Centralized Stochastic Multi-Agent Pathfinding under Partial Observability

Anonymous submission

Abstract

Multi-Agent Path Finding (MAPF) is the problem of finding paths for multiple agents where each agent aims to reach a given goal location without conflicting with the other agents. In MAPF applications with physical robots, we can expect the agents to have stochastic behavior and imperfect localization. Planning for such centrally-controlled agents can be viewed as a special case of Partially Observable Markov Decision Process (POMDP), but off-the-shelf POMDP solvers cannot scale to plan for even a very small number of agents, due to the exponentially large size of the state and action spaces. Instead, we propose the Online Prioritized Planning (OPP) approach, where each agent computes and follows its individually-optimal policy until a potential conflict is detected. OPP resolves detected potential conflicts by replanning online for a subset of the agents so as to avoid positions that are potentially occupied by other agents. We describe how OPP can be implemented and propose two extensions that encourage the agents to leverage localization actions when needed. We evaluate OPP and its extensions empirically to highlight the pros and cons of our approach and show it can scale better than an offline baseline.

1 Introduction

In classical multi agent path finding (MAPF) agents navigate in an environment represented as a graph. Each agent must move from its current position to a given goal position without conflicting with other agents (**?**). Real-world MAPF applications include automated logistic centers (**??**), airport towing (**?**), autonomous vehicles, robotics (**?**), and digital entertainment (**?**). However, MAPF research has mainly focused on deterministic and fully observable domains, i.e., where the effects of each action are known and deterministic and agents can fully observe their environment. Both assumptions, deterministic effects and full observability, are often violated in real-world problems, especially when the agents are physical robots. This work addresses MAPF problems where both types of uncertainty exist, which we refer to as *Stochastic MAPF with Partial Observability* (SMAPF-PO). Classical MAPF solvers cannot solve SMAPF-PO problems without compromising agents' safety.

We consider SMAPF-PO problems in which the agents are controlled in a centralized manner, e.g., a warehouse management system. Such SMAPF-PO problem can be reduced to a single-agent Partially Observable Markov Decision Process (POMDP) over states and actions that represent the agents' joint positions and actions. Solving this exponentially large POMDP is only practical for extremely small problems, even with modern online POMDP solvers. Even Dec-POMDP solvers, which are designed for solving multi-agent problems (**?**) can scale to solve problems with approximately 180K states, while SMAPF-PO with only 4 agents on a 20x10 grid reaches 1G states, 625 actions, and 2K observations. Thus, off-the-shelf POMDP and Dec-POMDP solvers only handle tiny SMAPF-PO problems.

Inspired by *Prioritized Planning* (PP) (**?**) for classical MAPF, we propose the *Online Prioritized Planning (OPP)* approach for solving SMAPF-PO problems. OPP starts by computing a single-agent policy for each agent independently, ignoring all other agents. The agents execute these policies until the belief over the agents' current location indicates a conflict may occur. At this stage, OPP attempts to resolve potential conflicts by replacing the individually-optimal policy of some agents with policies that avoid the conflict. This planning is done online, to avoid the need to plan a-priori for every contingency. Finally, OPP employs a dedicated mechanism to encourage agents to optimize their policy over time. We discuss several design choices and challenges when implementing OPP, and propose two extensions designed to encourage the agents to intelligently perform localization action when needed.

Our main contribution is a decoupling method that allows us to avoid directly solving the joint problem. The proposed method, OPP, allow us to scale to problems many orders of magnitude larger than a standard POMDP solver can handle over the joint problem. A secondary contribution is a set of problem-specific modifications for the underlying POMDP solver, that provide additional leverage in the navigation and localization task that we define. Experimental results over 12 SMAPF-PO problems show that, as expected, POMDP solvers over the joint problem cannot scale beyond tiny grids with two agents in reasonable time. OPP with our extensions, however, can solve non-trivial problems with up to 4 agents.

2 Background

MAPF. A classical MAPF problem is defined by a tuple $\langle k, G, \{s_1, \ldots, s_k\}, \{g_1, \ldots, g_k\}\rangle$, where k is the number of agents, $G = (V, E)$ is an undirected graph where V repre-

sents positions agents may occupy over time; $s_i, g_i \in V$ are the initial and target position of agent i, respectively. Time is discretized into time steps. At every time step, each agent either stays in its current position or moves to a neighboring one along an edge. A single-agent plan π_i for agent i in a MAPF problem is a path in G that starts in s_i and ends in g_i.

A *solution* to a classical MAPF problem is a set of single-agent plans $\Pi = \pi_1, \ldots, \pi_k$ that do not *conflict*. We consider two types of conflicts, *vertex* conflicts ad *swapping* conflicts, which occur in plans when agents occupy the same location or swap locations over the same edge at the same time step. The cost of a MAPF solution Π is the sum of steps in the single-agent plans. Prioritized Planning (PP) (**??**) is an extremely popular framework for solving MAPF problems suboptimally. In PP, each agent is assigned a priority such that a total order is formed over the agents based on these priorities. Then a single-agent pathfinding algorithm such as A* (**?**) is used to find a path for each agent, one at a time in order of the agents' priorities. Importantly, when finding a path for an agent PP avoids conflicts with the paths found for all higher priority agents. While PP is neither complete nor optimal, it is very efficient computationally, and often returns surprisingly high quality solutions. PP is also the industry standard in MAPF applications such as autonomous warehouses, and has many sophisticated extensions (**???**).

