Abstract

Multi-agent path finding (MAPF) is the problem of planning a set of non-conflicting paths on a graph, for a set of agents. Online MAPF extends MAPF by considering a realistic problem in which new agents may appear over time. While planning, an online solver does not know of agents that will join in the future. Therefore, the quality of a solution to an online MAPF problem may be lower than the quality of a solution to an equivalent offline MAPF problem, where the solver is preinformed of all the agents that will appear in the future. We examine the matter of optimality in classic online MAPF, as well as two variants of the problem. In the first variant, rerouting an agent is associated with a cost (COR). In the second variant, agents are assigned different priorities. We show how properties of online MAPF affect solution quality in these three different settings, and present the trade-offs involved with each setting.

1 Introduction

Multi-agent path finding (MAPF) is an important multi-agent planning problem, where the task is to find a set of plans for multiple agents. A plan is a sequence of vertices such that traversing all vertices in order, starting from the corresponding agent’s source, will bring the agent to its goal. This solution is constrained so that the agents are not allowed to collide with each other while following their individual plans concurrently. MAPF can be found in real-world applications such as autonomous warehouses, autonomous vehicles, and multi-robot systems (Ma et al. 2019; Čap, Vokřínek, and Kleiner 2015; Dresner and Stone 2008; Ho et al. 2019). Efficient algorithms that can solve the MAPF problem optimally for a large number of agents were proposed (Li et al. 2019; Lam et al. 2019; Gange, Harabor, and Stuckey 2019; Felner et al. 2017).

The standard MAPF problem assumes that all agents start moving at the same time, finish their task when arriving at their goals, and that new agents cannot appear after agents have started moving. However, in real-life applications new agents might arrive while some agents are still executing their plans. Moreover, the problem may continue after the last agent has arrived at its goal, e.g., a new autonomous vehicle can get on the road.

Online MAPF (Švancara et al. 2019) is a MAPF setting in which agents can arrive at different times. We will focus on a variant of this setting called the Intersection Model (Stern et al. 2019), which is inspired by autonomous vehicles passing through an intersection. Because of the online nature of this problem, a solver does not know in advance where and when new agents will appear. As a result, an existing agent may be rerouted in a way that seems efficient at one point in time, but eventually results in many conflicts with agents that appear in the future. Therefore, the standard definition of an optimal solution in offline MAPF is inadequate for online MAPF.

Švancara et al. (2019) suggested a new type of optimality for online MAPF called snapshot optimality. A solution is snapshot-optimal if it is optimal for the current known agents, i.e., assuming no new agent will appear. While some algorithms for finding snapshot-optimal solutions were proposed, the quality of snapshot-optimal solutions in terms of the cost function that is used has not been evaluated yet.

Another, closely related, generalization of offline MAPF is multi-agent pickup and delivery (Ma et al. 2017). In this online setting, agents cannot leave the problem space, and new agents may not join. Instead, new pickup-and-delivery tasks arrive over time, and are then assigned to agents.

In this paper, we examine different definitions of optimality in online MAPF, and the different factors affecting the quality of solutions for this problem, in terms of the cost of solutions. We analyze three different online MAPF settings, suggest different solvers for each setting, and compare them analytically and experimentally.

First, we evaluate the quality of snapshot optimal solutions compared to offline-optimal solutions. An offline-optimal solution is the minimal cost solution among all possible solutions. No online solver can guarantee to return an offline-optimal solution (Švancara et al. 2019). An offline-optimal solution can be guaranteed if the solver knows in

---

1This is reminiscent of the free space assumption (Koenig and Likachev 2002) which assumes that a cell in a grid that is not particularly known as an obstacle is considered free.
advance when and where new agents will appear. We compare the two types of optimality both theoretically and empirically, showing the pros and cons of snapshot-optimality.

Second, we evaluate different online solvers under a setting where rerouting an agent is associated with a specific cost (COR), in addition to the commonly used sum of costs (SOC). Changing the plan that an agent is currently executing requires communication with the agent. This may incur some overhead cost such as contributing to the congestion of a communications network. For this setting we suggest a new cost function called COMBINED-COST, which considers both SOC and COR. We suggest a new solver called SNAPSHOT-COR which finds snapshot-optimal solutions for COMBINED-COST. We compare the quality of its solutions with a snapshot-optimal solver that performs as many reroutes as necessary to minimise plan length, and performs well when COR is low. We also compare with a sub-optimal solver that performs no reroutes at all, and performs well when COR is high. We show that the more balanced SNAPSHOT-COR solver usually achieves better solutions in terms of cost.

Third, we examine a variant of online MAPF where agents are assigned different priorities, called online MAPF with priorities. An offline version of this setting was suggested by Okoso, Otaki, and Nishi (2019). We suggest a new solver for this setting called StratifiedPriorities. This solver finds solutions separately for groups of agents with equal priority, using any MAPF solver, without conflicting with the solutions of other groups. We implemented Stratified-Priorities with two different MAPF solvers, and compared them against three other online MAPF solvers.

