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

Exploiting Geometric Constraints in Multi-Agent Pathfinding

Dor Atzmon[1], Sara Bernardini[1], Fabio Fagnani[2], David Fairbairn[3]

[1]Royal Holloway, University of London, United Kingdom
[2]Politecnico di Torino, Italy
[3]Tharsus Limited, United Kingdom
dor.atzmon@rhul.ac.uk, sara.bernardini@rhul.ac.uk, fabio.fagnani@polito.it, david.fairbairn@tharsus.co.uk

Abstract

In tackling the multi-agent pathfinding problem (MAPF), we study a specific class of paths that are constructed by taking the agents' shortest paths from the start to the goal locations and adding *safe delays* at the beginning of the paths, which guarantee that they are non-conflicting. Safe delays are calculated by exploiting a set of fundamental *geometric constraints* among the distances between all agents' start and goal locations. Those constraints are simple, but the MAPF problem reformulated in terms of them remains computationally hard. Nonetheless, based on safe delays, we devise a new, fast and lightweight algorithm, called Delayed Shortest Path (DSP), to find solutions to the MAPF problem. Via an extensive experimental evaluation on standard benchmarks, we show that, in many cases, our technique runs several orders of magnitudes faster than related methods while addressing problems with thousands of agents and returning low-cost solutions.

1 Introduction

Considering a set of agents that simultaneously move in an environment, *Multi-Agent Pathfinding* (MAPF) (Stern et al. 2019) is the problem of finding conflict-free paths (a *plan*), where each path leads an agent from its start position to its end position. Conflicts between agents occur when two agents are simultaneously at the same location or want to swap locations. MAPF is relevant in several real-world applications, from self-driving cars (Wen, Liu, and Li 2022) to automated warehouses (Li et al. 2020) and squads of flying drones (Ho et al. 2019). In MAPF, the path's cost is equal to its length and a common way to define the plan cost is to consider the sum of the costs of its paths. Finding the lowest-cost (optimal) solution for MAPF is NP-complete (Surynek 2010; Yu and LaValle 2013). Nevertheless, a number of algorithms solve MAPF optimally for many agents, including *M** (Wagner and Choset 2015), *Increasing Cost Tree Search (ICTS)* (Sharon et al. 2013), and *CBS* (Sharon et al. 2015).

In this paper, we devise a simple technique, which we call *Delayed Shortest Path* (DSP), based on the idea that non-conflicting paths can be generated by considering the agents sequentially according to a given order, taking their shortest paths from start to goal location and adding delays at the beginning of the paths to avoid conflicts. The algorithm *SEQ* (Sequence) (Ma 2021) does that but, for each agent, naively chooses a relative delay that is equal to the length of the shortest path of the agent that precedes it in the order. Hence, in SEQ, one agent at a time follows its shortest path to destination while the other agents wait. We improve SEQ by calculating *safe delays*, i.e. relative delays that are possibly shorter than the shortest path of the previous agent in the order but are long enough to safely avoid conflicts.

We calculate safe delays by identifying a set of fundamental *geometric constraints* among the distances between all start and goal locations of the agents that, when satisfied, guarantee that shortest paths are non-conflicting. Starting from safe delays, which we show are the best possible delays that can be calculated based on information about the agents' start and goal locations, we approximate *minimal* delays via an iterative method. The resulting algorithm is lightweight and fast and can handle thousands of agents at a cost that is often several orders of magnitude lower than related techniques. Our algorithm has several benefits. It is effective while using very little information about the agents (their start and goal locations and corresponding shortest paths), and involving no planning. It also minimizes the time the agents spend moving on the environment and, hence, their use of resources (e.g. battery) as, when they move, they always follow their shortest path to destination. Finally, our technique does not impose the agents to take any particular shortest paths among those available, which means that the agents can preserve their privacy. In some applications (e.g. self-driving cars), this is a valuable feature.

After introducing MAPF and safe delays formally in Sec. 2 and 3, we present the geometric constraints to calculate safe delays in Sec. 4. In Sec. 5, we prove that finding minimal safe delays is NP-complete and put forward the DSP algorithm that finds approximate solutions to such a problem in Sec. 6. To offer evidence of the power of our technique, in Sec. 7, we present an extensive experimental evaluation on several benchmark grids. We end our paper with related work (Sec. 8) and conclusions (Sec. 9).

2 Multi-Agent Pathfinding

We consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a finite set of k agents $\mathcal{A} = \{1, \ldots, k\}$. Every agent $i \in \mathcal{A}$ is associated with a start vertex $s_i \in \mathcal{V}$ and a goal vertex $g_i \in \mathcal{V}$, with $s_i \neq$

g_i. For any two agents i and j ($i \neq j$), $s_i \neq s_j$ and $g_i \neq g_j$. Given these vectors collecting all start and goal vertices s and g, we call the tuple $(\mathcal{G}, k, s, g)$ a *MAPF instance*.

A *path* in $\mathcal{G}$ from $x \in \mathcal{V}$ to $y \in \mathcal{V}$ is any function $\pi : \{a, a+1, \ldots, b\} \to \mathcal{V}$ such that $(\pi(t), \pi(t+1)) \in \mathcal{E}$ for every $t = a, \ldots, b-1$ and such that $\pi(a) = x$ and $\pi(b) = y$. We assume that $\exists (x, x) \in \mathcal{E}$ for every $x \in \mathcal{V}$. This means that, at every step, an agent following a path can move to an adjacent vertex or remain still at its current vertex. We denote $a_\pi = a$, $b_\pi = b$, and $l_\pi = b - a$ to be, respectively, the starting time, the arrival time and the length of path π. We exclusively consider paths with starting time $a_\pi \geq 0$.

Given $x, y \in \mathcal{V}$, we indicate with $d(x, y)$ the geodetic distance between x and y in $\mathcal{G}$, namely, the minimum length of any path from x to y. A common assumption in MAPF is that $d(x, y)$ between any two vertices x and y is known. This information can be calculated in advance in a preprocessing computation and stored in memory for later use.

Let Π_i be the set of all minimum length paths π in the graph from s_i to g_i and Π_i^t the set of minimum length paths from s_i to g_i with starting time equal to t (specifically, Π_i^0 means starting time 0). Given two paths $\pi_i \in \Pi_i$ and $\pi_j \in \Pi_j$ of agents $i, j \in \mathcal{A}$, let $a_{\pi_i,\pi_j} = \max(a_{\pi_i}, a_{\pi_j})$, $b_{\pi_i,\pi_j} = \min(b_{\pi_i}, b_{\pi_j})$ be, respectively, the maximum starting time and the minimum arrival time of the two paths.

We assume that all agents are present outside the environment in the beginning; each agent can start moving immediately at time 0 or can be delayed outside the environment. Hence, an agent moving according to a path $\pi : \{a, a+1, \ldots, b\} \to \mathcal{V}$ is only present in the environment at times between a and b, while it is outside the environment at all the other times. This assumption, which is common in warehouse environments, implies that an agent starts moving when it appears at s_i (at 0 or after a delay) and disappears when it arrives at g_i (*disappear at target* (Stern et al. 2019)).

We have the following definition of conflict (here and below, intervals are always intended on the integers).

Definition 1. *Two paths $\pi_i \in \Pi_i$ and $\pi_j \in \Pi_j$ are called* non-conflicting *if the following two conditions hold:*

(a) $\forall t \in [a_{\pi_i,\pi_j}, b_{\pi_i,\pi_j}] : \quad \pi_i(t) \neq \pi_j(t)$

(b) $\forall t \in [a_{\pi_i,\pi_j}, b_{\pi_i,\pi_j} - 1] : \quad (\pi_i(t), \pi_i(t+1)) \neq (\pi_j(t+1), \pi_j(t))$

The conflict defined in condition (a) is called a *vertex conflict*, where two agents are simultaneously positioned at the same vertex, and the conflict in condition (b) is called a *swapping conflict*, where two agents swap positions in two consecutive time steps.