POMDP. A POMDP is a tuple $\langle S, A, Tr, R, G, \Omega, O \rangle$ where S is a set of states; A is a set of actions; $Tr : S \times A \times S \rightarrow [0, 1]$ is the transition function, returning the probability that executing the action $a \in A$ from state $s \in S$ will lead to state $s' \in S$; $R : S \times A \rightarrow \mathbb{R}$ is the reward for taking action $a \in A$ from state $s \in S$; $G \in S$ is a set of terminal states; Ω is a set of observations; $O : S \times A \times \Omega \rightarrow [0, 1]$ is the observation function, returning the probability of observing $o \in \Omega$ when performing $a \in A$ in state $s \in S$. We focus here on finite state, action, and observation spaces. POMDPs solvers often maintain a *belief state* b, where $b(s)$ is the probability that the current state is s. Offline POMDP solvers such as (FSVI) (**?**) and others (**???**) compute a policy that maps belief states to actions, and use it during execution. Online POMDP solvers (**???**) such as POMCP (**?**) perform little or no computation before execution and compute during the execution which action to execute next. In our experiments we used FSVI and POMCP, which are popular examples of offline and online POMDP solvers. FSVI is highly appropriate for environments where sensing actions can be executed en route to the goal, and is thus highly appropriate for our application. However, replacing FSVI in our approach with any other point-based algorithm is trivial.

3 Problem Definition

SMAPF-PO is a MAPF variant: there are k agents moving along the edges of an undirected graph $G = (V, E)$, starting in a set of vertices $s_1, \ldots, s_k$ and aiming to reach a set of goal positions $g_1, \ldots, g_k$. A subset of the vertices $BE \subset V$ contains *beacons*, which can be sensed remotely by the agents. We assume a distance function $d : V \times BE :\rightarrow \mathbb{R}$ that quantifies the distance between vertices and beacons, such that nearby beacons can be sensed more reliably. Time is discretized, and at every time step each agent performs either a *wait action*, a *move action*, a *ping action*, or a *declare action*. We define these actions below. At every time step all agents act concurrently. The vector of all single-agent actions is called a *joint action*.

Wait and move actions: As in a regular MAPF problem, agents can move to nearby vertices, or stay in the same vertex. A wait action results in the agent staying in its place. While movement actions in classic MAPF applications are from vertex to vertex, under partial observability, we are often unsure where the agent is. Hence, we assume a set of abstract move actions $a_1, ..., a_M$, that apply to all vertices. For example, in grids such abstract move actions can be moving up, down, left, and right. Move actions may have stochastic effects, which are defined as follows. For each action a_m and vertices $v_i, v_j, Tr(v_i, a_m, v_j)$ denotes the probability that the agent reaches v_j after applying a_m when at v_i.

Conflicts are strictly forbidden, that is, an agent cannot perform a move action if there is a non-zero probability that it will conflict with another agent. More formally, let b be a belief over the current locations of agents, where $b_{i,j}$ is the probability that agent i is at vertex j. An action a_m is forbidden if there exist two agents i, j, and nodes v_k, v_n, v_l s.t. $b_{i,k} > 0, b_{j,n} > 0, Tr(v_k, a_m, v_l) > 0, Tr(v_n, a_m, v_l) > 0$. That is, if both agents may currently be at vertices v_k, v_n, where executing a_m may bring them to a node v_l.

Ping and declare actions: After performing move actions, there is often uncertainty about the current location of agents. The ping action provides some information over that uncertainty by sending a signal to a beacon, receiving back a noisy estimation of its distance from the beacon. More formally, a *ping action* is defined by a pair (v, be) and corresponds to sending a signal to beacon $be \in BE$ when the agent is at position $v \in V$. During ping actions the agent always stays in place. In addition, it receives an observation $o \in \mathbb{R}$ that is related to the *distance* between v and be. The observation function $O_i(v, be, o)$, specifies the probability that agent i receives observation o when performing a ping action (v, be).

Even with exact ping actions, let alone with noisy observations, there is still uncertainty over the agents' current location, and thus uncertainty over when all agents have reached their goals. A *declare action* is a special action the agent must perform to declare that it is sufficiently confident that it has reached its goal. We assume that the agent cannot perform any other action after performing a declare action and it is removed from the graph, similar to Švancara et al. (**?**). Other assumptions are also possible.

Objective: navigate all agents to their goal vertices without conflicts, performing the least actions possible, followed by each agent performing a declare action. This complex objective is captured by a reward function $R_i(v, a)$ where i is an agent, v is its position, and a is the action it performed. For a declare action, if $v = g_i$ then $R(v, a)$ is a large positive number, corresponding to a success, and large negative number if $v \neq g_i$. For all other actions $R_i(v, a)$ is some small negative number, corresponding to a cost for each step.

