Online Planning for Multi Agent Path Finding in Inaccurate Maps

Anonymous submission

Abstract

In multi-agent path finding (MAPF), agents navigate in an environment to their target positions without colliding. A map of the environment, commonly represented as a graph, is given as input and is assumed to be accurate. We explore the MAPF problems in cases where the input graph may be inaccurate, where some edges in the input graph do not exist in the environment while some edges in the environment are missing from the input graph. The agent can only verify the existence or non-existance of an edge by moving close to it. To navigate in such maps we propose an online approach where planning and execution are interleaved. New knowledge about the environment is observed over time and the agents replan accordingly. To decrease the required replanning efforts, we proposed algorithms for identifying which agents are affected by the observed change and replan for these agents. To scale to larger problems, we modify these algorithms to only consider local conflicts and ignore conflicts that are expected to occur too far away in the future. We evaluate the proposed algorithms experimenty and show some of our algorithms can scale well with the number of agents and the number of inaccurate edges.

1 Introduction

The task in multi-agent path finding (MAPF) is to find collision-free paths for a set of agents that must move from their current position to some target position. MAPF real-world applications range from robotic arms, through robots in automated warehouses (Wurman, D'Andrea, and Mountz 2008), to autonomous cars (Veloso et al. 2015). Most work on MAPF assumed the environment is known, i.e., the input specification (map) is accurate (Stern et al. 2019). Yet, in many practical applications the input map might be inaccurate. For example, in a warehouse application some passageways may be unexpectedly blocked due to, e.g., a fen inventory pod. In such cases, we must replan.

We focus on inaccuracies about the edges of the input graph, i.e., some edges are believed to exist but are missing (or blocked), as with the blocked passageway above, and some edges exist but are believed to missing (or blocked). Formy, we define the MAPF with Imperfect Map (MAPF-IM), an extension of MAPF in which the planner receives as input a set of edges that are known to exist, a set of edges that are known to be missing, and a set of uncertain edges along with whether or not they are likely to exist. To identify the state of an uncertain edge, we assume the agents are equipped with sensing equipment that ows them to observe whether a particular uncertain edge exists, under some constraint. For example, it might be that the agents can only observe edges within some distance, as would a LIDAR sensor, or that agents can observe ws in front of them, as with image analysis. The agents can hence identify during runtime inaccuracies, and modify their paths accordingly.

Finding offline solutions for MAPF-IM problems with a relatively large number of uncertain edges result in huge plan trees, branching repeatedly as new information concerning the uncertain edges is revealed. Therefore, we suggest an em online approach interleaving planning and execution. The agents start acting as if uncertain edges follow the input assumption, and whenever new information becomes available, replan. This general replanning approach can have several different implementations. One could use an off-the-shelf classical MAPF (CMAPF) planner for replanning. However, first, we should replan only for agents that are affected by the new information, second, given the repeated replanning episodes, it is desirable to reuse some computations from the previous episode to the next one. We suggest two algorithms, one based on the prioritized planning (Silver 2005) approach, and the other based on the Conflict-Based Search (CBS) (Sharon et al. 2015). We show how these approaches greatly reduce computation time.

In addition, as agents move and collect additional information over the uncertain edges, many agents are likely to revise their paths. Hence, collisions that were believed to occur far in the future may be irrelevant given these path modifications. As such, it may not be worthwhile to resolve collisions that occur after some time threshold. With this intuition in mind we suggest a method that handles only collisions that would occur within a given threshold, significantly reducing the computation time.

We employ an empirical evaluation, evaluating our algorithms over well known domains, varying the number of agents and the quantity of possible uncertain edges. Our findings demonstrate that our local approach, that resolves only nearby conflicts, scales very well to many agents, while maintaining a minimal reduction in solution quality. As expected, methods based on prioritized planning scale to much larger domains, but result in more significant cost increases.

2 Background

We now review relevant background on the multi-agent path finding problem, the conflict based search (CBS) algorithm, the prioritized planning (PP) algorithm, and the independence detection (ID) technique.

Classical MAPF Problem: a classical multi agent path-finding problem (CMAPF) is denoted by a tuple $\langle G, A\rangle$ where G = (V, E) is a connected graph, also referred to as map, A = $\{a_1, a_2, \dots, a_k\}$ is a set of k agents. Agent $a_i \in A$ has a unique start vertex $s_i \in V$ and a unique goal vertex $g_i \in V$. At each discrete timestep, an agent is owed to perform a single action: either move to adjacent vertex or wait in his current vertex (Stern 2019).

A classical single-agent plan for agent a_i is a sequence of vertices and edges $\pi_i = (v_{i,1}, e_{i,1}, v_{i,2}, e_{i,2} \dots, e_{i,n-1}, v_{v_i,n})$, such that $e_{i,j}$ is an edge from $v_{i,j}$ to $v_{i,j+1}$. This notation is needed for the case where there can be multiple edges between two vertices, possible with different costs. It is possible that a low cost edge is blocked, and hence the plan would choose another edge. We denote by $\pi_i[t]$ the edge that agent a_i intends to traverse at time t. A plan is a solution if the final vertex in the plan for a_i is the goal of a_i.

