Implicitly Coordinated Multi-Agent Path Finding under Destination Uncertainty

Bernhard Nebel Thomas Bolander Thorsten Engesser
Robert Mattmüller

July 6, 2018

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 NP-complete problems or have to tolerate exponentially longer executions. In the setting with destination uncertainty, 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 [31], a street intersection [11], an airport ground-traffic scenario [19], or a video game [22], 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 a (perhaps directed) graph $G = (V, E)$, a set of agents A, an initial configuration assigning agents to distinct vertices, and a final configuration with another assignment of agents to distinct vertices, the question is how one can transform the initial configuration into the final one 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 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:

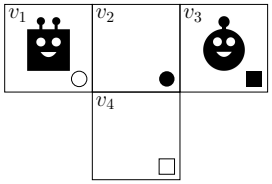


Figure 1: Multi-agent coordination example

1. C moves to v_2 and then to v_4,
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, we consider in Section 2 planning as well as execution as 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*. This result follows from more general results in epistemic planning [5]. 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 3 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 [12]. 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 cost of a branching plan for a MAPF/DU instance is polynomially bounded.

As a third step, in Section 4, 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 5, 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 for a fixed number of agents is polynomial.

In Section 6, we review related work and in Section 7 we summarize our results.

2 Distributed MAPF

As mentioned in the Introduction, the spatial environment is modeled using a graph $G = (V, E)$. Throughout the paper, we consider the most general case of a *directed graph*. 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 ***final 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***.

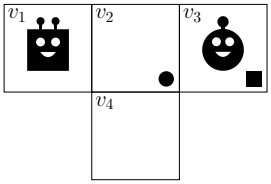
As Kornhauser et al. [21] have shown, solvability on undirected graphs can be decided in polynomial time; and if a plan exist, then its length can be bounded by a polynomial and it can be generated in polynomial time. For the generalization to directed graphs, Botea and Surynek [6] have shown that these bounds are still valid. If we are interested in shortest plans, then the corresponding problem of bounded existence for undirected graphs becomes NP-complete, as has been shown by Ratner and Warmuth [23]. As is easy to see, this also holds for the case of directed graphs.

2.1 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 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. And if an action was not anticipated by other agents, then they 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. 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.

In the example from the Introduction, everything will probably work out. Assume that 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



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

Figure 2: Plans for the multi-agent coordination example

and might come up with π_S with the first action removed, and joint execution will be successful.

2.2 Success Guarantees for Joint Execution

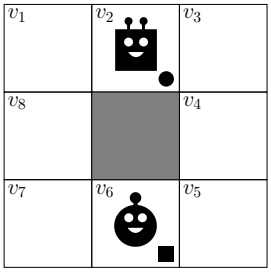
The interesting question is, whether we can find conditions that guarantee success for such joint executions with replanning in the general case. In order to demonstrate one of the issues, let us assume that S comes up with the plan π_C (expecting C to act first), and that C comes up with π_S (expecting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

Let π be a movement plan. We say that π is a ***lazy plan*** *with respect to agent i and a given class 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 as in the scenario described above [5].

We call i an ***eager agent*** if it never generates lazy plans (with respect to itself and the given class 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 class of plans). The reason is that there does not exist *any* prefix such that after the prefix the agent unnecessarily chooses an action of another agent.

With eager agents, we avoid deadlock situations, since there is always at least one agent that can act, as has been shown for the more general setting of epistemic planning [5]. In particular, if both S and C from our initial example are eager, 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 class of all cycle-free plans) have serious problems. The first problem is that eager agents may plan to make unnecessary moves, even when choosing a movement of another agent could lead to a shorter plan. The second problem, resulting from



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

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

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

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

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

Figure 3: Example for infinite execution

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 class of plans Π to ***shortest plans***. We call a plan ***optimally eager*** with respect to agent i if it is not lazy with respect to agent i and the class of shortest plans. Agents producing only such plans are called ***optimally eager agents*** [5]. As is easy to see, such agents are always successful, provided the instance is solvable at all, as spelled

out in the next Proposition.[1] 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.*

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. Actually, this should not come as surprise. If the only form of "communication" we allow is the actions of the other agents, then these actions should in some way reveal the underlying strategy of the agent. Simple eager agents are not very helpful in this respect, because some of their actions might just be unnecessary. Optimally eager agents, however, express with each action execution that their action is a part of an optimal plan. And this can then be used to anticipate the continuation.