Definition 1 (SMAPF-PO) *A SMAPF-PO problem is a tuple* $\langle \Pi, BE, \{Tr_i\}_i, \{O_i\}_i, \{R_i\}_i \rangle$ *where* Π *is a classical MAPF problem, and* Tr_i*,* O_i*, and* R_i *are the transition, ob-*

servation, and reward function of agent i. The objective of a SMAPF-PO solver is to output a joint action in every time step such that the expected sum of discounted reward collected by all agents is maximized, for a given discount factor.

3.1 Grid-based SMAPF-PO

Next, we describe the specific type of SMAPF-PO problem we used in our experiments, where the graph G represents a 4-neighborhood grid where agents' move actions are only up, down, left, and right. We emphasize that the theoretical contributions of this work are not limited to this specific SMAPF-PO problem. For every agent i, we define $Tr(v, a, v') = 0.8$ if $v' = v_a$ and 0.1 for v', the grid cells that are clockwise and counter-clockwise from the direction of v_a from v. For example, if the agent is currently at grid cell x, y and it executes the down action, then it arrives at cell $x, y + 1$ with probability 0.8, and moves to cells $x - 1, y$ and $x + 1, y$ with probability 0.1. Some grid cells may be blocked, in which case moving towards them results in staying in place. After an agent reaches its goal position, it must execute a declare action. The agent disappears from the grid after the declare action was used, receiving a positive reward if it is indeed at the goal, and a smaller penalty cost if it is not at the goal.

Observations are noisy, providing inexact biased estimates of the Manhattan distance, denoted $d(v, be)$, between the agent's current location v and the chosen beacon be. In particular, the observation never underestimates the Manhattan distance to the beacon. Beacons have a finite (very limited) range. When the agent is too far from the beacon, it observes an infinite distance. If an agent pings a beacon be and receives an observation value of zero, it can deduce that it is exactly at be. This is natural if signal strength weakens as the receiver gets far from the beacon.

More formally, each beacon $be \in BE$ has a maximal range $be.r > 0$. If an agent performs a ping action (v, be), and $d(v, be) > be.r$, it will observe ∞, i.e., $O_i(v, be, \infty) = 1$. Otherwise, the observation is between $d(v, be)$ and $be.r$, with the following observation function:

$$O(v, be, o) = \begin{cases} \frac{2^{be.r-o+d(v,be)}}{2^{be.r+1}-2^{d(v,be)}} & o \in \{d(v, be), be.r\} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Following preliminary experiments, we defined the reward $R_i(v, a)$ to be 50 when an agent declares at its goal, -20 if it declared not at its goal, and -0.04 if it did not declare.

3.2 Reduction to Single Agent POMDP

As the agents in SMAPF-PO are controlled in a centralized manner, any SMAPF-PO problem $\langle \Pi, BE, \{Tr_i\}_i, \{O_i\}_i, \{R_i\}_i \rangle$ can be reduced to a POMDP problem $\langle S, A, Tr, R, G, \Omega, O \rangle$, as follows. The set of states S in this POMDP is the cross product of all single-agent positions $\times_{i=1}^{k} V$ and a special terminal state s_f. The set of actions A is the set of all joint actions. The transition function Tr is the product of the respective single-agent transition functions Tr_i except that all outcomes in which a conflict has occurred reach the special terminal state s_f. The reward function R is the sum of the corresponding reward functions R_i, and a large negative value for reaching s_f due to a conflict. The set of terminal states G includes s_f and any state reached after all agents have used a declare action. The observations Ω is $\mathbb{R}^k$, corresponding to the joint observation of all agents, and the observation function O is the product of the respective single-agent observation functions O_i.

This reduction allows solving SMAPF-PO problem using off-the-shelf POMDP solvers such as FSVI (**?**). Unfortunately, current POMDP solvers cannot solve this POMDP in reasonable time beyond very small problems, as the size of the state space, action space, and observation space are all exponential in the number of agents. Recall that solving POMDPs is in general undecidable (**?**).

Algorithm 1: OPP(Π, b_0)

```
1:  b ← b_0
2:  for agent r do
3:      π_r ← Compute individually optimal policy for r
4:  end for
5:  π ← {π_1, ..., π_k}
6:  while Not IsDone(b) do
7:      InConflict ← DetectPotentialConflicts(π, b)
8:      if InConflict ≠ ∅ then
9:          if ResolvePotentialConflicts(InConflict)==False then
10:             Fail
11:         end if
12:     end if
13:     {ω_1, ..., ω_k} ← Step(π, b)
14:     b ← UpdateBelief(b, {ω_1, ..., ω_k})
15:     ForgetConflicts(π, b)
16: end while
```

4 Online Prioritized Planning

To enable scaling up to larger problems, we propose the *Online Prioritized Planning* (OPP) approach for solving SMAPF-PO problems. As its name suggests, OPP is an online approach that is inspired by the Prioritized Planning (PP) popular MAPF framework (**?**). It is *online* in the sense that it does not generate a policy for every possible contingency, but interleaves execution and planning. It is based on PP as it constructs a multi-agent solution by computing only single-agent plans and ensuring they do not conflict. When resolving potential conflicts, we take a prioritized approach, where lower priority agents yield to higher priority agents. We now detail OPP along with several extensions.

