# Improved Complexity Results and an Efficient Solution for Connected Multi-Agent Path Finding

Isseïnie Calviac
Univ Rennes, Inria, CNRS
Rennes, France
isseinie.calviac@ens-rennes.fr

Ocan Sankur
Univ Rennes, Inria, CNRS
Rennes, France
ocan.sankur@cnrs.fr

François Schwarzentruber
Univ Rennes, CNRS
Rennes, France
francois.schwarzentruber@ens-rennes.fr

## ABSTRACT

Connected multi-agent path finding (CMAPF) consists in computing paths for multiple agents which must reach a goal configuration while remaining connected at all steps. We prove the PSPACE-hardness of the problem when the underlying graph is a subgraph of a 3D grid and with range-based connectivity. Moreover, we provide an application of the WHCA* algorithm and show that it outperforms previously given algorithms by an order of magnitude in terms of the sizes of the instances it can handle.

## KEYWORDS

AI planning; multi-agent path finding; connectivity; CA*

**ACM Reference Format:**
Isseïnie Calviac, Ocan Sankur, and François Schwarzentruber. 2023. Improved Complexity Results and an Efficient Solution for Connected Multi-Agent Path Finding. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

## 1 INTRODUCTION

Multiple agents may have to cooperate in various situations in order to achieve a goal, such as search and rescue missions, or nuclear decommissioning. Some applications require agents to remain connected during the mission, for example, in order to transmit a video stream or other data to human operators [1].

In this article, we study the so-called Connected Multi-Agent Path Finding (CMAPF) problem [3, 6, 11, 16] which is the extension of the multi-agent path finding problem [8] to deal with connectivity constraints between agents. In this setting, the environment is modeled by a graph whose nodes are locations that can be occupied by a single agent at any time. There are two types of edges: movement and communication edges. Agents start in some starting locations (sources) and have to reach target locations, while forming a connected graph through communication edges at each step.

In [16], the authors prove that CMAPF is PSPACE-complete in general graphs, and give one deterministic and two randomized algorithms which consist in greedily selecting a successor configuration at each step so as to build an execution towards the targets. Surprisingly, their algorithms do not consider the collision constraints (i.e. allow collisions between agents). This study however does not fully settle the theoretical complexity of the problem. In fact, PSPACE-hardness result requires arbitrary graphs. Although the movement edges form a planar graph in their reduction, the communication graph has a very particular shape, and is far from being planar. So the PSPACE-hardness holds but on graphs that may look artificial. Many applications of MAPF are actually restricted to subgraphs of grids [7], and communication is often determined by range [1], that is, two agents can communicate whenever their distance is smaller than a given threshold. Our first objective in this paper was therefore to establish the complexity of the problem for subgraphs of grids and range-based communication. Second, algorithmic solutions given in [16] do not scale beyond about 10-20 agents (despite allowing collisions between agents). Our second objective was to show that it is possible to derive much more efficient algorithmic solutions for the CMAPF problem by exploiting the multi-agent planning literature. Moreover, we enforce collision constraints, since we believe that the CMAPF problem only makes sense when both collision and connectivity constraints are taken into consideration.

This paper provides two contributions.

(1) First we show that CMAPF is PSPACE-hard even when agents move in a subgraph of a 3D grid, and with range-based communication, that is, when two agents can communicate when their distance is within a given range. Our reduction is from *non-deterministic constraint logic* (NCL) [5], and is based on [16]. NCL is a computation model based on a so-called AND/OR graph in which edges are to be flipped sequentially. The main technical challenge we solve is to provide gadgets that mimic the sequential flips of edges.
(2) Second, we provide an algorithmic solution based on *windowed hierarchical cooperative $A^*$* (WHCA*), and provide a randomized conflict resolution mechanism well adapted to connectivity constraints, which scales to instances with an order of magnitude more agents than [16]. This solution has the advantage of being simple, and finding plans often quickly, and thus significantly improves over the previously given algorithms. Moreover, the randomization we introduce allows us to address the incompleteness of the basic WHCA* algorithm and solve more instances. For example, on some benchmarks, we were able to solve instances with about 80 agents where the performance of the previous algorithm dropped after about 10-20 agents.

*Related Work.* Connectivity constraints are very different in nature than collision constraints alone, so most techniques developed for MAPF in the literature do not easily apply to connectivity constraints. In fact, while collisions are often local (occur at a given time step) and mostly involve two or a few agents, connectivity

*Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.* © 2023 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

constraints are *global* and *continuous*, that is, they involve the set of all agents and a violation of connectivity can span a large time window, if not the whole execution. So these conflicts cannot be dealt locally, unlike collision constraints.

