3. After that, C could move to its destination. All in all, a branching plan could look as depicted in Figure 6. A visualization of the branching plan that is much easier to read is the plan tree as depicted in Figure 7.

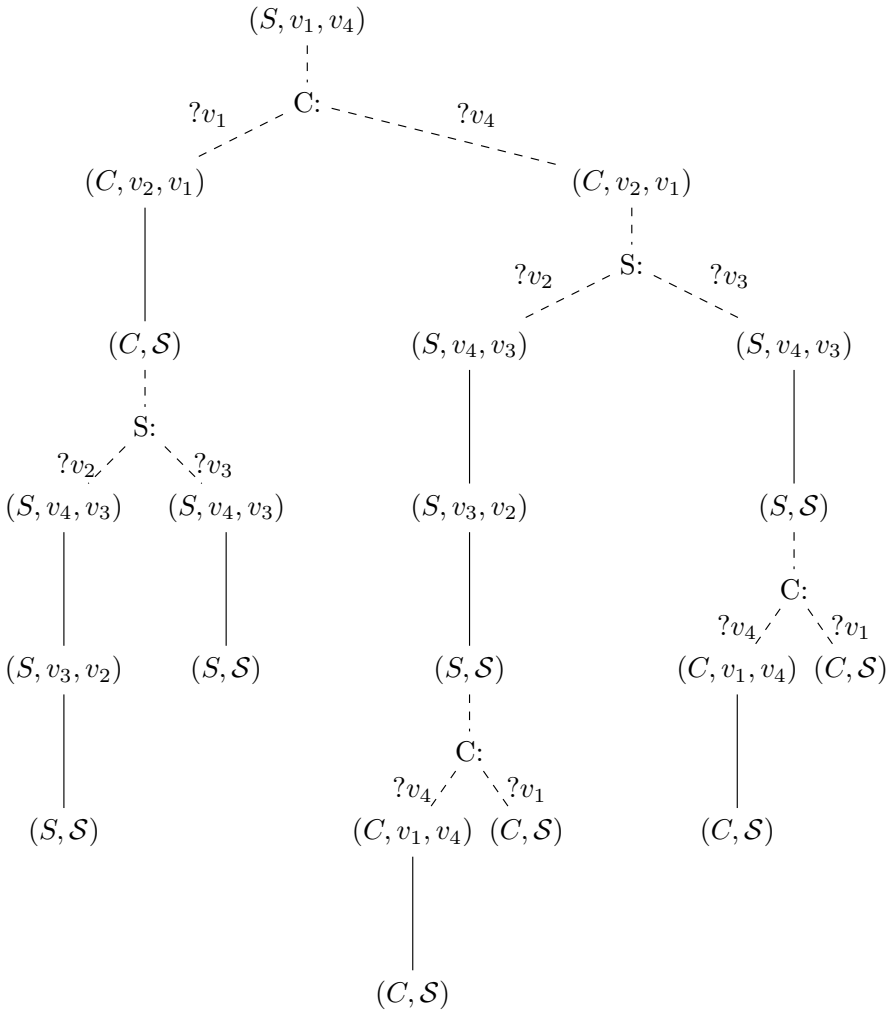


Figure 7: Branching plan depicted as a plan tree

An ***execution trace*** of a branching plan π is a sequence of *basic actions* and ***destination assumptions*** of the form $(i : v_i)$ for agent i with destination v_i. An execution trace of the plan $\pi = \sigma$ is formed by the sequence σ of basic actions. If the plan contains also a perspective shift, i.e., $\pi = \sigma\delta$, and $\delta = [j : (?v_{j_1} : \pi_{j_1}, \ldots, ?v_{j_m} : \pi_{j_m})]$, then one execution trace of π is σ followed by the destination assumption $(j : v_k)$, followed by an execution trace of π_{j_k}, for some $k \in \{1, \ldots, m\}$. If we have an unconditional perspective shift $\delta = [j : \pi]$, then we use the destination assumption $(j : \bot)$, i.e., j does not make any assumption about its destination. Using again our example, one execution trace of the branching plan depicted in Figure 7 formed by following the leftmost branches would

look as follows:

$$(S, v_1, v_4), (C : v_1), (C, v_2, v_1), (C, \mathcal{S}), (S : v_2), (S, v_4, v_3), (S, v_3, v_2), (S, \mathcal{S}). \tag{1}$$

Such an execution trace describes one possible execution of the plan. Movements change the configuration α, success announcements change the common knowledge β, and destination assumptions change the subjective perspective from one agent to another one. In order to be able to say when such an execution is successful, we define the subjective view of executing a trace formally.

The *subjective semantics of an execution trace* is defined by mapping the subjective state of the acting agent i and a given execution trace to a subjective state of an agent j (where $i = j$ is possible). Given a graph $G = (V, E)$, a subjective state $s^i = (\alpha, \beta, i, v)$ and an execution trace χ of a plan π, we define the ***subjective outcome of executing a trace*** χ ***in*** s^i as follows (where we assume that $a; \chi'$ is the trace formed by action a followed by the trace χ'):

$$\text{sexec}(s^i, \chi) \quad = \quad \begin{cases} s^i, & \text{if} \quad \chi = \langle\,\rangle \\ \text{sexec}(\text{sexec}(s^i, a), \chi'), & \text{if} \quad \chi = a; \chi'. \end{cases} \tag{2}$$

In other words, if the execution sequence is empty, then the subjective state is mapped to itself. If it is non-empty, then a new subjective state is computed based on the first action of the trace and then the subjective outcome function is applied to the new subjective state and the remaining part of the trace. The ***subjective outcome of executing a single action*** a ***in*** $s^i = (\alpha, \beta, i, w)$ is then defined as:

$$\text{sexec}((\alpha, \beta, i, w), a) \quad = \quad \begin{cases} (\alpha[i/v'], \beta, i, w) & \text{if} \quad a = (i, v, v'), \\ & \quad \beta(i) \neq \emptyset, \\ & \quad \alpha(i) = v, (v, v') \in E, \\ & \quad \text{and there is no } j : \alpha(j) = v', \\ (\alpha, \beta[i/\emptyset], i, w), & \text{if} \quad a = (i, \mathcal{S}) \text{ and } \alpha(i) = w, \\ (\alpha, \beta, j, v) & \text{if} \quad a = (j : v), \\ \text{undefined} & \quad \text{otherwise.} \end{cases} \tag{3}$$

The first clause in Eq. 3 applies to executing movement action (i, v, v') and requires that the moving agent i is the one mentioned in the subjective state, that the agent should be allowed to move according to $\beta(i)$ and that the move should be legal. The effect is that the configuration α changes. Applied to our example in Figure 5 and assuming that the initial subjective state of agent S is $(\alpha_0, \beta_0, S, v_3)$, then executing the first action (S, v_1, v_4) of our trace (1) leads to the following subjective state:

$$sexec((\alpha_0, \beta_0, S, v_3), (S, v_1, v_4)) = (\alpha_0[S/v_4], \beta_0, S, v_3). \tag{4}$$

The second clause in Eq. 3 describes under which conditions a success announcement can be made and what the result of it is. The third clause describes the effect of executing a destination assumption. Let us again apply it to our example and compute the subjective state that results from applying the second action of our trace (1) to the subjective state $(\alpha_0[S/v_4], \beta_0, S, v_3)$:

$$sexec((\alpha_0[S/v_4], \beta_0, S, v_3), (C : v_1)) = (\alpha_0[S/v_4], \beta_0, C, v_1) \tag{5}$$

An execution trace is called ***successful*** if its outcome is defined and in the resulting state it is common knowledge that all destinations have been reached, i.e., $\beta(i) = \emptyset$ for all agents i. We call a plan i***-successful***, if when started in the subjective state $(\alpha_0, \beta_0, i, \alpha_*(i))$, all its execution traces are successful.

Furthermore, we call a plan i***-covering***, if for each configuration α' with $\alpha'(j) \in \beta(j)$, for all j, and $\alpha'(i) = \alpha_*(i)$, there exists a successful execution trace ending in the configuration α', when execution is started in $(\alpha_0, \beta_0, i, \alpha_*(i))$.

Note that in principle an i-successful branching plan does not need to be i-covering, because there may be some combinations of destinations missing. Neither is an i-covering plan necessarily i-successful, because it could be that some traces lead, e.g., to undefined results.

Finally, we say that an execution trace is ***cycle-free*** if it never visits the same objective state (α, β) twice. This implies that one could revisit a configuration α if the uncertainty had been reduced meanwhile. A plan is said to be ***cycle-free***, if all of its execution traces are cycle-free.