OPP starts by generating for each agent i an *individually-optimal policy* π_i, which is an optimal policy for moving i to its goal while ignoring all the other agents. The agents then execute these policies until a *potential conflict* is detected. A potential conflict is a future event that may occur according to the current belief and agents' policies in which a conflict — either vertex or swapping — occurs. At this stage, OPP employs a *potential conflict resolution* mechanism which updates the policy of at least one agent to resolve the conflicts. The agents continue to execute their (revised) policies until either a new potential conflict has been detected, all agents have declared a goal, or sufficient steps have been performed without any agent detecting potential conflicts. The latter condition attempts to capture the case where the agents are

no longer in danger of collision, and can return to moving optimally towards their goals. When this occurs, OPP recomputes the individually-optimal policy for the agents that have diverted from their individually-optimal policies to avoid a potential conflict, starting from the current belief.

Algorithm 1 provides the pseudo-code for OPP. The statements "Step(π, b)" and "UpdateBelief(b, $\{\omega_1, \ldots, \omega_k\}$)" correspond to performing the joint action specified for the belief state b in policy π, and updating the belief b based on the observations collected after performing the previous joint action. Observe that as the transition and observation functions are factored and the agents share observations, the belief state b is naturally factored to a set of belief states $\{b_i\}_i$ where b_i is a distribution over the possible position agent i may currently occupy. Thus, creating the joint action from b (line 13) involves collecting the actions of the different agents by using their single-agent policies over their respective single-agent beliefs. Note that OPP is incomplete: it may detect a conflict that it cannot resolve, in which case it fails (line 10).

4.1 OPP Components

OPP includes several key steps, described below,: (1) generating individually-optimal policies, (2) detecting potential conflicts, (3) resolving potential conflicts, and (4) deciding when to recompute the individually-optimal policies.

Generating an Individually Optimal Policy: To generate the individually optimal policy for agent i, we solve the POMDP Π_i whose state space is V (the set of positions an agent may occupy), the actions are all agent i's actions, and the transition function, observations, observation function, and reward function are Tr_i, $\mathbb{R}$, O_i, and R_i, respectively. This POMDP is exponentially smaller than the POMDP described above, and thus solving it is more practical. In our implementation, we used FSVI (**?**), a popular POMDP solver. We run FSVI until it either converged or a memory and runtime limit has been reached. As FSVI does not provide optimality guarantees, we are not guaranteed find an individually-optimal policy.

Detecting Potential Conflicts: Detecting potential conflicts that may occur in the future requires tracking for all future positions that every agent may occupy, which is prohibitively expensive to compute (**?**). Moreover, in an online setting, the agent can limit its attention to conflicts that are possible in the particular execution it is running. Thus, we detect only potential conflicts that may occur in the next d steps, where d is a parameter. This is performed by expanding each agent's belief using the actions dictated by its current policy, considering only sequences of d actions. If there is some vertex v that appears in the expanded beliefs of two agents, then a potential conflict has been detected. If we identify a possible conflict between agents i and j, and another potential conflict between agents j and k, then we assume a potential conflict between i, j, k.

Resolving Potential Conflicts: To resolve one or more potential conflicts between a pair of agents i and j, one agent yields to the other agent. That is, OPP updates the policy of one agent to avoid all possible positions the other agent may occupy when following its current policy. Let F_i and F_j be these sets of possible future positions for agent i and j, respectively (computed in the previous step). We say that a policy for agent i is *safe* with respect to agent j if it avoids all positions in F_j. To find a safe policy for agent i wrt agent j, we solve a POMDP similar to the one described above for computing the individually-optimal policy for i, except that it forbids every action that may result in a non-zero belief in which i is occupying a state in F_j. Note that a safe policy for i may not be able to reach the goal. After the conflict is no longer possible, agent i may continue moving towards the goal. If OPP fails to find a safe policy for i wrt to j, it attempts to find a safe policy for j wrt to i. If OPP fails to find any safe policy, both agents change their policies to stay in their current positions. If there are potential conflicts that involve more than two agents, we repeat the above process but search for a policy that is safe with respect to all other agents. We continue this until all conflicts are resolved or we have failed to find a safe policy for an agent in conflict.

Algorithm 2: Resolve(Π, *InConflict*)

```
1: R_try ← ∅
2: while InConflict is not empty and InConflict \ R_try ≠ ∅ do
3:     r ← ChooseAgent(InConflict, R_try)
4:     Add r to R_try
5:     π_r ← FindSafePolicy(r, Π^{-r})
6:     if π_r is safe wrt all other agents then
7:         Update Π with π_r
8:         Remove r from InConflict
9:     end if
10: end while
11: return InConflict = ∅
```

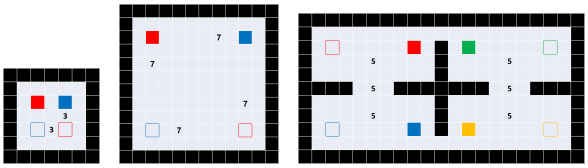
Algorithm 2 lists the pseudo code for the potential conflict resolution method described above. The input is the set of agents that were detected in a potential conflict, denoted *InConflict*. As long as *InConflict* is not empty and we have not attempted to update the policies of all the agents, then one of the conflicting agents is chosen. We search for a safe policy for that agent with respect to the current policies of all other agents (denoted by Π^{-r} in Algorithm 2). If a safe policy has been found, we update the current set of individual policies and remove r from *InConflict*. Otherwise, we iterate and choose a different agent to attempt to find a safe policy. If all conflicts have been resolved, or we have tried to update the policy of all agents, the algorithm stops ,reporting whether all conflicts have been resolved.