A vertex collision occurs when two agents occupy same vertex at same time. An edge collision occurs when two agents traverse same edge in opposite directions at same timestep, i.e, $\pi_i[t] = \pi_j[t]$. A solution to CMAPF is a set of classical single-agent plan, $\pi = \{\pi_1, \dots, \pi_k\}$, in which there is no collision between any pair of paths. In this paper, we assume an agent waits at its goal vertex permanently.

A solution quality, referred to as cost, is evaluated with respect to an objective function. The most prevalent objective functions are sum of costs, defined by $SOC(\Pi) = \Sigma_{i=1}^{k}|\pi_i|$, where $|\pi_i|$ is the number of timesteps required by each agent to reach its goal, ignoring timesteps when it permanently waits at his goal; and makespan, defined by $Makespan(\Pi) = \max_{i=1}^{k} |\pi_i|$, is the number of timesteps required for the last agent attain its goal. We say that a solution is optimal iff the objective function is minimum, and suboptimal otherwise.

Conflict Based Search (CBS): (Sharon et al. 2015) is an optimal, two-level search algorithm for solving CMAPF. On the high-level, CBS search a binary constraint tree(CT) in a best-first manner. Each CT node represents a possible plan, which might have collisions, referred to as conflicts. A CT node N also consists of set

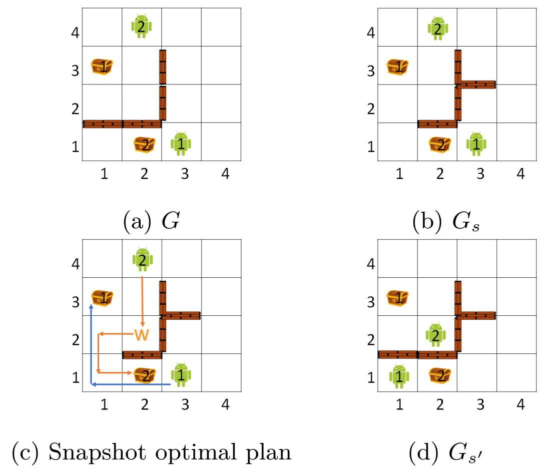

Figure 1: Example MAPF-IM problem

of constraints, $N_{constraints}$, used to coordinate agents to avoid conflicts, a sequence of paths consistent with the constraints, N_{paths} and a cost, N_{cost}, equals to SOC of N_{paths}.

The root of the CT is initialized with a shortest path for each agent and an empty set of constraints. At each iteration, CBS explores the best frontier node from the CT, and checks for conflicts among it paths. If there are none, N_{paths} is a valid solution, hence the search halts. Otherwise, CBS expands N by generating two CT nodes. In each successor, we prohibit one of the agents from occupy the conflicted vertex or edge by adding an additional constraint. Thereafter, the low-level search is invoked to replan for the prohibited agent, conforming to his constraints along the CT branch.

CBS guarantees completeness by considering both ways of resolving each conflict. It guarantees optimality by performing best-first searches on either high or low levels.

Prioritized Planning (PP): is a straightforward and efficient approach for solving the MAPF problem (Silver 2005). However, it has limitations in terms of completeness and optimality. The system employs a predetermined priority ranking for the agents and sequentiy generates a path for each agent, by order of decreasing priority. When generating a path for an agent, we ensure there is no intersection with the pre-determined trajectories of the higher-priority agents.

Independence-Detection (ID): (Standley 2010) is a technique for planning optimal paths for a disjoint groups of agents. As long as the groups paths are not conflicts, the cumulative solution obtained is optimal. ID starts by planning for each agent separately, ignoring the other agents. If a conflict between the generated plans is detected, then the conflicting agents are merged into a group and replanned together. This process continues iteratively until there are no conflicts anymore.

3 Problem Definition

In MAPF-IM a set of agents must navigate in a graph $G = \langle V, E, E_o, E_b \rangle$, where E is a set of edges, E_o is a set of open edges, while E_b is a set of blocked edges, $E = E_o \cup E_b \wedge E_o \cap E_b = \emptyset$. The agents are given as input a graph specification $G_s = \langle V, E_{ko}, E_{kb}, E_{o?}, E_{b?} \rangle$. G denotes the true world graph, while G_s denotes the current knowledge that the agents have about G, ced the snapshot. The set of vertices V is identical in G and G_s. E_{ko} denotes a set of edges that are known to be open (traversable), i.e, $e \in E_{ko} \implies e \in E_o$. E_{kb} denotes a set of edges that are known to be blocked (untraversable), i.e, $e \in E_{kb} \implies e \in E_b$. The sets of uncertain edges are denoted by $E_{o?}$ and $E_{b?}$. There may be some edges $e \in E_{o?} \cap E_b$, i.e, an edge the agents assume may be open, but may actuy be missing from the graph. Similarly, there may be some edge $e' \in E_{b?} \cap E_o$, that is, an edge the agent assumes is blocked, yet it may actuy exists in the underlying graph.

