Multi Agent Path Finding Under Obstacle Uncertainty

Anonymous submission

Abstract

In multi-agent path finding (MAPF), several agents must move from their current positions to their target positions without colliding. In this paper we focus on a MAPF setting where a set of obstacles may exist at known positions. When an agent is close to such a position, it can sense whether the obstacle exists, and plan accordingly. In this setting, the solution for an agent is a plan tree, branching on the observations. While we focus here on centralized planning, we consider two modes of execution, where the agents share information concerning observed obstacles during execution, and where no such communication is allowed. We study two optimization criteria — optimizing for the best case, and optimizing for the worst case. For conflict resolution, we use the CBS method, adapting constraints to be conditioned on observations. We provide experiments demonstrating how our approach scales up with respect to the problem size, number of agents, and the amount of possible obstacles.

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 we have full observability over the environment. That is, MAPF researchers often assume that we know, in advance, which cells of a grid that we navigate in will be blocked. However, in many practical robotic applications, the robot can only identify obstacles through sensors, typically only within some proximity (Lenser and Veloso 2003).

In this paper we extend the MAPF problem to partial observability over obstacles. We assume that while some obstacles (e.g. walls), are known in advance, other obstacles (e.g. heavy objects or closed doors) can only be sensed once the agent reaches the proximity of the obstacles. We call this problem MAPF under obstacle uncertainty (MAPFOU).

Solving MAPFOU problems is harder than classical MAPF problems (CMAPF), because the plan may require different solutions given different obstacles. This can be formalized a plan tree (Hoffmann and Brafman 2005), branching on obstacle observations. These plan trees can be exponential in the number of uncertain obstacles.

While we assume centralized planning, we investigate two cases for plan execution. First, execution can be centralized, that is, all agents share their observations. Hence, the plan tree of one agent can branch on the observations of other agents. In many cases, however, communication is impossible, and each agent must rely only on its own observations, leading to decentralized execution.

To avoid collisions, one may compute a plan jointly, that is, searching in the space of joint plans, avoiding movements that lead to a collision. However, planning for several agents together increases the search space exponentially. Hence, CMPAF research focuses on suggesting methods to plan for each agent individually, while maintaining safe (collision-free) plans. We follow their footsteps here, focusing on methods that plan for each agent individually.

CMAPF research often requires optimal plans (Stern et al. 2019). We suggest two different optimality criteria, focusing on optimizing either for the best case, where all obstacles do not exist, or the worst case, where all obstacles exist.

We report a set of experiments, analyzing the performance of our algorithms over different settings, with respect to the grid size, the number of agents, and the amount of obstacles.

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 pathfinding 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 vertexes.

$s = [s_1, s_2, ..., s_k], t = [t_1, t_2, ..., t_k]$ define source and target vertexes 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 either moves to a neighbor vertex, or waits. Actions can be defined as a function $a : V \rightarrow V$ such that $a_{i,j}$ moves the agent from v_i to v_j. The set of actions is $A = \{a_{i,j} : (v_i, v_j) \in E\}$. In addition, for every vertex v_i, $a_{i,i}$ denotes waiting at v_i for one step.

A classical single-agent plan for agent i is a sequence of actions $\pi = (a_1, ..., a_n)$, in A, such that $a_1 = a_{s_i,j}$, and $a_n = a_k, t_i$. 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). We leave 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. 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 locations, 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, that may incur additional cost. In a centralized setting, we assume a single central computing power which 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 introdued over the set

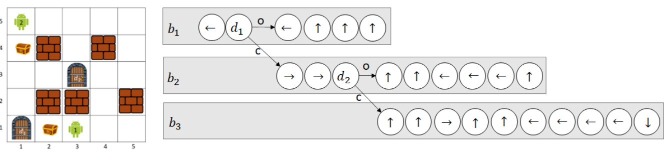


Figure 1: Example grid. Agents must reach their designated treasure boxes. Doors can be open or closed. A possible plan tree for agent 1. Arrows denote moving actions. d_i denotes sensing whether door i is open. o, c denote open and closed.