The above approach to resolve conflicts is but one design choice. An alternative way to resolve potential conflicts includes prioritizing the agents, and limiting the set of forbidden states when planning for each agent to the states potentially occupied by higher-priority agents. In addition, we can repeat the process of searching for a safe policy multiple times for each agent, as the set of potential conflicts may have been changed when trying to resolve some of the conflicts. We have found our implementation to be effective, but future research can explore other options.

Concluding Resolved Conflicts and Replanning: The safe policies created when resolving a conflict may be significantly worse than the corresponding individually optimal policies. Moreover, they are overly pessimistic, in the sense

Grid	Size	Cells	k	Joint $\|S\|$	Joint $\|A\|$	Joint $\|O\|$	Decoupled $\|S\|$	Decoupled $\|A\|$	Decoupled $\|O\|$
S_1	7x5	33	2	1056	49	16	33	7	4
S_2	5x5	23	2	506	49	25	23	7	5
S_3	5x5	25	2	600	49	25	25	7	5
S_4	5x5	25	2	600	49	25	25	7	5
M_1	10x10	94	2	8.7×10^3	49	25	94	7	5 d
M_2	10x10	94	2	8.7×10^3	64	49	94	8	7
M_3	10x10	100	2	9.9×10^3	81	81	100	9	9
M_4	10x19	190	2	35×10^3	121	81	190	11	9
M_5	14x11	118	2	13×10^3	81	36	118	9	6
L_1	15x10	138	3	2×10^6	1,331	343	138	11	7
L_2	10x10	100	4	94×10^6	6,561	2,401	100	9	7
L_3	9x19	152	4	512×10^6	14,641	2,401	152	11	7

Table 1: SMAPF-PO benchmark properties, the joint and the decoupled POMDP model parameters.



(a) S_3 (b) M_3 (c) L_3

Figure 1: Example grids. Each color represents an agent. Initial cells filled, goal cells bordered.

that they forbid reaching any position the other agents may occupy in the next d steps. But, after performing some steps, the set of possible positions the other agents occupy may change, based on the current location of all agents. Thus, it can be beneficial to replan for the agents following a safe policy after every step. On the other hand, replanning requires solving a POMDP and is thus computationally expensive.

As a middle ground, we implemented in OPP the following mechanism. After agent i updates its individually optimal policy to a safe policy in order to resolve a conflict, it executes that policy for T steps. If during those T steps another conflict is detected, then a new safe policy is computed and the step counter is restarted. If no conflicts were detected in T time steps, agent i recomputes an individual policy.

Extensions: We implemented two extensions to the baseline OPP described above. The first extension encourages the agents to perform ping actions instead of declaring failure when a conflict could not be resolved (line 10 in Algorithm 1. We denote this by *Forced Localization* (FL). The second extension is to modify the way the individually optimal policies are computed to explicitly include ping actions, addressing the following known deficit of FSVI. FSVI guides its sampling of the belief space — forward traversals — by solving the underlying MDP. Thus, it is less likely to perform sensing actions, which are ping actions in SMAPF-PO, during its forward traversal, since sensing actions do not provide value in the underlying MDP. Thus, it can be useful to explicitly consider sensing actions during forward traversals. We implemented this concept as follows. A belief is called *movable* if there is a moving action that may not end in a forbidden state. If in the belief collection phase of FSVI the current belief is not movable, the agent will execute every ping action possible and collect the resulted beliefs, thus explicitly considering the sensing actions when needed. We call this OPP extension the *Modified FSVI* (MF).

5 Experimental Results

We implemented OPP and conducted an experimental evaluation of its behavior on grid SMAPF-PO problems.

Benchmarks: Solving SMAPF-PO is much more difficult compared to classical MAPF, and none of our algorithms were able to scale to solve SMAPF-PO problems on standard MAPF benchmarks (**?**). Instead, we create a suite of 12 SMAPF-PO problems of different sizes, configurations, and number of agents. Figure 1 illustrates some of these problems visually and Table 1 provides additional information. The supplementary material contains a visualization of all problems in our benchmark.

Agents' start and goal positions are filled and bordered cells, respectively, where each agent has a different color. Cells with beacons are marked with a number that specifies the beacon's maximal range. Our benchmark selection is designed to examine different interactions between the agents and different difficulty levels. Most of these benchmarks are very difficult given noisy partial observability — there are no beacons at the goal positions allowing the agent to know it has arrived at the goal. Goal positions are also not located in places that are easy to reach blindly, such as corners. As such, achieving near perfect success is highly unlikely.

As can be seen in Table 1, the joint problem rapidly becomes unmanageable, due to the huge joint state space, but also due to the joint action and observation spaces. Each additional beacon adds another action, and the number of observations depends on the range of the beacons.