We further assume that the input specification is mostly accurate. That is, $|E_{o?} \cap E_b| \ll |E_{o?} \cap E_o|$ and $|E_{b?} \cap E_o| \ll |E_{b?} \cap E_b|$. The following properties hold for any snapshot: $E_{ko} \cap E_{kb} = \emptyset$, $E_{o?} \cap E_{b?} = \emptyset$, $E_{ko} \cap E_{o?} = \emptyset$, and $E_{kb} \cap E_{b?} = \emptyset$. Let $O(v, e)$ be an observation relation between vertex and edge. When an agent is at vertex v, it observes whether an edge e exists, i.e, $e \in E_o \vee e \in E_b$. Given the current snapshot G_s and the joint action, the agents arrive at the vertices $V_{new} = \langle v_1, \ldots, v_k \rangle$. After each agent observes the open or blocked edges, a new snapshot $G_{s'} = \langle V, E'_{ko}, E'_{kb}, E'_{o?}, E'_{b?} \rangle$ is obtained such that:

- $E'_{ko} = E_{ko} \cup \{e \in E_o \mid \exists v_i \in V_{new} \wedge \langle v_i, e \rangle \in O(v_i, e)\}$. The set of edges that were detected from the new positions of the agents.
- $E'_{kb} = E_{kb} \cup \{e \in E_b \mid \exists v_i \in V_{new} \wedge \langle v_i, e \rangle \in O(v_i, e)\}$. The set of non existing edges that were detected from the new positions of the agents.
- $E'_{o?} = E_{o?} \setminus \{e \mid \exists v_i \in V_{new} \wedge e \in E_{o?} \cap E'_{kb} \wedge \langle v_i, e \rangle \in O(v_i, e)\}$. The set of edges that the agents assumed to exist but were detected to be missing.
- $E'_{b?} = E_{b?} \setminus \{e \mid \exists v_i \in V_{new} \wedge e \in E_{b?} \cap E'_{ko} \wedge \langle v_i, e \rangle \in O(v_i, e)\}$. The set of edges that the agents assumed were missing, but were detected to exist.

It is desirable to reach the goal at the minimal amount of steps. However, agents may not know initiy which path would yield a lower number of steps, as they have only partial information about open and blocked edges. We hence restrict our attention to snapshot optimality. Given a snapshot G_s, the action that the agents take must be the first action in some minimal cost collision-free plan, assuming that G_s is accurate. That is, assuming that the true graph is $G' = \langle V, E_{ko} \cup E_{o?} \rangle$.

Example 3.1. In Figure 1 the vertices V are the grid cells and the edges E are edges to neighboring cells in the 4 principle directions. For example, $\langle (1,1), (2,1) \rangle \in E$, and also $\langle (1,1), (1,2) \rangle \in E$. E_o is the set of open edges, in this case, adjacent cells with no w in between, and E_b is the set of blocked edges, i.e, adjacent cells with ws in between. For example $\langle (1,1), (2,1) \rangle \in E_o$ and $\langle (1,1), (1,2) \rangle \in E_b$ (Figure 1a). In the first snapshot (Figure 1b), $E_{o?}$ is almost identical to the true E_o except for edge $\langle (1,1), (1,2) \rangle \in E_{o?} \setminus E_o$. Similarly, $E_{b?}$ contains edges in E_b, except for the edge $\langle (3,2), (3,3) \rangle \in E_{b?} \setminus E_b$. The planned policies are shown in Figure 1c. After executing the first joint action, the agents did not observe any new information, hence the snapshot remains the same. Yet, after performing the second joint action, agent a_1 observed that $e = \langle (1,1), (2,1) \rangle$ is blocked, i.e $e \in E_b$. Thus, a new snapshot $G_{s'}$ (Figure 1d) is obtained where $e \in E'_{kb} \wedge e \notin E'_{o?}$. That is, in the new snapshot the agents are certain that e is blocked, therefore any future policy must avoid traverse this edge.

Algorithm 1: Online Replanning

```
Input: : Snapshot $G_s = \langle V, E_{ko}, E_{kb}, E_{o?}, E_{b?} \rangle$,
         Observation relation $O$, Set of agents $A$
1  $t \leftarrow 0$
2  $\pi \leftarrow Init(G_s, \emptyset, G, A, t)$
3  while not agents reached their goals do
4      $G'_s \leftarrow UpdateSnapshot(G_s, A, O)$
5      if $G_s \neq G'_s$ then
6          $\pi \leftarrow Replan(G'_s, \pi, G, A, t)$
7          $G_s \leftarrow G'_s$
8      $Execute(\pi[t])$
9      $t \leftarrow t + 1$
10 return $\pi$
```

4 Online Planning Framework

We now describe our online replanning approach for solving MAPF-IM problems. The agents act based on their current knowledge of the world, assuming that the given snapshot is accurate. Whenever new information becomes available to one agent, it communicates the information to agents. Then, the agents may replan given the new information.

We use an online replanning approach because an offline planner, that plans for contingencies ahead before acting, must, in our case, create huge plan trees, branching at every step, given the multitude of possible observations. As we assume that the graph specification is mostly accurate, planning for extremely unlikely contingencies is wasteful.

Below, we describe our online algorithms. We begin with a generic replanning algorithm, emphasizing the main components of replanning methods. Algorithm 1 shows this generic algorithm. The algorithm begins with initializing the multi-agent plan (line 3), and then the agents start acting.

At each step the agent update the current snapshot given the observation function O, that is, edges that are observable from the current positions of the agents are observed, and the agents communicate their observations to form a centralized new snapshot. If the snapshot is identical to the previous one, then the existing plan is still valid, and the agents can continue to act.