While this is good news, the bad news is that this implies that the agents have to solve an NP-complete problem not only once, but potentially after each action execution. And the question may come up whether a computationally simpler version would be possible.

Instead of trying to find the shortest 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.[2] 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 class 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-complete 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 after node names separated by a colon. Bidirectional arcs do not have an arrow.

In this example, each agent a_i wants to move from its initial location $v_{i,1}$ to its destination $v_{i,2}$. An eager plan for agent a_1 could look like as follows. Agent

[1] Note that this a special case of Proposition 9 in the paper by Bolander et al. [5].

[2] This is very similar to tabu search [14] with an unlimited tabu list.

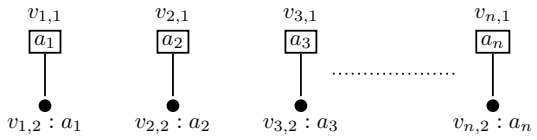


Figure 4: Example, which may result in an exponentially long execution

a_1 moves down to $v_{1,2}$. Since cycles are not allowed, now another agent has to move move, say a_2. After that, a_1 needs to move again (because it is eager), and then a_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. When it comes to the joint execution, however, it could happen that the order of execution is chosen in way that corresponds to the bit change in a *Gray counter* [18]. In this case, the corresponding agents will always move (since they are eager) and they 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.*

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-complete problems or in producing and remembering potentially exponentially long plans. One might hope that there is some middle ground between these two extremes. However, avoiding NP-completeness and exponential executions is probably only possible by using a polynomial approximation algorithm with guaranteed bounds for the MAPF problem. And we are not aware of any such algorithm.

3 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 goal. 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 there is no destinations to be reached, i.e., $\beta(i) = \emptyset$, because the agent already arrived. We require further that all combinations of possible destinations are consistent, i.e., $\beta(i) \cap \beta(j) = \emptyset$ for all $i \neq j \in A$.[3] We,

[3]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

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. In order not to trivialize the problem, 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.

3.1 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 final configuration α_*. Movement actions change the configuration α, while success announcements change the destination function 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.[4] 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: $s_v^j = (\alpha, \beta, j, v)$ 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.

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.

[4] We assume that all such announcements are truthful.

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). Such a ***branching plan*** is formally defined as follows. Let a_k^i be an ***atomic action*** by agent i. 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 atomic actions a_k^i by agent i, one can form a (perhaps empty) ***sequence***:

$$\sigma^i ::= a_1^i, \ldots, a_n^i, n \geq 0.$$

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

$$\pi ::= \sigma^i \mid \sigma^i \delta^j,$$

with

$$\delta^j ::= [j : (?v_1^j : \pi_1^j, \ldots, ?v_m^j : \pi_m^j)] \mid [j : \pi^j].$$

In the former case, one branches according to the possible destinations of j: $\beta(j) = \{v_1^j, \ldots, v_m^j\}$, i.e., agent i considers all destination possibilities for agent j. We call such a perspective shift ***branching point*** of the plan. In the latter case, the perspective is shifted to agent j, but no split on the possible destinations of j is performed.

We call a branching plan ***well-formed*** if for any sequence σ^i, only agent i is acting and if for each perspective shift δ^j, the split is either on all possible destinations of j or the shift is unconditional. Plans of this kind correspond roughly to what has been termed *policy* in the more general context of *implicitly coordinated epistemic planing* [5, 12], and we will only consider such well-formed plans in the following.

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 $\pi = \sigma^i$ is formed by the sequence σ^i of basic actions. If the plan contains also a perspective shift, i.e., $\pi = \sigma^i \delta^j$, and $\delta^j = [j : (?v_1^j : \pi_1^j, \ldots, ?v_m^j : \pi_m^j)]$, then one execution trace of π is σ^i followed by the destination assumption $(j : v_k)$, followed by an execution trace of π_k^j, for some $k \in \{1, \ldots, m\}$. If we have an unconditional perspective shift $\delta^j = [j : \pi^j]$, then we use the destination assumption $(j : \bot)$, i.e., j does not make any assumption about its destination.