Baselines: An experimental comparison between OPP and classical MAPF algorithms is not possible, as the latter assume only one (deterministic) outcome for each action and perfect localization. The main contribution of our approach is the decoupling method, that allows use to solve smaller single agent problems, rather than tackle the complete joint problem. We hence compare methods that solve the joint problem to our decoupling approach. For the joint problem, we used both an offline solver, FSVI (**?**), and an online solver, POMCP (**?**), with a domain specific heuristic that moves towards the goal. We refer to the joint baselines as *Joint-FSVI, and* Joint-POMDP.

For our decoupled approach, we also use both FSVI and POMCP to solve the decoupled single agent problems. We denote the OPP POMCP version, which we ran with our forced localization method, as *POMCP+FL*. We also experiment with several variations of OPP with FSVI: (1) a vanilla method without forcing localizations and without our modified version of FSVI, denoted *FSVI* (2) OPP with the modified FSVI, denoted *FSVI+MF*, (3) OPP with regular FSVI and forced localization, denoted *FSVI+FL*, and (4) OPP with the modified FSVI and forced locatization, denoted *FSVI+MF+FL*. We use FSVI as preliminary experiments with other point-based approaches, namely, Persues and PBVI, did not work as well. That being said, any other

offline or online solver can also be used.

The implementation is in Java. Experiments were run on an AMD® EPYC 7702P 64-Core with 32 GB of RAM.

Experiment Design and Metrics: For every problem in our benchmark we run each of the evaluated algorithms 50 times. Every such run was terminated when either all agents have declared that their goal has been reached or when the overall number of steps exceeded 200. In the latter case, we forced all remaining agents to perform a declare action.

The main performance metric we considered is the average sum of discounted rewards (ADR), which is a common solution quality measure in the POMDP literature. We also report the standard error of these averages, the average runtime in seconds, and the *success rate*, which is the percentage of runs where all agents reach their respective goal states and performed the declare action appropriately. ADR is not directly correlated with the success rate: when one agent reaches the goal and the other does not, declaring done will produce a positive ADR will not be considered a success. This can be seen in our results, where some cases with 0% success rate still have positive ADR.

P	Alg	ADR	RT	% S	#C	#R
S_1	Joint-FSVI	80 ± 4	671	85%	-	-
	Joint-POMCP	56±6	78	48%	-	-
	POMCP+FL	55 ± 6	3	68%	0.04±0.03	0.04±0.03
	FSVI	80 ± 4	3	88%	0.5±0.1	0.5±0.2
	FSVI+MF	76 ± 4	2	80%	0.3±0.1	0.4±0.1
	FSVI+FL	$\mathbf{79 \pm 4}$	3	88%	0.5±0.1	0.5±0.2
	FSVI+MF+FL	74 ± 5	5	76%	2±2	4±4
S_2	Joint-FSVI	62 ± 6	692	72%	-	-
	Joint-POMCP					
	POMCP+FL	63 ± 4	106	68%	130±28	131±28
	FSVI	61 ± 5	11	64%	5±0.8	7±1
	FSVI+MF	49 ± 5	23	40%	6±3	10±4
	FSVI+FL	64 ± 6	41	80%	27±5	10±11
	FSVI+MF+FL	48 ± 7	50	62%	41±6	80±13
S_3	Joint-FSVI	3 ± 4	3516	10%	-	-
	Joint-POMCP					
	POMCP+FL	0 ± 4	331	12%	90±6	170±10
	FSVI	21 ± 1	17	0%	2.7±0.1	4.3±0.2
	FSVI+MF	17 ± 3	21	0%	2.4±0.1	3.9±0.2
	FSVI+FL	11 ± 5	68	18%	66±7	128±13
	FSVI+MF+FL	3 ± 4	93	10%	70±6	138±13
S_4	Joint-FSVI	57 ± 6	4735	63%	-	-
	Joint-POMCP					
	FSVI	22 ± 1	9	0%	2±0	3±0
	POMCP+FL	-12.2 ± 0	422	0%	99.7±0.06	199±0.1
	FSVI+MF	22 ± 0.6	5	0%	2±0	3±0
	FSVI+FL	21 ± 5	57	36%	58±6	112±13
	FSVI+MF+FL	-12 ± 0	203	0%	100±0	200±0

Table 2: Results for small problems (S_1, S_2, S_3, and S_4).

Results The results for small problems are shown in Table 2 . Columns "P", "RT", and "%S" specify the problem name, runtime in seconds, and the success rate, respectively. We also report the number of potential collisions detected and the number of times an agent replanned, denoted as "#C" and "#R", respectively. $\#C$ and $\#R$ are always zero for *Offline*, since its initial policy is conflict-free.

As expected, Joint-FSVI could only solve in reasonable time the smaller 5x5 problems, S_1, S_2, S_3, and S_4. On the other hand, Joint-FSVI managed in some problems to provide the best policy in terms of success rate and ADR. This is expected, as an optimal POMDP solver on the joint problem, given enough runtime, can return an optimal policy. Joint-POMCP could not scale beyond the smallest problem, due to the large number of joint actions and observations.

