Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)

Multi Agent Path Finding Under Obstacle Uncertainty

Bar Shofer, Guy Shani, Roni Stern

Software and Information Systems Engineering, Ben Gurion University
shoferb@post.bgu.ac.il, shanigu@bgu.ac.il, roni.stern@gmail.com

Abstract

In multi-agent path finding (MAPF), agents must move from their current positions to their target positions without colliding. Prior work on MAPF commonly assumed perfect knowledge of the environment. We consider a MAPF setting where this is not the case, and the planner does not know a-priori whether some positions are blocked or not. To sense whether such a position is traversable, an agent must move close to it and adapt its behavior accordingly. In this work we focus on solving this type of MAPF problem, for cases where planning is centralized but cannot be done during execution. In this setting, a solution can be formulated as a plan tree for each agent, branching on the observations. We propose algorithms for finding such plans trees for two modes of executions: centralized, where the agents share information concerning observed obstacles during execution, a decentralized, where such communication is not allowed. The proposed algorithms are complete and can be configured to optimize solution cost, measured for either the best case or the worst case. We implemented these algorithms and provide experimental results demonstrating how our approach scales with respect to the number of agents and the number of positions we are uncertain about. The results show that our algorithms can solve non-trivial problems, but also highlight that this type of MAPF problems is significantly harder than classical MAPF.

1 Introduction

In multi-agent path finding (MAPF), we must plan for several agents to move from their current positions to some target positions without colliding (Stern et al. 2019). This is an important task with numerous real-world applications, ranging from the movement of robotic arms, through robots in automated warehouses, to autonomous cars. Research in MAPF has mainly focused on the classical case, where the environment is fully observable. However, in many practical robotic applications, the robot can only identify obstacles through sensors, typically only within some proximity (Lenser and Veloso 2003). We model this type of MAPF problem, where some obstacles (e.g., walls) are known in advance while other obstacles (e.g., heavy objects or closed doors) can only be sensed once the agent reaches the proximity of the obstacles. We refer to these obstacles as *potential obstacles* and call this problem *MAPF under obstacle uncertainty* (*MAPFOU*).

In *MAPFOU*, the best action for each agent may depend on which obstacles have been sensed so far during execution. Therefore, a solution to a *MAPFOU* can be formalized as a set of plan tree (Hoffmann and Brafman 2005), one per agent, that branch on obstacle observations. Each of these plan trees can be exponential in the number of uncertain obstacles. Thus, finding a solution for a *MAPFOU* problem is harder than for a classical MAPF problem (CMAPF).

Throughout this paper, we assume that planning is done in a centralized manner. In the first part of the paper, we consider a *centralized plan execution*, where all agents share their observations. Under this assumption, the plan tree of one agent can branch on the observations of other agents. In the second part of the paper, we consider a *decentralized plan execution*, where the agents cannot communicate during execution, and each agent must rely only on its own observations. We propose complete algorithms for solving *MAPFOU* problems under the centralized and the decentralized plan execution modes. These algorithms build on existing work for CMAPF, namely the conflict-based search (CBS) algorithm (Sharon et al. 2015), and generalize it to *MAPFOU*. We also show how these algorithms can be adjusted to guarantee different types of cost optimality.

We implemented our algorithms and analyzed their performance experimentally for different grid sizes, number of agents, and amount of potential obstacles. Our results show that the proposed algorithms can be used to solve *MAPFOU* problems. The results also highlight that solving *MAPFOU* is much more difficult than CMAPF, and scaling to a large number of agents is a major open challenge.

2 Background and Related Work

We now review relevant background on multi-agent path finding problems, the CBS algorithm, and contingent planning under partial observability and sensing actions.

Classic MAPF Problem A *classical multi agent path-finding problem* (CMAPF) (e.g. Stern et al. 2019) for $k \geq 2$ agents is a tuple $\langle G, s, t \rangle$ where $G = (V, E)$ is an undirected graph, and s, t are lists of vertices. $s = [s_1, s_2, ..., s_k], t = [t_1, t_2, ..., t_k]$ define source and target vertices for each agent, i.e., s_i is the initial vertex where agent i begins, and

t_i is the target vertex where agent i should arrive. Time is discrete, and at every time step each agent performs a single action: moving to a neighbor vertex or waiting in its current vertex. A classical *single-agent plan* for agent i is a sequence of actions corresponding to moving i from s_i to t_i. A pair single-agent plans π^i and π^j is said to have a *conflict* if they plan to occupy the same vertex at the same time. This is called a vertex conflict (Stern et al. 2019). In our implementation, we allowed *swapping conflicts*, i.e., where agents traverse the same edge from opposite directions at the same time (Surynek 2015), but the theory presented in this work can be extended to support swapping and similar types of conflicts. We leave the discussion of other conflict types to future research. A *solution* to a CMAPF problem is a set of classical single-agent plans $\Pi = \{\pi^i\}_{i=0}^{k}$, one for each agent, such that all pairs of single-agent plans do not conflict. We assume that agents that arrive at their target positions exit the graph, and hence, cannot participate in additional conflicts. This assumption, while less common in the MAPF literature, has been made in prior work on MAPF (Svancara et al. 2019; Morag et al. 2022). Extending our algorithms to agents that remain in the graph until all agents arrive at their target is straightforward.

A solution to a CMAPF problem can be optimal with respect to a given objective function. Two common objective functions in the CMAPF literature are *makespan* — the amount of time-steps until the last agent reaches its target location, and *sum of costs* (SOC) — the total amount of actions required by all agents to reach their target location. In this paper, we focus on the SOC, but conversion to makespan would not require a significant change.

As with other multi-agent problems, during planning, MAPF problems can be either *distributed* or *centralized*. In a distributed setting, each agent plans independently, and information sharing must be done through pre-designed communication protocols, which may incur additional cost. In a *centralized* setting, we assume a single central computing power that plans and searches for a solution for all agents.

In this paper, we focus on centralized planning. However, in our case, one must consider whether the execution is also centralized, i.e., whether agents share information over the obstacles they observe while executing the plans.