The popular conflict-based search algorithm for MAPF [13] (without connectivity) is difficult to generalize connectivity constraints although some attempt was made [10] which allows collisions between agents and only focus on connectivity; but the scalability seemed limited. We also observe that allowing collisions has little interest in practice, and it renders the problem easier (this can also be observed in the experiments of [16]). In instances with particular communication graphs called sight-moveable, and in the presence of a *basis vertex* to which the group of agents are to be constantly connected, the problem was shown to be in LOGSPACE [3]. However, this does not apply to grid graphs with range-based communication.

MAPF with imperfect information has been considered in the literature, for instance [17] or [9]. The closer work seems to be [11]. They also use topological graphs with both movement and communication edges. The connectivity is then taken into account to compute the knowledge of each agents. In our work, we consider the perfect information case where connectivity must be maintained.

Improvements over WHCA$^*$ have been considered e.g. [2] that specifically target conflicts due to collisions. Some other algorithms that target collisions are [18, 19]. Algorithms combining plans for groups of agents such as [15] are also difficult to apply here due to connectivity being a global constraint on all agents.

*Outline* In Section 2 we recall the background about CMAPF. In Section 3, we first recall NCL and then explain our reduction to prove that CMAPF is PSPACE-hard for 3D instances. Section 4 provides our algorithm and its performance against the state-of-art ones. Section 5 is the conclusion.

## 2 SETTING

The environment of the CMAPF problem is represented by a so-called *topological graph* $G = (V, E_M, E_C)$ with a non-empty finite set $V$ of vertices, and with two types of undirected edges: the movement edges (set $E_M$) and the communication edges (set $E_C$). We denote $G_M = \langle V, E_M \rangle$ and $G_C = \langle V, E_C \rangle$. Figure 1 shows such a graph which contains 8 vertices. The solid lines represent the movement edges and the dashed ones the communication edges. The agents can thus move along the solid lines and communicate with other agents along the dashed ones.

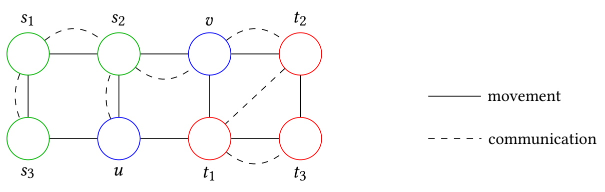


**Figure 1: Example of a topological graph for the CMAPF problem.**

We consider $n$ agents that must move in $G_M$ from their initial vertices to their target ones.

A *configuration* $c$ of $n$ agents in $V$ is a tuple of $n$ distinct vertices of $V$, denoted $(c_1, ...c_n)$ where for $i \in \{1, ..., n\}$, $c_i$ is the position of the agent $a_i$. The initial configuration is $s = (s_1, ..., s_n)$ and the final one $t = (t_1, ..., t_n)$. Two configurations $c$ and $c'$ of length $n$ are consequent if and only if for each $i \in \{1, ..., n\}$ we have $(c_i, c'_i) \in E_M$; thus, each agent makes one move in $G_M$. We allow agents to idle. Moreover, the agents must stay connected along their movements. We say that a configuration $c$ of $n$ agents is connected if and only if it forms a connected sub-graph of $G_C$. In Figure 1, assume that we have 3 agents that must move from $(s_1, s_2, s_3)$ to $(t_1, t_2, t_3)$. The configuration $(s_1, s_2, s_3)$ is connected because $(s_1, s_2), (s_1, s_3) \in E_C$. Intuitively, agent at $s_3$ can communicate with the agent at $s_2$ via the agent at $s_1$ (multi-hop).

An execution $e$ of length $\ell$ is a sequence of configurations, denoted $(c^1, ..., c^\ell)$ such that for each $i \in \{1, ..., \ell - 1\}$, $c^i$ and $c^{i+1}$ are consequent. An execution $e$ of length $\ell$ is connected in the graph of communication $G_C$ if for each $i \in \{1, ..., \ell\}$, $c^i$ is connected. We want a connected execution from $s$ to $t$.

Importantly, as in MAPF (unlike [16]), we suppose that agents do not collide. In other words, agents have distinct positions in all configurations. We thus consider here a simple form of collision constraints; one could as well consider forbidding taking the same edge in opposite directions [20].