The online algorithms perform well on S_1 and S_2, where the agents' shortest path to their goals do not conflict. However, they perform poorly in S_3 and S_4, which represent very difficult problems, where agents must move close together in a tight space. As such, the number of conflicts between the shortest paths is very high, many potential conflicts will be detected and the agents will replan very often. This is clearly observable in the "#C" and "#R" results, which are high and are inversely correlated with the ADR.

Another observation from these results is that when many conflicts are expected (S_3 and S_4), the modified FSVI technique is not beneficial, but the forced localization works better than the vanilla FSVI. This is because in such small grids localizing is very helpful, allowing agents to move at least one step without colliding, and then localizing again. Looking at the run time of the different algorithms, as expected, computing the joint policy is much more computationally demanding, requiring many orders of magnitude additional run time compared to the online algorithms. Indeed, the offline algorithm could not scale to any of the larger problems.

The POMCP+FL is slower than all other online methods. This is because even the single agent problems have a relatively large number of actions (movements + pinging beacons), and many observations. Thus, the branching factor is difficult to handle. On the other hand, POMCP+FL creates in some cases good policies, and in S_2, even the best policies.

Results on large problems Table 3 lists the results for the larger problems (M_1, M_2, M_3, M_4, M_5, L_1, L_2, and L_3), where the joint problem is too difficult for FSVI and POMCP. In these problems, the benefit of the forced localization (FL) method are obvious. For example, without FL and MF the success rate of the vanilla FSVI was 0 for M_3, M_4, M_5, and L_2, while FSVI+FL and FSVI+MF+FL often succeed. This advantage is also shown in the ADR. For example, in L_1 using FL approximately doubled the ADR achieved.

The benefit of modifying FSVI (MF) to include more pings, however, is less pronounced. Only in M_1 FSVI+MF provides better results than FSVI+FLm while in most cases, using only FSVI+MF in not as good as FSVI+FL. On the other hand, the combined FSVI+FL+MF provides additional leverage, at least in the largest problem, L_3.

POMCP-FL provided competetive results in some cases, but in most problems did not work as well as FSVI-FL. On the largest problems, POMCP-FL failed to provide strong policies, probably because the longer needed planning horizon requires deeper trees, which take a long time to construct.

In some sense, M_5 is the hardest of the large problems, as the single-agent path of each agent to its goal is conflicting in M_5, requiring much coordination for navigating the narrow corridors without collisions. Additional research is needed to develop methods that can better handle such problems that require strong synchronization. It may be that in such corridors, solving a small, short range joint problem is better.

Summary For very small problems, finding a solution to the joint problem provides the best policy, but takes a very long time. As such, only OPP was able to scale beyond very small problems, especially when using the FL method. This demonstrates that solving the naive joint MAPF POMDP problem is not practical, and hence, one would need to use some

type of decoupling technique, creating single agent problems that allow for much better scaling up. We also see here that domain specific techniques can help POMDP solvers in creating better policies, with respect to the synchronization of policies. Our localization additions were successful because, when considering a single agent, localization is less important, unless we are near the goal. However, when other agents are nearby, localization can reduce significantly the possibility for collisions, and hence, the need to replan, and the constraints during replanning.

6 Related Work

MAPF with unexpected delays has been studied in different forms (**????**). Yet limiting stochastic effects to delays simplifies the problem compared to SMAPF-PO, and these works all assumed perfect observability. UM^* (**?**) does support uncertainty over agents' locations but it allows agents to risk conflicts as long as their probability of occurring is beneath a given threshold. Similarly, prior work on multi-robot navigation assigned cost to collision and aim to minimize it. The SMAPF-PO is fundamentally different since we require avoiding any chance for having a conflict. Recent work (**?**) on MAPF with partial observability differs from SMAPF-PO in that they assumed control is not centralized and agents' visibility is based on line of sight. The latter assumption is different from our beacons-based observation model, precluding empirical comparison. MAPF under movement uncertainty is a special case of Multi-Agent MDP (MMDP) (**?**). General-purpose MMDP solvers (**?**) do not exploit the specific structure of the SMAPF-PO problem, such as the limited interactions between agents. Similarly, online approaches for solving large MDPs, such as FF-Replan(**?**), are not expected to be effective in our domain and do not deal with partial observability. Dec-POMDP (**?**) is often used to model multi-agent problems under partial observability, where agents plan jointly, but execute their policies independently. This is different from our setting, where agents are controlled in a centralized manner and share their observations. Also, Dec-POMDP algorithms tend to scale poorly unless the interactions between the agents are limited to small predefined areas in the state space (**?**). Multiagent POMDP (**?**), where both planning and control are centralized, is an appropriate model for SMAPF-PO. **?** suggest an algorithm based on POMCP which exploits the locality of agent interactions. Their algorithm given singleton sets is similar to our POMCP, which does not scale as well as FSVI. Our approach can be viewed as a particular implementation of FT-POMCP leveraging the structure of the SMAPF-PO problem for factorization, and using prioritized planning to ensure independence. Factorizing the decision-making process in POMDPs has long roots (**??**, e.g.), and is particularly useful given several independent tasks. Factorization is less effective, however, when the coupling between the tasks increases, as happens in our problem where agents may collide. In tasks with high uncertainty concerning the current state, such coupling is common, and useful factoring is difficult. We used FSVI and POMCP but other alternatives for offline and online POMDP solvers exist (**???**, e.g.). Our contribution is not in an adaptation of a particular POMDP solver to SMAPF-PO, but in our factoring and prioritization schemes. Replacing POMCP with, e.g., ABT, is likely to scale up only slightly, while improvements to the factorization can make a huge difference.