The *subjective semantics of an execution trace* is defined by transforming the subjective state of the acting agent i. 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*** χ ***in*** s^i as follows:

$$\text{sexec}(s^i, \chi) \quad = \quad \begin{cases} s^i, & \text{if} \quad \chi = \langle\,\rangle \\ \text{sexec}(\text{sexec}(s^i, a), \chi'), & \text{if} \quad \chi = a; \chi'. \end{cases} \tag{1}$$

The ***subjective outcome of executing a single action*** a is then defined as:

$$\text{sexec}((\alpha,\beta,i,w),a) \quad = \quad \begin{cases} (\alpha[i/v'],\beta,i,w) & \text{if} \quad a=(i,v,v'), \\ & \quad \beta(i)\neq\emptyset, \\ & \quad \alpha(i)=v,(v,v')\in E, \\ & \quad \text{and there is no } j:\alpha(j)=v \quad (2) \\ (\alpha,\beta[i/\emptyset],i,w), & \text{if} \quad a=(i,\mathcal{S}) \text{ and } \alpha(i)=w, \\ (\alpha,\beta,j,v) & \text{if} \quad a=(j:v), \\ \text{undefined} & \quad \text{otherwise.} \end{cases}$$

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.

An execution trace is said to be ***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.

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 $(\alpha_0,\beta_0,i,\alpha_*(i))$.

In analogy to the notion of *strongness* in planning under partial observability for one agent [3], 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 at all branching points that the plan branches over all possible destinations of an agent.

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'. ■

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 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

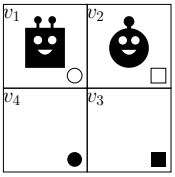


Figure 5: Small MAPF/DU example

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

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

Figure 6: Branching plan for configuration in Figure 5

to read is the plan tree as depicted in Figure 7. The depicted plan is an *S-strong plan*, because 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 edges 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 C cannot know what the right assumption is after S has executed the $(S, \mathcal{S})$ action.

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 atomic actions and not to perspective shifts. Second, we are interested in the worst-case costs, i.e., the longest execution. We define the ***execution cost of a branching plan*** to be the number of atomic actions of the longest execution trace. As mentioned above, there exists 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

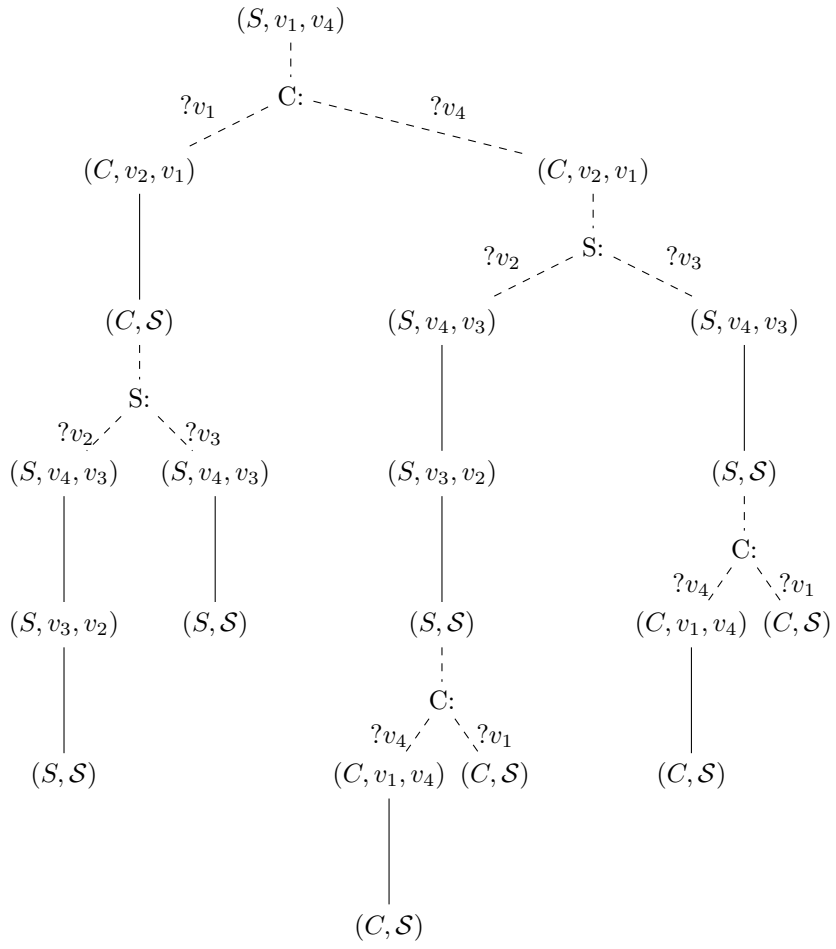


Figure 7: Branching plan depicted as a plan tree

of view.