2 Background

The input of the Multi-Agent Path Finding (MAPF) problem is a tuple \( \langle G, A \rangle \), where \( G = (V, E) \) is the problem graph, and \( A = \{a_1, \ldots, a_k\} \) is a set of \( k \) agents. Each agent \( a_i \) is associated with a source location \( s_i \in V \) and a goal location \( g_i \in V \) (Stern et al. 2019). A solution to a MAPF problem is a set \( \pi = \{\pi_1, \ldots, \pi_k\} \) of individual agent plans such that each agent \( a_i \in A \) is associated with a single plan \( \pi_i \) that starts at the agent’s source and ends at the agent’s goal. A plan is a sequence of vertices such that \( \pi_i(t) \in V \) is the planned location for agent \( a_i \) at time \( t \), where time is discrete, and \( \pi_i[x] \in V \) is the location that the agent would reach after performing \( x \) moves according to its plan. Thus, \( \forall a_i, (\pi_i[0] = s_i \land \pi_i[|\pi_i|] = g_i) \).

Each vertex in the plan is either a neighbor of the preceding vertex, or the same vertex, i.e. \( \forall x (\pi_i[x-1] = \pi_i[x] \lor (\pi_i[x-1] = \pi_i[x] \in E) \).

The length of a plan is the difference between the time that the plan starts on and the time that it ends on \((len(\pi_i) = |\pi_i| - 1)\). A MAPF solution is valid only if none of the plans within it conflict. Two plans \( \pi_i \) and \( \pi_j \) conflict if at any time \( t \) the two agents are planned to occupy the same vertex \((\pi_i(t) = \pi_j(t))\), which is called a vertex conflict, or are planned to swap their locations \((\pi_i(t) = \pi_j(t-1) \land \pi_j(t-1) = \pi_i(t))\), which is called a swapping conflict. A common cost function in MAPF is the Sum Of Costs (SOC). SOC is defined as the sum all individual plans’ lengths, \( SOC(\pi) = \sum_{i=1}^{k} len(\pi_i) \). A MAPF solution is optimal if it is valid and has the minimal cost among all valid solutions.

Conflict-Based Search (CBS) (Sharon et al. 2015) is a state-of-the-art optimal solver for the multi-agent path finding problem. CBS finds optimal multi-agent solutions by performing a best first search according to solution cost on a binary called the constraint tree (CT). This is called the high-level search. The CT starts with a root node, containing a solution, where individual plans are created for each agent while ignoring all other agents. These plans are created using a single-agent search algorithm, referred to as the low-level search. This solution may contain conflicts between the plans of different agents. A conflict is a tuple \( \langle a_i, a_j, v, t \rangle \) where the plans of agents \( a_i \) and \( a_j \) have them colliding on vertex \( v \) at time \( t \). CBS resolves these conflicts by putting constraints on agents, thus limiting the actions they are allowed to make. A constraint is a tuple \( \langle a_i, v, t \rangle \) such that \( a_i \) is an agent that is prevented from being at vertex \( v \) at time \( t \).

Every time a node in the CT is expanded, a conflict is picked and two constraints are created, one for each agent in the conflict. A node is then created for each constraint, and the solution in each node is updated to comply with the node’s constraint. The node is then put into OPEN. CBS halts when a node with no conflicts is polled from OPEN.

3 Online MAPF

Most previous works on MAPF focused on an offline MAPF setting, where plans are found before the agents start their movement. As soon as a solution is chosen, it is assumed that the agents can execute that solution, without any modification during execution (Felner et al. 2017). Recently, an online MAPF setting was suggested (ˇSvancara et al. 2019).

In this setting, new agents may appear over time and wish to join the problem space while existing agents are still executing their plans. This problem is relevant to real-world problems, such as autonomous intersections, robot warehouses, airport traffic and more.

The input for an online MAPF problem \( \langle G, A \rangle \) contains a graph, similarly to the offline MAPF problem, but instead of a set of agents it contains a stream of sets of agents \( \mathcal{A} = \{A_1, \ldots, A_n\} \). At each time step \( t > 0 \), a set \( A_t \) of 0 or more new agents is produced from the stream. \( n \) is the time of arrival of the last set of new agents. Each agent \( a_i \) is associated with a time of appearance \( TOA(a_i) \). Accordingly, each plan in an online MAPF solution starts at that agent’s time of arrival \((\forall a_i, (\pi_i[0] = \pi_i(TOA(a_i)) = s_i))\).