*Example 2.1.* Figure 2 shows an example of a connected execution from the Figure 1. Agents start in configuration $(s_1, s_2, s_3)$ (see Figure 1(a)). Note that agent $a_2$ communicates indirectly with agent $a_3$, via the agent $a_1$. Then the agents move synchronously to configuration $(s_2, v, u)$: agent $a_1$ moves to $s_2$ by taking the movement edge $(s1, s2) \in E_M$, agent $a_2$ takes the edge $(s2, v) \in E_M$ and $a_3$ takes $(s3, u) \in E_M$. At step 2, the configuration is $(v, t_2, t_1)$. Note that now agent $a_2$ communicates directly with agent $a_3$ via the communication edge $(t_2, t_1) \in E_C$; $a_1$ communicates with $a_3$ indirectly via $a_2$.

*Definition 2.2 (CMAPF Problem).* Given $\langle G, s, t \rangle$ decide if there is a connected execution $(c^1, ..., c^\ell)$ such that $c^1 = s$ and $c^\ell = t$. We say that the execution is from $s$ to $t$.

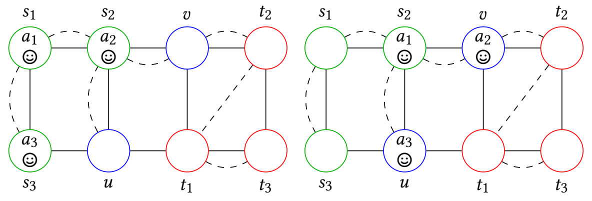
The CMAPF problem was proven to be PSPACE-complete on general graphs [16]. However, the reduction requires a connectivity graph $G_C$ that is arbitrary and unrealistic. In the next section, we establish the PSPACE-hardness of the problem on instances defined on subgraphs of 3D grids, and with range-based communication.

## 3 PSPACE-COMPLETENESS IN 3D

We focus on 3D grid instances where the communication is range-based.

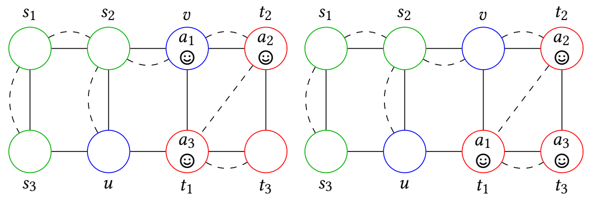
*Definition 3.1.* A 3D grid topological graph is a triplet $(G, pos, \rho)$ where $G = (V, E_M, E_C)$ is a topological graph, $pos$ is a position function $pos : V \longrightarrow \mathbb{N}^3$, which is injective, and a radius $\rho$ which respects the following conditions:

(1) for all $(u, v) \in E_M$,
$$|\text{pos}(u).x - \text{pos}(v).x| + |\text{pos}(u).y - \text{pos}(v).y| + |\text{pos}(u).z - \text{pos}(v).z| \leq 1,$$
where $\text{pos}(u).x, \text{pos}(u).y, \text{pos}(u).z$ denote the three components of $\text{pos}(u)$;



(a) **Step 0: agents are at the initial configuration** $(s_1, s_2, s_3)$

(b) **Step 1: agents move to configuration** $(s_2, v, u)$



(c) **Step 2: agents move to configuration** $(v, t_2, t_1)$

(d) **Step 3: agents are at the final configuration** $(t_1, t_2, t_3)$

**Figure 2: Example of connected execution.**

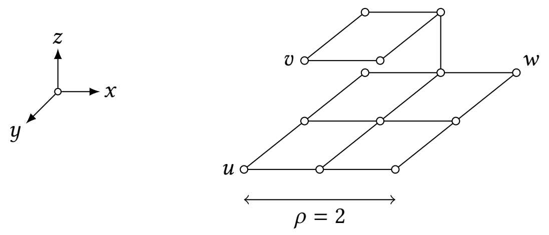


**Figure 3: 3D grid topological graph, with 13 vertices and with a radius $\rho = 2$.**

(2) and $(u, v) \in E_C$ if and only if $d(\text{pos}(u), \text{pos}(v)) \leq \rho$, where $d$ is the Euclidean distance (range-based communication).

*Example 3.2.* Figure 3 shows a 3D grid topological graph with 13 vertices and with a radius $\rho = 2$. The graph consists of two layers: the bottom layer ($z = 0$) contains 9 vertices, while the top layer ($z = 1$) contains 4 vertices. As shown, the graph can be seen a subgraph of the 3D grid. For instance, $pos(u) = (0, 2, 0)$, $pos(v) = (0, 1, 1)$ and $pos(w) = (2, 0, 0)$. An agent can move from layer 0 to layer 1 throw a movement edge from the node at position $(1, 0, 0)$ to the node at position $(1, 0, 1)$. We have:

- $(u, v) \in E_C$ because $\sqrt{0^2 + 1^2 + 1^2} = \sqrt{2} \leq 2$;
- $(u, w) \notin E_C$ because $\sqrt{2^2 + 2^2 + 0} = \sqrt{8} > 2$.