In analogy to the notion of *strongness* in planning under partial observability for one agent (Bertoli, Cimatti, Roveri, & Traverso, 2006), we call a branching plan ***i-strong*** for an objective state (α, β), if it is i-covering, i-successful and cycle-free. If such an i-strong plan exists for a MAPF/DU instance $\mathcal{M}_{DU}$, then $\mathcal{M}_{DU}$ is said to be i-solvable. Note that in our setting, i-success already implies i-covering, because we require that at all branching perspective shifts all possible destinations of an agent are considered.

Proposition 3 *If a plan is i-successful then it is also i-covering.*

Proof: Let $\mathcal{M}_{DU} = \langle A, G, (\alpha_0, \beta_0), \alpha_* \rangle$ be a MAPF/DU instance, π be an i-successful plan, and α' a configuration such that $\alpha'(j) \in \beta(j)$, for all j, and $\alpha'(i) = \alpha_*(i)$. Extract an execution trace χ from π by selecting at each branching point with a perspective shift to agent j the branch corresponding to $\alpha'(j)$. This is possible since we required that for each perspective shift to agent j, the split is either on all possible destinations $\beta_0(j)$ or the shift is unconditional $(j : \bot)$. Since π is i-successful, χ is successful. Because of the semantics of success announcements $(j, \mathcal{S})$, agent j can only announce its destination if it is on the vertex that had been mentioned in the last destination assumption $(j : \alpha'(j))$, which implies that in the resulting state of χ, all agents j have reached their respective destinations $\alpha'(j)$. Since α' was chosen arbitrarily, this holds for all possible α'. ■

The plan shown in Figure 6 (depicted as a tree in Figure 7) is an *S-strong plan*, because it is cycle-free, all its execution traces lead to states in which all destinations are common knowledge, and all possible destinations of C are covered. Some of the execution traces look peculiar, though. If we follow the rightmost branches in Figure 7, we notice that after the move by S, we make the assumption that C's destination is v_4. Further down in the plan tree, we then make the assumption that C's destination is v_1. While this sounds inconsistent, and in fact no possible execution will follow this path, it is nevertheless necessary to consider this case, because when S plans from the perspective of C after having made it success announcement, it cannot know what the right assumption is, and so it has to consider both possibilities (again).

Defining a cost measure for branching plans is not as straight-forward as it is for linear plans. First of all, we only assign costs to basic actions and not to perspective shifts. Second, we are interested in the worst-case costs, i.e., the longest execution. We define the ***execution costs of a branching plan*** to be the number of basic actions (movements and success announcements) of the longest execution trace. As mentioned above, there exist execution traces of a branching plan with inconsistencies concerning the assumptions about the destinations of the agents as in the rightmost path of the plan in Figure 7. We consider these traces as relevant, though, because they reflect the belief about possible executions from a subjective point of view.

4.2 Stepping Stones and a Polynomial Cost-Bound

When solving a MAPF/DU instance, one encounters configurations that are a great step forward to a solution. In a configuration, where an agent i can reach all its possible destinations without the necessity that other agents have to move out of the way, the agent has the freedom to move to its true destination and to announce its success. If one can now guarantee that for each possible destination of i, the remaining problem can be solved, then one has made a significant step ahead. We call such situations *stepping stone* configurations. Formally, an objective state (α, β) is a ***stepping stone*** for i if:

1. For each $v \in \beta(i)$, the node v can be reached by i from the configuration α without the necessity that other agents have to move, resulting in $\alpha[i/v]$;
2. for all $v \in \beta(i)$, there exists an i-strong plan from the objective state $(\alpha[i/v], \beta[i/\emptyset])$.

An example for a stepping stone for agent S appears during the execution of the plan in Figure 7 after having made a perspective shift to C assuming that the destination is v_4 and having executed (C, v_2, v_1). S can now freely move to v_2 or v_3 and in each case the remaining problem is solvable. We call the perspective shift to agent S and the following movements up to the announcements a ***stepping stone utilization***.

Stepping stone utilization is actually the backbone for solving MAPF/DU problems. As a matter of fact, a branching that is not a stepping stone utilization can be simplified. In a non-stepping stone branching point, it is enough to consider one branch that does not end in a success announcement and to prune the other branches away, as spelled out in the next Lemma.

Lemma 4 *Let π be an i-strong branching plan for a MAPF/DU instance $\langle A, G, (\alpha_0, \beta_0), \alpha_* \rangle$ and let $\pi' = \sigma[j : (?v_{j_1} : \sigma_1\delta_1), \ldots, (?v_{j_m} : \sigma_m\delta_m)]$ be a sub-plan of π, such that for some k, $1 \leq k \leq m$, σ_k does not end in a success announcement action. Then the plan π^*, where π' is replaced by $\sigma[j : \sigma_k\delta_k]$, is still an i-strong plan.*

Proof: Assume π and π' as in the proposition. Assume χ is the execution trace leading to π'. Shifting the perspective to j results in the subjective states $(\alpha, \beta, j, v_l), 1 \leq l \leq m$.

If $\beta(j) = \emptyset$, then j has already announced success and cannot move anymore. This means that all σ_l must be empty (and none of them ends in an announcement). Since all execution traces of π are successful, starting at the subjective state of i, all execution traces of δ_l for all $1 \leq l \leq m$ concatenated to χ must be successful. So we can simply choose one and replace π' by $\sigma[j : \sigma_k\delta_k]$, for any k. The resulting plan will be still i-successful.

If $\beta(j) \neq \emptyset$, then j still has to move to its destination. Let σ_k be a sequence not ending in $(j, \mathcal{S})$. This means that σ_k does not affect β. Since all execution traces of $\sigma_k\delta_k$ concatenated to χ are successful, these traces will lead to $\beta(h) = \emptyset$ for all agents h. So it is enough to use the sub-plan $\sigma[j : \sigma_k\delta_k]$ instead of π' inside π, resulting in π^*, which is then still i-successful.

Since by removing parts of the plan, we never can add an execution cycle, the plan will still be cycle-free. Because i-success implies i-covering (Proposition 3), the resulting plan will still be i-strong. ■

As an example application of the Lemma, consider again the plan in Figure 7. The first split on destinations of C is actually not necessary. We could simply use the right branch unconditionally and prune away the left branch. All in all, this means that the only necessary branching points in a plan are stepping stone utilizations.

Theorem 5 (Stepping Stone) *If there exists an i-strong plan with execution costs k, then there exists an i-strong plan with execution costs k or less such that the only branching points are stepping stone utilizations.*

Proof: Let π be an i-strong plan with execution costs k. By Lemma 4, all branching points of π that are not stepping stone utilizations can be removed. Since pruning a branching plan can only decrease execution costs, the resulting plan cannot have higher execution costs than the original plan. ■

One interesting consequence of the *Stepping-Stone Theorem* is that if an instance is solvable, then it is possible to generate a branching plan such that none of its execution traces will contain inconsistent destination assumptions, because each execution trace contains only one destination assumption different from $\perp$ for each agent. Another consequence is an upper bound for the execution costs of branching plans.

Theorem 6 (Polynomial cost bound) *If $\mathcal{M}_{DU} = \langle A, (V, E), (\alpha_0, \beta_0), \alpha_* \rangle$ is i-solvable, then there exists an i-strong branching plan with execution costs bounded by $O(|V|^4)$.*

Proof: By Theorem 5, we only need to consider branching plans that split on destinations when it is a stepping stone utilization. This means that each execution trace proceeds from one stepping stone to the next one. As Kornhauser et al. Kornhauser et al. (1984) have shown, there are at most $O(|V|^3)$ many movements necessary to transform one configuration into another one. So, if there exists any i-strong branching plan, then, since $|A| < |V|$, there will be one that has a worst-case execution trace of length $O(|V|^4)$. ■

5. Joint Execution of MAPF/DU Branching Plans

As in the case with full information, we would like to execute branching plans jointly and guarantee that we reach the goal state after finitely many steps—perhaps using replanning on the way. *Joint execution* should mean here that all agents follow their plans using their own perspective. In particular, when an agent i comes to the point where its plan branches according to its own possible destination, it should follow its private knowledge $\alpha_*(i)$. As in the full information case, we will assume an asynchronous execution regime, where one of the agents that wants to act is chosen and its movement or announcement is executed. Afterwards the other agents follow their original plans, if they are still compatible, or they have to replan.