In online MAPF, while the stream of new agents \( \mathcal{A} \) is not exhausted, time is incremented by 1, resulting in time \( t \). The agents that are already in the problem space are advanced to their planned locations at \( t \), and if \( \mathcal{A} \) is queried to reveal the set \( A_t \) of new agents appearing at \( t \). If \( A_t \) is not empty, a new solution \( \pi^t \) that contains plans for both the existing agents (starting from their current positions) and the new agents, must then be found. It is important to note that the new agents are only revealed to an online solver at the time of their arrival.

The solution to an online MAPF problem is a sequence
of all the solutions found at the different times when new agents appeared \( \Pi = \{ \pi^0, ..., \pi^n \} \). A partial solution \( \pi^x \{x : y\} \) is the part of a solution \( \pi^x \) that is planned for time steps \( x, x + 1, ..., y \). The executed solution \( Ex(\Pi) \) represents the plans that the agents ended up following. It is derived from \( \Pi \) thusly \( Ex(\Pi) = \pi^0[t_1 = 0 : t_1 = 1] \circ \pi^1[t_2 = 1 : t_2 = 2] \circ ... \circ \pi^n[n : \infty] \), where \( \circ \) represents the concatenation of partial plans. An online MAPF solution \( \Pi \) is valid only if none of the plans in \( Ex(\Pi) \) has a conflict. The cost of an online solution is the sum of costs of its executed solution \( SOC(Ex(\Pi)) \).

ˇSvancara et al. (2019) defined two types of online MAPF: the warehouse model, and the intersection model. In this work we focus on the intersection model, where new agents appear over time and existing agents may leave the problem. Therefore, appearing agents are allowed to wait outside the problem space before entering, and agents leave the problem when they arrive at their goal. This is akin to agents appearing from within a private garage at their start location, and leaving into one at their goal location.

### 3.1 Offline-Optimal

Any online MAPF problem may be converted into an equivalent offline problem. This is simply done by informing the solver of all the agents that will appear in the problem, rather than hiding them until their time of appearance. By optimally solving the equivalent offline problem, we can find the lowest cost possible for a solution to the online problem. We call such a solution an offline-optimal solution. Naturally, online problems can not be solved offline in practice. However, the cost of the offline-optimal solution is useful for theoretical comparisons. We use these solutions to establish how close snapshot-optimal is to offline-optimal.

We implemented an offline-optimal solver for online MAPF, making the following modifications to CBS:

1. The problem’s graph is modified. A private vertex is added for each agent, and connected with a directed edge to that agent’s source location. This allows agents to wait before entering the problem space. The move from the private vertex to the start location is not included in the length of the agent’s plan.

2. Once an agent has reached its goal, it no longer conflicts with other agents. For instance, an agent may reach its goal \( g \) at time \( t \), and another agent may occupy \( g \) at time \( t + 1 \) without generating a conflict.

3. All agent plans are associated with a start time. Since agents appear at different times, it is no longer possible to infer conflicts using plan indices. As an example, assume two plans \( \pi_1 = [v_1, v_2, v_3] \) and \( \pi_2 = [v_4, v_3, v_2] \) for agents \( a_1 \) and \( a_2 \) respectively. If both plans start at time \( 0 \) (as is the case in offline MAPF), a swapping conflict would occur at time \( 2 \), on edge \( (v_2, v_3) \). If agent \( a_2 \) was to instead appear at time \( 1 \), the swapping conflict would not occur, but a vertex conflict would occur at time \( 2 \) on vertex \( v_3 \). If agent \( a_2 \) was to appear at any other time, no conflict would occur between these two agents.

### 3.2 Snapshot Optimal

An online solver for the online MAPF problem does not know when and what agents may appear in the future. Consequently, a solution with a lower cost than that of the executed solution may exist. No online MAPF solver can guarantee to find a solution with cost equal to the offline-optimal cost of an equivalent offline problem (ˇSvancara et al. 2019). An algorithms that always returns a solution for the current set of agents that is optimal assuming no new agents arrive in the future, is called snapshot optimal (ˇSvancara et al. 2019).

Two snapshot-optimal algorithms for online MAPF have been suggested:

- In Replan All (ˇSvancara et al. 2019), whenever new agents appear, a new solution is found by optimally replanning for all agents from their current positions. This means all existing plans are discarded.

- Online Independence Detection (OID) (ˇSvancara et al. 2019), based on Standley’s Independence Detection (Standley 2010), finds snapshot-optimal solutions, while attempting to minimise the number of times agents are re-planned for. In OID, whenever new agents arrive, they are each assigned to a singleton set, and planned for optimally. Then, if conflicts exist between two sets (containing new agents, existing agents, or both), they are joined into one set and re-planned optimally as a group. These set assignments and plans are then preserved so that they may be used again when new agents appear in the future.