Conflict Resolution Once a conflict between several agents has been identified, it must be resolved to ensure a safe solution. Given a conflict between a subset of agents, we can plan jointly for the subset, avoiding joint actions that lead to a collision (Standley 2010). However, planning for multiple agents together increases the problem complexity exponentially. Thus, such methods scale up poorly.

The *conflict based search* (CBS) (Sharon et al. 2015) algorithm employs a different conflict resolution method in which constraints are introduced over the set of vertices an agent can visit at a given time. We denote by $\langle i, v, t\rangle$ a constraint on agent i not to visit v at time t. We can then use standard pathfinding algorithms to solve individually for each agent subject to these constraints. To ensure optimality, when two agents i, j have a potential conflict on vertex v at time t, we can create two possible constraints $\langle i, v, t\rangle$ and $\langle j, v, t\rangle$. We can then solve once for each constraint, replanning only for the agent that received the constraint. We can then observe the two solutions, and choose the one that is better, e.g., with a lower sum of costs.

As there can be multiple conflicts, CBS maintains a set of constraints on each agent, and can be considered to search the space of possible constraint sets. This search process can be maintained as a binary tree, called the *constraint tree*.

Contingent Planning In classical automated planning agents fully observe the current state. In contingent planning under partial observability (Albore, Palacios, and Geffner 2009; Brafman and Shani 2012; Bonet and Geffner 2011), some aspects of the problem are hidden, but can be either directly observed using sensing actions, or reasoned about given some observations. A solution to such problems can be formalized as a plan tree, or more compactly, a plan graph (Muise, Belle, and McIlraith 2014; Maliah, Komarnitsky, and Shani 2021). To compute such plan graphs we can plan until the next sensing action, and then replan following each possible observation (Bonet and Geffner 2011; Maliah, Komarnitsky, and Shani 2021). Multi-agent extensions were also suggested (Brafman, Shani, and Zilberstein 2013; Bazinin and Shani 2018) in a decentralized setting.

MAPFOU problem can be modeled as a contingent planning problem (for centralized execution), or as a multi-agent QDec-POMDP (for decentralized execution). However, current contingent solvers support much more complicated settings, such as complex relationships between the unobserved aspects, and as such, do not scale as well. Our approach can be considered as a domain-specific adaptation of replanning-based contingent solvers, scaling to much larger problems.

The concept of optimal plan trees is also not trivial. It is unclear how one can aggregate over multiple branches to compare plan trees. Averaging over branches makes the underlying assumption of uniform distribution. As the probability of obstacle existence is unknown, any such assumption may lead to unjustified preference over plan trees (Shmaryahu, Shani, and Hoffmann 2019). In this paper, we focus on either optimizing for the best case, i.e., when all obstacles are absent, or for the worst case, where all obstacles exist. We make no guarantees over other plan tree branches.

Our problem is also related to the Canadian Traveler Problem (CTP) (Papadimitriou and Yannakakis 1991; Bnaya, Felner, and Shimony 2009), a shortest path problem where some edges are blocked with some probability. In *MAPFOU*, however, we do not have a probability distribution over the potentially blocked vertices.

MAPF Variants with Uncertainty Various forms of uncertainty in MAPF were studied in the past. Nebel et al. (2019) explored uncertainty over the agents' possible destinations, where the agents cannot directly communicate except to announce that they reached their destination. In MAPF with delay probabilities (MAPF-DP) each agent action may be delayed with some probability (Wagner and Choset 2017; Hönig et al. 2016; Atzmon et al. 2020a). In stochastic MAPF (Levy, Shani, and Stern 2022) an agent may stochastically move in a different direction than intended. Our problem is fundamentally different, as we do

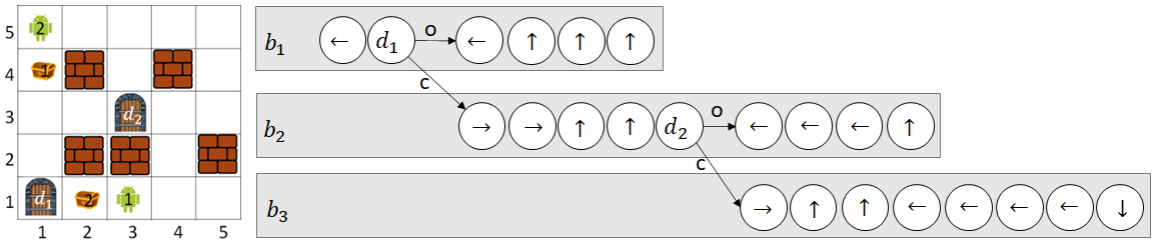


Figure 1: Example grid. Agents must reach their designated treasure boxes. Doors can be open or closed. A possible plan tree for agent 1. Arrows denote moving actions. d_i denotes sensing whether door i is open. o, c denote open and closed.

not assume known probabilities over the potential obstacles' state. MAPF variants under known stochastic uncertainty were also suggested. Atzmon et al.'s (2020b) work on robust MAPF finds solutions to CMAPF problems that are robust to delays up to a fixed threshold. Shahar et al. (2021) explored MAPF with time uncertainty (MAPF-TU), given a lower and upper bound over action duration. They proposed both an offline approach and an online approach, and distinguished between free and limited communication, as we do. They did not propose an offline solution that is optimal and considers the observed information.

3 Problem Definition

We now describe an extension to MAPF that we call *multi-agent path finding under obstacle uncertainty* (*MAPFOU*). In *MAPFOU*, a subset of vertices in the graph may be blocked by obstacles. We call these vertices *potential obstacles* (POs). The *state* of a PO is either *blocked* or *unblocked*. The agents do not initially know the state of the potential obstacles, but can use sensing actions to reveal them. Formally, a *MAPFOU* problem is a tuple $\langle G, s, t, O \rangle$, where G, s, t are as in a CMAPF problem and $O \subseteq V$ is the set of POs. An *obstacle configuration* $c : O \to \{blocked, unblocked, b|u\}$ is a mapping between POs and their possible state, where $c(p) = b|u$ represents that the state of $p \in O$ is not yet known. We say that an obstacle configuration c is *complete* if every PO is mapped to either *blocked* or *unblocked*, i.e., there is no PO p such that $c(p) = b|u$. We say that two POs c and c' are *consistent*, denoted $cons(c, c')$ if they agree on the state of every PO they know about, i.e.,

$$\forall p \in O : c(p) = c'(p) \vee c(p) = b|u \vee c'(p) = b|u \quad (1)$$

In our setting, there is a single obstacle configuration that is true and does not change throughout the execution. A PO is either blocked or unblocked and its state does not change. The solver, of course, does not initially know the true obstacle configuration. Problems where the state of a PO can change arbitrarily throughout the execution are perhaps less interesting, as there is no benefit in communicating an observed PO to the other agents, because the state may change by the time other agents will approach the PO.