In order to make this notion of joint execution more precise, let us first introduce the notion of an ***observed action sequence*** ω, which is the finite sequence of basic actions executed by the agents so far. The semantics of such a sequence is given by the ***objective outcome of executing*** ω in objective state $s = (\alpha, \beta)$:

$$\mathrm{oexec}(s, \omega) \quad = \quad \begin{cases} s, & \text{if} \quad \omega = \langle\,\rangle \\ \mathrm{oexec}(\mathrm{oexec}(s, a), \omega'), & \text{if} \quad \omega = a; \omega', \end{cases} \tag{6}$$

where the ***objective outcome of executing*** a ***in*** (α, β) is defined as:

$$\mathrm{oexec}((\alpha, \beta), a) \quad = \quad \begin{cases} (\alpha[i/v'], \beta) & \text{if} \quad a = (i, v, v'), \\ & \quad \beta(i) \neq \emptyset, \\ & \quad \alpha(i) = v, (v, v') \in E, \\ & \quad \text{and there is no } j : \alpha(j) = v', \\ (\alpha, \beta[i/\emptyset]) & \text{if} \quad a = (i, \mathcal{S}), \\ \text{undefined} & \quad \text{otherwise.} \end{cases} \tag{7}$$

Comparing the subjective semantics (Eq. 2–3) with the objective one (Eq. 6–7), one notes that the only difference is the absence of destination assumptions in the objective semantics, both as part of the state and as a possible action.

We say that an observed sequence ω ***matches*** an execution trace χ of a plan π, if the action sequence is a prefix of the execution trace, ignoring all destination assumptions. The remaining part of the trace is called the ***unmatched tail***, denoted by $\chi \backslash \omega$. Further, an ***i-compatible execution trace*** is a trace such that all destination assumptions for i use the actual destination of agent i, i.e., they are of the form $(i : \alpha_*(i))$.

Joint execution of MAPF/DU branching plans now proceeds as follows. All agents i that have formed an i-strong plan π^i from some objective state (α^i, β^i) consider all i-compatible execution traces χ^i_k that match the observed action sequence ω^i, which started in (α^i, β^i). If for agent i, the first action of all unmatched tails $\chi^i \backslash \omega^i_k$ is one of i's basic actions, we say that ***agent i wants to act***. One of these agents that want to act is then chosen and the corresponding action, say a, is executed modifying the current objective state (α, β) to $(\alpha', \beta') = oexec((\alpha, \beta), a)$ and extending the individual observed action sequences to $\omega'^i = \omega^i; a$ for all agents i. All agents that have a plan

that contains i-compatible execution traces matching ω'^i do not need to replan. All others have to replan from (α', β'). This continues until no agent wants to act any more.

Hopefully, an objective state (α_*, β) with $\beta[i] = \emptyset$ for all i has been reached by then. However, while such a goal state might not be reached, because, e.g., all agents want others to act or agents go into infinite execution cycles, it is clear that the outcome of executing an observed action sequence of finite length ω generated by joint execution of i-strong plans is always defined (because ω is always the prefix of an execution trace of some i-strong plan). Moreover, if an objective state is reached such that all agents have announced success, then the agents must have reached the goal configuration α_* (because all agents use only i-compatible execution traces to choose their actions).

Proposition 7 *Let ω be the observed action sequence resulting from joint execution of i-strong plans on the MAPF/DU instance $\langle A, G, (\alpha_0, \beta_0), \alpha_* \rangle$ that cannot be extended any more by the execution process. Then $oexec((\alpha_0, \beta_0), \omega)$ is defined. Further, if $(\alpha, \beta) = oexec((\alpha_0, \beta_0), \omega)$ and $\beta(i) = \emptyset$ for all i, then $\alpha = \alpha_*$.*

5.1 Objective and Subjective Solvability

In contrast to the full information case, it now can happen that agents have a different perspective and therefore judge the solvability differently. Consider the example in Figure 8, where our usual suspects are joined by triangular agent T.

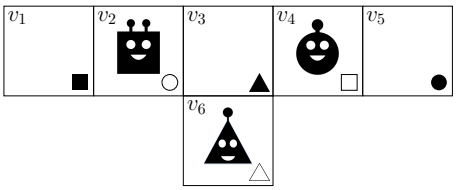


Figure 8: Subjectively but not objectively solvable MAPF/DU instance

From T's perspective, the instance does not appear to be solvable, i.e., there is no T-strong plan. The reason is that from T's perspective, it could be possible that S has to move to the empty square and C has to move to the empty circle, and there is no way that the two agents can both reach these destinations. However, S is able to form an S-strong plan: First go to v_1, then announce success. This results in a stepping stone for C, so that C can reach its destination and announce success, after which T finally can make the last move and announce success as well. From C's perspective, it looks similar. So, this instance is S- and C-solvable, but not T-solvable. We call an instance ***subjectively solvable***, if it is i-solvable for some agent i. In contrast, we also consider ***objective solvability***, which means that the objective state (α_0, β_0) is solvable without having information about the destination of the agent that moves first. A branching plan accomplishing that, i.e., a plan that is i-strong for every agent i, is called ***objectively strong plan***.[6] Clearly, objective solvability implies subjective solvability, but not the other way around.

Note that plans that are i-strong but not objectively strong have a special structure. They consist of an initial movement sequence by agent i, followed by a success announcement, followed by a perspective shift to another agent. The reason for the success announcement immediately before the perspective shift is that without it, the plan would be objectively strong, which we assumed it is not.

One question that may come up in this context is whether it could happen, that a MAPF/DU instance is i-solvable for all agents i, but not objectively solvable. The example in Figure 8 can be easily modified to arrive at such an example. Let us eliminate agent T. Then it is clear that

6. Note that a problem being objectively solvable does not mean that it is only solvable when the destinations are known, that is, it is not planning from an omniscient, centralized perspective. It is still planning under destination uncertainty, it just means that it is planning where *all* destinations are taken to be uncertain.

the instance is S- and C-solvable. However, there does not exist an objectively strong plan. In order to show this, let us consider plans, where agent C starts to move. We could initially split on C's destinations. If it is v_5, then C moves there, announces success, and then S could move to its destination. If instead C's destination is v_2, then it could move to v_6. However, this is not a stepping stone for S, because if C's destination is v_5, then this might not be reachable after S moved to v_4. So, there is no objectively strong plan.

One might argue, however, that after agent C moved to v_6, agent S knows that C's destination cannot be v_5. Otherwise it would have moved there immediately and had announced success, enabling S to complete the task. Based on the conclusion that C's destination cannot be v_5, S can move to its destination (whether it is v_1 or v_4) announcing success, and then wait for C to complete the task. While this sounds rational, it is not a objectively strong plan in the sense we defined it here. The reason is that we took into account information from the history in order to prune possible destinations. This appears to be very similar to what has been called *forward induction* in game theory (Battigalli & Siniscalchi, 2002), however, it is not clear how to incorporate this into our framework in a general way.

5.2 Success Guarantees for Conservative Eager MAPF/DU Agents

As in the full information case, lazy agents will probably be able to provoke a deadlock. So, what is a lazy plan in this context? It is a plan where at some point an action of another agent is planned for, although an action by the original agent would have been possible. So, formally, an i-strong plan is a ***lazy branching plan*** relative to agent i and a set of plans if it contains an execution trace such that the $k+1$th action is by agent $j \neq i$ and there exists another i-strong plan in this set of plans containing an execution trace that is identical up to the kth action, but the $k+1$th action is one by agent i. The branching plan depicted in Figure 7 is lazy with respect to agent S and all S-strong plans since S could have moved to v_3 after its first move, and still one could extend this plan successfully to an S-strong plan. As in the full information case, agents that could generate lazy plans are called ***lazy agents***. Agents that do not generate lazy plans are called again ***eager agents***.

That eager agents avoid deadlocks may be less obvious than in the full information case. However, assume that we have a deadlock for an objectively solvable instance. Then there must be an agent i which prescribes the execution of an action of agent j, which in turn has a plan that prescribes that another agent should act. However, the plan by i, which must be an objectively strong plan, could obviously be executed by j. Since j is eager, it should have adapted this plan instead of one, where it does not act. In case of instances that are only subjectively solvable, one agent i must have formed an i-strong plan, and so no deadlock is possible.

However, we run into the same problem as in the full information case, i.e., joint executions with replanning might lead to non-terminating executions. In the full information case, one way to guarantee termination of executions was to insist on conservatism. In the new setting, we require that for an agent i to be ***conservative*** it needs to replan from the initial configuration (α_0, β_0) generating a ***conservative i-compatible branching plan***, i.e., a plan that contains an i-compatible execution trace that matches the observed action sequence.

This requirement, however, is not enough. In order to account for instances that are only subjectively solvable as in Figure 8, which we want to be able to deal with, we need to redefine the notion of conservatism somewhat. In order to be able to generate a conservative plan for T in Figure 8 after S has announced success, we need to modify the initial belief state. Changing the initial belief state in a form so that the true destination of S is known in the initial state, would be enough.

So, for ***conservative replanning***, we require that after each success announcement of an agent i, the initial objective state (α_0, β_0) will be modified to an objective state such that all agents know

the destination of agent i who just announced success:

$$(\alpha_0', \beta_0') = \Big(\alpha_0, \beta_0\big[i/\{\alpha_*(i)\}\big]\Big). \tag{8}$$

This permits all agents to form a plan that contains as an initial part the already executed actions and will perhaps prune away some branches of the original plan, potentially reducing the execution costs. Agents using this replanning scheme and are eager with respect to conservative plans are called again ***conservative eager agents***.