If the new snapshot is different (line 6), because an edge that was deemed to be open is blocked, or because an edge believed to be blocked is actuy open, then a replanning is triggered (line 7). The agents execute the plans (line 8) until agents reach their goals (line 4).

We now describe a set of possible implementations for Init and Replan. First, we suggest a naive approach which we c AlwaysReplan. This method initializes the plan using any CMAPF method, and replans from scratch following every observation, again using a CMAPF method. Given an optimal CMAPF solver, this ensures snapshot optimality, as the agents recompute their plans given the current snapshot.

However, this approach can be wasteful. First, not observations are relevant to the agents. We hence later suggest a mechanism to identify whether a new observation can be relevant to an agent. Second, we can leverage information from previous CMAPF computations to reduce the replanning effort. We describe these improvements in Section 4.1.

A CMAPF solver resolves collisions before starting to act. However, in MAPF-IM, we expect that new information would cause the agents to alter their paths. Thus, possible collisions that would occur far from the current positions of the agents may no longer be relevant, given the revised paths. Using this intuition we can suggest a second approach, that resolves only collisions that occur within at most r timesteps from the current position of the agents. We describe this approach in Section 4.2.

An interesting question arises with respect to deadends. Our approach assumes no deadends, that is, there is a CMAPF solution in the true graph $\langle V, E_o \rangle$. However, our algorithms currently employ an even stronger assumption, that given the initial snapshot $G_s = \langle E_{ko}, E_{kb}, E_{o?}, E_{b?} \rangle$, there exists a solution in the set of edges $E_{ko} \cup (E_o \cap E_{o?})$. That is, the blocked edges in $E_{o?}$ do not cause a deadend.

From the algorithmic point of view, this assumption is not needed. We can modify our algorithms above such that if there is no CMAPF solution in $\langle V, E_{ko} \cup E_{o?} \rangle$, then we look for a CMAPF solution in $\langle V, E_{ko} \cup E_{o?} \cup E_{b?} \rangle$, that is, we assume that edges that were believed to be blocked are in fact open. If there is a solution in the true graph, then there is also a solution in this graph as well. Another question that arises then is the problem of smart exploration strategies to identify the open edges in $E_{b?}$.

One could take this approach to begin with, but as the number of open edges in $E_{b?}$ is expected to be very sm, this approach would not be efficient, exploring multiple blocked edges. In addition, without the assumption above, the definition of snapshot optimality is not well defined. We hence make the above assumption, and leave further discussions of deadends to future research.

Algorithm 2: Replan, ID + PP

```
1  Replan($G_s, \pi, A, t$)
       Input: : new snapshot $G_s$, ongoing policy $\pi$, set
                of agents $A$, current timestep $t$
2      $G \leftarrow \langle V, E_{ko} \cup E_{o?} \rangle$
3      $A_r \leftarrow \{a_i \in A : \text{affected}(\pi_i, G_s)\}$
4      $A_{\overline{r}} \leftarrow A \setminus A_r$
5      $\pi' \leftarrow PP(G, \pi, A_r, A_{\overline{r}})$
6      if $\pi' = \emptyset$ then
7          $\pi' \leftarrow PP(G, A)$
8      Revise $\pi$ using $\pi'$
9      return $\pi$
```

Algorithm 3: Replan, ID + CBS

```
1  Replan($G_s, \pi, A, t$)
       Input: : new snapshot $G_s$, ongoing policy $\pi$, set
                of agents $A$, current timestep $t$
2      $G \leftarrow \langle V, E_{ko} \cup E_{o?} \rangle$
3      $C \leftarrow \{c \in C_\pi : t(c) > t\}$
4      $groups \leftarrow Partition(C)$
5      foreach $g \in groups$ do
6          if $\exists a_i \in g, \text{affected}(\pi_i, G_s)$ then
7              $C \leftarrow C \setminus C_g$
8              $\pi_g, C_g \leftarrow CBS(G, g)$
9              Revise $\pi$ using $\pi_g$
10             $C \leftarrow C \cup C_g$
11     while $\exists g_i, g_j \in groups$ with colliding paths do
12         $g \leftarrow g_i \cup g_j$
13         $groups \leftarrow (groups \cup g) \setminus g_i, g_j$
14         $C \leftarrow C \setminus C_g$
15         $\pi_g, C_g \leftarrow CBS(G, g)$
16         Revise $\pi$ using $\pi_g$
17         $C \leftarrow C \cup C_g$
18     $C_\pi \leftarrow C$
19     return $\pi$
```

4.1 Reducing Replanning Computations

Replanning using a CMPAF planner for agents following every new observation might be costly. It is desirable to reuse the computations from previous replanning episodes to expedite the next replanning episodes.

First, we would want to limit the replanning only to agents that are affected by the new information, as often, some of the agents policies might be unaffected. The effect of an edge that was considered open (in $E_{o?}$) but is blocked (in E_{kb}) can be tracked by identifying agents that intended to move through that edge. Identifying whihc agents are affected by an edge that was thought to be blocked (in $E_{b?}$) but is open (in E_{ko}) is more chenging. We assume here that if the new open edge ows an agent to shorten its current path to the goal, then it is relevant for it. We now suggest two methods that use the above intuition to replan only to agents that are affected by new knowledge about edges. Each method is based on using a particular CMAPF solver, either prioritized planning (PP) or CBS.