Definition 2. *A* solution *to a MAPF instance $(\mathcal{G}, k, s, g)$ is a family of paths $\Pi = \{\pi_i \in \Pi_i \mid i \in \mathcal{A}\}$ such that, for every $i, j \in \mathcal{A}$ with $i \neq j$, π_i and π_j are non-conflicting. The cost of a solution is defined as follows:*

$$C(\Pi) = \sum_{i \in \mathcal{A}} b_{\pi_i}$$

This cost is known as *sum-of-costs*, which is one of the most commonly used metrics in MAPF (Stern et al. 2019).

We denote the set of solutions to the MAPF instance $(\mathcal{G}, k, s, g)$ by $\mathcal{M}_{(\mathcal{G},k,s,g)}$. MAPF aims to solve the following optimization problem:

$$C_{(\mathcal{G},k,s,g)} = \min \left\{ C(\Pi) \mid \Pi \in \mathcal{M}_{(\mathcal{G},k,s,g)} \right\} \tag{1}$$

3 Safe Delays

A simple way to generate solutions to a MAPF instance is to start from a family of minimum-length paths, one for each agent, with starting time equal to 0 and then delay the entrance of some of the agents into the environment to avoid conflicts. We formalize this idea below.

Given $i \in \mathcal{A}$, $\pi \in \Pi_i^0$, and $\tau \in \mathbb{N}_0$, we define the mapping $\pi^\tau : [a_\pi + \tau, b_\pi + \tau] \to \mathcal{V}$ as follows:

$$\forall t \in [a_\pi + \tau, b_\pi + \tau] : \quad \pi^\tau(t) = \pi(t - \tau)$$

Hence, τ is the *delay* of agent i. When $\tau > 0$, the agent waits $\tau - 1$ time steps outside the environment and then starts its motion from s_i to g_i at time τ. We now give the concept of *safe delays*, which are introduced to avoid conflicts.

Definition 3. *A family of delays $\{\tau_i \mid i \in \mathcal{A}\}$ is called* safe *for a MAPF instance $(\mathcal{G}, k, s, g)$ if, for every choice of $\pi_i \in \Pi_i^0$, the set of paths $\{\Pi_i^{\tau_i} \mid i \in \mathcal{A}\}$ is a solution to $(\mathcal{G}, k, s, g)$. We assemble families of delays into vectors $\tau \in \mathbb{N}_0^k$ and call them* delay assignments.

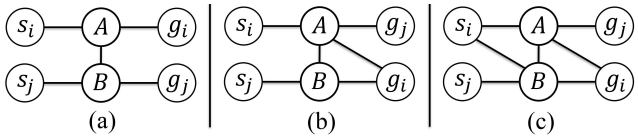

Figure 1: Examples for safe delays.

Example 1. *Figure 1 illustrates three instances with two agents. The shortest paths are: (a) $\pi_i = (s_i, A, g_i)$ and $\pi_j = (s_j, B, g_j)$; (b) $\pi_i = (s_i, A, g_i)$ and $\pi_j = (s_j, B, A, g_j)$; (c) $\pi_i = (s_i, A, g_i)$ (or (s_i, B, g_i)) and $\pi_j = (s_j, B, A, g_j)$. In (a), all values of τ_i and τ_j are safe. In (b), for instance, $\tau_i = 0$ and $\tau_j = 0$ are safe, while $\tau_i = 1$ and $\tau_j = 0$ are not safe. In (c), for instance, $\tau_i = 0$ and $\tau_j = 1$ are safe, while $\tau_i = 0$ and $\tau_j = 0$ or $\tau_i = 1$ and $\tau_j = 0$ are not safe.*

Our aim is to give sufficient conditions for delay assignments to be safe so that it is easy to generate solutions for a MAPF instance. To keep our approach simple and lightweight, we put forward conditions that are purely based on *fundamental geometric relationships* between the start and goal vertices of the agents involved in a MAPF instance. To express this geometric information, we augment MAPF instances $(\mathcal{G}, k, s, g)$ with a set of three $k \times k$ matrices $\Delta = (\Delta^s, \Delta^{sg}, \Delta^g)$ and call Δ the *distance profile* of $(\mathcal{G}, k, s, g)$. The matrix Δ^s contains all possible pairwise distances between the agents' start vertices; Δ^{sg} contains all pairwise distances between the agents' start and goal vertices; and Δ^g contains all pairwise distances between the agents' goal vertices. In formulas,

$$\Delta_{ij}^s = d(s_i, s_j), \ \Delta_{ij}^{sg} = d(s_i, g_j), \ \Delta_{ij}^g = d(g_i, g_j) \tag{2}$$

Often, we work with distance profiles Δ without referring to any particular MAPF instance. A triple of non-negative matrices $\Delta = (\Delta^s, \Delta^{sg}, \Delta^g)$ (with $\Delta^{sg}_{ii} > 0$ for every i) is the distance profile of some MAPF instance if and only if all the following inequalities, corresponding to distance triangular inequalities between the four points, are satisfied.

$$\begin{array}{l}
\Delta^s_{ij} \le \Delta^{sg}_{ii} + \Delta^{sg}_{ji},\ \Delta^s_{ij} \le \Delta^{sg}_{jj} + \Delta^{sg}_{ij} \\
\Delta^g_{ij} \le \Delta^{sg}_{ii} + \Delta^{sg}_{ij},\ \Delta^g_{ij} \le \Delta^{sg}_{jj} + \Delta^{sg}_{ji} \\
\Delta^{sg}_{ii} \le \Delta^s_{ij} + \Delta^{sg}_{ji},\ \Delta^{sg}_{ii} \le \Delta^g_{ij} + \Delta^{sg}_{ij} \\
\Delta^{sg}_{jj} \le \Delta^s_{ij} + \Delta^{sg}_{ij},\ \Delta^{sg}_{jj} \le \Delta^g_{ij} + \Delta^{sg}_{ji} \\
\Delta^{sg}_{ij} \le \Delta^s_{ij} + \Delta^{sg}_{jj},\ \Delta^{sg}_{ij} \le \Delta^g_{ij} + \Delta^{sg}_{ii} \\
\Delta^{sg}_{ji} \le \Delta^s_{ij} + \Delta^{sg}_{ii},\ \Delta^{sg}_{ji} \le \Delta^g_{ij} + \Delta^{sg}_{jj}
\end{array} \tag{3}$$

We call $\mathcal{T}_\Delta$ the set of delay assignments that are safe with respect to any MAPF instance having distance profile Δ and have the following definition.

Definition 4. *Given a distance profile Δ, we denote $\mathcal{T}_\Delta$ the set of delay assignments τ such that, for every MAPF instance $(\mathcal{G}, k, s, g)$ whose distance profile is Δ and for every choice of shortest paths $\pi_i \in \Pi^0_i$, the set of paths $\{\Pi^{\tau_i}_i \,|\, i \in \mathcal{A}\}$ is a solution to $(\mathcal{G}, k, s, g)$.*

Given a distance profile Δ, our goal is to analyze the following optimization problem:

$$C_\Delta = \min_{\tau \in \mathcal{T}_\Delta} \sum_{i \in \mathcal{A}} \tau_i \tag{4}$$

Notice that, by construction, for every MAPF instance $(\mathcal{G}, k, s, g)$ whose distance profile is Δ, we have that

$$C_{(\mathcal{G},k,s,g)} \le C_\Delta + \sum_{i \in \mathcal{A}} d(s_i, g_i) \tag{5}$$

To study the problem in expression (4), we need an explicit description of $\mathcal{T}_\Delta$, which is given in the next section.

4 Geometric Constraints

Recall that the non-conflicting property between two paths given in Definition 1 is expressed in terms of pairwise conditions. A similar pairwise description holds for the set $\mathcal{T}_\Delta$. If we denote by $\Delta^{(ij)}$ the triple of matrices $(\Delta^s, \Delta^{sg}, \Delta^g)$ restricted to agents i and j, we have that

$$\tau \in \mathcal{T}_\Delta \ \Leftrightarrow\ (\tau_i, \tau_j) \in \mathcal{T}_{\Delta^{(ij)}}\ \forall i \neq j \tag{6}$$

We now give an explicit description of the sets $\mathcal{T}_{\Delta^{(ij)}}$. To this aim, we introduce two additional $k \times k$ matrices Λ and Ψ defined as follows:

$$\Lambda_{ij} = \Delta^{sg}_{ii} - \Delta^{sg}_{ji}, \quad \Psi_{ij} = \Delta^s_{ij} + \Delta^g_{ij} - \Delta^{sg}_{ii} - \Delta^{sg}_{jj}$$

Theorem 5 below shows that the delays $\mathcal{T}_{\Delta^{(ij)}}$ can be established based on Λ_{ij} and Ψ_{ij}: if $\Psi_{ij} > 0$, then agents i and j cannot possibly interfere with one another and so any delay can be chosen for τ_i and τ_j. However, if $\Psi_{ij} \le 0$, agents i and j can potentially conflict and, to avoid that, τ_i and τ_j need to be chosen appropriately. Note that the theorem gives *necessary and sufficient conditions* for the delays to be safe, which implies that no additional delays between two agents can be calculated based on the available information $\Delta^{(ij)}$[1].