Interestingly, this will lead to the following behavior when replanning conservatively on instances that are only subjectively solvable. If an instance is not objectively but only subjectively solvable, this means that all of the agents that have an i-strong plan will have planned to move to their known destination $\alpha_*(i)$ and announce success, as mentioned in Section 5.1. Now, if one agent i is chosen to make the first move, no other agent will be able to replan conservatively, i.e., the other agents "loose" their solutions. The reason is that after the first move of i, all of the other agents have to replan starting from the initial state for i to move first, implying a perspective shift to i initially. Since the instance is not objectively solvable, the other agents cannot find such a plan and the only agent with a plan will be agent i which moved first. Only, after i reaches its destination and announces success, then the remaining configuration is objectively solvable, because i had to plan for all possible destinations of the other agents. And now all other agents can come up with a conservative plan using the modified initial objective state.

This behavior can be seen in the example of Figure 8. After S moves to v_1, C needs to replan. Replanning conservatively, it tries to find a plan with S making the first move, but will not find it. Only after the success announcement by S on v_1, both C and T are able to form a plan starting at the modified initial state.

Since the state space is finite, and all acting agents know how the plan can be completed, conservative replanning is always successful. However, since distributed MAPF as characterized in Section 3 is a special case of MAPF/DU, executions can, of course, grow to a length that is exponentially larger than the execution costs of the shortest plan.

Proposition 8 *For MAPF/DU instances, joint execution and replanning of movement plans generated by conservative eager agents is always successful, provided the instance is solvable. However, executions can be exponentially longer than the execution costs of an i-strong plan.*

5.3 Optimally Eager MAPF/DU Agents

In the full information case, focusing on minimal-length plans saved us. The hope could be that it is possible to generalize this to branching plans with minimal execution costs. Agents that are eager with respect to plans with minimal execution costs are called ***optimally eager agents***, as in the full information case.

In the more general case of epistemic planning it has been shown that optimally eager agents can still end up in infinite executions (Bolander et al., 2018). Whether it leads to infinite executions in our more specialized setting case is not immediately clear. We will now show that indeed this can happen.

Consider the MAPF/DU instance in Figure 9 with two agents named 1 and 2 initially located at v_1 and v_6, respectively. The non-actual possible destinations are marked by agent identifiers in parenthesis after the colon, while the actual destinations are shown without parenthesis. Edges labelled by numbers are shorthand for simple paths of that length. For each agent i and configuration α, we define $dist(i, \alpha)$ as the length of the shortest unblocked path from $\alpha(i)$ to the actual destination of agent i. Some relevant values of $dist(i, \alpha)$ are provided in Table 1. Furthermore, for each agent i and all vertices v_j we define $maxdist(i, v_j)$ to be the maximal distance from v_j to any possible destination of agent i. All values of $maxdist(i, v_j)$ are provided in Table 2. We also define $dist^+(i, \alpha) = dist(i, \alpha) + 1$ and $maxdist^+(i, v_j) = maxdist(i, v_j) + 1$.

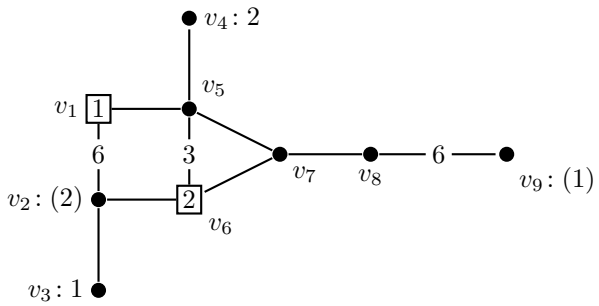


Figure 9: Counter example: Initial state

i	$\alpha(1)$	$\alpha(2)$	$dist(i, \alpha)$
2	v_1	v_6	3
1	v_1	v_7	6
1	v_5	v_6	8

Table 1: Some relevant values of $dist(i, \alpha)$.

v_j	$maxdist(1, v_j)$	$maxdist(2, v_j)$
v_1	9	4
v_2	9	4
v_3	10	5
v_4	9	4
v_5	8	3
v_6	8	3
v_7	7	2
v_8	6	3
v_9	10	9

Table 2: All values of $maxdist(i, v_j)$.

In any configuration, agent i can consider three different types of strategies:

MoveToGoal Agent i moves directly to its actual destination, announces success and then agent $3 - i$ completes the task.

MoveToTemp Agent i moves to some temporary position after which agent $3 - i$ executes its first action. After this, there can be any interleaving of actions between the two agents until the goal is reached.

Wait Agent i waits for agent $3 - i$ to execute the first action. After this first action, there can be any interleaving of actions between the two agents until the goal is reached.

Now note that when agent i considers any optimally eager strategy from a configuration α, the worst-case total number of actions of agent $3 - i$ must be at least $maxdist^+(3 - i, \alpha(3 - i))$: agent i doesn't know the true destination of agent $3 - i$, so in the worst case agent $3 - i$ has to move to its furthest possible destination followed by an announcement of success. When i in particular considers the MoveToGoal strategy, the worst-case number of actions of agent $3 - i$ is *exactly* the value $maxdist^+(3 - i, \alpha(3 - i))$: Agent i first moves to its actual destination and announces success, and thereafter agent $3 - i$ moves to its furthest possible destination and announces success (this rests on the property that no shortest path to any free vertex is blocked when agent i is in its actual destination). So the execution costs of the full MoveToGoal strategy for agent i in a configuration α must be $dist^+(i, \alpha) + maxdist^+(3 - i, \alpha(3 - i))$. Table 3 provides some relevant values of the execution costs of the MoveToGoal strategy, derived from Tables 1 and 2.

When agent i considers the MoveToTemp and Wait strategies, the worst-case total number of actions of i itself must be at least $maxdist(i, \alpha(i))$: Agent i moves to its destination after at least

i	$\alpha(1)$	$\alpha(2)$	$dist^+(i,\alpha) + maxdist^+(3-i,\alpha(3-i))$
2	v_1	v_6	14
1	v_1	v_7	10
1	v_5	v_6	13

Table 3: Some execution costs of the MoveToGoal strategy

$\alpha(1)$	$\alpha(2)$	$maxdist^+(1,\alpha(1)) + maxdist^+(2,\alpha(2))$
v_1	v_6	14
v_1	v_7	13
v_5	v_6	13
v_5	v_7	12
v_5	v_8	13

Table 4: Some lower bounds on the execution costs of the MoveToTemp and Wait strategies.

one action of agent $3-i$, and the number of actions of i will hence be assessed from the perspective of agent $3-i$. So optimally eager MoveToTemp or Wait strategies from a configuration α will have a execution costs of at least $maxdist^+(1,\alpha(1)) + maxdist^+(2,\alpha(2))$. In Table 4 we provide some relevant values of this lower bound on the MoveToTemp and Wait strategies, derived from Table 2.

We now demonstrate that for optimally eager agents it is possible to create an execution sequence that leads to an execution cycle in the objective state space. We start by considering the initial configuration α from Figure 9.

$\alpha(1) = v_1, \alpha(2) = v_6$: We consider the possible strategies of agent 2 in this configuration. From Table 4 we can conclude that the MoveToTemp and Wait strategies have at least execution costs 14. The execution costs of the MoveToGoal strategy are also 14, according to Table 3. Hence an optimally eager agent 2 will choose the MoveToGoal strategy, and in the execution where it gets to move first, it will move to the first vertex on the shortest unblocked path to its destination, which is v_7. This leads to the following configuration.

$\alpha(1) = v_1, \alpha(2) = v_7$: We consider the possible strategies of agent 1 in this configuration. From Table 4 we can conclude that the MoveToTemp and Wait strategies have at least execution costs of 13. The execution costs of the MoveToGoal strategy are 10, according to Table 3. Hence an optimally eager agent 1 will choose the MoveToGoal strategy, and in the execution where it gets to move first, it will move to the first vertex on the shortest unblocked path to its destination, which is v_5. This leads to the following configuration.