The set of actions of an agent in a MAPFOU problem contains the move and wait actions of a CMAPF problem, as well as *sense* actions for each $v \in O$. In this paper, we assume that a sensing action takes a single time step and can only be executed when the agent is near an obstacle. That is, an agent located at vertex v_i can observe the occurrence of obstacle in v_j only if there exists an edge $(v_i, v_j) \in E$. Extending the sensing model to allow sensing from a finite range should not significantly influence our methods. However, a sensing model that allows agents to observe remotely, e.g., from high ground from which several POs can be observed, may raise value of information concerns, which are difficult to model in non-stochastic settings.

A single-agent plan in a MAPFOU problem is a plan tree τ, where nodes are labeled by actions, and edges are labeled by observations. For move and wait actions, there is a single outgoing edge labeled by the $null$ observation. For sense actions, there are two outgoing edges labeled *blocked*(v) and *unblocked*(v), where v is the sensed vertex. A node n in the tree is associated with a time step $n.t$, a vertex $n.v \in V$, and an obstacle configuration $n.c$, which are the time step, agent's position, and agent's knowledge about the state of the POs, respectively, when it reaches n. In our sensing model, $n.c$ corresponds to the set of potential obstacles that were sensed on the path from the root to n.

Example 1. In Figure 1, the vertices of the G are the grid cells, and edges denote neighboring cells. In this case, movement actions correspond to moving from the current grid cell in one of the principal directions. Doors here, that may be open or closed, correspond to the possible obstacles. That is, $O = \{\langle 1, 1 \rangle, \langle 3, 3 \rangle\}$. Figure 1 also shows an example of a plan tree. Nodes are labeled by actions, where arrows denote movement, and d_i denote sensing actions over the vertex where d_i resides. Each such sensing action can have two possible outcomes — the doors are either open or closed, and hence, the vertex is either blocked or unblocked, respectively. In this plan tree agent 1 first attempts to move through door d_1. If it observes, once it is near the door, that the door is open, it moves up toward its treasure box (branch b_1). If d_1 is closed, it attempts to go to door d_2, to move through it to the box (b_2). If that door is closed as well, the agent is forced to go around the walls (b_3).

To generalize the notion of a conflict, we observe that a branch in a single-agent plan tree τ represents a classical single-agent plan. Formally, for a single-agent plan tree τ and an obstacle configuration c, we denote by *tree2plan*(τ, c) the classical single-agent plan represented by the corresponding branch in τ.

Definition 1 (Conflict between Plan Trees)**.** A pair of plan trees τ^i and τ^j has a conflict if there exists a complete obsta-

cle configuration c for which the classical single-agent plans *tree2plan*(τ^i, c) and *tree2plan*(τ^j, c) have a conflict.

Notice that, in the worst-case, identifying a conflict between plan trees is exponential in the number of POs. A MAPFOU solution is a set of plan trees, one per agent. For every leaf node n of τ^i, $n.v = t_i$ and there are no conflicts between its constituent single-agent plan trees.

Objective Function We assume that sensing actions do not have an inherent cost, and only consider the cost of the agents' move and wait actions. Following prior work on MAPF with non-deterministic delays (Shahar et al. 2021), we consider two alternative objective functions: *best-case* SOC and *worst-case* SOC. This also conforms with solution cost measures that have been proposed for single-agent plan trees (Shmaryahu, Shani, and Hoffmann 2019). Let $SOC(\{\pi^i\})$ denote the SOC of a set of classical single-agent plans $\{\pi^i\}$, and let $\mathcal{C}(\mathcal{P})$ denote all possible obstacle configurations for a MAPFOU problem $\mathcal{P}$:

$$\text{best-case SOC: } \min_{c \in \mathcal{C}(\mathcal{P})} SOC(\{\textit{tree2plan}(\tau^i, c)\}) \quad (2)$$

$$\text{worst-case SOC: } \max_{c \in \mathcal{C}(\mathcal{P})} SOC(\{\textit{tree2plan}(\tau^i, c)\}) \quad (3)$$

Observation 1. For any MAPFTOU problem $\mathcal{P}$, if $\{\tau^i\}$ is an optimal solution wrt the best-case SOC objective function then its cost is equal to the SOC of the single-agent plans corresponding to the obstacle configuration where all potential obstacles are *unblocked*. Similarly, the optimal worst-case SOC is obtained when assuming all POs are *blocked*.

Centralized vs. Decentralized Execution We assume planning is centralized, and consider two possible execution paradigms: *centralized* and *decentralized*. In centralized execution, agents constantly share their observations and coordinate their actions. That is, when one agent senses a potential obstacle, all agents are immediately notified of the outcome of this sensing action. Then, an agent can branch its plan tree based on observations of other agents. In decentralized execution, communication during execution is prohibited, and each agent must rely only on its own observations. It is also possible that communication has costs, and agents must decide whether to sense themselves or obtain information from other agents. We leave this for future research. In summary, we consider in this work four variants of the MAPFOU problem: Centralized Execution while optimizing for the Best Case SOC (CEBC), Centralized Execution while optimizing for the Worst Case SOC (CEWC), Decentralized Execution while optimizing for the Best Case SOC (DEBC), and Decentralized Execution while optimizing for the Worst Case SOC (DEWC). All variants are at least NP-Hard, since CMAPF, which is NP-Hard (Nebel 2020; Surynek 2010; Yu and LaValle 2013), is a special case of *MAPFOU* without any POs. We conjecture that *MAPFOU* is EXP-SPACE complete, because the size of the plan trees can be exponential in the number of POs.