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 the 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). A common approach to computing such plan graphs is to plan until the next sensing action, and then replan following each possible observation (Bonet and Geffner 2011; Maliah, Komarnitsky, and Shani 2021). Extensions to multiple agents where also suggested (Brafman, Shani, and Zilberstein 2013; Bazinin and Shani 2018) in a decentralized setting.

The MAPFOU problem that we suggest here 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. One may consider our suggested approach 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 studies in the past. 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 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.

More 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 $c : O \rightarrow \{\text{blocked}, \text{unblocked}, \text{b}|\text{u}\}$ is a mapping between POs and their possible state. We included here an artificial state b|u, to enable representing incomplete knowledge of a PO's state. 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 $c(p) = \text{b}|\text{u}$. We say that two POs c and c' are consistent, denoted $\text{cons}(c, c')$ if they agree on the state of every PO they know about, i.e.,

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

In our setting, there is a single obstacle configuration that is true and does not change throughout the execution. That is, POs are either blocked or unblocked even before the agents start moving, and obstacles do not appear or disappear throughout the execution. The solver, of course, does not initially know the true obstacle configuration.

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 sensing actions 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 $(v_i, v_j) \in E$. That being said, this problem setting can be extended to remote sensing, although other algorithms may be needed for such 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 vertex $n.v \in V$ and a time step $n.t$, which are the position of the agent in the graph and the time step, respectively, when it reaches n. Each node n in the plan tree is also associated with a partial obstacle configuration $n.K$, representing the agent's knowledge about the state of the potential obstacles when it reaches n. In our sensing model, $n.K$ corresponds to the set of potential obstacles that were sensed on the path from the root to n.

Example 1. For the example 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 is 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 towards 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 obstacle configuration c for which the classical single-agent plans tree2plan(τ^i, c) and tree2plan(τ^j, c) have a conflict.

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 $\mathcal{P}$:

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

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

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 equal to the SOC of the single-agent plans corresponding to the obstacle configuration where all potential obstacles 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 the 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).

Algorithm 1: Centralized execution, best case

```
1  CEBC
       Input:  : Graph $G$, Start vertices $S$, Target
                 vertices $T$
       Input:  : Obstacle configuration $K$, Unknown
                 POs $U$
2      $G' \leftarrow$ AssumeObs $(G, K \cup \{v : \text{unblocked} | v \in U\})$
3      $\{\pi^1, \ldots, \pi^k\} \leftarrow$ PlanCMAPF$(G', S, T)$
4      foreach agent $i$ do
5          $(n^i_1, \ldots n^i_{|\pi^i|}) \leftarrow$ CreateBranch$(\pi^i)$
6      for $j = 1 \ldots \max(|\pi^i|) - 1$ do
7          Find $p \in U$ such that
             $p \notin K \wedge \exists i : \langle p, n^i_j.p \rangle \in E$
8          if there exists such $p$ then
9              $\{\tau'^i\} \leftarrow$
                 $CEBC\left(G, [n^i_j.p]_i, T, K \cup \{p : \text{blocked}\}, U \setminus \{p\}\right)$
10             for every agent $\ell$ do
11                 Add a new node $n_{sense}$ with action
                     $Sense(p)$ before $n^\ell_j$
12                 Add an outgoing edge from $n_{sense}$,
                     marked $unblocked(p)$, with child $n^\ell_j$
13                 Add an outgoing edge from $n_{sense}$,
                     marked $blocked(p)$, with child $\tau'^\ell$
14             Add $p$ : unblocked to $K$
15             Remove $p$ from $U$
16     For every agent $i$, create plan tree $\tau^i$, with root
         $n^i_1$
17     return $\{\tau^i\}$
18 Main
19     CEBC$(G, s, t, \emptyset, O)$
```

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

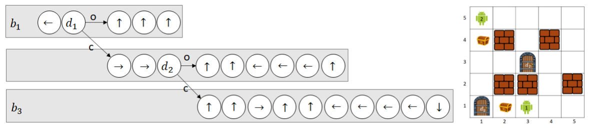
For the MAPFOU centralized execution variants (CEBC or CEWC), we propose an iterative approach in which the algorithm focuses on one obstacle configuration at a time, building the set of plan trees one branch at a time. Observe that a MAPFOU with a single obstacle configuration is essentially a CMAPF 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 psuedo code for our algorithm for CEBC variant (omitting some details for ease of exposition). The input to our CEBC algorithm is (1)