Figure 1 demonstrates a scenario where the offline solver would find a better solution than a snapshot-optimal solver. Agent \( a_1 \) appears at time 0 and wishes to get from location \( s_1 \) to \( g_1 \). Agent \( a_2 \) appears at time 2, and wishes to get from \( s_2 \) to \( g_2 \). The offline solver would know about both agents in advance, and choose the plan that goes around the obstacle for agent \( a_1 \) (marked by a dashed line), and the direct plan for agent \( a_2 \) (marked by a solid right-facing arrow). Note that \( len(\pi_1^1) = 11 \), and \( len(\pi_2^1) = 5 \), resulting in a SOC of 16 for this solution. The snapshot-optimal solver does not know about agent \( a_2 \) at time 0, and so it chooses the direct plan for \( a_1 \) (marked by a left-facing solid line, with \( len(\pi_1^1) = 7 \)), because it results in a minimal SOC of 7 at that time. At time 2, agent \( a_1 \) has already moved one cell to the left, and agent \( a_2 \) appears. Now, a new snapshot-optimal solution is chosen. \( a_2 \) waits outside the problem space until \( a_1 \) passes location \( s_2 \) (\( len(\pi_2^2) = 10 \)), while \( a_1 \) executes the remainder of its existing plan (\( len(\pi_1^2) = 6 \)), for a SOC of 16 for the solution at time 2. However, \( SOC(Ex(\Pi)) = 17 \) as it includes the steps taken by \( a_1 \) before \( a_2 \) appeared, meaning the snapshot-optimal solution has a higher cost than the offline-optimal solution in this case.

### 4 Optimality in Online MAPF

In this work, we explore how optimal snapshot optimal is, by comparing offline optimality and snapshot optimality, both analytically and empirically.

Let \( P \) be an online MAPF problem. Let \( \Pi_{oo}(P) \) and \( \Pi_{so}(P) \) be the solutions to \( P \) generated by an offline-optimal solver and snapshot-optimal solver, respectively. Let
$P^+$ be an online MAPF problem that is identical to $P$ except that there is an additional agent $a_i$ and $\forall a_j \neq a_i : TOA(a_i) > TOA(a_j)$. Let $t^+$ be the time at which this new agent has appeared. Let $\Delta_{SOC}(P)$ be the difference in SOC between $\Pi_{so}(P)$ and $\Pi_{oo}(P)$, that is,

$$\Delta_{SOC}(P) = SOC(\Pi_{so}(P)) - SOC(\Pi_{oo}(P))$$

We assume that the snapshot-optimal solver is deterministic, always returning the same result given the same input.

In examining the differences between snapshot-optimal and offline-optimal, we see that the adding a new agent caused the cost difference to increase if $\Delta_{SOC}(P^+) > \Delta_{SOC}(P)$.

**Observation 1.** The following is a necessary condition for which $\Delta_{SOC}(P^+) > \Delta_{SOC}(P)$: For every minimum-cost single-agent plan of the new agent there exists a conflict with the single-agent plans of the other agents in $\Pi_{so}(P)$.

**Proof.** By contradiction, assume $\Delta_{SOC}(P^+) > \Delta_{SOC}(P)$ and the condition is not satisfied.

Because condition 1 is not satisfied, there exists a minimal length plan $\pi_i^*$ for the new agent $a_i$ that does not conflict with any plan in $\Pi_{oo}(P)$. Since $\Pi_{oo}(P)$ was already optimal and $\pi_i^*$ does not conflict with it, $\Pi_{so}(P) \cup \pi_i^*$ is a valid and optimal solution for $P^+$. Therefore

$$SOC(\Pi_{so}(P^+)) = SOC(\Pi_{so}(P)) + len(\pi_i^*)$$

Remember that $\Delta_{SOC}(P^+) > \Delta_{SOC}(P)$. Therefore,

$$SOC(\Pi_{so}(P^+)) - SOC(\Pi_{so}(P)) > SOC(\Pi_{oo}(P^+)) - SOC(\Pi_{oo}(P)) = len(\pi_i^*)$$

and so

$$SOC(\Pi_{so}(P^+)) > SOC(\Pi_{so}(P)) + len(\pi_i^*)$$

$\Pi_{so}(P) \cup \pi_i^*$ is a valid solution for $P^+$ and its cost is

$$SOC(\Pi_{so}(P)) + len(\pi_i^*)$$

Therefore $\Pi_{so}(P^+)$ is not a snapshot-optimal solution to $P^+$, contradicting the assumption.

**Observation 2.** The following is a necessary condition for which $\Delta_{SOC}(P^+) > \Delta_{SOC}(P)$: For every snapshot-optimal solution to $P^+$ either there exists an old agent whose single-agent plan is longer than its single-agent plan in $\Pi_{so}(P)$, or the single-agent plan for new agent is not a minimum-length plan for the new agent.

**Proof.** By contradiction, assume $\Delta_{SOC}(P^+) > \Delta_{SOC}(P)$ and the condition is not satisfied. Let $len(\pi_i^*)$ be the minimal length of a plan for agent $a_i$.