$\alpha(1) = v_5, \alpha(2) = v_7$: We consider the possible strategies of agent 2 in this configuration. This case is more involved. Agent 2 cannot follow a MoveToGoal strategy, as all paths to its destination are blocked. So agent 2 has to follow either a Wait or MoveToTemp strategy. We first consider Wait. By the Wait strategy, agent 1 first executes a number of action k for some $k \geq 1$. Since agent 2 is blocking access to one of the possible destinations of agent 1, these k actions doesn't bring agent 1 to its destination in the worst case. Afterwards, agent 2 executes l actions for some $l \geq 1$. These l actions are seen from the perspective of agent 1, since agent 2 acts after agent 1. In the worst case, these l actions can then not end with agent 2 announcing success, since one of the possible destinations of agent 2 is blocking the destination of agent 1. Hence, after these first $l + k$ actions leading to a configuration α', there are still execution costs of $maxdist^+(1,\alpha'(1)) + maxdist^+(2,\alpha'(2))$. Note that the first k moves of agent 1 cannot have led it to any vertex on the path from v_7 to v_9 since this path was blocked by agent 2 when agent 1 had moved. Hence

$maxdist^+(1, \alpha'(1)) \geq 9$, according to Table 2. From the same table we get that $maxdist^+(2, \alpha'(2)) \geq 3$. Hence the total execution costs of the Wait strategy are at least $l+k+9+3 \geq 1+1+9+3 = 14$.

Consider now the possible MoveTemp strategies of agent 2. A concrete MoveTemp strategy for agent 2 is to move to v_6, followed by 1 moving to its destination and announcing success, followed by 2 completing the task. The execution costs of this strategy are $1+maxdist^+(1, v_5)+maxdist^+(2, v_6) = 14$, using Table 4. We will now show that this strategy has minimal execution costs among the MoveTemp strategies. Consider any MoveTemp strategy where agent 2 initially makes k moves. Now note that since the shortest path between v_2 and v_4 (the two possible destinations of agent 2) is 4, then $maxdist(2, v_j) \geq 2$ for all nodes v_j. Then depending on the value of k, we can give the following lower bounds on the execution costs:

- $k = 1$: We already considered the case of agent 2 moving to v_6. The only other option is moving to v_8. This gives a lower bound on the execution costs of $k + maxdist^+(1, v_5) + maxdist^+(2, v_8) = 1 + 13 = 14$, using Table 4.
- $k \geq 2$: A lower bound on the execution costs is $k + maxdist^+(1, v_5) + min\{maxdist^+(2, v_j) \mid v_j \text{ is any node}\} \geq 2 + 9 + 3 = 14$, consulting Table 2.

This proves that moving to v_6 is optimal among the MoveToTemp strategies of agent 2. Since the Wait strategy has the same costs, moving to v_6 is the first step on an optimally eager plan for agent 2. An execution where this move is made by agent 2 leads to the following configuration.

$\alpha(1) = v_5, \alpha(2) = v_6$: We consider the possible strategies of agent 1 in this configuration. From Table 4 we can conclude that the MoveToTemp and Wait strategies have at least execution costs of 13. The execution costs of the MoveToGoal strategy is also 13, according to Table 3. Hence an optimally eager agent 1 will choose the MoveToGoal strategy, and in the execution where it gets to move first, it will move to the first vertex on the shortest unblocked path to its destination, which is v_1. This finally brings the execution back to its initial state.

We have now shown how it is possible to create an execution cycle in the objective state space. So, in contrast to our positive result for distributed MAPF in Proposition 1, we now have a negative result.

Proposition 9 *For MAPF/DU instances, joint execution and replanning generated by optimally eager agents is not always successful, even when the instance is objectively solvable.*

5.4 Conservative, Optimally Eager MAPF/DU Agents

The above example highlights two problems. First, since an acting agent knows its actual destination in the beginning, it considers the execution costs differently from all the other agents. Second, by replanning from a configuration another agent has created, it can happen that the execution costs increase, leading to the cycle shown above.

One way to address this problem is conservatism. As was noted above in Proposition 8, *conservatism* alone is already enough for eager agents to guarantee success. However, we want, of course, to get rid of the exponential execution length! And so we consider ***conservative, optimally eager agents*** which plan conservatively as described above and among the conservative i-compatible plans only consider those with the lowest execution costs. While one would expect that in this case the execution length is bounded polynomially, this is not immediately obvious. In particular, one might fear that with each replanning step, the execution length could increase. However, it is possible to show that this cannot happen. The main argument is that after the initial movements of one agent i, all still active agent have to create objectively strong plans, because of the initial perspective shift to i. This forces them to align their perspectives and so they might generate different plans, but all plans will have the same execution costs.

Theorem 10 *For solvable MAPF/DU instances, joint execution and replanning by conservative, optimally eager agents is always successful and the execution length is polynomial.*

Proof: Since the instance is solvable, there is at least one agent which has formed an i-strong plan. Moreover, since all agents are eager, there must be one that wants to execute an action. Assume that agent i is chosen to execute its action. After the action, the other agents will have to replan coming up either with no plan (e.g., if the instance was only locally solvable) or with an objectively strong plan with an initial perspective shift to agent i. These objectively strong plans will all have the same execution costs because all agents $k \neq i$ plan optimally and have the same knowledge when planning since they are now planning with an initial perspective shift to agent i, which makes them "forget" their own destinations. Agent i may be chosen to execute further actions (provided the initial action was not a success announcement), leading to further replanning for the remaining agents. However, assuming n to be the number of vertices in the graph, no more than $n-1$ movement actions in a row by agent i are possible, because otherwise the plan by i would not be cycle-free. So, after at most $n-1$ movement actions by agent i, (1) either another agent j will act or (2) i will announce success.

In case (1), replanning will lead to objectively strong plans with making a perspective shift to i for all agents $k \neq i$ as described in the previous paragraph. Agent i will form a plan with an initial sequence of its own actions (knowing his own destination), then shifting the perspective to agent j. This plan must also be an objectively strong plan, because after its first perspective shift to j, it must cover all possible destinations of i. For this reason, all agents, planning optimally and using the same prefix, will come up with plans having the same execution costs m.

In case (2), the initial objective state is modified according to Equation 8. After that, because i had an i-strong plan, all remaining agents $k \neq i$ agents are guaranteed to find an objectively strong plan (with an initial perspective shift to i). Again, all these plans will have the same execution costs m, because the agents have the same knowledge and plan optimally.

In future replanning steps, the execution costs of the plans can never increase because all agents follow these plans, so in replanning it must always be possible to find a plan with execution costs of at most m.

In summary, this means that no more than $n+m$ steps will be executed, where m has an upper bound of $O(n^4)$ by Theorem 6. ■

Let us reconsider the example from Section 5.3. We showed that optimally eager agents could end in an infinite cyclic execution. By the theorem above, this cannot happen when the agents are conservative, optimally eager. Let us illustrate that on this example starting from the initial configuration in Figure 9.

$\alpha(1) = v_1, \alpha(2) = v_6$: As before, agent 2 will choose the strategy MoveToGoal, and when acting first it will move to v_7, leading to the next configuration, which is the same as in the counter example.

$\alpha(1) = v_1, \alpha(2) = v_7$: Now agent 1 might replan and has to take into account the prefix of agent 2 moving to v_7 (thereby forgetting its own actual destination).

As we know from Theorem 5, we only need to consider plans where all branching points are stepping stones. Since there are no stepping stones for agent 2 initially (because one possible destination of agent 1 is blocked by one possible destination of agent 2), any objectively strong plan will consists of movements by the two agents creating a stepping for agent 1 followed by agent 1 moving to its destination and then agent 2 moving to its destination. Interestingly, the initial movement of agent 2 destroyed the stepping stone for agent 1. However, agent 1 now has to consider to move first or to let agent 2 continue in order to create a stepping stone (followed by the rest).

Let us consider the case that agent 1 *waits* for agent 2 to create a stepping stone. Note that moving back to v_6 is not an option for agent 2, because it would create a cycle. Agent 2 could move to v_5 and then to either v_4 or the first anonymous node below v_5 (which we will call v_5') (all in all 2 actions). After that, agent 1 could move to its destination and announce success

($maxdist^+(1, v_1) = 10$ actions). If agent 2 needs to move from v_4, we have another 5 actions (including the success announcement). From v'_5 the maximum number of actions is 4. So the maximum execution costs would be 16, assuming agent 1 initially waits.

Let us now consider the case that agent 1 *moves*. It could move to the first anonymous node below v_1 (which we call v'_1) or to v_5 (1 action). Since the former move will increase the maximal distance for agent 1 from 9 to 10, we will not consider it. Now agent 2 needs to act and go to v_6 and then to the first anonymous node above v_6, which we call $\hat{v_6}$ (2 actions), creating the stepping stone for agent 1, who needs in the worst case $maxdist^+(1, v_5) = 9$ actions to reach its destination and announce success. Then agent 2 needs another 3 moves in the worst case and the success announcement (4 actions). All in all, 16 actions. Being eager, agent 1 clearly chooses to move to v_5. Since this is the worst case for the scenario, let us assume that this happens. Again, we reach the same configuration as in the counter example.

$\alpha(1) = v_5, \alpha(2) = v_7$: In so far, we have replicated the movements from our counter example. However, from now on, since both agents have acted now, both agents will only generate objectively strong plans with the same execution costs and both will avoid cycles. So, agent 2 will plan to go back to v_6 (as in our counter example), but agent 1 will never even consider to move back to v_1. Rather, it will wait for agent 2 to create a stepping stone by moving to $\hat{v_6}$. From then on, there will not be any conflict any longer.