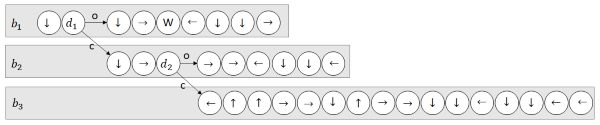
G, the underlying graph of the MAPFOU problem, (2) S, the currently assumed locations of the agents, (3) T, the target location of the agents, (4) K, an obstacle configuration representing an assumption over the state of some of the potential obstacles, and (5) U, a set of potential obstacles whose state in not specified in K. For a MAPFOU problem $\langle G, s, t, O \rangle$ the initial call to our CEBC algorithm sets $S = s$, $T = t$, $K = \emptyset$, and $U = O$, representing that the agents are in their initial locations and we do not assume anything yet about the the states of the potential obstacles.

We begin (line 2) by constructing an obstacle configuration that is consistent with K 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 a 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 behaving differently depending on whether given that potential obstacle 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 K 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.

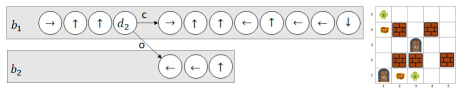
Example 2. Figures 2a and 2b show the plan trees created by CEBC. We first call CEBC with $K = \emptyset$, $U = \{d_1, d_2\}$, the agent start positions: $s_1 = \langle 3, 1 \rangle$, $s_2 = \langle 1, 5 \rangle$, and the agent target positions, where the boxes are located: $t_1 = \langle 1, 4 \rangle$, $t_2 = \langle 2, 1 \rangle$. 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 as to 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 $K = \{blocked(d_1)\}$,



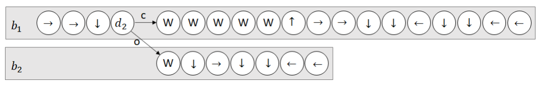
(a) CEBC, Agent 1



(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 denote sensing whether door i is open or closed, o, c denote the open and closed observations.

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 $K = \{blocked(d_1), blocked(d_2)\}$, $U = \emptyset$, $s_1 = \langle 4, 1 \rangle$, 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 less 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.

We use our own implementation of CBS, using A^* as the internal planner, to solve the CMAPF problems.

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 a guaranteed that our algorithm terminates in finite time.
Optimality. Consider the 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. □

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. We take an approach similar to CEBC (Algorithm 1), using a single-agent solver (A^*) instead of a CMAPF solver (line 3) to generate single-agent plans. 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 replan their plan trees accordingly.

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.

Definition 2 (Conflict between Plan Tree Nodes). A pair of plan tree nodes $n^i \in \tau^i$ and $n^j \in \tau^j$ for agents i and j (where $i \neq j$), respectively, form a conflict if (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 nodes $\langle n^i, n^j \rangle$ from different plan trees form a conflict if they represent the same location and time, and their obstacle configurations are consistent. We can connect Definition 1 (tree conflict) and Definition 2 (node conflict).

Observation 2. Plan trees τ^i and τ^j have a conflict iff there exists two nodes $n^i \in \tau^i$ and $n^j \in \tau^j$ that form conflict.

In decentralized execution, we check if a pair of plan trees has a conflict by searching for a pair of nodes in them that form a conflict, i.e., reside in non-conflicting branches. 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 3 (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 for agent i violates this MAPFOU constraint if there exists a node $\hat{n} \in \tau$ where $\hat{n}.v = v$, $\hat{n}.t = t$, and for every $p \in O$ the following conditions holds

$$(c_i(p) \neq \text{b}|\text{u}) \rightarrow (\hat{n}.c(p) = c_i(p) \vee \hat{n}.c(p) = \text{b}|\text{u}) \quad (4)$$
$$(c_i(p) = \text{b}|\text{u}) \rightarrow (\hat{n}.c(p) = c_j(p) \vee \hat{n}.c(p) = \text{b}|\text{u}) \quad (5)$$

Intuitively, the first condition (Equation 5) ensures that the obstacle configuration of $\hat{n}$ is on the same branch in the plan tree as c_i, and the second condition (Equation 5) ensures that if $\hat{n}$ is deeper than n^i in the plan tree then every POs it sensed is consistent with c_j.