Because condition 2 is not satisfied,

$$SOC(\Pi_{oo}(P^+)) = SOC(\Pi_{oo}(P)) + len(\pi_i^*)$$

$\Pi_{so}(P)$ is an optimal solution for the group of agents in $P$, so with the addition of agent $a_i$ in $P^+$,

$$SOC(\Pi_{so}(P^+)) \geq SOC(\Pi_{oo}(P)) + len(\pi_i^*)$$

So,

$$SOC(\Pi_{oo}(P^+)) - SOC(\Pi_{so}(P)) \geq len(\pi_i^*)$$

$$= SOC(\Pi_{so}(P^+)) - SOC(\Pi_{so}(P))$$

Meaning,

$$SOC(\Pi_{so}(P^+)) - SOC(\Pi_{oo}(P^+)) \leq SOC(\Pi_{so}(P)) - SOC(\Pi_{oo}(P))$$

Contradicting the assumption that

$$\Delta_{SOC}(P^+) > \Delta_{SOC}(P)$$

A solution $\pi$ is applicable at time $t$ for an online solution $\Pi$ if $\forall a_i : Ext(\Pi)_i(t) = \pi_i(t)$. This includes agents that have already reached their goal.

A solution $\Pi'$ is a prefix of $\Pi$ up to time step $t$ if it is applicable for every time step $t' < t$ for $\Pi$.

**Observation 3.** The following is a necessary condition for $\Delta_{SOC}(P^+) > \Delta_{SOC}(P)$: For every offline-optimal solution $\Pi_{oo}(P^+)$ to $P^+$ and for every snapshot optimal solution $\Pi_{so}(P)$ to $P$ it holds that $\Pi_{so}(P)$ is not a prefix of $\Pi_{oo}(P^+)$ up to time step $t^+$.

**Proof.** By contradiction, assume $\Delta_{SOC}(P^+) > \Delta_{SOC}(P)$ and the condition is not satisfied.

$\Pi_{oo}(P^+)$ is an offline-optimal solution $\Pi_{oo}(P^+)$ and snapshot optimal solution $\Pi_{so}(P)$ such that $\Pi_{so}(P)$ is a prefix of $\Pi_{oo}(P^+)$. $\Pi_{so}(P^+)$ can be split into two partial solutions: $\Pi_{oo}(P^+)[0 : t^+ - 1]$ and $\Pi_{oo}(P^+)[t^+ : \infty]$. Because $\Pi_{so}(P)$ is a prefix of $\Pi_{oo}(P^+)$,

$$SOC(\Pi_{oo}(P^+)[0 : t^+ - 1]) = SOC(\Pi_{so}(P))[0 : t^+ - 1])$$

Be definition,

$$SOC(\Pi_{oo}(P^+)) \leq SOC(\Pi_{so}(P^+))$$

So,

$$SOC(\Pi_{oo}(P^+)[0 : t^+ - 1]) + SOC(\Pi_{oo}(P^+)[t^+ : \infty]) \leq SOC(\Pi_{so}(P))[0 : t^+ - 1]) + SOC(\Pi_{so}(P^+))[t^+ : \infty]$$

But $\Pi_{so}(P^+)$ is applicable at $t^+ - 1$ for $\Pi_{so}(P)$, so by the definition of snapshot-optimal,

$$SOC(\Pi_{so}(P^+)[t^+ : \infty]) \geq SOC(\Pi_{so}(P^+))[t^+ : \infty]$$

Therefore,

$$SOC(\Pi_{oo}(P^+)[t^+ : \infty]) = SOC(\Pi_{so}(P^+))[t^+ : \infty]$$
\[ SOC(\Pi_{so}(P^+))[t^+ : \infty] - SOC(\Pi_{so}(P^+))[t^+ : \infty] = \Delta SOC(P^+) = 0 \]

By definition, \( \Delta SOC(P) \geq 0 \), so, contrary to the assumption that \( \Delta SOC(P^+) > \Delta SOC(P) \),

\[ \Delta SOC(P^+) \leq \Delta SOC(P) \]

From these observations we conclude that for a significant difference in quality to exist between an offline-optimal solution and a snapshot-optimal solution for an online MAPF problem, that problem would have to contain many situations where many of the conditions are met. Such situations can be contrived, however, we hypothesize that they are rare in practice, and conduct experiments to examine that hypothesis.

As mentioned before, the cost of an online solution is the sum of costs of its executed solution. Additionally, one might consider the number of reroutes in the solution. A reroute is when an agent has a plan \( \pi_i^1 \) at time \( t_1 \), and then at time \( t_2 \) it is replaced by a new plan which is different from the existing partial plan \( \pi_i^1[t_1 : \infty] \). Rerouting an agent may incur real world costs, such as congestion of wireless networks. There exist online MAPF solvers that find sub-optimal solutions, snapshot-optimal solutions, or snapshot-optimal solutions with a minimal number of reroutes (Svanecara et al. 2019; Ma et al. 2017). We suggest a new cost function called Cost Of Reroute (COR), which balances minimizing SOC and the number of reroutes.