6. Computational Complexity of MAPF/DU Planning

It looks as if MAPF/DU planning is harder than MAPF planning. In fact, bounded plan existence for strong plans is PSPACE-complete.

Theorem 11 *Deciding whether there exists a MAPF/DU i-strong or objectively strong plan with execution costs k or less is PSPACE-complete.*

Proof: Membership in NPSPACE follows from Theorem 6. If there exists a plan with execution costs k, then we can guess all its traces and verify that they are successful in polynomial space. Since NPSPACE=PSPACE, the problem is in PSPACE.

We prove hardness by a reduction from QUANTIFIED 3SAT, which is known to be PSPACE-complete (Stockmeyer & Meyer, 1973). Given the quantified Boolean formula

$$\Psi = \forall x_1 \exists x_2 \ldots \phi(x_1, x_2, \ldots, x_\ell),$$

with ℓ variables, ℓ_e existentially quantified, ℓ_u universally quantified, where $\phi(x_1, x_2, \ldots, x_\ell)$ is in conjunctive normal form and contains m clauses C_j with exactly three literals, we construct a MAPF/DU instance with the property that there exists a (globally or x_2-) strong plan with execution costs

$$k = [\ell \cdot (\ell+1)/2] + [(m + m \cdot \ell) \cdot \ell] + [m \cdot (2 + \ell_e)] + [m \cdot (m+1)/2] + [\ell_e \cdot (\ell_e+1)/2] + [\ell_e + \ell + m + m \cdot \ell] \quad (9)$$

if and only if Ψ is true.

We proceed by constructing three gadgets, which we call *choice sequencer*, *clause evaluator*, and *collector*, respectively. We illustrate the construction using the example in Figure 10.

The task of the *choice sequencer* is first of all to enforce the sequence of truth-value choices of the variables corresponding to the quantifier sequence. Each of the variable agents x_i have to go to one of the nodes $v^T_{i,2}$ or $v^F_{i,2}$. Note that for the universally quantified variables, there are two possible destinations and a branching plan needs to consider both possibilities. For the existentially quantified variable, there is just one destination, which is occupied by an *shadow agent*. Here we only have to plan for one of the two possibilities. Once the choices have been made, the clause agents c_j can move to the left until they are ready to enter the clause evaluator. The *clause evaluator* is

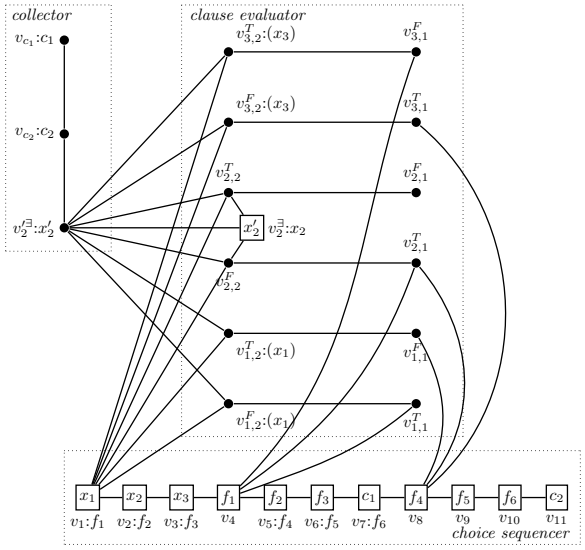


Figure 10: Example construction for $\forall x_1 \exists x_2 \forall x_3 : (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

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 nodes for all clause agents and for the *shadow agents*. The overall effect of the construction is that for all satisfying truth assignment, the number of steps in one execution trace is bounded by k. For unsatisfying assignments, there might still be successful execution traces, but they are necessarily longer.

The *choice sequencer* consists of a sub-graph with $\ell + m(\ell + 1)$ nodes, which are named v_1 to $v_{\ell+m(\ell+1)}$. These nodes are connected linearly, i.e., there is an edge between v_{i+1} and v_i. The nodes v_1 to v_ℓ are occupied by *variable agents* named x_1 to x_ℓ. In addition we have *clause agents* $c_j, 1 \leq j, \leq m$ on the nodes $v_{\ell+j(\ell+1)}$, respectively. The rest of the nodes are filled with *filler agents* f_p for all the not yet occupied nodes. The (deterministic) destination for each filler agent f_p is the node with an index ℓ 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 two pairs of nodes: $v^F_{i,1}, v^T_{i,2}$, and $v^T_{i,1}, v^F_{i,2}$ with an edge between each pair. For a universally quantified variable x_i, the nodes $v^F_{i,2}$ and $v^T_{i,2}$ are the two potential destinations of variable agent x_i. Note that we do not care what the actual destination is because we are interested in x_2-strong plans or objectively strong plans (implying that the agent moving first does not know any of the true destinations). These destination nodes represent the truth assignment false and true, respectively. For each existentially quantified variable x_i, there exists an additional node $v^\exists_i$, which is connected to both $v^F_{i,2}$ and $v^T_{i,2}$ and which is the deterministic destination for agent x_i. Initially, another *shadow agent* x'_i occupies $v^\exists_i$.

The node v_1 has edges to the nodes $v^F_{i,2}$ and $v^T_{i,2}$ of the choice sequencer. Since the shadow agents x'_i do not have any place they can immediately go to without blocking the way for the clause agents to their final destination, this implies that in the beginning the variable agents x_1 to x_ℓ will all move to $v^F_{i,2}$ or $v^T_{i,2}$, whereby for the universally quantified variables, both choices have to be considered in evaluating whether the plan is a strong plan. Furthermore, the choices have to be made in the same order as in the quantifier prefix of the formula. Once all the x_i agents have reached their nodes, the remaining agents in the choice sequencer can move ℓ nodes to the left, i.e., from v_p to $v_{p-\ell}$ bringing all the filler agents f_p to their respective destinations. Further, all clause agents c_j have to go from

$v_{\ell+j(\ell+1)}$ to $v_{j(\ell+1)}$, whereby these latter nodes are connected to the clause evaluator in the following way. The node $v_{j(\ell+1)}$, which will hold clause agent c_j after all agents moved ℓ steps to the left, is connected to $v^T_{i,1}$ iff the clause C_j contains x_i positively and it is connected to $v^F_{i,1}$ iff C_j contains x_i negated.

Finally, the collector gadget provides the destinations for all the clause agents c_j and the shadow agents. Assuming we have the the set of existentially quantified variables $\{x_{j_1}, \ldots, x_{j_{\ell_e}}\}$, we create the following sequence of nodes $v'^{\exists}_{j_{\ell_e}}, \ldots, v'^{\exists}_{j_1}, v_{c_m}, \ldots, v_{c_1}$, connected linearly. From all nodes $v^F_{i,2}$, $v^T_{i,2}$ and $v^{\exists}_i$ there is an edge to $v'^{\exists}_{j_{\ell_e}}$. If there is no existential variable, then the nodes are connected directly with v_{c_m}.

Now, any successful execution trace must at least contain the following actions:

- all of the ℓ variable agents x_i have to move to their truth value choice node $v^X_{i,2}$, which takes i moves for agent x_i: $\sum_{i=1}^{\ell} i = \underline{\ell \cdot (\ell+1)/2}$ moves;
- all of the m clause agents x_j and $m \cdot \ell$ filler agents f_l have to move ℓ nodes to the left in the choice sequencer gadget: $\underline{(m + m \cdot \ell) \cdot \ell}$ moves;
- all of the m clause agents have to move through the clause evaluator gadget to the first node in the collector gadget (3 moves) and then to the node just before the first destination for the clause agents ($\ell_e - 1$ moves): $\underline{m \cdot (2 + \ell_e)}$ moves;
- next, all of the clause agents have to move to their destination: $\sum_{j=1}^{m} j = \underline{m \cdot (m+1)/2}$ moves;
- all of the n_e shadow agents x'_i have to move to their destinations: $\sum_{i=1}^{\ell_e} = \underline{\ell_e \cdot (\ell_e + 1)/2}$ moves;
- and, all of the agents have to announce their success once they have reached their destination: $\underline{\ell_e + \ell + m + m \cdot \ell}$ announcements.

This adds up to k actions as specified in Equation (9). This is clearly a lower bound. If the path of one of the clause agents is blocked by all variable agents, then one needs more actions.

This construction is obviously polynomial in the size of the QBF formula. We now have to show that it is indeed a reduction.