Algorithm 1: Centralized execution, best case

```
1  CEBC
       Input: : Graph G, Start vertices S, Target vertices T
       Input: : Obstacle configuration c, Unknown POs U
2      G' ← AssumeObs (G, c ∪ {v : unblocked | v ∈ U})
3      {π^1, ..., π^k} ← PlanCMAPF(G', S, T)
4      foreach agent i do
5          (n^i_1, ... n^i_{|π^i|}) ← CreateBranch(π^i)
6      for j = 1... max(|π^i|) − 1 do
7          Find p ∈ U such that ∃i : ⟨p, n^i_j.p⟩ ∈ E
8          if there exists such p then
9              {τ'^i} ←CEBC( G, [n^i_j.p]_i, T,
10                                c ∪ {p : blocked}, U \ {p} )
11             for every agent ℓ do
12                 Add a new node n_sense with action
                     Sense(p) before n^ℓ_j
13                 Add an outgoing edge from n_sense,
                     marked unblocked(p), with child n^ℓ_j
14                 Add an outgoing edge from n_sense,
                     marked blocked(p), with child τ'^ℓ
15             Add p : unblocked to c
16             Remove p from U
17     For every agent i, create plan tree τ^i, with root n^i_1
18     return {τ^i}
19 Main
20     CEBC(G, s, t, ∅, O)
```

4 Solving MAPFOU Using Replanning

We now present a set of algorithms for finding optimal solutions for all the MAPFTO variants we consider in this work.

4.1 Centralized Execution

For the MAPFOU centralized execution variants (CEBC or CEWC), we propose an iterative approach in which the algorithm focuses on one obstacle configuration at a time, building the set of plan trees one branch at a time. Observe that a MAPFOU with a single obstacle configuration is essentially a CMAPF problem. We can use a standard CMAPF solver to solve this CMAPF problem, resulting in a set of safe single-agent plans for that particular obstacle configuration. We convert each single-agent plan to a branch in the set of plan trees, adding a sense action whenever an agent is adjacent to a potential obstacle that has not been sensed yet. For each of these sense actions, we call again the CMAPF solver starting from the agent's expected location and the observation outcome that was not explored yet.

Algorithm 1 presents the pseudo code for our algorithm for CEBC variant (omitting some details for ease of exposition). The input to our CEBC algorithm is (1) G, the underlying graph of the MAPFOU problem, (2) S, the currently assumed locations of the agents, (3) T, the target location of the agents, (4) c, an obstacle configuration representing an assumption over the state of some of the potential obstacles, and (5) U, a set of potential obstacles whose state is un-

known (i.e., $b|u$) in c. For a MAPFOU problem $\langle G, s, t, O\rangle$ the initial call to our CEBC algorithm sets $S = s$, $T = t$, $c = \emptyset$, and $U = O$, representing that the agents are in their initial locations and we do not assume anything yet about the states of the potential obstacles.

We begin (line 2) by constructing an obstacle configuration that is consistent with c and assumes all vertices in U are unblocked. Then we create a corresponding graph G', removing all vertices that are considered blocked (the **AssumeObs** function), define a CMAPF problem $\langle G', S, T\rangle$ and solve it using an off-the-shelf CMAPF solver (the **PlanCMAPF** function). This results in a conflict-free set of classical single-agent plans $\Pi = \{\pi^i\}$ (line 3). Each single-agent plan in this set is then transformed into a branch in the plan tree of the corresponding agent (line 4). We then traverse these branches, searching for the earliest node in which an agent is planned to occupy a vertex that is adjacent to a potential obstacle p that was not sensed yet (line 7). Then, we add a $Sense(p)$ action to all plan trees (line 12) creating a branch in the plan trees to allow all agents to behave differently depending on whether given that potential obstacle is blocked or not. In this, we abuse notation, as only one agent actually senses whether p is blocked, and all other agents only observe the result. We recursively create plan trees for the case where p is blocked (line 10), and link the resulting trees following the sensing nodes (line 14). Finally, we add the assumption that p is unblocked to c and remove it from U. Hence, if another agent must later pass through p, it no longer needs to sense whether it is blocked. We ignore many details. For example, it might be that multiple agents pass through possibly blocked vertices at the same time step, requiring a more delicate, but straightforward, treatment with multiple consecutive sensing actions.

Example 2. Figures 2a and 2b show the plan trees created by CEBC. We first call CEBC with $c = \emptyset, U = \{d_1, d_2\}$, the agent start positions: $s_1 = \langle 3, 1\rangle$, $s_2 = \langle 1, 5\rangle$, and the agent target positions, where the boxes are located: $t_1 = \langle 1, 4\rangle$, $t_2 = \langle 2, 1\rangle$. First, assuming doors are open, the CMAPF solver generates (lines 3-5) the topmost branches (b_1 in both trees) , without the sensing action (d_1). The CMAPF solver has identified a conflict and resolved it by moving agent 2 to the side and waiting for agent 1 to get to its target. Then, agent 2 can move to its own target. After moving left in the first action, agent 1 can sense whether d_1 is open. As observations are shared during the central execution, we add the sensing action in both plan trees (line 12), allowing agent 2 to act differently if d_1 is closed. When d_1 is closed, we call CEBC recursively (line 10), with $c = \{blocked(d_1)\}$, and $U = \{d_2\}$, and S is the current position of the agents: $s_1 = \langle 2, 1\rangle$, and $s_2 = \langle 1, 4\rangle$. Now, it is beneficial for both agents to try and pass through d_2. Agent 2 moves to the left of d_2, and senses. Then, again, we branch in both trees. We call CEBC recursively with $c = \{blocked(d_1), blocked(d_2)\}$, $U = \emptyset$, $s_1 = \langle 4, 1\rangle$, and $s_2 = \langle 2, 3\rangle$.