[1]The conditions can be easily reformulated under different assumptions than those made in this paper.

Example 2. *Consider again our examples in Figure 1. We can see that (a) $\Psi_{ij} = 2, \Lambda_{ij} = -1, \Lambda_{ji} = -1$; (b) $\Psi_{ij} = 0, \Lambda_{ij} = 0, \Lambda_{ji} = 1$; (c) $\Psi_{ij} - 1, \Lambda_{ij} = 0, \Lambda_{ji} = 1$. In (a), as mentioned, any pair of delays is safe. However, in (b), τ_i and τ_j are safe if $\tau_j - \tau_i \in (-\infty, -1) \cup [0, +\infty)$ (defined formally below), which means that $\tau_j - \tau_i = -1$ is not safe (e.g., $\tau_i = 1$ and $\tau_j = 0$). And, in (c), $\tau_j - \tau_i \in (-\infty, -1) \cup (0, +\infty)$ is safe.*

Theorem 5. *For every distance profile Δ and for every $i \neq j$, $(\tau_i, \tau_j) \in \mathcal{T}_{\Delta^{(ij)}} \Leftrightarrow \tau_j - \tau_i \in L_{ij}$ where*

$$L_{ij} = \begin{cases} \mathbb{Z} & \text{if } \Psi_{ij} > 0 \\ (-\infty, -\Lambda_{ji}) \cup (\Lambda_{ij}, +\infty) \cup & \\ \quad (\{-\Lambda_{ji}, \Lambda_{ij}\} \cap (\Delta^s_{ij} + 1 + 2\mathbb{Z})) & \text{if } \Psi_{ij} = 0 \\ (-\infty, -\Lambda_{ji}) \cup (\Lambda_{ij}, +\infty) & \text{if } \Psi_{ij} < 0 \end{cases} \tag{7}$$

Proof We first prove $\Leftarrow$. We fix any MAPF problem instance $(\mathcal{G}, k, s, g)$ having distance profile Δ and two distinct agents i and j. We now choose any two paths $\pi_i \in \Pi^0_i$ and $\pi_j \in \Pi^0_j$. Suppose that the delay pair $(\tau_i, \tau_j) \in \mathbb{N}_0 \times \mathbb{N}_0$ is such that $\pi^{\tau_i}_i$ and $\pi^{\tau_j}_j$ are conflicting in a vertex, namely there exists $\max\{\tau_i, \tau_j\} \le t \le b_{\pi^{\tau_i}_i, \pi^{\tau_j}_j}$ such that $\pi^{\tau_i}_i(t) = \pi^{\tau_j}_j(t)$. This yields $\pi_i(t_i) = P = \pi_j(t_j)$ where $t_i = t - \tau_i$ and $t_j = t - \tau_j$. We have that

$$\begin{array}{rcl} d(s_i, s_j) + d(g_i, g_j) & \le & d(s_i, P) + d(P, s_j) + \\ & & d(g_i, P) + d(P, g_j) \\ & = & d(s_i, g_i) + d(s_j, g_j) \end{array} \tag{8}$$

where the inequality follows from the triangular inequality and the equality follows from the fact that P belongs to a minimum path from s_i to g_i and a minimum path from s_j to g_j. This yields $\Psi_{ij} \le 0$. Moreover, the following holds true

$$\begin{array}{rcl} \tau_j - \tau_i = t_i - t_j & = & d(s_i, P) - d(s_j, P) \\ & = & [d(s_i, P) + d(P, g_i)] \\ & - & [d(s_j, P) + d(P, g_i)] \\ & \le & d(s_i, g_i) - d(s_j, g_i) \end{array} \tag{9}$$

This yields $\tau_j - \tau_i \le \Lambda_{ij}$. A symmetric argument also yields $\tau_i - \tau_j \le \Lambda_{ji}$. Suppose now that $\Psi_{ij} = 0$, $\tau_j - \tau_i = \Lambda_{ij}$, and that $\Lambda_{ij} - d(s_i, s_j)$ is an odd number. The condition $\Psi_{ij} = 0$ implies that all the relations in (8) must be equalities so that, in particular, $d(s_i, s_j) = d(s_i, P) + d(P, s_j)$. Combining this with the equality $\tau_j - \tau_i = d(s_i, P) - d(s_j, P)$ obtained from expression (9), we deduce that $\tau_j - \tau_i - d(s_i, s_j) = -2d(P, s_j)$ contradicting that $\tau_j - \tau_i - d(s_i, s_j)$ is odd. This shows that if delays are chosen such that $\tau_j - \tau_i \in L_{ij}$, the delayed paths $\pi^{\tau_i}_i$ and $\pi^{\tau_j}_j$ satisfy condition (a) in Definition 1. We claim that they also satisfy condition (b). Indeed, if by contradiction they did not, then both pairs $\pi^{\tau_i+1}_i, \pi^{\tau_j}_j$ and $\pi^{\tau_i}_i, \pi^{\tau_j+1}_j$ would present a vertex conflict. However, if $\tau_j - \tau_i \in L_{ij}$, necessarily at least one of the two numbers $\tau_i + 1 - \tau_j$ or $\tau_i - \tau_j - 1$ is in L_{ij} and thus would lead to paths without vertex conflict. This contradiction implies that condition (b) must also be satisfied.

We now prove $\Rightarrow$. To this aim, we take any pair of distinct agents i and j and a pair of delays (τ_i, τ_j) such that $\tau_j - \tau_i \notin$

L_{ij}. Notice that this necessarily implies that $\Psi_{ij} \leq 0$. We split this condition into three possible cases:

$$\begin{array}{ll} (a)\ \Psi_{ij} < 0, & -\Lambda_{ji} \leq \tau_j - \tau_i \leq \Lambda_{ij} \\ (b)\ \Psi_{ij} = 0, & -\Lambda_{ji} \leq \tau_j - \tau_i \leq \Lambda_{ij},\ \tau_j - \tau_i - \Delta^s_{ij} \text{ even} \\ (c)\ \Psi_{ij} = 0, & -\Lambda_{ji} < \tau_j - \tau_i < \Lambda_{ij},\ \tau_j - \tau_i - \Delta^s_{ij} \text{ odd} \end{array} \tag{10}$$

When (10) is satisfied, we will prove that there exists a MAPF problem instance $(\mathcal{G}, k, s, g)$ having distance profile Δ and two paths $\pi_i \in \Pi^0_i$ and $\pi_j \in \Pi^0_j$ such that $\pi_i^{\tau_i}$ and $\pi_j^{\tau_j}$ are conflicting in a vertex (if (a) or (b) in expression (10) hold true) or in an edge (if (c) in expression (10) holds true). As only agents i and j will play a role in this construction, we assume from now on that $k = 2$ and $\mathcal{A} = \{i, j\}$.