3.2 Stepping Stones and a Polynomial Cost-Bound

When solving a MAPF/DU instance, one encounters configurations, which 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 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, 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 end in an announcement). Since all execution traces of π are successful starting at the subjective state of i, all executions 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) *Given an i-solvable MAPF/DU instance, there exists an i-strong branching plan such that the only branching points are stepping stone utilizations.*

Proof: Assume that the instance is i-solvable and let π be an i-strong plan. By Lemma 4, all branching points of π that are not stepping stone utilizations can be removed. ■

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 $\bot$ 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 cost 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. [21] 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 $O(|V|^4)$. ■

4 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 which wants to act is chosen and its movement or announcement is executed. Afterwards the other agents follow their original plan, if it is 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 atomic 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 $s = (\alpha, \beta)$:

$$\mathrm{oexec}(s, \omega) \quad = \quad \begin{cases} s, & \text{if} \quad \chi = \langle\,\rangle \\ \mathrm{oexec}(\mathrm{oexec}(s, a), \omega'), & \text{if} \quad \omega = a; \omega', \end{cases} \tag{3}$$

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

$$\text{oexec}((\alpha,\beta),a) \quad = \quad \begin{cases} (\alpha[i/v'],\beta) & \text{if} \quad a=(i,v,v'), \\ & \quad \beta(i) \neq \emptyset, \\ & \quad \alpha(i)=v, (v,v') \in E, \\ & \quad \text{and there is no } j : \alpha(j) = v', \\ (\alpha,\beta[i/\emptyset]) & \text{if} \quad a=(i,\mathcal{S}), \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{4}$$

Comparing the subjective semantics (Eq. 1–2) with the objective one (Eq. 3–4), 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 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 atomic 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 contain 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 final 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 $\alpha = \alpha_*$.*

4.1 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.

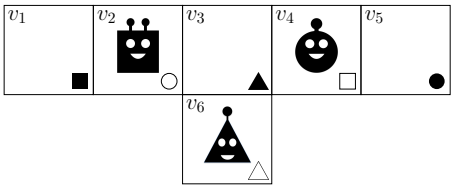


Figure 8: Subjectively but not objectively solvable MAPF/DU instance

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***.[5] 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 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

[5] Note that a problem being objectively solvable does not mean that it is only solvable when the goals 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.

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 on possible destinations. This appears to be very similar to what has been called *forward induction* in game theory [2], however, it is not clear how to incorporate this into our framework in a general way.

4.2 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 plan*** relative to agent i and a class 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 class 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. Agents that do not generate lazy plans are called again ***eager agents***.

Forbidding lazy plans will avoid deadlocks, 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 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:

$$(\alpha'_0, \beta'_0) = \Big(\alpha_0, \beta_0\big[i/\{\alpha_*(i)\}\big]\Big). \tag{5}$$

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 cost.

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 4.1. 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.

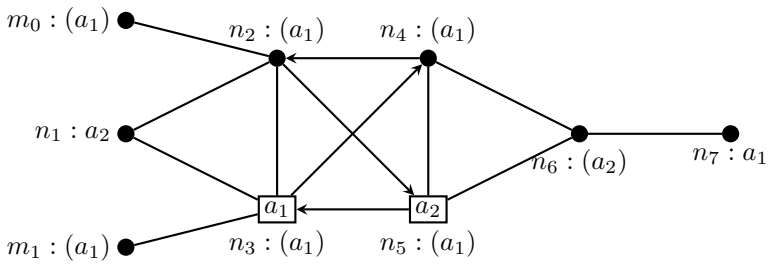
Since the state space is finite, and all acting agents know how the plan can be completed, conservative replanning is always successful. However, since distributed MAPF as characterized in Section 2 is a special case of MAPF/DU, executions can, of course, blow up the execution length.

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.*

4.3 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. In the more general case of epistemic planning it has been shown that optimally eager agent can still end up in infinite executions [5]. Whether it leads to infinite executions in our more specialized setting case is not immediately clear.