Assume that the QBF formula is true. Then we can generate an objectively strong plan with execution costs of k as follows. The universally quantified variable agents move to their respective destinations branching on their destinations. These are stepping stone utilizations, provided that the formula is true. The existentially quantified variables choose one of $v^F_{i,2}$ and $v^T_{i,2}$ as the temporary location. After all variable agents are in place, the clause agents can move one after the other to their destinations in the collector gadgets. Since the formula is true, we know that for each choice of the universal variable agents and for appropriate choices of the existential variable agents, each clause contains one true literal corresponding to an unblocked path through the clause evaluator to the destination. After the clause agents have reached their destination, the shadow agents x'_i have to go to their respective destinations, which makes room for the existentially quantified variable agents x_i move to their destinations, after which all agents have reached their destinations and have not used more than k actions.

Conversely, assume that there exists an objectively strong plan (or an x_2-strong plan) with execution costs k. Then the movements and branching points are as above and if the clause agents can pass through the clause evaluator to the collector with an overall execution costs of k, it means that the choices made by the variable agents led to a satisfying assignment. Since the plan is strong, this holds for all possible assignments, hence the formula must be true. ■

As in the fully observable case, adding eagerness does not change the picture. Since eagerness is defined on the set of plans, i.e., the set of plans with execution costs less then k, the decision problem does not becomes harder.

This increase in computational complexity when going from distributed MAPF to MAPF/DU probably does not come as a surprise, and it seems to rule out applications as the ones envisioned in the Introduction, namely, implicit coordination in a human-robot context or when agents are not able to communicate. However, when looking at the reduction, one sees that it is very involved and does not seem to be close to situations one encounters in real life. In particular, it is probably very seldom that one encounters $\ell + m \cdot (\ell + 1)$ agents (for moderately large ℓ and m) at the same time. For a fixed number of agents, the problem is fortunately solvable in polynomial time.

Proposition 12 *For a fixed number c of agents, deciding whether there exists a MAPF/DU i-strong or objectively strong plan with execution costs of k or less can be computed in time $O(n^{c^2+c})$, where n is the number of nodes in the graph.*

Proof: Given a MAPF/DU instance $\mathcal{M}_{DU} = \langle A, (V, E), (\alpha_0, \beta_0), \alpha_* \rangle$ with $|V| = n$ and a fixed number of agents $c = |A|$, the following algorithm returns the execution costs of an optimal plan in $T(n, c) = O(n^{c^2+c})$ steps:

If $c = 1$, return the length of a shortest path for the agent to its goal position. For example, using Dijkstra's algorithm this can be done in $O(n^2)$. In the case of $c > 1$, proceed as follows:

1. Check for each of the $O(n^c)$ possible placements α_i whether for one agent all possible destinations are reachable without moving the other agents. Memorize the shortest path length for each agent/destination pair. Again, using Dijkstra this can be done in $O(n^c \cdot c \cdot n^2) = O(n^{c+2})$.
2. For each such possible stepping stone, compute a shortest movement plan to reach it from α_0. This can be done by computing shortest paths on the product graph in time $O(n^c \cdot (n^c)^2) = O(n^{3c})$.
3. For each such possible stepping stone and the respective agents that can reach all their destinations from there, apply our algorithm recursively. The subproblems are the ones where the agent has already reached its destination and announced success. The number of subproblems which have to be solved is bounded by $c \cdot n^c$, so the runtime for solving all the subproblems is $O(n^c)T(n, c-1)$.
4. Add the path lengths from steps (2) and (3) to the recursively obtained execution costs. Maximize over the destination candidates for each agent. Memoize these costs for all stepping stone/agent pairs and return the minimum. This can be done in $O(n^c \cdot c \cdot n) = O(n^{c+1})$.

Our algorithm has a runtime of $T(n, c) = O(n^c)T(n, c-1) + O(n^{3c})$, with $T(n, 1) = O(n^2)$. We can expand this to $T(n, c) = O(n^c)^{c-1}O(n^2) + O(n^c)^{c-2}O(n^{3c}) = O(n^{c^2+c})$. ∎

For two agents, this would result in a runtime of $O(n^6)$. However, one would probably expect a significantly lower practical runtime, provided the environment is not too complicated.

7. Summary and Outlook

We have generalized the well-known MAPF problem to a distributed setting with uncertainty about the destinations of the other agents. First we considered the case, where the agents have common knowledge about their destinations, but have to plan in a distributed fashion, cannot communicate, and have to execute their plan in a distributed way. As we have shown, agents are guaranteed to succeed, provided they are *eager* and either *conservative* when replanning or they are *planning optimally*. However, in the first case executions might last exponentially long while in the latter case the agents have to solve a sequence of NP-hard problems. All in all this demonstrates the value

of communicating about plans. When such communication is possible, planning and/or execution time can be significantly shortened.

Going one step further, we dropped the assumption that destinations are common knowledge, resulting in the MAPF/DU problem. In order to solve this problem, one first has to come up with a reasonable solution concept. We propose to use branching plans, which branch on the possible destinations of the executing agent. In particular, these plans anticipate actions of the agents, although these anticipation might turn out to be wrong. Branching plans that for a subjective state of agent i are successful for all branches are called i-strong and capture what has been termed *implicitly coordinated plans* or *policies* (Engesser et al., 2017). One important result of this paper is the identification of *stepping stone configurations* as a backbone for generating i-strong plans. Using this result, we can show that the execution costs of such plans are polynomially bounded.

Similar to the distributed MAPF setting, we investigate under which conditions we can guarantee success. It turns out that in the MAPF/DU setting, agents have to be *eager* and *conservative*. Because of non-uniform knowledge, *optimally eager* agents are not universally successful, as we have demonstrated. Nevertheless, if we add *conservatism*, we can show again that success is guaranteed and that the worst-case execution length is bounded polynomially.

While this success guarantee is encouraging, the computational costs are unfortunately even worse than in the distributed MAPF setting. Deciding the bounded i-strong and objectively strong plan existence problem is PSPACE-complete. This reinforces the conclusion above: Communication can have a significant effect on lowering computational costs. Furthermore, it suggests that this technique is probably not meant to be used in a real-time, online fashion. However, if there are only a few agents present in the environment, things look much more promising. For a fixed number of agents the bounded MAPF/DU plan existence problem is polynomial.

The paper gives a first idea of what issues arise when solving the MAPF/DU problem. However, there are also a number of open problems. First of all, all results were proven for general undirected graphs. Whether the results also hold for planar graphs or for graphs resulting from a grid map is not obvious. Second, although we have proven only PSPACE-completeness for the bounded plan existence problem, it seems likely that also the general plan existence problem is PSPACE-complete. However, it is not obvious how to prove it. Third, one might ask whether the non-overlapping constraint for possible destinations is really necessary. Perhaps, one can even allow for more expressive goal configuration descriptions. Fourth, one may want to relax the solvability constraints. For example, it might be considered desirable to find plans for situations, when no strong plans exist. Fifth, one might argue that the asynchronous execution model is unrealistic. Whether one could come up with a reasonable parallel execution model is not obvious, though. Sixth, we have just started to explore the space of implicit coordination. The only kind of communication currently allowed is the announcement of success. Perhaps, with more communication other success guarantees could be established. In particular, one could imagine that the agents could signal each other their destination by using a particular protocol (similar to a bee dance). However, we prefer to make minimal assumptions about the knowledge the agents share. Seventh, it is conceivable that agents could be more aggressive in making conclusions from the movements of other agents. Something similar to *forward induction* known from game theory (Battigalli & Siniscalchi, 2002) might help in order to coordinate implicitly. Finally, it would be interesting to look into more efficient solutions for the MAPF/DU problem as the one presented in Proposition 12 and to explore how large an environment and agent group can become for the distributed MAPF problem and the MAPF/DU problem.

Acknowledgements

We would like to thank all the people who contributed by making suggestions and pointing out problems. In particular, Andreas Herzig, Michael Thielscher, and the three anonymous reviewers made valuable comments. All remaining errors are ours, though.

This research was partly supported by DFG as part of the PACMAN project within the HYBRIS research group (NE 623/13-1) and partly by DAAD as part of the Australia-Germany Joint Research Cooperation Scheme 2017/18, project 57319997, and as part of the France-Germany Joint Research Cooperation 2017/18, project 57317892.