We first assume that condition (a) or (b) in (10) is satisfied, and we study the presence of vertex conflicts. We construct a MAPF instance based on the graph depicted in Fig. 2 (left), which represents the two starting points s_i, s_j, the two goal points g_i, g_j, the point P and the paths $\alpha_i, \alpha_j, \beta_i, \beta_j, \mu_{ij}, \nu_{ij}, \lambda_{ij}, \lambda_{ji}$ (all with starting time 0) obtained connecting these points and possibly containing extra vertices. We assume that

$$l_{\mu_{ij}} = \Delta^s_{ij},\ \ l_{\nu_{ij}} = \Delta^g_{ij},\ \ l_{\lambda_{ij}} = \Delta^{sg}_{ij},\ \ l_{\lambda_{ji}} = \Delta^{sg}_{ji}$$

We now define $x_i = l_{\alpha_i}, x_j = l_{\alpha_j}, y_i = l_{\beta_i}, y_j = l_{\beta_j}$ and consider the following set of equalities and inequalities:

$$\begin{cases} x_i + \tau_i = x_j + \tau_j \\ x_i + y_i = \Delta^{sg}_{ii} \\ x_j + y_j = \Delta^{sg}_{jj} \\ x_i + x_j \geq \Delta^s_{ij} \\ y_i + y_j \geq \Delta^g_{ij} \\ x_i + y_j \geq \Delta^{sg}_{ij} \\ x_j + y_i \geq \Delta^{sg}_{ji} \end{cases} \tag{11}$$

We notice that the second and third equalities and the last four inequalities imply that all paths $\alpha_i, \alpha_j, \beta_i, \beta_j, \mu_{ij}, \nu_{ij}, \lambda_{ij}, \lambda_{ji}$ are shortest paths. In particular, they imply that the geodetic distances among all the points s_i, s_j, g_i, g_j coincide with those assigned by Δ. If we define π_i and π_j as the concatenation of the pair of paths, respectively, α_i, $\beta_i^{l_{\alpha_i}}$ and α_j, $\beta_j^{l_{\alpha_j}}$, the second and third equalities imply that $\pi_i \in \Pi^0_i$ and $\pi_j \in \Pi^0_j$. Finally, the first equality implies that $\pi_i^{\tau_i}$ and $\pi_j^{\tau_j}$ are conflicting in P.

Therefore, the only thing that remains to be done is to show the feasibility of our construction, namely that there exist non negative integer values x_i, x_j, y_i, and y_j solving the set of equalities and inequalities in (11). We notice first that, based on the first three equalities, we can write x_j, y_i, y_j in terms of the variable x_i:

$$\begin{array}{l} x_j = x_i + \tau_i - \tau_j, \quad y_i = -x_i + \Delta^{sg}_{ii} \\ y_j = -x_i + \tau_j - \tau_i + \Delta^{sg}_{jj} \end{array} \tag{12}$$

Substituting (12) into the last four inequalities of (11), we now obtain

$$\begin{cases} x_i \geq \frac{1}{2}[\Delta^s_{ij} + \tau_j - \tau_i] \\ x_i \leq \frac{1}{2}[\Delta^{sg}_{ii} + \Delta^{sg}_{jj} - \Delta^g_{ij} + \tau_j - \tau_i] \\ \Delta^{sg}_{jj} - \tau_i + \tau_j \geq \Delta^{sg}_{ij} \\ \tau_i - \tau_j + \Delta^{sg}_{ii} \geq \Delta^{sg}_{ji} \end{cases} \tag{13}$$

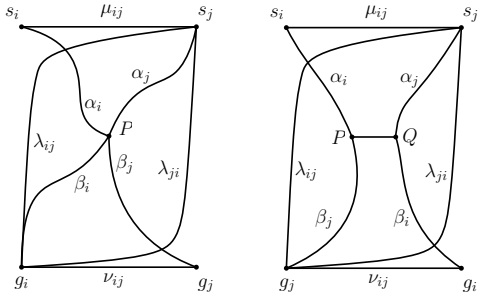

Figure 2: Proof auxiliary graphs.

Finally, the non negativity constraints on x_i, x_j, y_i, and y_j are equivalent to

$$\begin{cases} x_i \geq 0 \\ x_i \geq \tau_j - \tau_i \\ x_i \leq \Delta^{sg}_{ii} \\ x_i \leq \tau_j - \tau_i + \Delta^{sg}_{jj} \end{cases} \tag{14}$$

So, we are left to show that there exists an integer solution x_i to the inequalities in (13) and (14). Notice first how the last two inequalities in (13) do not depend on x_i and are always true thanks to the inequalities in (10). The remaining inequalities are equivalent to prove that given

$$\begin{array}{l} A_1 = 0, A_2 = \tau_j - \tau_i, \quad A_3 = \frac{1}{2}[\Delta^s_{ij} + \tau_j - \tau_i] \\ B_1 = \Delta^{sg}_{ii}, B_2 = \tau_j - \tau_i + \Delta^{sg}_{jj}, \\ B_3 = \frac{1}{2}[\Delta^{sg}_{ii} + \Delta^{sg}_{jj} - \Delta^g_{ij} + \tau_j - \tau_i] \end{array} \tag{15}$$

there exists a non negative integer x_1 such that $A_i \leq x_1 \leq B_j$ for every $i, j = 1, 2, 3$. We split the proof into three cases:

- Case 1: $\max A_i \in \{A_1, A_2\}$. In this case, since A_1, A_2 are both integer, it is sufficient to prove that $A_i \leq B_j$ for $i = 1, 2$ and $j = 1, 2, 3$. $A_1 \leq B_1$ follows from the fact that Δ^{sg}_{ii} is by definition non negative. $A_1 \leq B_2$ and $A_1 \leq B_3$ both follow from the inequalities in (10):

$$\begin{array}{rcl} B_2 & = & \tau_j - \tau_i + \Delta^{sg}_{jj} \geq -\Lambda_{ji} + \Delta^{sg}_{jj} \\ & = & -\Delta^{sg}_{jj} + \Delta^{sg}_{ij} + \Delta^{sg}_{jj} = \Delta^{sg}_{ij} \geq 0 \end{array} \tag{16}$$

$$\begin{array}{rcl} B_3 & = & \frac{1}{2}[\Delta^{sg}_{ii} + \Delta^{sg}_{jj} - \Delta^g_{ij} + \tau_j - \tau_i] \\ & \geq & \frac{1}{2}[\Delta^{sg}_{ii} + \Delta^{sg}_{jj} - \Delta^g_{ij} - \Delta^{sg}_{jj} + \Delta^{sg}_{ij}] \\ & \geq & \frac{1}{2}[\Delta^{sg}_{ii} - \Delta^g_{ij} + \Delta^{sg}_{ij}] \geq 0 \end{array} \tag{17}$$

  where the last inequality follows from Δ being a distance profile. We now consider A_2. Notice that, because of the inequalities in (10), we have:

$$A_2 \leq \Lambda_{ij} = \Delta^{sg}_{ii} - \Delta^{sg}_{ji} \leq \Delta^{sg}_{ii} = B_1$$

  while $A_2 \leq B_2$ follows again from the non negativity of Δ^{sg}_{jj}. Finally, notice that $A_2 \leq B_3$ is equivalent to $\tau_j - \tau_i \leq \Delta^{sg}_{ii} + \Delta^{sg}_{jj} - \Delta^g_{ij}$ and this follows again from (10) and Δ being a distance profile.
- Case 2: $\max A_i = A_3$, $\min B_j \in \{B_1, B_2\}$. As B_1 and B_2 are integer, it is sufficient to prove that $A_3 \leq B_1$ and $A_3 \leq B_2$. The inequality $A_3 \leq B_1$ follows again from applying (10) and the fact that Δ is a distance profile:

$$A_3 = \frac{1}{2}[\Delta^s_{ij} + \tau_j - \tau_i] \leq \frac{1}{2}[\Delta^s_{ij} + \Delta^{sg}_{ii} - \Delta^{sg}_{ji}] \leq \Delta^{sg}_{ii} \tag{18}$$

The inequality $A_3 \leq B_2$ is equivalent to $\Delta^s_{ij} \leq \tau_j - \tau_i + 2\Delta^{sg}_{jj}$ and this follows again from $\tau_j - \tau_i \geq -\Lambda_{ji}$.

- Case 3: $\max A_i = A_3$, $\min B_j = B_3$. Notice that $B_3 - A_3 = -\frac{1}{2}\Psi_{ij} \geq 0$. If $\Psi_{ij} \leq -1$, this means that $B_3 - A_3 \geq 1/2$ and, as a consequence, there exists an integer $A_3 \leq x_1 \leq B_3$. Instead, if $\Psi_{ij} = 0$, we have that $B_3 = A_3$. In this case, thanks to conditions (b) of (10), we know that $\Delta^s_{ij} + \tau_j - \tau_i$ is even and thus A_3 is an integer.