Optimizing for the worst case, i.e., when all obstacles are assumed to be blocked, is very similar. CEWC (centralized execution worst case) differs from CEBC (Algorithm 1) in that we assume in the CMAPF problem (line 2) that all unobserved vertices are blocked. The recursive call (line 9), assumes that p is unblocked, allowing a possible shortcut over the worst case. The children of the sensing nodes are also attached to the opposite case than in CEBC.

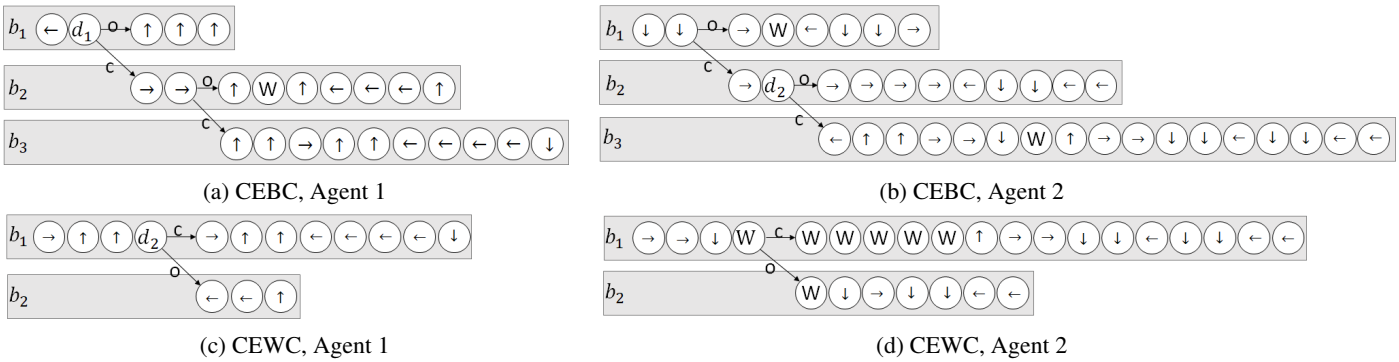
Example 3. Figures 2c and 2d show the plan trees for the worst case. Here, we assume that all doors are closed. Hence, the CMAPF solver must move the agents around the doors, with a long wait (denoted W) for agent 2 until agent 1 moves past the conflict area. It happens that here both agents move next to d_2 — agent 1 on its shortest path, and agent 2 as it needs to move aside allowing agent 1 to pass. Then, one agent can sense for d_2, and if it happens to be open, both agents can design a shorter path (branch b_2).

We can see that for our running example, as often happens in our experiments, the worst case avoids possible obstacles, and thus the plan trees have fewer branches. For plan cost, we can see that the best case in CEBC has a sum of costs of 12 (ignoring sensing), and 16 for CEWC. For the worst case, the sum of costs is 32 in CEBC, and 29 in CEWC.

Theorem 1. Given a sound, complete, and optimal CMAPF solver, CEBC and CEWC algorithms are also sound, complete, and optimal, i.e., guaranteed to return a solution, if such exists, which has no conflicts and is optimal wrt to the chosen objective function (best- or worst-case SOC).

Proof. **Soundness.** Every branch in the plan trees returned by our algorithm is created by running a CMAPF solver under the assumption of a specific obstacle configuration for all agents. As the CMAPF solver is sound, the branches of the plan trees do not contain any conflict, and all agents end at the goal in that branch. In addition, the main loop (lines 6-15) ends after inspecting all actions in the active branch, and hence the plan tree does not contain any unexplored branches. Thus, the solution is sound, i.e., conflict-free and in all leaves all agents reached their target.
Completeness. As the underlying graph G is undirected and sensing actions do not change it, the agents can always backtrack to any joint state they occupied before. Thus, adding nodes and edges to the plan trees never results in a dead-end. The number of potential obstacles in finite and every recursive call to our algorithm removes one potential conflict from U. Hence, we are guaranteed that our algorithm terminates in finite time.
Optimality. Consider the best-case SOC objective function. We know that the optimal best-case SOC is equal to the solution of the CMAPF problem that corresponds to assuming all potential obstacles are unblocked (Observation 1). By construction, our algorithm is guaranteed to include that solution as branches in the resulting set of plan trees. □

The computation complexity of CEBC can be, unfortunately, very high. In the worst case, CEBC calls a CMAPF solver for every possible obstacle configuration, i.e., $2^{|O|}$ times. This worst case does not arise in pathological cases only. When every PO is visited by at least one agent, then all obstacle configurations must be considered. As finding optimal solutions to CMAPF is NP-Hard (Nebel 2020; Surynek 2010; Yu and LaValle 2013), the worst-case complexity of CEBC is exponential in both the number of agents and the



(a) CEBC, Agent 1

(b) CEBC, Agent 2

(c) CEWC, Agent 1

(d) CEWC, Agent 2

Figure 2: Plan trees for centralized execution for the running example. Arrows denote movement actions, W denotes waiting, d_i denotes sensing whether door i is open or closed, and o, c denotes the open and closed observations.

number of POs. If optimality is not required and the underlying graph is undirected, then a polynomial-time CMAPF solver can be used (Daniel Kornhauser 1984), in which case CEBC complexity is only exponential in the number of POs.

4.2 Decentralized Execution

For decentralized execution, agents do not share their observations, and hence, different plan trees branch on different observations. We thus take a different approach, inspired by the CBS algorithm (Sharon et al. 2015). First, each agent computes a complete plan tree ignoring all other agents. Then, we search for possible conflicts between these plan trees. If a conflict is detected, we resolve it by imposing constraints on each of the conflicting agents and revising their plan trees accordingly. We now describe the key steps of this algorithm: plan tree generation, conflict detection, and the constraints imposed to resolve conflicts.

Plan Tree Generation The plan tree generation method we use in this algorithm is similar to the one used in CEBC, except that it uses a single-agent solver (A^*) instead of a CMAPF solver. That is, we maintain an obstacle configuration c which initially assumes all POs are unknown. Then, we run A^* to find a shortest path π for the respective agent assuming all POs are *unblocked*. Following, for every potential obstacle $p \in O$ that is on the shortest path π and is assumed to be *unblocked* (in the best case scenario) in c, we recursively call the plan tree generation method with an obstacle configuration c' identical to c except for assuming that p is blocked. This plan tree generation method is called for each agent, creating an initial set of plan trees $\tau^1, \ldots, \tau^k$. For worst case optimization we first assume that all POs are *blocked*, and then, whenever passing near a PO, recursively construct the plan tree where it is *unblocked*.