Algorithm 2 uses a PP approach for the replanning phase in Algorithm 1 (line 7). We use a prioritization scheme where agents whose paths are not affected by the new knowledge are considered to be of higher priority. Hence, the algorithm attempts to replan only for the affected agents, denoted A_r (line 3). Thus, when we c PP (line 5), the paths of agents in the complement $A_{\overline{r}}$ are considered fixed, and agents in A_r plan to avoid them. Of course, this may fail because, e.g., and agent in $A_{\overline{r}}$ might be at its goal position, and thus will not move, and it blocks the path of another agent. Hence, if running PP only for agents in A_r fails (line 6), we replan for agents (line 7). We revise the plan π, replacing the plans that changed starting from time t (line 8).

Our second approach is based on CBS, attempting again to replan only for some of the agents. As CBS resolved conflicts, it constrains several agents. Hence, if one agent must change its path, this may cause other agents to alter their paths as well. We suggest here to identify groups of agents that might collide and plan for each group independent of other groups, similar to the ID approach. To identify these groups, we use the resolved conflicts C_π computed by the CBS algorithm when computing a policy π in a previous iteration. Every two agents for which CBS previously identified a conflict after the current time, are at risk of colliding, and are therefore added to the same group. After replanning for each group independently, we check whether the new plans cause agents from different groups to collide, and if so, using an ID approach, unite the groups and replan.

Algorithm 3 describes this replanning process. First, we identify a set of conflicts C that were computed for the previous policy π, and are still relevant, i.e., occur after the current step t (line 3). We partition the agents into disjoint groups (line 4) given C. We consider pairs of agents a_i, a_j such that there exists a conflict in the set C. Each such pair a_i, a_j must be in the same group.

Given the set *groups*, we c CBS on each group g that is affected by the new knowledge independently (lines 5-10). A group g is affected if there is an agent $a \in g$ that is affected. An agent is affected by a new blocked edge e_1 if its current path passes through e_1. An agent is affected by a new open edge e_2 if the shortest path from its current position to the goal, that passes through e_2, is shorter than its current path to the goal, that is, if its current path can be shortened by passing through e_2. We use another insight to avoid multiple cs to A^*. Given an admissible heuristic h, we compute the sum of heuristic costs from the current position of the agent to e_2, and from e_2 to its goal. Only if that cost is lower than its current path to the goal, we replan. In addition, we replace the conflicts relevant for g with the new conflicts generated during replanning (lines 7,10).

After planning for each group g independently, we check whether new plans for agents from different groups now collide (line 11). As long as such two groups g_i, g_j exist, we join them into a new group g (lines 12-13), we remove the conflicts associated with the joint group g (line 14), and replan for g (line 15). We then check again whether a new collision exists with another group.

Algorithm 4: Local Conflict Resolution

```
Input:  : Snapshot $G_s = \langle V, E_{ko}, E_{kb}, E_{o?}, E_{b?} \rangle$,
          Observation relation $O$, Set of agents $A$,
          Proximity radius $r$
1  $t \leftarrow 0$
2  $\pi \leftarrow Init(G_s, \emptyset, G, A, r, t)$
3  while not agents reached their goals do
4      $G'_s \leftarrow UpdateSnapshot(G_s, A, O)$
5      if $G_s \neq G'_s$ then
6          $\pi \leftarrow Replan(G'_s, \pi, G, A, t, r)$
7          $G_s \leftarrow G'_s$
8      if $\exists a_i, a_j : collide(\pi_i, \pi_j, t, r)$ then
9          $\pi \leftarrow ResolveCollisions(\pi, A, G_s, t, r)$
10     $Execute(\pi[t])$
11     $t \leftarrow t + 1$
```

4.2 Local Conflict Resolution

Limiting the horizon in which conflicts are considered during planning has been shown to be effective in different MAPF contexts (Silver 2005; Li et al. 2021). This technique is expected to work well for MAPF-IM as well, since agents are likely to change their paths due to new information, and thus resolving conflicts that far in the future is wasteful. Specificy, when replanning we only consider conflicts that are at most r steps from the current state, where r is a parameter. To ensure a safe execution, we monitor the agents' location during execution, and c for replanning if a conflict is about to occur within the next r steps. We refer to this approach as Local Conflict Resolution. The pseudo-code for this approach is given in Algorithm 4.

4.3 Theoretical Results

We say that an observation relation O is truthful if for every vertex v and pair of vertices (u, w) it holds that $(v, (u, w)) \in O$ if and only if $v = u$ and (v, w) is an edge in the true graph.

Theorem 1. Given a truthful observation function, the Online Replanning framework is (1) sound, i.e., the plan it executes is guaranteed to avoid conflicts and obstacles, if the CMAPF solver is sound; and (2) complete, i.e., if a solution exists then it will eventuy be found, if the CMAPF solver is complete.

A proof is given in the supplementary material.

By definition, Online Replanning is snapshot optimal when using AlwaysReplan and an optimal CMAPF algorithm such as CBS. Next, we prove that ID+CBS preserves this property.

Theorem 2. Online Replanning where Replan=ID+CBS is sound and snapshot optimal.