We are left with studying case (c) of (10). In this case, we construct a MAPF instance based on the graph depicted in Fig. 2 (right). Using the same definitions and notations of the previous case, we can prove that showing the feasibility of the construction and the existence of a swapping conflict is equivalent to showing the existence of non negative integer values x_i, x_j, y_i, and y_j that solve the following system:

$$\begin{cases} x_i + \tau_i = x_j + \tau_j \\ x_i + y_i = \Delta^{sg}_{ii} - 1 \\ x_j + y_j = \Delta^{sg}_{jj} - 1 \\ x_i + x_j \geq \Delta^{s}_{ij} - 1 \\ y_i + y_j \geq \Delta^{g}_{ij} - 1 \\ x_i + y_j \geq \Delta^{sg}_{ij} \\ x_j + y_i \geq \Delta^{sg}_{ji} \end{cases} \tag{19}$$

If we define the triple $\tilde{\Delta} = (\tilde{\Delta}^s, \tilde{\Delta}^{sg}, \tilde{\Delta}^g)$ by setting

$$\tilde{\Delta}^s_{ij} = \Delta^s_{ij} - 1, \quad \tilde{\Delta}^g_{ij} = \Delta^g_{ij} - 1$$
$$\tilde{\Delta}^{sg}_{ii} = \Delta^{sg}_{ii} - 1,\ \tilde{\Delta}^{sg}_{jj} = \Delta^{sg}_{jj} - 1,\ \tilde{\Delta}^{sg}_{ij} = \Delta^{sg}_{ij},\ \tilde{\Delta}^{sg}_{ji} = \Delta^{sg}_{ji}$$

we claim that a solution exists by applying the same considerations used to prove the existence of a solution to (11). By arguing as above, a solution to (19) is equivalent to finding a non negative integer x_1 that solves the two set of inequalities (13) and (14) with Δ replaced by $\tilde{\Delta}$. Notice that the last two inequalities in (13) become

$$\tau_j - \tau_i \geq \tilde{\Delta}^{sg}_{ij} - \tilde{\Delta}^{sg}_{jj} = -\Lambda_{ji} + 1$$
$$\tau_j - \tau_i \leq \tilde{\Delta}^{sg}_{ii} - \tilde{\Delta}^{sg}_{ji} = \Lambda_{ij} - 1$$

which are true thanks to the relations in condition (c) of (10). As for the rest, we can repeat the same arguments as above noting that the only facts needed in the proofs of the various steps are the following:

(i) $\tilde{\Delta}^{sg}_{ii}, \tilde{\Delta}^{sg}_{jj}, \tilde{\Delta}^{sg}_{ij}, \tilde{\Delta}^{sg}_{ji}$ are non negative;

(ii) the triangular inequalities $\tilde{\Delta}^{sg}_{ii} + \tilde{\Delta}^{sg}_{ij} \geq \tilde{\Delta}^{g}_{ij}$ and the analogous ones replacing ii with jj and ij with ji hold true;

(iii) $\tilde{\Delta}^{s}_{ij} + \tau_j - \tau_i$ is even.

Now note that: (i) holds because of the assumption that $s_i \neq g_i$ and $s_j \neq g_j$. (ii) holds by inspection. (iii) holds because of condition (c) of (10) and the definition of $\tilde{\Delta}^s_{ij}$.

The proof is now complete. ∎

Theorem 5 and relation (6) yield an algebraic description of the set $\mathcal{T}_\Delta$ in terms of the pairwise conditions described in expression (7). For every distance profile Δ, $\tau \in \mathbb{N}_0^k$ belongs to $\mathcal{T}_\Delta$ if and only if, for every $i, j \in \mathcal{A}$ such that $\Psi_{ij} \leq 0$, we have that $\tau_j - \tau_i \in L_{ij}$. Notice that in the special case of $\Lambda_{ij} + \Lambda_{ji} < 0$, condition (7) is always satisfied. This means that all pairs of agents such that $\Psi_{ij} > 0$ or $\Lambda_{ij} + \Lambda_{ji} < 0$ are not constrained in their corresponding delays.

5 Complexity Analysis

Despite the explicit description of the set $\mathcal{T}_\Delta$ given in Sec. 4, the minimization problem in expression (4) is hard. More precisely, let us consider the corresponding decision problem, stated below.

Problem 1. *Given a distance profile Δ and $m \in \mathbb{N}$, it exists $\tau \in \mathcal{T}_\Delta$ such that $\sum_{i \in \mathcal{A}} \tau_i \leq m$.*

We show that Problem 1 is NP-complete by proving that it is equivalent to a well-known NP-complete scheduling problem (membership in NP is trivial). Let $J_1, J_2, \ldots, J_N$ be N jobs to be scheduled on one single machine. For each job J_i, we call S_i its starting time, p_i its processing time and C_i its completion time. Each job is also characterized by a release time $r_i \geq 0$, which is its earliest possible starting time. The problem we consider is traditionally indicated as $n \mid 1 \mid r_i \geq 0 \mid \sum_i C_i$ and can be formulated as follows. Find a schedule Z that minimizes the sum of the completion times, i.e. $\sum_{i=1}^N C_i$, and satisfies the following constraints:

$$\begin{array}{ll} \forall i & S_i \geq r_i \\ \forall i, j \in Z & S_j - S_i \geq p_i \ \text{ or } \ S_i - S_j \geq p_j \end{array} \tag{20}$$

Lenstra, Rinnooy Kan, and Brucker (1977) demonstrate that this problem is NP-complete by exhibiting a reduction from the 3-Partition problem.

We now show that the scheduling problem $n \mid 1 \mid r_i \geq 0 \mid \sum_i C_i$ can be reformulated into the problem given in expression (4). Let us put $\tau_i = S_i - r_i$. The constraints in expression (20) can be rewritten as follows:

$$\begin{array}{ll} \forall i & \tau_i \geq 0 \\ \forall i, j \in T & \tau_j - \tau_i \geq p_i + r_i - r_j \text{ or } \tau_i - \tau_j \geq p_j + r_j - r_i \end{array} \tag{21}$$

As for the objective function, the scheduling problem $n \mid 1 \mid r_i \geq 0 \mid \sum_i C_i$ aims to find a schedule Z such that: $\operatorname{argmin}_Z \sum_{i=1}^N C_i$. The following equivalences hold.

$$\operatorname*{argmin}_Z \sum_{i=1}^N C_i = \operatorname*{argmin}_Z \sum_{i=1}^N S_i = \operatorname*{argmin}_T \sum_{i=1}^N \tau_i$$

We notice that the last expression is the same used for our problem and that the constraints match those in our problem as long as we can find a distance profile Δ for which $\Lambda_{ij} = p_i + r_i - r_j - 1$ and $\Psi_{ij} < 0$ for every pair of jobs $i \neq j$.

Proposition 6. *For every matrix $\bar{\Lambda} \in \mathbb{R}^{\mathcal{V} \times \mathcal{V}}$, there exists a distance profile $\Delta = (\Delta^s, \Delta^{sg}, \Delta^g)$ such that*

$$\Lambda_{ij} = \bar{\Lambda}_{ij}, \quad \Psi_{ij} < 0 \quad \forall i \neq j$$

Proof Recall that a triple of non-negative matrices $\Delta = (\Delta^s, \Delta^{sg}, \Delta^g)$ is a distance profile if all related triangular inequalities given in expression (3) are satisfied. Imposing that $\Lambda = \bar{\Lambda}$, we can rewrite these inequalities as follows:

$$\begin{array}{l} \max\{|\bar{\Lambda}_{ij}|, |\bar{\Lambda}_{ji}|\} \leq \Delta^s_{ij} \leq \min\{\bar{\Lambda}_{ij} + 2\Delta^{sg}_{ji}, \bar{\Lambda}_{ji} + 2\Delta^{sg}_{ij}\} \\ \max\{|\bar{\Lambda}_{ij} + \Delta^{sg}_{ji} - \Delta^{sg}_{ij}|, |\bar{\Lambda}_{ji} + \Delta^{sg}_{ij} - \Delta^{sg}_{ji}|\} \leq \Delta^g_{ij} \leq \\ \leq \min\{\bar{\Lambda}_{ij} + \Delta^{sg}_{ji} + \Delta^{sg}_{ij}, \bar{\Lambda}_{ji} + \Delta^{sg}_{ij} + \Delta^{sg}_{ji}\} \\ \Delta^s_{ij} + \Delta^g_{ij} < \bar{\Lambda}_{ij} + \bar{\Lambda}_{ji} + \Delta^{sg}_{ij} + \Delta^{sg}_{ji} \end{array} \tag{22}$$