Conflict Detection One may detect conflicts between plan trees τ^i and τ^j by enumerating all possible complete obstacle configurations $c \in \mathcal{C}(\mathcal{P})$, checking for conflicts between the corresponding single agent plans *tree2plan*(τ^i, c) and *tree2plan*(τ^j, c). We suggest a more efficient method, based on the following observation.

Observation 2. Plan trees τ^i and τ^j have a conflict iff there exists two nodes $n^i \in \tau^i$ and $n^j \in \tau^j$ such that (1) $n^i.v = n^j.v$, (2) $n^i.t = n^j.t$, and (3) $n^i.c$ and $n^j.c$ are consistent.

That is, a pair of plan tree nodes satisfy the conditions in Observation 2 if they represent the same location and time, and their obstacle configurations are consistent. Our *MAPFOU* algorithm for decentralized execution checks if a pair of plan trees has a conflict by searching for a pair of nodes in them that satisfy these conditions. This conflict detection method can be much more efficient than iterating over all obstacle configurations.

Conflict Resolution CBS resolves conflicts by imposing constraints of the form $\langle i, v, t \rangle$, representing that agent i cannot visit vertex v at time t. In our case, however, a conflict formed by a pair of nodes at vertex v and time t is only relevant under the respective nodes' obstacle configurations. Therefore, the constraint we use to resolve a conflict formed by a pair of plan-tree nodes must include information about the obstacle configurations where it is relevant. We define the following type of MAPFOU constraint:

Definition 2 (MAPFOU Constraint)**.** A MAPFOU constraint is defined by a tuple $\langle i, v, t, c_i, c_j \rangle$ where i is an agent, v is a vertex, t is a time step, and c_i and c_j are obstacle configurations. A plan tree τ^i for agent i violates this MAPFOU constraint if there exists a node $\hat{n}_i \in \tau^i$ where $\hat{n}_i.v = v$, $\hat{n}_i.t = t$, and for every $p \in O$, where O is the set of all POs, the following conditions hold:

$$(c_i(p) \neq b|u) \rightarrow (\hat{n}_i.c(p) = c_i(p) \vee \hat{n}_i.c(p) = b|u) \quad (4)$$

$$(c_i(p) = b|u) \rightarrow (\hat{n}_i.c(p) = c_j(p) \vee \hat{n}_i.c(p) = b|u) \quad (5)$$

Intuitively, the first condition (4) ensures the obstacle configuration of $\hat{n}_i$ is on the same branch in the plan tree as c_i, and the second condition (5) ensures that if $\hat{n}_i$ is deeper than n^i in the plan tree. To ensure optimality, one can follow the same framework as CBS, where both options to resolve any identified conflict are checked. This is managed by maintaining a constraint tree in which each node represents a set of plan trees and a set of constraints. A best-first search over this tree on the desired objective function is guaranteed to be sound, complete, and optimal following the same proof as in the original CBS.

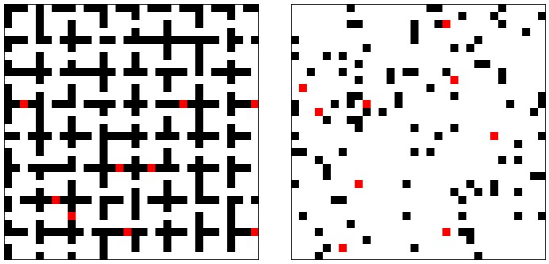


Figure 3: The grids used in our experiments: rooms (left) and open grid (right). Red grid cells denote POs.

Theorem 2. DEBC is sound, complete, and optimal.

The proof, as well as an example, can be found in the supplementary material. The runtime of DEBC is challenging to analyze, as it depends on the number of conflicts found between the different plan trees. In the worse case, the complexity of generating a plan tree for a single agent is exponential in the number of POs, and DEBC generates a plan tree whenever a node in the constraint tree is generated. The number of nodes in the constraint tree is exponential in the number of conflicting plan trees generated by the algorithm. Analyzing exactly the number of conflicts considered by CBS in CMAPF is a known challenge, and also in our case. However, as a rough estimation, as conflicts occur between plan tree nodes, a conservative estimate is that the number of conflicts is polynomial in the number of vertices in the graph, the number of time steps until all agents reached their goals, and the number of obstacle configurations. The latter is exponential in the number of POs, and thus the worst-case complexity of DEBC is doubly exponential.

This complexity analysis, and difficulty of implementation, lead us to simplify our implementation of DEBC in two ways. First, we only considered the condition in Equation 5 when considering MAPFOU constraints. This does not hinder completeness or optimality, but may cause the constraint tree to have more nodes. Second, we searched the constraint tree with a greedy best-first search, instead of using optimal search such as A^*. This does have implications on completeness and optimality. Yet, we observed almost no quality reductions in our experiments.

5 Empirical Evaluation

In this section, we report on a set of experiments, designed to evaluate the performance of the proposed algorithms in different settings. Our methods are implemented in Java and the source code is available in the supplementary material. Experiments were run on a an $i7$, 2.8GHz CPU, with 16GB RAM, but we limited memory usage to only 4GB.

Benchmark Domains Following most prior MAPF work, we experimented on different 4-neighborhood grids. We designated a set of grid cells as potential POs, placing them at strategic points in the grid, ensuring there is always a path between any two cells even if all POs are blocked. We experimented on two types of grids: *rooms*, where the grid is split into rooms with one cell doorways, and *open* grids with randomly places obstacles, some of which were chosen to be PO. Figure 3 shows these grids, where the red grid cells are possible locations of POs. In our experiments we varied the size of these grids, the number of agents, and the number of POs (sampled from the red cells in Figure 3).