Consider the MAPF/DU instance in Figure 9. Here, the possible destinations are marked by agent identifiers in parenthesis after the colon, while the



Agent a_1 has a shortest plan with 7 steps (a_1 starting, moving to n_4)
Agent a_2 has a shortest plan with 8 steps (a_2 starting, moving to n_4)

Figure 9: Counter example: Initial state

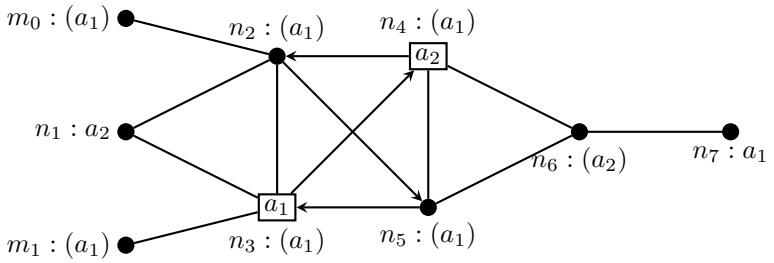
actual destinations are shown without parenthesis. Note that there are four directed edges, which can only be passed in the direction of the arrow. We now demonstrate that for optimally eager agents it is possible to create an execution sequence that leads to an execution cycle in the objective state space.

Note that the only stepping stone situations for agent a_1 are the ones where agent a_2 is on node n_1, which is also the actual destination of a_2. So, agent a_2 will plan to move to n_1 in order to create a stepping stone and to get to its goal. An eager a_2-strong plan would move a_2 to n_1 via n_4 and n_2 and announce success in 3+1 steps. After that, the plan branches on the possible destinations of a_1 adding in the worst case 3 move actions and an announcement action (3+1 steps), which leads to an overall worst-case cost of 8.

The alternative would be for a_2 to wait for a_1 to move out of the way to n_4 (1 step), then a_2 moves to n_1 to create the stepping stone (2 steps). Note that in this case, a_2 cannot announce success, since when the plan starts by a_2 taking the perspective of a_1, the rest of the plan has to be verifiable from the perspective of a_1. So, the plan continues by a_1 moving to its destination and announcing success (worst case $3 + 1$ steps) followed by finalizing the task by a_2 (worst case $3 + 1$ steps). In total, we have 11 steps and in addition the plan is not eager, so a_2 prefers the first one.

Planning from a_1's perspective, it could move to n_7 announcing success (3+1 step), which creates a stepping stone for a_2 with worst case cost of (2+1 steps), totalling 7 steps. An alternative for a_1 would be to wait for a_2 to move first, creating a stepping stone (3 steps). After that a_1 could go to its destination with worst-case cost (3+1), i.e., this plan would be longer than 7 steps.

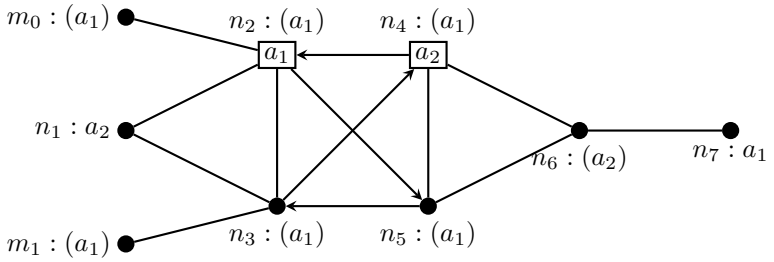
Let us assume that agent a_2 starts to execute the first plan mentioned above, leading to the situation in Figure 10. In this situation, agent a_1 can either wait for agent a_2 to move out of n_4, since agent a_1 knows that agent a_2 is trying to get to n_1, or agent a_1 can take the path $n_3n_2n_5n_6n_7$. The execution cost of the first plan is the cost of creating the stepping stone (2 steps by a_2) plus the



Agent a_1 has a shortest plan with 8 steps (a_1 starting, moving to n_2)
Agent a_2 has a shortest plan with 7 steps (a_2 starting, moving to n_2)

Figure 10: Counter example: After execution of (a_2, n_5, n_4)

worst-case cost for a_1 to move to its destination and announce success $(3 + 1)$ plus the worst-case cost of agent a_2 moving to its destination and announce success $(3 + 1)$, in total 9. The execution cost of the second plan is the cost of a_1 to move to the destination and announce success $(4 + 1)$ plus the worst-case cost of agent a_2 moving to its destination and announce success $(2 + 1)$, in total 8. Hence, an optimally eager agent will choose the second of these plans. Alternatively, a_2 might follow its plan described above, leading to 7 steps (one less than in the initial situation shown in Figure 9). Let us assume, that agent a_1 executes the next step, then it will move to n_2 (see Figure 11).