The variables Δ^s_{ij}, Δ^g_{ij}, Δ^{sg}_{ji}, Δ^{sg}_{ij} must all be non negative for every $i \neq j$. Moreover, it must hold that, for every $i \in \mathcal{V}$,

$$\bar{\Lambda}_{ij} + \Delta^{sg}_{ji} = \bar{\Lambda}_{ik} + \Delta^{sg}_{ki} > 0 \quad \forall j, k \neq i \tag{23}$$

This guarantees that $\Lambda_{ij} = \bar{\Lambda}_{ij}$ can be achieved by setting $\Delta^{sg}_{ii} > 0 = \bar{\Lambda}_{ij} + \Delta^{sg}_{ji}$. We now fix a job, say 1, and we set

$$\Delta^{sg}_{ij} = \bar{\Lambda}_{j1} - \bar{\Lambda}_{ji} + 2q, \quad \Delta^s_{ij} = \Delta^g_{ij} = q$$

for $q \in \mathbb{N}$. It follows from relations (22) and (23) that, if q is sufficiently large, all conditions are satisfied. ■

Corollary 7. *Problem 1 is NP-complete.*

While, in our proof, we focus on the sum-of-costs objective function, following Lenstra, Rinnooy Kan, and Brucker (1977), other objectives (as makespan) may be considered.

6 Delayed Shortest Path (DSP)

In DSP, the agents follow their shortest paths, according to a given total priority order $PO = (\hat{1}, \ldots, \hat{k})$. PO states that, for every two agents $\hat{i}, \hat{j} \in PO$ such that $\hat{j} < \hat{i}$, agent $\hat{j}$ has a higher priority than agent $\hat{i}$. Let $PO_{<\hat{i}} = PO \backslash \{\hat{j} | \hat{i} \leq \hat{j}\}$. Before start moving, the agents might be delayed to avoid conflicts with agents with higher priority. As the first agent $\hat{1}$ has the highest priority, it can follow its shortest path without delays, i.e. $\tau_{\hat{1}} = 0$. The delay of an agent $\hat{i}$ may be influenced by any other agent in $PO_{<\hat{i}}$. Hence, the value of $\tau_{\hat{i}}$ must be safe when it is paired with value $\tau_{\hat{j}}$ of any agent $\hat{j}$ in $PO_{<\hat{i}}$.

The pseudo-code of DSP is presented in Algorithm 1. An empty solution Π and an empty list of delays τ are initialized in lines 2-3. We use τ to maintain the delays of the agents. In lines 4-9, we go over each agent $\hat{i}$ according to the priority order PO and calculate its safe delays $\tau_{\hat{i}}$ by invoking the function *SafeDelays*. The minimal safe delay returned by the function is stored in τ (line 8). In line 9, the algorithm augments the plan Π with the path of agent $\hat{i}$, which is delayed by $\tau_{\hat{i}}$ time steps and then follows its shortest path.

SafeDelays calculates safe delays for the agent $\hat{i}$, stores them in T (by shifting $L_{\hat{i},\hat{j}}$) and returns the minimum.

Note that DSP does not perform a search in the graph. It only uses the costs of the shortest paths to determine when each agent should start following its shortest path. Therefore, the complexity of DSP is mainly influenced by the number of agents and not by the length of their paths.

7 Experimental Study

To examine the performance of DSP, we compare it with a few well-known MAPF algorithms as described below.

7.1 Algorithms

SEQ. An algorithm closely related to DSP is *SEQ* (Ma 2021), which was originally proposed for theoretical purposes on online MAPF (or well-formed) instances. According to a given total priority order of agents, a single agent at a time follows its shortest path to the goal vertex while all the other agents wait at their start vertices. The algorithm ends when all agents have reached their goals.

Algorithm 1: Delayed Shortest Path (DSP)

```
1  Main(MAPF instance, Priority Order PO)
2      Init Π with an empty solution
3      Init τ with an empty list
4      foreach î ∈ PO do
5          τ_î ← 0
6          if τ is not empty then
7              τ_î ← SafeDelays(PO, τ, î)
8          τ[î] = τ_î
9          Π ← Π ∪ {π^{τ_i}}
10     return π
11 SafeDelays(Priority Order PO, Delays τ, Agent î)
12     Init T ← ℕ_0
13     foreach ĵ ∈ PO_{<î} do
14         τ_ĵ ← τ[ĵ]
15         T ← T ∩ {τ_ĵ − L_{î,ĵ}}
16     return min_{τ'∈T}(τ')
```

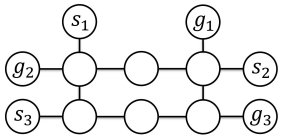

Figure 3: Example in which DSP outperforms PP.

Prioritized Planning (PP). In PP (Erdmann and Lozano-Pérez 1987), the agents may not necessarily follow their shortest paths. According to a given total priority order of agents, PP computes paths for them, one by one. When PP searches for a path, the path must not conflict with the paths of higher priority agents (prior in the given order). This can be done, for instance, by searching in a space-time configuration, e.g., by using Space-Time A* (Silver 2005). Here too, agents may delay their start times. We use the pre-calculated costs of the shortest path between any two cells as a heuristic for PP. PP is not optimal nor complete. Practically, it usually performs well and, in many cases, returns near-optimal solutions. Under our assumptions on agents' appearance and disappearance, PP is complete (all instances below are solved). We also consider a PP version that uses one shortest path for each agent and executes PP on that path (**SPP**).

While PP plans paths for the agents and DSP only defines safe delays, in some cases, DSP outperforms PP when both use the same priority order. Fig. 3 illustrates such a case with three agents: $1, 2$, and 3. Let us assume that the priority order is $(1, 2, 3)$ for DSP and PP. According to DSP, the first agent is not delayed ($\tau_1 = 0$), the second agent is delayed 3 time steps ($\tau_2 = 3$), and the third agent is not delayed ($\tau_3 = 0$). This results in a solution of cost 15. In PP, the first agent is not delayed, and its path costs 4. However, the second agent follows a path of cost 6 (which passes through the row below its start and goal vertices) because it chooses the shortest path that avoids conflicts with the first agent. As a result, the third agent follows a path of cost 6, with a resulting solution of cost 16, which is higher than the one created by DSP.

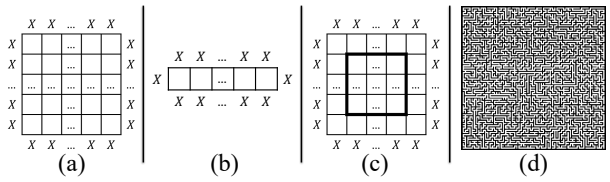

Figure 4: (a) Empty. (b) Corridor. (c) Grid w/ obs. (d) Maze.

7.2 Priority Orderings

All algorithms DSP, SEQ, SPP, and PP use a given priority order for the agents. We now present existing priority ordering functions (Ma et al. 2019) as well as a new one based on our geometric constraints. **SH (Shortest path first)** gives a higher priority to agents with shorter paths. **LH (Longer path first)** is the opposite of SH and gives a higher priority to agents with longer paths. **RND** prioritizes the agents randomly. **LD (Lowest delays first)** is a greedy method that prioritizes agents according to the lowest safe delay. Starting with an empty priority order PO and an empty set of delays τ, we call *SafeDelays* (Algorithm 1) for each agent. Then, we choose the agent with the lowest delay, adding it to PO and its delays to τ. We continue this process until PO contains all agents. LD breaks ties according to LH.