Grid	Agents	Obstacles	Joint	CEBC
6×6	3	2	2.400	0.027
8×8	3	2	313.780	0.041
13×13	2	3	11.400	0.140
15×15	2	3	39.830	0.200
17×17	2	3	133.330	0.300

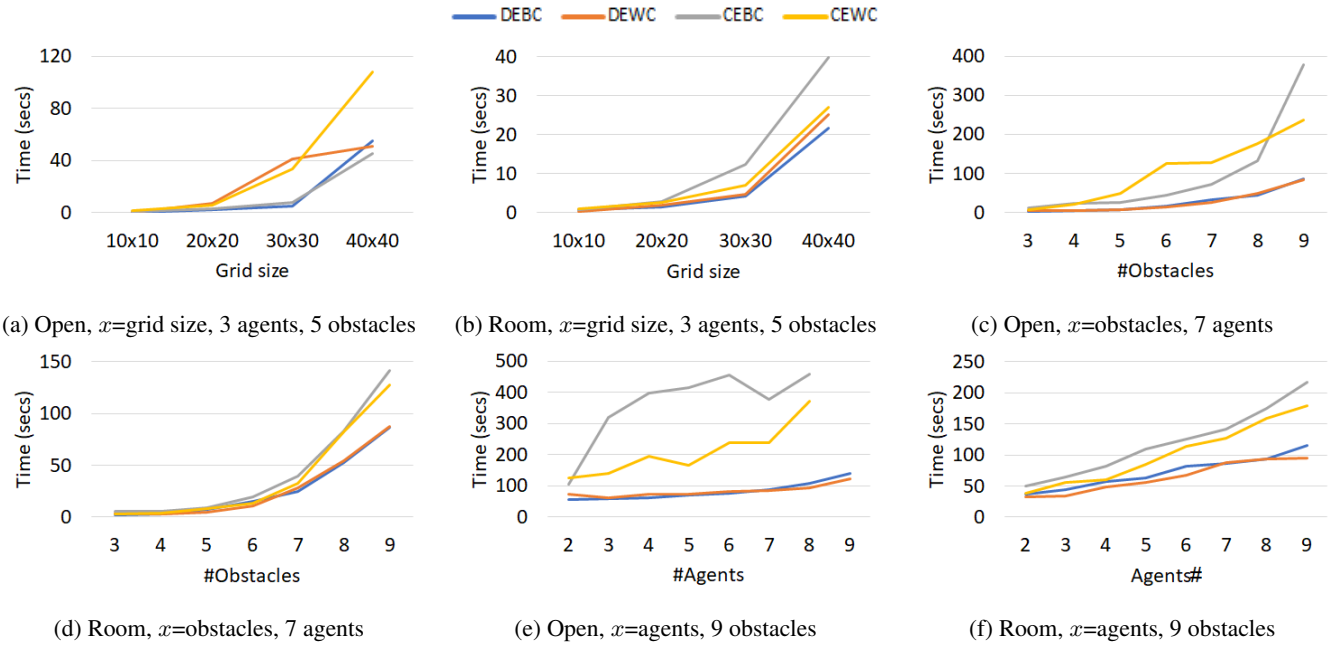
Table 1: Comparing runtime (sec) between solving the joint problem, and our CEBC solver.

Protocol For each of the grid types, we generate 10 test problems. Given k, the number of agents, we randomly choose for each agent a start and target position in the grid. In a grid with N predefined possible obstacle positions, given $n \leq N$, the number of obstacles in the test problem, we randomly choose n of the possible obstacle positions, assuming the rest of the possible obstacles do not exist. Below, we report average results over the 10 executions. When testing scaling up, we set a time limit of 15 minutes for a given experiment. We limit our experiments to cases where all 10 problems were solved within the given time.

Baseline Since this is the first work on *MAPFOU*, there is no natural baseline to compare our algorithms with. Instead, we considered as a baseline a naive implementation of a contingent planner (Hoffmann and Brafman 2005) that runs an A^* search over the joint state space of all agents for every possible obstacle configuration. A state in this search space consists of the positions of all agents, and an action is a joint action, that is, a combination of single-agent actions (one per agent) except those directly leading to a conflict.

Results Table 1 shows the results. Clearly, the joint space grows exponentially, and we cannot handle problems with more than 2-3 agents and 2-3 obstacles, even for small grids. Figures 4a and 4b show the runtime as the grid grows. As expected, the runtime grows exponentially with the grid size. The runtime in open grids grows faster than for room grids. This is because in open grids, when agents take the shortest path, their paths intersect much more often, and the amount of conflict resolutions grows, requiring more time to resolve. This is more pronounced in the centralized execution case, as agents branch on observations from other agents, over obstacles that they do not necessarily pass through.

Figure 4 compares the run time of our algorithms as the number of agents and obstacles increases. As we can see (Figures 4c and 4d), increasing the number of obstacles results in an exponential increase. This is because observations over obstacles result in a split in the plan tree, which requires replanning for exponentially more branches. We placed the POs strategically to avoid positions that are completely irrelevant (i.e., not sensed by any agent in any branch). Such irrelevant obstacles are implicitly ignored by our algorithms and do not affect the runtime. Our approach would scale



(a) Open, x=grid size, 3 agents, 5 obstacles

(b) Room, x=grid size, 3 agents, 5 obstacles

(c) Open, x=obstacles, 7 agents

(d) Room, x=obstacles, 7 agents

(e) Open, x=agents, 9 obstacles

(f) Room, x=agents, 9 obstacles

Figure 4: Comparing execution time, given a growing grid size (a,b), obstacles (c,d), and number of agents (e,f)

	Worst			Best		
Agents	2	3	4	2	3	4
Cen. better	44%	50%	38%	50%	44%	38%

Table 2: Advantage in solution costs execution modes.

much better in environments with POs at completely random positions, as many of them would be irrelevant. Again, centralized execution results in splitting over observations that are not necessarily relevant. This is most obvious in the open environment, where movement is less restricted, and hence fewer obstacles are relevant for an agent. An interesting direction for future research is to allow agents to ignore observations that are not relevant to them.