COR is a cost function over a solution for a set of agents in a MAPF problem. COR gives a cost to the action of rerouting an agent, i.e. changing its current plan. Since there is no concept of rerouting in offline MAPF, COR is only meaningful in an online MAPF setting. COR is meant to be used in conjunction with a standard cost function, such as Sum Of Costs (SOC).

Formally: Let \( A_{t_1} = \{a_1, \ldots, a_k\} \) be the group of agents that are currently executing their individual plans, in the current solution \( \pi_i^1 \). Let \( A_{t_2} \) be a group of new agents who are joining the system at time \( t_2 > t_1 \). Let \( \pi_i^2 \) be a new valid plan, for the agent group \( A_{t_1} \cup A_{t_2} \).

We will first define the helper function \( \text{isRerouted}(a_i, \pi_i^1, \pi_i^2) \) which indicates whether an agent’s plan changed between two consecutive solutions:

\[
\text{isRerouted}(a_i, \pi_i^1, \pi_i^2) = \begin{cases} 
0, & \pi_i^1[t_1 : \infty] = \pi_i^2 \\
1, & \text{otherwise}
\end{cases}
\]

We notice that if agent \( a_i \) reaches its goal before time \( t_1 \), then \( \pi_i^1[t_1 : \infty] = \emptyset = \pi_i^2 \). The COR of a solution found at time \( t_2 \) given the current solution \( \pi_i^1 \) is defined thusly, where \( \alpha \) is a parameter:

\[
COR(\pi_i^1, \pi_i^2, \alpha) = \sum_{a_i \in A_{t_1}} (\alpha \cdot \text{isRerouted}(a_i, \pi_i^1, \pi_i^2))
\]

Similarly, the COR of a solution to the online MAPF problem is the sum of the COR of the solutions contained within:

\[
COR(\Pi, \alpha) = \sum_i (COR(\Pi[i], \Pi[i + 1], \alpha))
\]

It is simple to incorporate COR into an online MAPF solver, by using the sum of an existing cost function and COR, as a new cost function, and adjusting \( \alpha \) to suit system demands. For instance, for a solver that uses \( SOC(\pi_i^2) \) as the cost of a solution found at time \( t_2 \), the new cost function would be \( \text{COR}(\pi_i^1, \pi_i^2, A, \alpha) = SOC(\pi_i^2) + COR(\pi_i^1, \pi_i^2, A, \alpha) \). With such a cost function, setting \( \alpha \leftarrow y \) would mean that rerouting a single agent is as costly to the system, as the solution including \( y \) more steps (distributed in some way amongst the executed plans of the various agents).

5 Online MAPF with Priorities

A delay caused to one agent may more harmful than a delay caused to another agent. For instance, if planning paths for autonomous vehicles in a city, it may be necessary to prioritise emergency vehicles over passenger vehicles. MAPF with Priorities (Okoso, Otaki, and Nishi 2019) is a setting where each agent is associated with a priority \( \text{Pri}(a_i) \in \mathbb{R} \).

In this setting, the cost of each agent’s plan is defined as the length of the agent’s plan, multiplied by its priority. Thus, the Sum of Costs with Priorities (SOPCP) of a solution \( \pi \) to a MAPF with priorities problem with \( k \) agents is

\[
SOPCP(\pi) = \sum_{i=1}^{k} \text{len}(\pi_i) \cdot \text{Pri}(a_i)
\]

Since our goal is to reduce delays caused to high-priority agents, we may alternatively consider only the delay caused to each agent, multiplied by its priority. The delay of the plan of an agent \( a_i \) is marked \( \Delta(\pi_i) \). The length of an optimal plan for an agent \( a_i \) when disregarding all collisions with other agents is marked \( \text{IndOpt}(a_i) \), so \( \Delta(\pi_i) = \text{len}(\pi_i) - \text{IndOpt}(a_i) \). We call this cost function the Sum of Delays with Priorities (SODP), and define it as

\[
SODP(\pi) = \sum_{i=1}^{k} \Delta(\pi_i) \cdot \text{Pri}(a_i)
\]

\( \text{IndOpt}(a_i) \) is a constant value for each agent. Therefore, minimizing SOPCP and minimizing SODP are equivalent as objective functions. However, when evaluating the quality of solutions, using SOPCP can show the impact of delays on overall path lengths, whereas using SODP may highlight the delays caused by conflicts between agents.