7.3 Domains

We perform experiments on four maps, presented in Fig. 4. Fig. 4(a) illustrates an **empty grid**, without obstacles, where the agents enter and leave the grid from the borders (marked by X in the figure), i.e., their start and goal cells are adjacent to the borders. We denote by R the number of rows in the grid and by C the number of columns. Fig. 4(b) presents a special case of the empty grid, where there is only a single row ($R = 1$). Thus, all agents must pass through this **long corridor**. Arguably, the long corridor is a challenging MAPF scenario as the agents must share cells along their paths and, therefore, may easily conflict. The map in Fig. 4(c) generalizes the empty grid to a **grid with obstacles**. To ensure that the problem remains solvable, the obstacles are only allocated in the bold rectangle area, i.e., in rows 2 to $|R|-1$ and in columns 2 to $|C|-1$. Allocating 100% obstacles means that the bold rectangle is full of obstacles. Fig. 4(d) shows a **maze grid**, called maze-128-128-1, from the *MovingAI* repository (Sturtevant 2012).

7.4 Experiments

In our experiments (performed by Intel Xeon E5-2660 v4,2.00GHz processor with 16GB RAM), we evaluate DSP, SEQ, SPP, and PP with each of the four priority orders (RND, SH, LH, and LD). Each result in our experiment is averaged on 50 randomly created instances. We measure the cost of the returned solution and the runtime (in sec).

Table 1 presents the average cost (in thousands) on (a) 100×100 empty grids (Fig. 4(a)), (b) 1×100 long corridor grids (Fig. 4(b)), (c) grids with randomly allocated obstacles (Fig. 4(c)), and (d) maze grids (Fig. 4(d)). In (a), (b), and (d), we perform experiments with 20, 40, 60, 80, and 100 agents and, in (c), we experiment with 100 agents and varied the percentage of obstacles. Table 2 shows representative results of the average runtime and the average preprocessing time of all algorithms (in sec). In all these experiments, the average runtime of DSP and SEQ with RND, SH, LH, and LD is less than 100ms and, thus, is not presented in the table.

As expected, when we increase the number of agents, the cost of all algorithms increases for all priority orders. In the empty grid (a), PP achieves a lower cost than SPP, DSP and SEQ. This is due to the fact that, in empty grids, many paths exist for every agent and, therefore, most agents are able to use one of their shortest paths without delays. This is also why all priority orders achieve a similar cost in PP.

As for the long corridor grid (b), the runtime of PP is the highest. DSP not only presents a lower runtime (sometimes, three orders of magnitude lower than PP) but also provides solutions of lower costs than PP. This is because (e.g. Fig. 3) in DSP, the agents use their shortest paths while, when waiting, they do not block other agents with lower priority. This shows that DSP is more beneficial in dense domains.

To examine the impact of the density on the algorithms, we run experiments on grids with 20%, 40%, 60%, 80%, and 100% randomly allocated obstacles (Fig. 4(c)), with 100 agents. The impact of increasing the percentage of obstacles is more prominent in SPP and PP than in SEQ and DSP in terms of cost. When only 20% obstacles are present, the cost of PP is more than 2 times lower than DSP but, when the percentage of obstacles increases, the cost becomes similar. PP consumes larger runtime when more obstacles exist. This is, again, a result of fewer shortest paths being present for each agent when the grid is denser. With 100% obstacles, DSP is four order of magnitude faster than PP. We obtain similar trends for the maze grid (Fig. 4(d)).

The fast algorithms SEQ and DSP with RND, SH, and LH allow us to experiment with thousands of agents. We measure the average cost. Fig. 5(a) presents the results of an experiment performed on a $1 \times 10,000$ long corridor grid (Fig. 4(b)) with $2,000$, $4,000$, $6,000$, $8,000$, $10,000$ agents. Fig. 5(b) presents the results of an experiment performed on the maze grid (Fig. 4(d)) with $1,000$, $2,000$, $3,000$, $4,000$ agents. In both experiments, DSP achieves a significantly lower cost than SEQ with any priority order. Compared to RND and SH, DSP obtains the lowest cost when it uses LH.

8 Related Work

Several studies have explored MAPF with imperfect execution. Shahar et al. (2021) show how to calculate a safe plan when there is uncertainty on the time it takes to perform a move action. Other papers (Wagner and Choset 2017; Atzmon et al. 2020b,a) focus on the idea of creating a plan that can withstand unexpected delays during execution and avoid conflicts. Finally, other authors (Ma, Kumar, and Koenig 2017; Hönig et al. 2019) show how to prevent conflicts during execution using execution policies that require agents with communication capabilities. Execution delays are different from the delays that we intentionally assign to the agents to create conflict-free plans in our approach.

Our research is related to studies that prioritize agents in MAPF. van den Berg and Overmars (2005) prioritize agents according to their *query distance*, which is similar to the

		DSP				SEQ				SPP				PP			
	k	RND	SH	LH	LD	RND	SH	LH	LD	RND	SH	LH	LD	RND	SH	LH	LD
(a)	20	3.4	3.5	3.3	3.0	20.0	14.5	25.2	19.6	2.2	2.1	2.1	2.1	**1.9**	**1.9**	**1.9**	**1.9**
	40	8.1	8.7	7.4	6.8	76.6	54.2	97.6	71.9	4.6	4.6	4.5	4.4	**3.7**	**3.7**	**3.7**	**3.7**
	60	14.7	16.7	12.6	11.2	171.7	123.3	220.9	162.2	7.7	8.0	7.3	7.1	**5.7**	**5.7**	**5.7**	**5.7**
	80	20.7	24.9	17.0	15.2	302.2	212.6	391.7	283.0	10.6	11.4	10.1	9.7	**7.5**	**7.5**	**7.5**	**7.5**
	100	27.9	34.5	22.3	19.5	468.9	328.3	608.2	437.2	13.7	14.6	12.7	12.4	**9.3**	**9.3**	**9.3**	**9.3**
	k																
(b)	20	1.3	1.4	**1.1**	**1.1**	7.1	4.8	9.8	8.0	1.4	1.5	1.4	**1.1**	1.5	1.5	1.5	**1.1**
	40	3.0	4.1	**2.4**	**2.4**	28.9	18.5	39.1	30.5	4.5	4.3	3.5	**2.4**	4.8	4.3	4.3	**2.4**
	60	5.3	8.2	**3.9**	**3.9**	67.2	43.0	91.2	69.6	8.5	9.2	6.1	**3.9**	9.8	9.1	8.7	**3.9**
	80	7.7	13.0	**5.3**	**5.3**	117.5	74.9	159.7	120.7	12.9	14.6	8.6	**5.3**	16.5	14.6	14.5	**5.3**
	100	9.8	18.3	**6.7**	**6.7**	182.4	115.7	249.7	186.8	18.1	20.8	10.9	**6.7**	24.6	21.3	21.6	**6.7**
	Obs.																
(c)	20%	27.8	32.7	22.3	20.2	478.1	337.0	615.5	347.6	14.3	15.1	13.1	12.7	9.6	**9.5**	9.6	**9.5**
	40%	30.8	40.6	26.6	22.3	508.9	344.5	672.6	355.2	27.5	34.2	24.5	19.7	12.1	11.9	12.4	**11.8**
	60%	31.9	41.1	26.8	22.6	505.5	347.4	673.9	357.9	31.7	39.5	26.6	21.3	16.0	14.9	17.6	**14.5**
	80%	31.2	40.8	26.4	22.9	514.3	350.8	674.2	361.4	33.2	41.6	27.1	21.5	21.9	19.2	22.5	**17.2**
	100%	30.8	40.6	26.5	22.6	509.5	345.8	670.9	356.4	32.9	40.9	26.1	21.5	44.8	34.9	21.7	**21.4**
	k																
(d)	20	10.8	11.4	10.6	10.2	76.6	52.6	100.1	83.0	12.0	11.3	11.5	10.1	**7.7**	**7.7**	**7.7**	**7.7**
	40	27.1	29.2	24.3	23.0	304.9	206.8	399.8	327.5	30.6	30.2	31.0	22.3	16.6	16.4	16.7	**16.3**
	60	43.9	50.4	39.0	36.2	670.1	453.1	879.6	691.4	58.9	57.3	55.1	35.2	26.0	25.6	26.3	**25.2**
	80	62.7	74.4	53.8	50.0	1,169.0	785.9	1,546.7	1,192.3	83.8	85.7	86.4	48.6	36.7	35.6	37.7	**34.7**
	100	86.7	103.1	72.2	65.3	1,842.3	1,242.9	2,448.5	1,903.9	128.4	129.0	124.8	62.9	49.3	47.5	51.2	**45.9**

Table 1: Average plan cost (in thousands) for DSP, SEQ, SPP, and PP, on (a) a 100 × 100 empty grid (Fig. 4(a)); (b) a 1 × 100 long corridor grid (Fig. 4(b)); (c) a 100 × 100 grid with obstacles and 100 agents (Fig. 4(c)); and (d) a maze grid (Fig. 4(d)).