P	Alg	ADR	RT	% S	#C	#R
M_1	FSVI	69 ± 4	21	74%	0.3±0.1	0.4±0.2
	POMCP+FL	52 ± 5	33	62%	15±12	16±13
	FSVI+MF	70 ± 4	19	78%	0.4±0.2	0.6±0.2
	FSVI+FL	61 ± 5	22	66%	0.2±0.1	0.3±0.2
	FSVI+MF+FL	70 ± 4	18	78%	0.4±0.2	0.6±0.2
M_2	FSVI	41 ± 5	146	46%	4±0.4	7±0.6
	POMCP+FL	28 ± 5	946	46%	300±20	300±20
	FSVI+MF	28 ± 4	230	30%	5±0.3	7±0.4
	FSVI+FL	48 ± 5	191	64%	6±0.5	9±0.6
	FSVI+MF+FL	55 ± 4	200	76%	5±0.4	8±0.5
M_3	FSVI	17.8 ± 0.4	253	0%	1±0	2±0
	POMCP+FL	57 ± 3	202	84%	79±15	83±15
	FSVI+MF	17.6 ± 0.3	418	0%	1±0	2±0
	FSVI+FL	56 ± 3	402	92%	10±1	16±3
	FSVI+MF+FL	48 ± 4	535	72%	10±2	18±3
M_4	FSVI	30 ± 3	133	10%	1±0	2±0
	POMCP+FL	73 ± 2	208	94%	63±16	63±16
	FSVI+MF	36 ± 4	91	20%	1±0	2±0
	FSVI+FL	77 ± 7	245	96 %	2±0.3	4±0.4
	FSVI+MF+FL	75 ± 3	125	94%	3±0.7	5±1
M_5	FSVI	-6 ± 3	381	0%	2.1±0.2	3.4±0.2
	POMCP+FL	-11.6 ± 0.6	275	2%	55±5	101±3
	FSVI+MF	5 ± 2	217	0%	2.0±0.1	3.3±0.2
	FSVI+FL	-6 ± 2	896	8%	29±3	55±6
	FSVI+MF+FL	0 ± 2	671	2%	20±3	37±5
L_1	FSVI	37 ± 6	254	8%	7.2±0.5	10.9±0.8
	POMCP+FL	64 ± 5	1954	64%	372±21	376±21
	FSVI+MF	29 ± 6	433	6%	6.8±0.5	10.5±0.6
	FSVI+FL	97 ± 3	222	78%	8±0.5	12±0.7
	FSVI+MF+FL	97 ± 4	386	74%	10±3	15±4
L_2	FSVI	-11 ± 2	80	0%	2.96±0.02	8.8±0.1
	POMCP+FL	69 ± 12	1950	26%	-	-
	FSVI+MF	-15.1 ± 0.3	57	0%	3±0	8.9±0.1
	FSVI+FL	67 ± 12	182	36%	25±3	50±6
	FSVI+MF+FL	73 ± 11	212	30%	20±3	40±6
L_3	FSVI	67 ± 6	202	8%	8±0.6	12±0.8
	POMCP+FL	35 ± 7	1663	12%	328±23	326±22
	FSVI+MF	59 ± 7	353	8%	9±1	13±1
	FSVI+FL	112 ± 5	183	62%	10±1	14±1
	FSVI+MF+FL	117 ± 6	423	68%	12±1	18±2

Table 3: Results for medium and large problems.

7 Conclusion

We studied the Stochastic MAPF with Partial Observability (SMAPF-PO) problem, which is a generalization of MAPF in which actions have stochastic outcomes and agents do not have perfect observability of the current state. We focused on a centralized control setup. While SMAPF-PO can be modeled as a single-agent POMDP problem, solving this POMDP is intractable even for small problems. We introduced the OPP approach, an online adaptation of Prioritized Planning. OPP has several non-trivial components which we describe, and we also propose two extensions that encourage the agents to actively localize. The results showed that OPP solves larger problems than an offline baseline. Yet, solving larger problems is still an open challenge for future work. Possimus placeat accusamus corporis officia temporibus, harum consequatur quidem.A pariatur quo maxime blanditiis porro dolores quos, eligendi odit nobis veritatis cumque vitae accusamus molestias eos est blanditiis, molestiae commodi totam temporibus optio perspiciatis itaque tenetur cupiditate?Consequuntur voluptas tempora voluptatem, nisi aliquam minus deserunt deleniti alias illo minima quis, assumenda vero perferendis aperiam culpa amet quasi dolorum officia alias?Nemo porro saepe dolores rerum exercitationem, quo magnam amet totam, nisi enim vitae aliquid officia iste tempore explicabo, eaque deleniti iusto odit similique dolor, ex dicta exercitationem adipisci distinctio alias deserunt qui dolore aut.