When increasing the number of agents (Figures 4e and 4f) the increase is almost linear for the distributed execution case. This is due to our distributed computation, where we create plan trees for each agent independently. As these grids are fairly large, with respect to the number of agents, conflicts do not happen too often. Here, centralized planning over the open grid presents the toughest problems to solve, and with 9 agents, both CEBC and CEWC failed to solve the problem within the 15 minutes timeout. This is because in the open grid less obstacles are relevant for multiple agents, yet reporting them requires substantial additional planning.

Centralized vs. decentralized execution. To compare the execution costs obtained with the centralized and decentralized execution modes, we performed a set of experiments on a 12×12 open grid with 4 obstacles and 2, 3, and 4 agents, with every possible PO configuration. Table 2 shows the percentage of PO configurations where the centralized execution mode obtained better solution cost, when optimizing for the best case and when optimizing for the worst case. As expected, in most cases the centralized execution, where agents can share information about the observed obstacles, leads to a solution cost that is lower than that of the decentralized execution, where agents cannot share information. The advantage of sharing information allowed agents to save up to 8 actions. In less than 6% of the cases, there was a small advantage to the decentralized execution, which may be because the planner optimized for the worst case (all obstacles blocked) and thus it may be suboptimal in other obstacle configurations. Similar results were obtained when optimizing for the best case.

6 Conclusion and Future Work

This paper explored a new MAPF variant called MAPFOU, where the problem solver has incomplete knowledge about the traversability of some vertices. We formally defined the MAPFOU problem, its solution, and two optimization criteria that focus on the best and worst case. We suggest methods for two alternatives — when information about observed obstacles is communicated, and when no communication is allowed. We show in our experiments how our methods scale up given the grid size, the number of agents, and the number of unknown obstacles. Our results show that MAPFOU problems are much more difficult than MAPF. Our work opens the path for many research questions, including solving MAPFOU problems where communication is delayed or costly, and how to consider different types of conflicts and more complex sensing models. A major research direction for future work is to develop algorithms that can scale to a larger number of agents, e.g., by using non-optimal MAPF solvers such as prioritized planning (Ma et al. 2019).

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Atzmon, D.; Stern, R.; Felner, A.; Sturtevant, N. R.; and Koenig, S. 2020a. Probabilistic Robust Multi-Agent Path Finding. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 29–37.

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2020b. Robust multi-agent path finding and executing. *Journal of Artificial Intelligence Research*, 67: 549–579.

Bazinin, S.; and Shani, G. 2018. Iterative planning for deterministic QDec-POMDPs. In *GCAI*, 15–28.

Bnaya, Z.; Felner, A.; and Shimony, S. E. 2009. Canadian Traveler Problem with Remote Sensing. In *IJCAI*, 437–442.

Bonet, B.; and Geffner, H. 2011. Planning under partial observability by classical replanning: Theory and experiments. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Brafman, R.; Shani, G.; and Zilberstein, S. 2013. Qualitative planning under partial observability in multi-agent domains. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Brafman, R. I.; and Shani, G. 2012. Replanning in Domains with Partial Information and Sensing Actions. *J. Artif. Intell. Res.*, 45: 565–600.

Daniel Kornhauser, P. S., Gary Miller. 1984. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *FOCS*, 241–250.

Hoffmann, J.; and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proc. ICAPS*, volume 2005, 71–80.

Hönig, W.; Kumar, T. S.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-Agent Path Finding with Kinematic Constraints. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, 477–485.

Lenser, S.; and Veloso, M. 2003. Visual sonar: Fast obstacle avoidance using monocular vision. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 1, 886–891. IEEE.

Levy, E.; Shani, G.; and Stern, R. 2022. An Online Approach for Multi-Agent Path Finding Under Movement Uncertainty. In *International Symposium on Combinatorial Search (SoCS)*, 299–301.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with consistent prioritization for multi-agent path finding. In *AAAI Conference on Artificial Intelligence*, 7643–7650.

Maliah, S.; Komarnitsky, R.; and Shani, G. 2021. Computing Contingent Plan Graphs using Online Planning. *ACM Trans. Auton. Adapt. Syst.*, 16(1): 1:1–1:30.

Morag, J.; Felner, A.; Stern, R.; Atzmon, D.; and Boyarski, E. 2022. Online Multi-Agent Path Finding: New Results. In Chrpa, L.; and Saetti, A., eds., *International Symposium on Combinatorial Search (SoCS)*, 229–233.

Muise, C.; Belle, V.; and McIlraith, S. 2014. Computing contingent plans via fully observable non-deterministic planning. In *AAAI Conference on Artificial Intelligence*.

Nebel, B. 2020. On the computational complexity of multi-agent pathfinding on directed graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 212–216.

Nebel, B.; Bolander, T.; Engesser, T.; and Mattmüller, R. 2019. Implicitly Coordinated Multi-Agent Path Finding under Destination Uncertainty: Success Guarantees and Computational Complexity. *J. Artif. Intell. Res.*, 64: 497–527.

Papadimitriou, C. H.; and Yannakakis, M. 1991. Shortest paths without a map. *Theoretical Computer Science*, 84(1): 127–150.

Shahar, T.; Shekhar, S.; Atzmon, D.; Saffidine, A.; Juba, B.; and Stern, R. 2021. Safe multi-agent pathfinding with time uncertainty. *Journal of Artificial Intelligence Research*, 70: 923–954.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219: 40–66.

Shmaryahu, D.; Shani, G.; and Hoffmann, J. 2019. Comparative criteria for partially observable contingent planning. *Autonomous Agents and Multi-Agent Systems*, 33(5): 481–517.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI Conference on Artificial Intelligence*, 173–178.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. S.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*.

Surynek, P. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *AAAI*.

Surynek, P. 2015. Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Svancara, J.; Vlk, M.; Stern, R.; Atzmon, D.; and Barták, R. 2019. Online Multi-Agent Pathfinding. In *AAAI Conference on Artificial Intelligence*, 7732–7739.

Wagner, G.; and Choset, H. 2017. Path Planning for Multiple Agents under Uncertainty. In *the International Conference on Automated Planning and Scheduling (ICAPS)*, 577–585.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI*.