		SPP				PP				Pre
	k	RND	SH	LH	LD	RND	SH	LH	LD	Pro.
(b)	20	0.0	0.1	0.0	0.0	0.1	0.2	0.1	0.2	0.1
	40	0.1	0.2	0.0	0.0	0.6	0.7	0.5	0.3	0.1
	60	0.3	0.5	0.1	0.0	1.4	1.8	1.0	0.6	0.1
	80	0.4	0.9	0.1	0.1	2.5	3.1	1.8	0.8	0.2
	100	0.5	1.3	0.1	0.1	3.7	4.6	2.5	1.0	0.3
	Obs.									
(c)	20%	0.8	1.1	0.5	0.4	0.1	0.1	0.1	0.3	2.5
	40%	4.0	6.3	3.2	1.4	4.2	3.6	5.2	4.0	2.0
	60%	5.3	8.1	3.9	1.7	5.9	5.2	8.4	5.2	0.5
	80%	6.4	9.7	4.7	1.9	9.6	8.1	9.5	5.6	0.4
	100%	8.0	11.5	5.3	2.4	26.8	20.6	9.6	5.8	0.3

Table 2: Average planning time (sec) for SPP and PP.

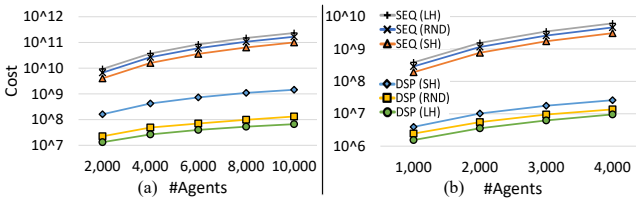

Figure 5: Average cost for thousands of agents.

LH priority ordering. Wu, Bhattacharya, and Prorok (2020) prioritize agents using communication between agents during execution to avoid conflicts. Buckley (1989) prioritize agents so as to prevent conflicts that may be present due to other agents' start and goal vertices. Zhang et al. (2022) use machine-learning methods to prioritize agents. We consider deterministic calculations for prioritizing the agents.

Online MAPF (OMAPF) (Švancara et al. 2019) extends MAPF to the case where new agents may appear over time. Lifelong MAPF (LMAPF) (Li et al. 2020; Wan et al. 2018) extends MAPF to the case where new tasks arrive over time. In both LMAPF and OMAPF, it is assumed that the solver is not aware of future agents/tasks. The solver gets information about new agents/tasks only when they are about to arrive. Therefore, it is common to plan paths when it is needed. This approach results in solving a new (offline) MAPF problem each time a new agent appears. The SEQ algorithm was originally defined for OMAPF. Our problem definition is also related to the definition of well-formed infrastructure (Čáp et al. 2015), in which each agent has a path that does not pass through other agents' start or goal vertices. However, we assume that agents appear at their start vertices and disappear when their goals are reached. In the *Push and Swap* and *Push and Rotate* algorithms (Luna and Bekris 2011; De Wilde, Ter Mors, and Witteveen 2014), each agent follows a shortest path in turn and it performs a push action to resolve conflicts. In our problem, as agents disappear at goals, no such actions are needed. Thus, under our assumptions, these two algorithms are similar to SEQ.

9 Conclusion and Future Work

In this paper, we define geometric constraints for MAPF and show how they can be used to determine safe delays for the agents. We also propose DSP, an algorithm that, given a priority order, approximates minimal safe delays for quickly finding a plan. We show experimentally that DSP is able to find low-cost solutions for instances with thousands of agents with a runtime that is often several orders of magnitudes smaller than related methods. In future work, we will use the geometric constraints for other algorithms, e.g. *priority based-search* (Ma et al. 2019), and extend them for solving related problems, such as OMAPF and LMAPF.

Acknowledgements

For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising. This work is partially supported by Innovate UK (VersaTile Grant - 10005401), Leverhulme Trust (Grant VP1-2019-037), and MIUR (Grant CUP: E11G18000350001).

References

Atzmon, D.; Stern, R.; Felner, A.; Sturtevant, N. R.; and Koenig, S. 2020a. Probabilistic Robust Multi-Agent Path Finding. In *ICAPS*, 29–37.

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

Buckley, S. 1989. Fast motion planning for multiple moving robots. *IEEE International Conference on Robotics and Automation*, 1: 322–326.

De Wilde, B.; Ter Mors, A. W.; and Witteveen, C. 2014. Push and Rotate: A Complete Multi-Agent Pathfinding Algorithm. *Journal of Artificial Intelligence Research*, 443—-492.

Erdmann, M.; and Lozano-Pérez, T. 1987. On Multiple Moving Objects. *Algorithmica*, 2(1–4): 477–521.

Ho, F.; Salta, A.; Geraldes, R.; Goncalves, A.; Cavazza, M.; and Prendinger, H. 2019. Multi-Agent Path Finding for UAV Traffic Management. In *AAMAS*, 131–139.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2019. Persistent and Robust Execution of MAPF Schedules in Warehouses. *IEEE Robotics and Automation Letters*, 4(2): 1125–1131.

Lenstra, J.; Rinnooy Kan, A.; and Brucker, P. 1977. *Complexity of Machine Scheduling Problems*, volume 1 of *Studies in Integer Programming*. Elsevier.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2020. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. In *AAAI*, 11272–11281.

Luna, R.; and Bekris, K. E. 2011. Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, 294—300.

Ma, H. 2021. A Competitive Analysis of Online Multi-Agent Path Finding. In *ICAPS*, 234–242.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *AAAI*, 7643–7650.

Ma, H.; Kumar, T. K. S.; and Koenig, S. 2017. Multi-Agent Path Finding with Delay Probabilities. In *AAAI*, 3605–3612.

Shahar, T.; Shekhar, S.; Atzmon, D.; Saffidine, A.; Juba, B.; and Stern, R. 2021. Safe Multi-Agent Pathfinding with Time Uncertainty. *JAIR*, 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.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195: 470–495.

Silver, D. 2005. Cooperative Pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 117–122.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *SoCS*, 151–159.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2): 144–148.

Surynek, P. 2010. An Optimization Variant of Multi-Robot Path Planning Is Intractable. In *AAAI*, 1261–1263.

Švancara, J.; Vlk, M.; Stern, R.; Atzmon, D.; and Barták, R. 2019. Online multi-agent pathfinding. In *AAAI*, 7732–7739.

van den Berg, J.; and Overmars, M. 2005. Prioritized motion planning for multiple robots. In *the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 430–435.

Wagner, G.; and Choset, H. 2015. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219: 1–24.

Wagner, G.; and Choset, H. 2017. Path Planning for Multiple Agents under Uncertainty. In *ICAPS*, 577–585.

Wan, Q.; Gu, C.; Sun, S.; Chen, M.; Huang, H.; and Jia, X. 2018. Lifelong Multi-Agent Path Finding in A Dynamic Environment. In *the International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 875–882.

Wen, L.; Liu, Y.; and Li, H. 2022. CL-MAPF: Multi-Agent Path Finding for Car-Like robots with kinematic and spatiotemporal constraints. *Robotics and Autonomous Systems*, 150: 103997.

Wu, W.; Bhattacharya, S.; and Prorok, A. 2020. Multi-Robot Path Deconfliction through Prioritization by Path Prospects. In *ICRA*, 9809–9815.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI*, 1444–149.

Zhang, S.; Li, J.; Huang, T.; Koenig, S.; and Dilkina, B. 2022. Learning a Priority Ordering for Prioritized Planning in Multi-Agent Path Finding. In *SoCS*, 208–216.

Čáp, M.; Novák, P.; Kleiner, A.; and Selecký, M. 2015. Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots. *IEEE Transactions on Automation Science and Engineering*, 12(3): 835–849.