Agent a_1 has a shortest plan with 7 steps (a_1 starting, moving to n_5)
Agent a_2 has a shortest plan with 8 steps (a_2 starting, moving to n_5)

Figure 11: Counter example: After execution of $(a_2, n_5, n_4), (a_1, n_3, n_2)$

Comparing the new state with the initial state, one notices that we have reached a configuration which is completely symmetric to the initial state. In other words, with two additional steps, we could reach the initial situation and would have created 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.*

4.4 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 cost 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 cost. 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 cost 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 make them"forget" their own goals. 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 cost m.

In case (2), the initial objective state is modified according to Equation 5. 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 cost 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 4.3. We showed that optimally eager agents could end in an infinite cyclic execution. By the theorem above, this can not happen when the agents are conservative, optimally eager. Let us illustrate this on the example.

We will again assume that in the situation depicted in Figure 9, a_2 is chosen to act, resulting in the situation in Figure 10. Agent a_2 does not need to replan and can still use the plan formed in the initial state. Agent a_1 needs to replan, however. Since this is a conservative plan, it will start with the action of agent a_2 moving to n_4 (1 step). One possibility to extend this plan would be to wait for a_2 to create a stepping stone by going to n_2 and then n_1 (2 steps). Then a_1 could move to its own destination (which because of the initial perspective shift to a_2 has been "forgotten" by a_1) and announce success (worst case 3+1 steps), followed by a_2 moving to its destination (worst case 3+1 steps). All in all 11 steps from the initial state. Alternatively a_1 could try to minimize its worst-case distance to its destination or it could try to shorten a_2's path to n_1 before a_2 creates a stepping stone. Neither is possible, however. So, although a_1 is eager, there is no plan with execution cost of 11 or less where a_1 makes a movement after the initial movement of a_2. The same is true after a_2 has executed its move to n_2 and n_1. Now a_2 can announce success on n_1. However, also a_1 can now come up with a plan, where he wants to act. Regardless of whether a_2 announces success or not, a_1 forms an objectively strong plan starting with the movements of a_2, perhaps its success announcement, followed by a perspective shift to itself. Then it will plan for all destinations and finally use only the a_1-compatible traces with the right destination assumption.

5 Computational Complexity of MAPF/DU Planning

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

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

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

We prove hardness by a reduction from QUANTIFIED 3SAT, which is known to be PSPACE-complete [27]. Given the quantified Boolean formula

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

with n variables, n_e existentially quantified, n_u universally quantified, where $\phi(x_1, x_2, \ldots, x_n)$ is in conjunctive normal form and contains m clauses C_j with exactly three literals l_{jk}, we construct a MAPF/DU instance with the property that there exists a (globally or x_2-) strong plan 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 12.

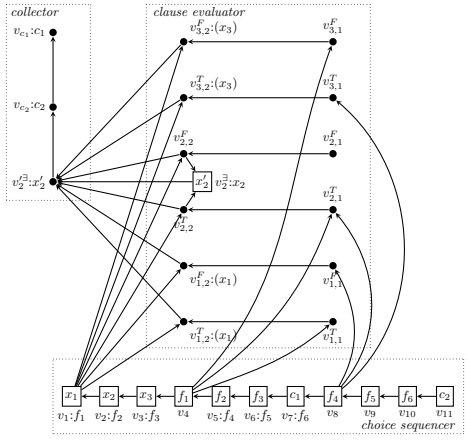


Figure 12: Example construction for $\forall x_1 \exists x_2 \forall x_3 : (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

The task of the choice sequencer is to enforce the sequence of truth-value choices of the variables and, after all choices have been made, to start the clause evaluation. It consists of a sub-graph with $n+m(n+1)$ nodes, which are named v_1 to $v_{n+m(n+1)}$. These nodes are connected linearly, i.e., there is a directed edge from v_{i+1} and v_i. The nodes v_1 to v_n are occupied by *variable agents*

named x_1 to x_n. In addition we have *clause agents* $c_j, 1 \leq j, \leq m$ on the nodes $v_{n+j(n+1)}$, respectively. The rest of the nodes are filled with *filler agents* f_k for all the not yet occupied nodes. The (deterministic) destination for each filler agent f_k is the node with an index n lower than the one f_k is starting from.