Appendix A. Terminology Index

References

Ambros-Ingerson, J. A., & Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)* (pp. 83–88). Retrieved from http://www.aaai.org/Library/AAAI/1988/aaai88-015.php

Andersen, M. B., Bolander, T., & Jensen, M. H. (2012). Conditional epistemic planning. In *Logics in Artificial Intelligence - 13th European Conference (JELIA-12)* (pp. 94–106). Retrieved from https://doi.org/10.1007/978-3-642-33353-8_8 doi: 10.1007/978-3-642-33353-8_8

Battigalli, P., & Siniscalchi, M. M. (2002). Strong belief and forward induction reasoning. *J. Economic Theory*, *106*(2), 356–391. Retrieved from https://doi.org/10.1006/jeth.2001.2942 doi: 10.1006/jeth.2001.2942

Bertoli, P., Cimatti, A., Roveri, M., & Traverso, P. (2006). Strong planning under partial observability. *Artif. Intell.*, *170*(4-5), 337–384. Retrieved from https://doi.org/10.1016/j.artint.2006.01.004 doi: 10.1016/j.artint.2006.01.004

Bhattacharya, S., Kumar, V., & Likhachev, M. (2010). Distributed optimization with pairwise constraints and its application to multi-robot path planning. In *Robotics: Science and Systems VI (RSS-10)*. Retrieved from http://www.roboticsproceedings.org/rss06/p23.html

Bolander, T., & Andersen, M. B. (2011). Epistemic planning for single and multi-agent systems. *Journal of Applied Non-Classical Logics*, *21*(1), 9–34. Retrieved from https://doi.org/10.3166/jancl.21.9-34 doi: 10.3166/jancl.21.9-34

Bolander, T., Engesser, T., Mattmüller, R., & Nebel, B. (2018). Better eager than lazy? How agent types impact the successfulness of implicit coordination. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference (KR-18)* (pp. 445–453). Retrieved from https://aaai.org/ocs/index.php/KR/KR18/paper/view/18070

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

Brafman, R. I., & Shani, G. (2012). Replanning in domains with partial information and sensing actions. *J. Artif. Intell. Res.*, *45*, 565–600. Retrieved from https://doi.org/10.1613/jair.3711 doi: 10.1613/jair.3711

Brenner, M., & Nebel, B. (2009). Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, *19*(3), 297–331. Retrieved from https://doi.org/10.1007/s10458-009-9081-1 doi: 10.1007/s10458-009-9081-1

Cáp, M., Vokrínek, J., & Kleiner, A. (2015). Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS-15* (pp. 324–332). Retrieved from http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10504

desJardins, M., Durfee, E. H., Ortiz, C. L., Jr., & Wolverton, M. (1999). A survey of research in distributed, continual planning. *AI Magazine*, *20*(4), 13–22. Retrieved from http://www.aaai.org/ojs/index.php/aimagazine/article/view/1475

de Wilde, B., ter Mors, A., & Witteveen, C. (2014). Push and rotate: a complete multi-agent pathfinding algorithm. *J. Artif. Intell. Res.*, *51*, 443–492. Retrieved from https://doi.org/10.1613/jair.4447 doi: 10.1613/jair.4447

Dresner, K. M., & Stone, P. (2008). A multiagent approach to autonomous intersection management. *J. Artif. Intell. Res.*, *31*, 591–656. Retrieved from https://doi.org/10.1613/jair.2502 doi: 10.1613/jair.2502

Engesser, T., Bolander, T., Mattmüller, R., & Nebel, B. (2017). Cooperative epistemic multi-agent planning for implicit coordination. In *Proceedings of the Ninth Workshop on Methods for Modalities (M4MICLA-17)* (pp. 75–90). Retrieved from https://doi.org/10.4204/EPTCS.243.6 doi: 10.4204/EPTCS.243.6

Felner, A., Stern, R., Shimony, S. E., Boyarski, E., Goldenberg, M., Sharon, G., ... 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)* (pp. 29–37). Retrieved from https://aaai.org/ocs/index.php/SOCS/SOCS17/paper/view/15781

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & OR*, *13*(5), 533–549. Retrieved from https://doi.org/10.1016/0305-0548(86)90048-1 doi: 10.1016/0305-0548(86)90048-1

Gmytrasiewicz, P. J., & Doshi, P. (2005). A framework for sequential planning in multi-agent settings. *J. Artif. Intell. Res.*, *24*, 49–79. Retrieved from https://doi.org/10.1613/jair.1579 doi: 10.1613/jair.1579

Goldman, C. V., & Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis. *J. Artif. Intell. Res.*, *22*, 143–174. Retrieved from https://doi.org/10.1613/jair.1427 doi: 10.1613/jair.1427

Goldreich, O. (2011). Finding the shortest move-sequence in the graph-generalized 15-puzzle is NP-hard. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation* (pp. 1–5). Retrieved from https://doi.org/10.1007/978-3-642-22670-0_1 doi: 10.1007/978-3-642-22670-0_1

Grady, D. K., Bekris, K. E., & Kavraki, L. E. (2010). Asynchronous distributed motion planning with safety guarantees under second-order dynamics. In *Algorithmic Foundations of Robotics IX - Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics (WAFR-10)* (pp. 53–70). Retrieved from https://doi.org/10.1007/978-3-642-17452-0_4 doi: 10.1007/978-3-642-17452-0_4

Gray, F. (1953). *Pulse Code Communication.* Retrieved from https://patents.google.com/patent/US2632058 (U.S. Patent 2,632,058, issued March 17, 1953)

Hatzack, W., & Nebel, B. (2013). Solving the operational traffic control problem. In A. Cesta (Ed.), *Proceedings of the 6th European Conference on Planning (ECP-01).* Toledo, Spain: AAAI Press. Retrieved from https://www.aaai.org/ocs/index.php/ECP/ECP01/paper/view/7284

Jansen, M. R., & Sturtevant, N. R. (2008). A new approach to cooperative pathfinding. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-08)* (pp. 1401–1404). Retrieved from http://doi.acm.org/10.1145/1402821.1402883 doi: 10.1145/1402821.1402883

Kornhauser, D., Miller, G. L., & 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)* (pp. 241–250). Retrieved from https://doi.org/10.1109/SFCS.1984.715921 doi: 10.1109/SFCS.1984.715921

Lawrence, R., & Bulitko, V. (2013). Database-driven real-time heuristic search in video-game pathfinding. *IEEE Trans. Comput. Intellig. and AI in Games*, *5*(3), 227–241. Retrieved from https://doi.org/10.1109/TCIAIG.2012.2230632 doi: 10.1109/TCIAIG.2012.2230632

Löwe, B., Pacuit, E., & Witzel, A. (2011). DEL planning and some tractable cases. In *Logic, Rationality, and Interaction - Third International Workshop (LORI-11)* (pp. 179–192). Retrieved from https://doi.org/10.1007/978-3-642-24130-7_13 doi: 10.1007/978-3-642-24130-7_13

Luna, R., & Bekris, K. E. (2011). Push and swap: Fast cooperative path-finding with completeness guarantees. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)* (pp. 294–300). Retrieved from https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-059 doi: 10.5591/978-1-57735-516-8/IJCAI11-059

Ma, H., Tovey, C. A., Sharon, G., Kumar, T. K. S., & 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-06* (pp. 3166–3173). Retrieved from http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12437

Muise, C. J., Belle, V., Felli, P., McIlraith, S. A., Miller, T., Pearce, A. R., & Sonenberg, L. (2015). Planning over multi-agent epistemic states: A classical planning approach. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)* (pp. 3327–3334). Retrieved from http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9974

Ratner, D., & Warmuth, M. K. (1986). 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-86)* (pp. 168–172). Retrieved from http://www.aaai.org/Library/AAAI/1986/aaai86-027.php

Schiffel, S., & Thielscher, M. (2014). Representing and reasoning about the rules of general games with imperfect information. *J. Artif. Intell. Res.*, *49*, 171–206. Retrieved from https://doi.org/10.1613/jair.4115 doi: 10.1613/jair.4115

Schofield, M. J., & Thielscher, M. (2017). The efficiency of the HyperPlay technique over random sampling. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)* (pp. 282–290). Retrieved from http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14560

Silver, D. (2005). Cooperative pathfinding. In *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-05)* (pp. 117–122). Retrieved from http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications_files/coop-path-AIIDE.pdf

Stockmeyer, L. J., & Meyer, A. R. (1973). Word problems requiring exponential time: Preliminary report. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC-73)* (pp. 1–9). Retrieved from http://doi.acm.org/10.1145/800125.804029 doi: 10.1145/800125.804029

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)*. Retrieved from http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1768

van den Berg, J. P., Lin, M. C., & Manocha, D. (2008). Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation (ICRA-08)* (pp. 1928–1935). Retrieved from https://doi.org/10.1109/ROBOT.2008.4543489 doi: 10.1109/ROBOT.2008.4543489

Wagner, G., & Choset, H. (2017). Path planning for multiple agents under uncertainty. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS-17)* (pp. 577–585). Retrieved from https://aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15756

Wang, K. C., & Botea, A. (2011). MAPP: A scalable multi-agent path planning algorithm with tractability and completeness guarantees. *J. Artif. Intell. Res.*, *42*, 55–90. Retrieved from https://doi.org/10.1613/jair.3370 doi: 10.1613/jair.3370

Wurman, P. R., D'Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, *29*(1), 9–20. Retrieved from http://www.aaai.org/ojs/index.php/aimagazine/article/view/2082

Yu, J., & 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)*. Retrieved from http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6111