To ensure optimality, we 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.

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

The proof, as well as an example, can be found in the supplementary material.

The above conditions and constraint-tree search are not trivial to implement. Thus, in our implementation we simplified 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 search instead of a best-first search. This does have implications on completeness and optimality. Yet, we observed almost no quality reductions in our experiments.

[Roni: Issues to raise: 1) This constraint is different from not allowing the agent to be at v and t all along the branch. Doing this is, I believe, incomplete. 2) This constraint is also different from replanning from the root up to the brach. Because, sometimes the branching may change. In other words, the plan tree my branch in different ways, after multiple replans. For example, one plan tree may branch first on obstacle $p1$ while after several replanning it might choose to branch first over obstacle $p2$. I am almost certain this can happen.]

5 Empirical Evaluation

We now discuss a set of experiments, designed to shed light on the properties of the various approaches that we suggest. Our methods are implemented in Java. Experiments were run on a an $i7$, 2.8GHz CPU, with 16GB RAM, but only 4GB were used. In the experiments below we use A^* for computing optimal paths, and our implementation of conflict resolution. We did not use an off-the-shelf CMAPF solver.

Benchmark Domains In our experiments, we follow the MAPF community (Stern et al. 2019), defining grids of various sizes and properties. In our grids, aside from the start and target locations of the agents, we also add a set of grid cells that may contain obstacles, placing them at strategic points in the grid, while ensuring that there is always a path between any two cells, even if all obstacles exist. We experiments on two types of grids — room grids, where the space is split into rooms with one cell doorways, and open grids, with a few known obstacles, and a number of possible obstacles. We use two different grid sizes.

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 obstacles 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 experiments. We limit our experiments to cases where all 10 problems were solved within the given time.

Grid	Agents	Obstacles	Joint	DEBC
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 DEBC solver.

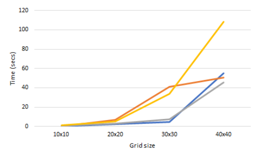
Results First, we implemented a solver over the joint space of all agents. In this case a search space node consists of the positions of all agents. Actions are joint, that is, all possible combinations of movements of all agents. We eliminate actions that lead to a collision, and hence a single search is sufficient, and there is no need for conflict detection and resolution. The joint solver plans for one branch at a time, similar to CEBC and CEWC.

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.

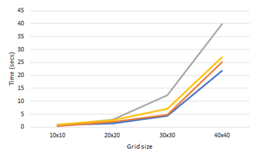
Figures 3a and 3b show the runtime as the grid grows. As expected, the runtime grows exponentially with the grid size. The runtime in open grids grows faster than for room grids. This is because in open grids, when agents take the shortest path, their paths intersect much more often, and the amount of 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.

We now analyze our algorithm sensitivity to the number of agents and obstacles. Figure 3 compares the run time of our algorithms as the number of agents and obstacles increases. As we can see (Figures 3c and 3d), 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. 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 less obstacles are relevant for an agent. An interesting future research, would allow agents to ignore observations that are not relevant for them.

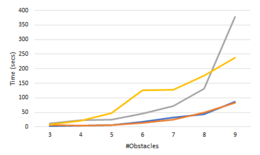
When increasing the amount of agents (Figures 3e and 3f) the increase is almost linear for the distributed execution case. This is due to our distributed computation, where we create plan trees for each agent independently. As these grids are fairly large, with respect to the number of agents, conflicts 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 most obstacles are not relevant for most agents, yet re-



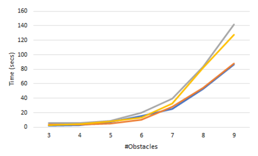
(a) Open, increasing grid size, 3 agents, 5 obstacles



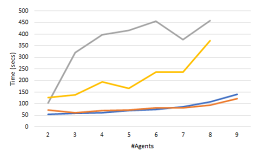
(b) Room, increasing grid size, 3 agents, 5 obstacles