We propose a novel solver for the online version of MAPF with priorities, called StratifiedPriorities. In StratifiedPriorities, agents are separated into groups based on their priority, so that each group contains agents of the same priority. StratifiedPriorities then orders the groups by priority, and finds solutions for each group, in order, starting from the highest priority, using any MAPF solver. If StratifiedPriorities uses prioritised planning as its MAPF solver, it behaves similarly.
to CA *pri (Okoso, Otaki, and Nishi 2019). After a solution for a group of agents is found, it is protected in a reservation table, and a solution must then be found for the next group without conflicting with the solutions in the reservation table. If a solution cannot be found for an agent group, it is merged with the previous agent group, the solution for the previous group is removed from the reservation table, and they are solved together. For online MAPF, this process is repeated whenever new agents arrive.

Theorem 5.1. If StratifiedPriorities uses a complete MAPF solver, StratifiedPriorities is complete.

Proof outline. Let $S$ be a complete MAPF solver. Let $[A_1, \ldots, A_n]$ be $n$ groups of agents, each containing agents of equal priority, and sorted by group priority in ascending order. StratifiedPriorities first looks for a solution to $A_n$. Since $S$ is complete and the reservation table is empty, a solution will be found for $A_n$. StratifiedPriorities then looks for a solution to $A_{n-1}$. If it cannot find a solution, the reservation table will be emptied, and a solution will be found for $A_n \cup A_{n-1}$ because $S$ is complete. It can be proven through induction that StratifiedPriorities will find a solution for all agents in $\bigcup [A_1, \ldots, A_n]$.

6 Experimental Results

In this section we show experiments comparing the quality of offline-optimal and snapshot-optimal solutions, and comparing different solvers for cost of reroute.

We used problem instances from a common MAPF benchmark (Stern et al. 2019), which we modified to create online MAPF problem instances. The benchmark contains a set of grid-maps, and a set of (offline) MAPF scenarios for each map. The maps are typically seen as 4-connected grids. We chose to use the offline problems in the benchmark as a base, and made the following adjustments to create online MAPF instances:

1. Each agent may have a unique pair of source and goal locations, or they may repeat. For each online scenario a set of 20 agents was selected randomly from an existing offline scenario, and used to create 20 source-goal pairs.

2. If source-goal pairs are repeated, they may repeat in different ways. Every time a new agent appears, a source-goal pair is drawn from a standard normal distribution over the set of pairs, meaning some pairs would repeat often and others would repeat seldomly. We chose this method as we felt it would reflect real-world scenarios.

3. Agents may appear at different frequencies. For every time step, the number of new agents that appear was drawn from a Poisson distribution. We experimented with different rates of appearance, which is expressed as different values for the $\lambda$ parameter of the Poisson distribution.

6.1 Offline-Optimal and Snapshot-Optimal

We compared the quality of offline-optimal and snapshot-optimal solutions on 50 online MAPF instances per map and appearance rate, for appearance rates ($\lambda$) ranging from 0.05 to 1.5 agent per time step, with 40 agents. We used two 32x32 4-connected grid maps: ‘random-32-32-20’, which has 20% random obstacles, and ‘room-32-32-4’ which is comprised of randomly connected 4x4 rooms (see Figure 2). The solvers were each allowed 300 seconds to solve each instance. For the instances solved by both solvers and on all arrival rates, we measured the average cost of offline-optimal and snapshot-optimal solutions, the proportion of instances where a cost difference was observed, the average number of reroutes in snapshot-optimal solutions, and the proportion of instances that included reroutes among snapshot-optimal solutions. The results of this experiment are found in table 1. Each row shows the same measurements for instances with different appearance rates.

6.2 Cost of Reroute Experiments

We compared the solution quality of 3 solvers: SNAPSHOT, COR-SNAPSHOT, and Replan Single (RS). SNAPSHOT is a snapshot-optimal solver that is blind to COR. It represents one extreme, where the solver makes as many reroutes as is necessary to minimise SOC. RS is a polynomial-time sub-optimal solver where new agents are simply integrated by planning for them while preserving the existing agents’ plans (Švancara et al. 2019). It represents another extreme, where no reroutes are made, regardless of the impact on SOC. COR-SNAPSHOT represents a compromise between the two extremes, by minimising COMBINED-COST, which incorporates both SOC and COR.

We ran 50 online MAPF instances per map with an appearance rate of 1 agent per time step, with 40 agents, on the same problems presented in table 1. Each solver was allowed 300 seconds to solve each instance. The results of this experiment are shown in Figure 3, where the y-axis represents the average COMBINED-COST of instances that were solved by all solvers, and the x-axis represents different values for the $\alpha$ parameters of the COR function. Therefore,
the right side of the graph represents higher costs for rerouting an agent. We see that as \( \alpha \) increases, SNAPSHOT performs worse in terms of COMBINED-COST, and Replan Single is unaffected. This is because Replan Single performs no reroutes, while SNAPSHOT performs an unconstrained number of reroutes. COR-SNAPSHOT performs as well as SNAPSHOT for low \( \alpha \) values, and as well as Replan Single for high \( \alpha \) values. However, the difference between Replan Single and SNAPSHOT for low \( \alpha \) values is small.