The clause evaluator contains for each variable x_i two pairs of nodes: $v^T_{i,1}, v^T_{i,2}$, and $v^F_{i,1}, v^F_{i,2}$ with a directed edge from the $i,1$ to $i,2$ nodes. 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 true and false, respectively. For an existentially quantified variable x_i, there exists an additional node $v^\exists_i$, which has a directed edge pointing to it both from $v^F_{i,2}$ and $v^T_{i,2}$ and which is the deterministic destination for agent x_i. Initially, another *copy agent* x'_i occupies $v^\exists_i$.

The node v_1 has a directed edge to the nodes $v^F_{i,2}$ and $v^T_{i,2}$ of the choice sequencer. Since the copy 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_n 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 from nodes v_k to v_{k-n} bringing all the filler agents f_k to their respective destinations. Further, all clause agents c_j have to go from $v_{n+j(n+1)}$ to $v_{j(n+1)}$, whereby these latter nodes are connected to the clause evaluator in the following way. The node $v_{j(n+1)}$, which will hold clause agent c_j after all agents moved n steps to the lower numbered nodes, 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 copies of the existential variable agents. Assuming we have the the set of existentially quantified variables $\{x_{j_1}, \ldots, x_{j_{n_e}}\}$, we create the following sequence of nodes $v'^\exists_{j_{n_e}}, \ldots, v'^\exists_{j_1}, v_{c_m}, \ldots, v_{c_1}$. From all nodes $v^F_{i,2}$, $v^T_{i,2}$ and $v^\exists_i$ there is a directed edge to $v'^\exists_{j_{n_e}}$. If there is no existential variable, then the directed edges point to x_{c_m}.

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 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 (universally quantified) 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 copy 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.

Conversely, assume that there exists an objectively strong plan (or an x_2-strong plan). Then the movements and branching points are as above and if the clause agents can pass through the clause evaluator to the collector, 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. ■

Solving the bounded plan existence problem is, of course, not easier, since on can reduce the general existence problem to the bounded existence problem with a polynomial given by Theorem 6. Membership follows again by nondeterministically checking all traces of the polynomial-depth (Theorem 6) plan.

Corollary 12 *Deciding whether there exists a MAPF/DU i-strong or objectively strong plan with execution cost of k or less is PSPACE-complete.*

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 $n + m \cdot (n + 1)$ agents (for moderately large n and m) at the same time. For a fixed number of agents, the problem is fortunately solvable in polynomial time.

Proposition 13 *For a fixed number c of agents, deciding whether there exists a MAPF/DU i-strong or objectively strong plan with execution cost of k or less can be computed in time $O(n^{c^2+c})$.*

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 cost 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 goals from there, apply our algorithm recursively. The subproblems are the ones where the agent has already reached its goal 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 goal 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.

6 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 et al. [21]. They considered memory contents moving over computer networks, formalized as *pebbles*. However, the results apply, of course, to agents in spatial environment represented as a graph 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.

The paper by Ratner and Warmuth [23] was the first paper that looked at the problem of generating optimal movement plans. They showed that the problem is NP-complete. Later, variations of this problem were analyzed, for instance, MAPF with simultaneous moves [28] or MAPF on directed graphs [6]. Furthermore, different metrics were considered [32]. 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 [30, 9, 13]. Taking some of the kinematics of real robots into account, the multi-robot path finding problem was studied as well [17, 4]. Finally, also uncertainty of the position or the movements are taken into account [29].

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 distributed planning approaches are used that combine plans generated by/for single agents [26, 20]. However, it is always assumed that the agents can communicate in order to resolve conflicts.

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, [10]. Brenner and Nebel [8], e.g., looked at this problem and proposed actually as one of their benchmarks the MAPF/DU problem. However, their solution did not include the anticipation of actions by other agents as we have done here.

Distributed POMDPs (decPOMPDs) allow for distributed execution [16] 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 [15]. 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 [24] could be used to model MAPF/DU instances, which could then be solved using an GDL-II game solver, e.g., HyperPlay [25]. Since the solver is sampling based, it is probably incomplete, though.

The approaches that come closest to the one proposed here are the papers on epistemic planning by Bolander et al. [5] and Engesser et al. [12]. As a matter of fact, the notions of *implicit coordination* and *i-strong plans* are directly borrowed from those papers. Since the MAPF/DU problem is much simpler than the general epistemic planning problem (which is undecidable), 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.

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 [1, 8, 7]. 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.