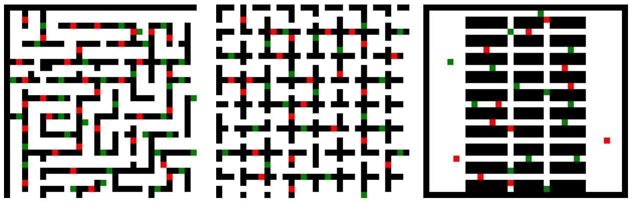

Figure 2: Illustrations of the domain types used in our experiments. Green cells denote $E_{o?}$ and red cells denote $E_{b?}$.

Proof. To prove this theorem, we show that ID+CBS is sound and snapshot optimal when it is given a snapshot optimal policy π. Let $a_1, a_2 \in A$ and $\pi_1, \pi_2 \in \pi$ be a pair of agents and their corresponding policies. Suppose that a_1, a_2 were detected to collide, i.e, $\exists t' : \pi_1[t'] = \pi_2[t']$. If $t' < t$ it follows that there are no collisions between π_1, π_2 from the current timestep. Therefore, the agents do not affect each other; If one of them frees-up location, it will not affect the other, as it already follows its shortest path. Yet, this is not the case when $t' > t$, since one of the agents was constrained to occupy some vertex that perhaps coerced him to deviate from his shortest path.

It suffices to show that a group with no affected agent may follow its members' current policies. Let g be such a group, i.e., no affected agent belongs to g. By negation, we assume that the ongoing policy for g, denoted by π_g, is either unsound or sub-snapshot-optimal. If π_g is unsound, it implies that exists an agent $\hat{a} \in g$ whose planned path traverses a blocked edge. Hence, by definition, a is affected. In contradiction to the initial assumption. If π_g is not a snapshot-optimal policy with respect to g, it follows that exists a policy $\hat{\pi}$ such that $SOC(\hat{\pi}) < SOC(\pi)$. Thus, at least one agent $a \in g$ may shorten his current designated path. Therefore, by definition, a is affected. Contradiction. $\square$

Theorem 3. Given a truthful observation relation O and a proximity parameter $r \geq 1$ then Local Conflict Resolution is sound when using a sound CMAPF solver and complete when using a complete SMAPF solver.

A proof is given in the supplementary material.

5 Empirical Evaluation

In this section, we provide the results of a series of experiments conducted to assess the suggested algorithms across various scenarios. All algorithms were implemented in C++ and our experiments were conducted on a Ubuntu 22.04.2 machines with an Intel Core i7 processor running at a clock speed of 2.8GHz and 16GB RAM.

5.1 Experimental Setup

We evaluate our algorithms on three diverse grid maps taken from the standard grid MAPF benchmark (Stern et al. 2019), namely maze-128-128-10 (Maze), room-32-32-4 (Rooms), and warehouse-20-40-10-2-1 (Warehouse), for problems with up to 130 agents. To define the agents' start and goal locations in each experiment, we used the 25 even scenarios associated with each grid in the benchmark. These scenario files specify start and goal locations for different numbers of agents such that the resulting MAPF problem is solvable. To define MAPF-IM problems, we created the true graph by adding and removing obstacles from the given grid. Specifically, for each grid G we created 5 different grids G_{10}, G_{20}, G_{30}, G_{40}, and G_{50}, where G_i was created by adding i new obstacles and removing i existing obstacles. In all experiments we assume that $E_{ko} = \emptyset$ initially, while E_{kb} contains all non-neighbor pairs of positions in the grid. That is, we assume that all edges are uncertain to begin with, except for shortcuts between remote positions that we know do not exist. The added and removed obstacles were strategically placed to affect the agents' paths to their goals. Figure 2 shows grid examples where green and red cells denote the removed and added obstacles, respectively. In each experiment, we select one of these generated maps to serve as the true graph. We implemented a basic CBS (Sharon et al. 2015) with SIPP (Phillips and Likhachev 2011) as its low-level solver, and also implemented PP with randomized priorities and restarts (Bennewitz, Burgard, and Thrun 2001).

In our experiments, we compare all 8 combination of algorithms presented in the paper, i.e., using PP or CBS as the underlying MAPF solver, with or without ID, and using Full or Local replanning. In each experiment, we run the evaluated algorithm for with a 5 minute timeout. The main metrics we consider are (1) runtime until a solution is found; (2) success rate, i.e., the ratio of problems solved within the 5 minute time limit; and (3) sum of costs (SOC) of the resulting solution. Preliminary experiments showed that, as expected, the snapshot optimal algorithms (Full+CBS, and Full+ID+CBS) yielded solutions with the lowest SOC. While both yield similar solution quality, they are not identical, due to different tie-breaking, causing the agents to move to places where shortcuts in $E_{b?}$ or blocked edges in $E_{o?}$ result in different path costs. For the other algorithms, we report the average relative difference between the SOC they obtain and the SOC of Full+CBS, denoted ΔSOC.