These results show that when considering the cost of reroute, sub-optimal solutions that minimise the number of reroutes may be superior to snapshot-optimal solutions that only minimise SOC, even when the cost of rerouting an agent is small. We think that similar results would be seen in any sufficiently open domain, and that sub-optimal solvers may therefore be more appropriate for this problem.

### 6.3 Online MAPF with Priorities Experiments

We compared the quality, as well as coverage, of five solvers for online MAPF with priorities: Offline-Optimal, Snapshot-Optimal, StratifiedPriorities with Snapshot (CBS-based), StratifiedPriorities with Prioritised Planning, and Replan Single. All solvers were given SOC as the cost function to minimise. For this experiment, we used maze-15-15, a 15 by 15 grid map of a maze with loops (see Fig 2). We compared the different solvers across 100 online MAPF instances for this map. Online MAPF instances were created with 80% of the agents given priority 1, and the remaining 20% given priority 10. Other then this change, the experiment protocol is the same as described in subsection 6.1.

Table 2 shows the results of this experiment. SOC and SODP columns show average cost, and coverage columns show the percent of instances that were successfully solved within the 300 second time limit. In terms of SOC cost, SODP cost, and coverage percent, we notice the following relation between solvers: Offline-Optimal < Snapshot-Optimal < StratifiedPriorities with Snapshot < Stratified-Priorities with Prioritised Planning < Replan Single. This demonstrates a relation in online MAPF with priorities between optimality guarantees, and computational difficulty and solution quality. Offline-Optimal guarantees offline-optimal solutions, and indeed had the lowest coverage and solution costs. Snapshot-Optimal finds snapshot-optimal solutions, considering both path lengths and priority. StratifiedPriorities with Snapshot attempts to prioritise high priority agents, but also finds snapshot-optimal solutions within groups of agents with equal priority. StratifiedPriorities with Prioritised Planning only prioritises high priority agents, with no guarantee on optimality. Finally, Replan Single does not prioritise high priority agents and provides not optimality guarantee, and accordingly has the highest coverage and solution costs.

Apart from Offline-Optimal, which is not applicable for real-world online MAPF, the tested solvers represent different options for the trade-off between computational efficiency and solution quality for online MAPF with priorities. For real-world applications, it is possible to first solve the problem with the most computationally efficient solver (Replan Single), and then use the rest of the available time to improve the solution using increasingly demanding solvers.

### 7 Conclusions and Future Work

We examined the matter of optimality in online Multiagent Path Finding. We defined offline-optimality in the online MAPF problem, and compared it to snapshot-optimality both analytically and empirically. We conclude that snapshot-optimal solutions are usually very similar in quality to offline-optimal solutions. We further evaluated a setting where rerouting an agent carries a specific cost. We found that in the tested scenarios, solving sub-optimally while completely avoiding reroutes produced higher quality solutions under most reroute costs. However, our suggested solver, SNAPSHOT-COR achieved average quality that is similar to the best of both other solvers for any cost of reroute. Finally, we examined a setting where agents are assigned different priorities. We suggested a new solver, StratifiedPriorities, and compared it and other solvers for this problem. We showed a trade-off between solution quality and problem coverage that should be considered when choosing between the solvers. Future work may consider different costs that are uniquely relevant to online MAPF, such as fairness, or the cost of delaying an agent after it had already been planned to arrive at its goal at a certain time.

<table>
<thead>
<tr>
<th>Map</th>
<th>Rate</th>
<th>Offline-Optimal</th>
<th>Snapshot-Optimal</th>
<th>Stratified w/ Snapshot</th>
<th>Stratified w/ Priorities Planning</th>
<th>Replan Single</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SOC</td>
<td>SOCP</td>
<td>SODP</td>
<td>SOCP</td>
<td>SODP</td>
<td>SOCP</td>
</tr>
<tr>
<td>maze-15-15</td>
<td>0.5</td>
<td>941.94</td>
<td>3.28</td>
<td>941.94</td>
<td>1.74</td>
<td>941.94</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>948.83</td>
<td>10.17</td>
<td>948.83</td>
<td>11.17</td>
<td>951.11</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>955.33</td>
<td>14.67</td>
<td>955.33</td>
<td>15.17</td>
<td>953.94</td>
</tr>
</tbody>
</table>

Table 2: Comparison of solutions using different solvers, for 40 agents with 80% priority 1 agents and 20% priority 10 agents, with varying appearance rates, showing average SOCP, SODP, and coverage.

Figure 3: Average COMBINED-COST of RS, COR-SNAPSHOT, SNAPSHOT as a function of alpha, for 40 agents, on room-32-32-4 (left) and Random-32-32-20 (right).
References