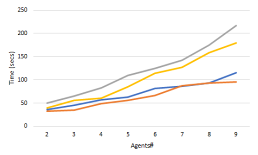
(c) Open, 32×32, increasing obstacles, 7 agents



(d) Room, 32 × 32, increasing obstacles, 7 agents



(e) Open, 32×32, increasing agents, 9 obstacles



(f) Room, 32 × 32, increasing agents, 9 obstacles

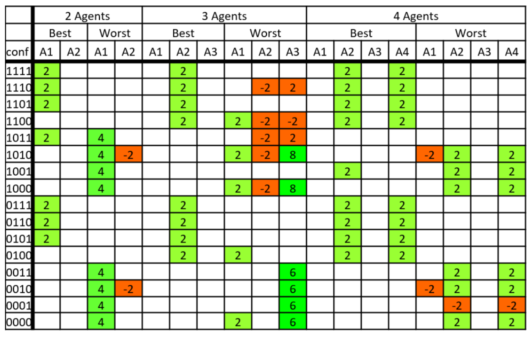
Figure 3: Comparing execution time

porting them requires substantial additional planning.

We now analyze the benefits of using a centralized execution, where agents share information about the observed obstacles. For the best and worst case that we focus on here, there is no difference, because for the best case all obstacles are missing, while for the worst case, all obstacles exist. As this is what the planner assumes, receiving information about the obstacles does not modify the extreme plans. However, differences may occur in other branches.

For analyzing these costs differences in branches other than the best and worst case, we used a smaller, 12 × 12 grid, with 4 obstacles, and vary the amount of agents. Figure 4 shows the difference between the centralized and decentralized execution, when there is a difference. With 2 agents, many obstacles are not relevant for one of the agents, and sharing information does not help. In one case information about an unblocked cell, for the worst case optimization, caused agent 1 to use a substantially shorter path.

For 3 agents, we get more pronounced differences. Agent 3 can save 6-8 actions when it gets information from other agents. For 4 agents, the differences are much smaller, because with 4 agents and such a small grid, there are many conflicts, constraining the agents in the paths they must choose. Hence, information about obstacles is less helpful.



	2 Agents				3 Agents						4 Agents							
	Best		Worst		Best			Worst			Best				Worst			
conf	A1	A2	A1	A2	A1	A2	A3	A1	A2	A3	A1	A2	A3	A4	A1	A2	A3	A4
1111	2					2						2		2				
1110	2					2			-2	2		2		2				
1101	2					2						2		2				
1100						2		2	-2	-2		2		2				
1011	2		4						-2	2								
1010			4	-2				2	-2	8					-2	2		2
1001			4									2				2		2
1000			4					2	-2	8						2		2
0111	2					2						2		2				
0110	2					2						2		2				
0101	2					2						2		2				
0100						2		2				2		2				
0011			4							6						2		2
0010			4	-2						6					-2	2		2
0001			4							6						-2		-2
0000			4					2		6						2		2

Figure 4: Difference between centralized and decentralized execution costs. Conf denotes obstacle configuration. 1 denotes unblocked. 0 denotes blocked. A_i denotes agent i.

6 Conclusion

In this paper we suggest a new variant of multi agent path finding, called MAPFOU, assuming that some vertices may be blocked. We provide formal definition for the new problem and its solution. We suggest two optimization criteria, focusing 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, number of agents, and the amount of unknown obstacles.

We believe that MAPFOU opens the path for many research questions. Future research can focus on handling problems with communication costs, on ignoring irrelevant obstacles in centralized execution, on handling other types of conflicts, on remote sensing, and much more.

We consider a multi-level jury problem in which experts We consider a multi-level jury problem in which

experts We consider a multi-level jury problem in which
experts We consider a multi-level jury problem in which
experts

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.

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), volume 15, 299–301.

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.

Muise, C.; Belle, V.; and McIlraith, S. 2014. Computing contingent plans via fully observable non-deterministic planning. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 28.

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 Proceedings of the AAAI Conference on Artificial Intelligence, volume 24, 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.

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.