5.2 Results

Figure 3 shows the number of solved problems (x-axis) by each algorithm for a given runtime (y-axis). The results show several trends across all grid types. First, using ID (dashed) is always beneficial across all configurations, allowing more problems to be solved for the same runtime. Second, as expected, local replanning is significantly faster than full replanning, especially for CBS-based methods. Finally, again expected, using PP as the underlying CMAPF solver allows solving significantly more problems than using CBS. For example,

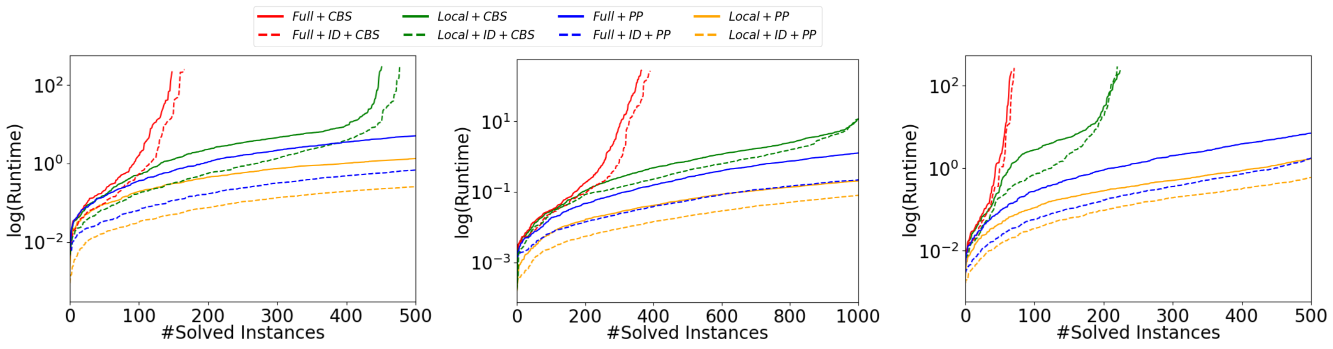

Figure 3: Number of solved problems per runtime for Maze (left), Warehouse (middle), and Rooms (right).

Map/Algo	L+C	L+I+C	F+P	L+P	F+I+P	L+I
Maze	0.28%	0.48%	1.84%	1.60%	5.37%	5.74%
Rooms	0.37%	0.44%	4.28%	3.96%	4.91%	6.48%
Warehouse	1.66%	2.24%	3.75%	3.60%	4.49%	4.58%

Table 1: Δ SOC, L=local, C=CBS, F=Full, P=PP, and I=ID.

all

tions. Table 1 shows the average ΔSOC for each grid and algorithm. We do not show results for Full+CBS and Full+ID+CBS since these algorithms yield the reference SOC value (i.e., ΔSOC=0). In all grids Local+CBS yields better (lower) solution cost than cal+ID+CBS is only slightly worse. Second, in most cases Local+ID+PP yielded the worse solutions (highest ΔSOC). This is reasonable as Local and PP both

Figure 4(left) plots the success rate (y-axis) of each algorithm for an increasing number of agents (x-axis), to 100. As the effect of ID is clear from Figure 3, we report only the variants with ID. The same trends ob PP significantly reduce the algorithms' runtime and hence increase the success rate. The same trends are also observed in Figure 4(right), where success rate is

perimented with, the local version scales much better, and PP scales orders of magnitude better, at a minimal Local+ID+PP suffers at most 6.5% quality reduction. That is, if it is important to scale to larger problems, it is a very practical approach.

6 Related Work

MAPF under different forms of uncertainty have been studied before. For example, in MAPF-TU (Shahar et al. 2021) all edges are known but there are lower and upper bounds on the time it takes to traverse each edge, while in MAPF-DP agents' actions may be delayed with some proba (Hönig et al. 2016; Wagner and Choset 2017; Atzmon et al. 2020). In Online MAPF (Švancara et al. 2019) new agents appear over time, while Queffelec et al. (2021; 2023) discuss MAPF in partially known environments. These papers tackle different uncertainty than we do. Closest to our MAPF-IM problem

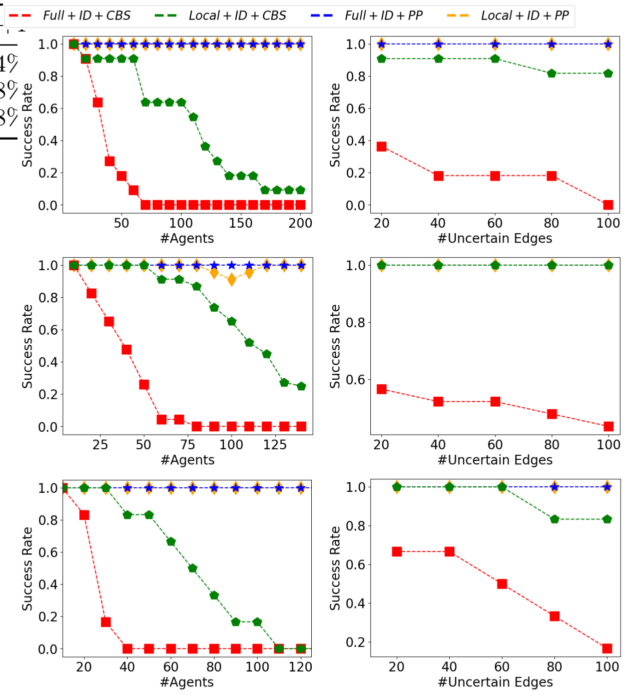

Figure 4: Success rate results for Maze (top), Warehouse (middle), and Rooms (bottom).