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-complete problems. A middle ground between these two extremes would require approximation algorithms for the optimal MAPF problem, and we are not aware of any such algorithm. 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* [12]. 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 minimal worst-case execution cost of such plans is 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 general and 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 the most general case of directed graphs. Whether the results also hold for undirected graphs, planar graphs, or graphs resulting from

a grid map is not obvious. While bounded MAPF/DU plan existence (Corollary 12) can be easily shown to be PSPACE-complete for undirected graphs, it is unclear, e.g., whether this still holds for general plan existence or for planar graphs. Second, 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. Third, 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. Fourth, 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. Fifth, 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. Finally, 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 [2] might help in order to coordinate implicitly.

References

[1] J. A. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pages 83–88, 1988.

[2] P. Battigalli and M. M. Siniscalchi. Strong belief and forward induction reasoning. *J. Economic Theory*, 106(2):356–391, 2002.

[3] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Strong planning under partial observability. *Artif. Intell.*, 170(4-5):337–384, 2006.

[4] S. Bhattacharya, V. Kumar, and M. Likhachev. Distributed optimization with pairwise constraints and its application to multi-robot path planning. In *Robotics: Science and Systems VI (RSS-10)*, 2010.

[5] T. Bolander, T. Engesser, R. Mattmüller, and B. Nebel. Better eager than lazy? How agent types impact the successfulness of implicit coordination. In *Proceedings of the ICAPS Workshop on Distributed and Multi-Agent Planning (DMAP-16)*, pages 42–49, 2016.

[6] A. Botea and P. Surynek. Multi-agent path finding on strongly biconnected digraphs. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence (AAAI-15)*, pages 2024–2030, 2015.

[7] R. I. Brafman and G. Shani. Replanning in domains with partial information and sensing actions. *J. Artif. Intell. Res.*, 45:565–600, 2012.

[8] M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, 19(3):297–331, 2009.

[9] B. de Wilde, A. ter Mors, and C. Witteveen. Push and rotate: a complete multi-agent pathfinding algorithm. *J. Artif. Intell. Res.*, 51:443–492, 2014.

[10] M. desJardins, E. H. Durfee, C. L. Ortiz, Jr., and M. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.

[11] K. M. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *J. Artif. Intell. Res.*, 31:591–656, 2008.

[12] T. Engesser, T. Bolander, R. Mattmüller, and B. Nebel. Cooperative epistemic multi-agent planning for implicit coordination. In *Proceedings of the Ninth Workshop on Methods for Modalities (M4MICLA-17)*, pages 75–90, 2017.

[13] A. Felner, R. Stern, S. E. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. R. Sturtevant, G. Wagner, and P. Surynek. 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)*, pages 29–37, 2017.

[14] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & OR*, 13(5):533–549, 1986.

[15] P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. *J. Artif. Intell. Res.*, 24:49–79, 2005.

[16] C. V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *J. Artif. Intell. Res.*, 22:143–174, 2004.

[17] D. K. Grady, K. E. Bekris, and L. E. Kavraki. 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)*, pages 53–70, 2010.

[18] F. Gray. Pulse code communication, 1953. U.S. Patent 2,632,058, issued March 17, 1953.

[19] W. Hatzack and B. Nebel. Solving the operational traffic control problem. In A. Cesta, editor, *Proceedings of the 6th European Conference on Planning (ECP'01)*, Toledo, Spain, 2013. AAAI Press.

[20] M. R. Jansen and N. R. Sturtevant. A new approach to cooperative pathfinding. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-08)*, pages 1401–1404, 2008.

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

[22] R. Lawrence and V. Bulitko. Database-driven real-time heuristic search in video-game pathfinding. *IEEE Trans. Comput. Intellig. and AI in Games*, 5(3):227–241, 2013.

[23] D. Ratner and M. K. Warmuth. Finding a shortest solution for the N × N extension of the 15-puzzle is intractable. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86).*, pages 168–172, 1986.

[24] S. Schiffel and M. Thielscher. Representing and reasoning about the rules of general games with imperfect information. *J. Artif. Intell. Res.*, 49:171–206, 2014.

[25] M. J. Schofield and M. Thielscher. The efficiency of the HyperPlay technique over random sampling. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 282–290, 2017.

[26] D. Silver. Cooperative pathfinding. In *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-05)*, pages 117–122, 2005.

[27] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC-73)*, pages 1–9, 1973.

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

[29] G. Wagner and H. Choset. Path planning for multiple agents under uncertainty. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS-17)*, pages 577–585, 2017.

[30] K. C. Wang and A. Botea. MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *J. Artif. Intell. Res.*, 42:55–90, 2011.

[31] P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–20, 2008.

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