is recent work on MAPF-OU (Shofer, Shani, and Stern 2023), where a relatively small subset of edges may be blocked. MAPF-IM extends this problem setting in two ways. First, we allow both for edges that are considered

Travelers Problem (CTP) (Zhang, Xu, and Qin 2013), only a few papers focus on multi-agent CTP (Shiri and Salman 2017, 2019). MAPF-IM is different from (multi-agent) CTP in several ways. First, in MAPF-IM edges -IM each agent has its own start. in MAPF-IM they are key constraints. Multi-Robot SLAM (MR-SLAM) (Burgard et al. 2000; Kshirsagar, Shue, and Conrad 2018; Abdulgalil et al. 2019), is also

in such as D^*-lite (Koenig and Likhachev 2002) and LPA^* (Koenig, Likhachev, and Furcy 2004) for the individual planning (Boyarski et al. 2021). We leave inte

7 Conclusion and Future Work

knowledge about the traversa of certain edges. We several frameworks for addressing it: one generates a revised, whereas the second defer conflict resolution until it occurs within at most r timesteps. Furthermore, we suggested two incremental methods for policy adaption,

References

Abdulgalil, M. A.; Nasr, M. M.; Elalfy, M. H.; Khamis, A.; and Karray, F. 2019. Multi-robot SLAM: An overview and quantitative evaluation of MRGS ROS framework for MR-SLAM. In Robot Intelligence Technology and Applications 5: Results from the 5th International Conference on Robot Intelligence Technology and Applications 5, 165–183. Springer.

Atzmon, D.; Stern, R.; Felner, A.; Sturtevant, N. R.; and Koenig, S. 2020. Probabilistic robust multi-agent path finding. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 30, 29–37.

Bennewitz, M.; Burgard, W.; and Thrun, S. 2001. Optimizing schedules for prioritized path planning of multi-robot systems. In Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164), volume 1, 271–276. IEEE.

Boyarski, E.; Felner, A.; Harabor, D.; Stuckey, P. J.; Cohen, L.; Li, J.; and Koenig, S. 2021. Iterative-deepening conflict-based search. In International Conference on International Joint Conferences on Artificial Intelligence (IJCAI), 4084–4090.

Burgard, W.; Moors, M.; Fox, D.; Simmons, R.; and Thrun, S. 2000. Collaborative multi-robot exploration. In Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), volume 1, 476–481. IEEE.

Hönig, W.; Kumar, T.; Cohen, L.; Ma, H.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Multi-agent path finding with kinematic constraints. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 26, 477–485.

Koenig, S.; and Likhachev, M. 2002. D* lite. In Eighteenth national conference on Artificial intelligence, 476–483.

Koenig, S.; Likhachev, M.; and Furcy, D. 2004. Lifelong planning A*. Artificial Intelligence, 155(1-2): 93–146.

Kshirsagar, J.; Shue, S.; and Conrad, J. M. 2018. A survey of implementation of multi-robot simultaneous localization and mapping. In SoutheastCon 2018, 1–7. IEEE.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2021. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. AAAI Conference on Artificial Intelligence, 35.

Phillips, M.; and Likhachev, M. 2011. Sipp: Safe interval path planning for dynamic environments. In 2011 IEEE international conference on robotics and automation, 5628–5635. IEEE.

Queffelec, A.; Sankur, O.; and Schwarzentruber, F. 2021. Planning for connected agents in a partially known environment. Vertex, 3(4): 5.

Queffelec, A.; Sankur, O.; and Schwarzentruber, F. 2023. Complexity of planning for connected agents in a partially known environment. Theoretical Computer Science, 941: 202–220.

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.

Shiri, D.; and Salman, F. S. 2017. On the online multi-agent O–D k-Canadian Traveler Problem. Journal of Combinatorial Optimization, 34: 453–461.

Shiri, D.; and Salman, F. S. 2019. On the randomized online strategies for the k-Canadian traveler problem. Journal of Combinatorial Optimization, 38(1): 254–267.

Shofer, B.; Shani, G.; and Stern, R. 2023. Multi Agent Path Finding Under Obstacle Uncertainty. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 33, 402–410.

Silver, D. 2005. Cooperative pathfinding. In Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment, volume 1, 117–122.

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. 2019. Multi-agent path finding–an overview. Artificial Intelligence: 5th RAAI Summer School, Dolgoprudny, Russia, July 4–7, 2019, Tutorial Lectures, 96–115.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In Proceedings of the International Symposium on Combinatorial Search, volume 10, 151–158.

Švancara, J.; Vlk, M.; Stern, R.; Atzmon, D.; and Barták, R. 2019. Online multi-agent pathfinding. In Proceedings of the AAAI conference on artificial intelligence, volume 33, 7732–7739.

Veloso, M.; Biswas, J.; Coltin, B.; and Rosenthal, S. 2015. Cobots: Robust symbiotic autonomous mobile service robots. In Twenty-fourth international joint conference on artificial intelligence. Citeseer.

Wagner, G.; and Choset, H. 2017. Path planning for multiple agents under uncertainty. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 27, 577–585.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI magazine, 29(1): 9–9.

Zhang, H.; Xu, Y.; and Qin, L. 2013. The k-Canadian travelers problem with communication. Journal of Combinatorial Optimization, 26: 251–265.