Now, a 3D grid instance for the CMAPF problem is described by a 3D topological graph $(G, pos, \rho, s, t)$ where $(G, pos, \rho)$ is a 3D grid topological graph, and two connected configurations $s$ and $t$. In this section, we prove the following theorem:

Theorem 3.3. *CMAPF is PSPACE-complete, even for a graph which is a 3D grid instance.*

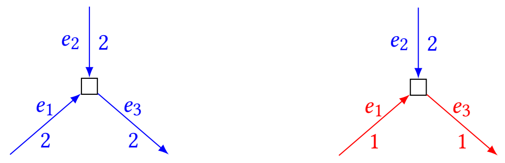


**Figure 4: The two types of nodes in an AND/OR graph: OR node (left) and AND node (right). An OR node has three incident edges of weight 2 (blue), while an AND has two incident edges of weight 1 (red) and one of weight 2.**

PSPACE membership holds because CMAPF is in PSPACE in the general case [16]. The PSPACE-hardness on arbitrary graphs relies on a reduction from the the PSPACE-complete problem related to Nondeterministic Constraint Logic (NCL) [5]. Here, we also provide a reduction from NCL, while our technical contribution consists in defining the reduction with the restrictions of a 3D grid graph, and only using range-based communication.

## 3.1 Nondeterministic Constraint Logic

Let us recall that a graph is planar if there exists an injective embedding of the set of nodes in $\mathbb{N}^2$ such that edges do not cross. An AND/OR graph is an *undirected* planar graph $\mathfrak{G}$ such that:

- each edge has a weight of either 1 or 2;
- each node is either an OR node, or an AND node (see Figure 4; ignore the direction of edges for now).

As shown in Figure 4, a node has three incident edges $e_1, e_2, e_3$. All are of weight 2 for an OR node. For an AND node, two of them, say $e_1, e_3$, are of weight 1 while the third one, $e_2$, is of weight 2.

Given an AND/OR graph $\mathfrak{G}$, a *configuration* $\gamma$ is an orientation of $\mathfrak{G}$, defining a direction for each edge in $\mathfrak{G}$; $\gamma$ is *valid* if the in-flow of each node (the sum of the weights of the incoming edges) is at least 2 (in the directed graph defined by the pair $(\mathfrak{G}, \gamma)$). Concretely, for an OR node, at least one edge must enter the node. For an AND node, either $e_2$ enters the node or $e_1$ and $e_3$ both enter the node. Given two valid configurations $\gamma, \gamma'$, we define an *elementary step* denoted, $\gamma \xrightarrow{\text{flip}} \gamma'$, iff $\gamma'$ is obtained from $\gamma$ by flipping the direction of exactly one edge. An NCL execution is a sequence of valid configurations $\gamma_0, \ldots, \gamma_\ell$ with $\gamma_i \xrightarrow{\text{flip}} \gamma_{i+1}$.

*Definition 3.4.* The NCL reconfiguration problem is defined as follows: given an AND/OR graph $\mathfrak{G}$ and two valid configurations $\gamma_0, \gamma_f$, does there exist an execution $\gamma_0 \xrightarrow{\text{flip}} \ldots \xrightarrow{\text{flip}} \gamma_f$?

Theorem 3.5. *[5] The NCL reconfiguration problem is PSPACE-complete.*

## 3.2 Description of the reduction

Let us consider a NCL instance $(\mathfrak{G}, \gamma_0, \gamma_f)$. We show how to construct in poly-time a 3D CMAPF instance written $tr(\mathfrak{G}, \gamma_0, \gamma_f) = (G, pos, \rho, s, t)$ such that $\gamma_0 \xrightarrow{\text{flip}} \ldots \xrightarrow{\text{flip}} \gamma_f$ iff there is a connected execution from $s$ to $t$.

The 3D grid topological graph $(G, pos, \rho)$ contains 5 layers as shown in Figure 5. Each layer lies in a $Oxy$-plane (a plane in which the $z$-coordinate is constant). As we will see, each layer contains some gadgets. As indicated in Figure 5, layer 0 and 1 are at distance 2, and the $Oxy$-plane between layer 0 and 1 does not contain any node (in other words, it is an obstacle). In the same way, layer 1 and 2 are at distance 8 and are separated by 7 $Oxy$-planes without any node (obstacle), and so on. Let $\rho = 10$ be our communication radius.

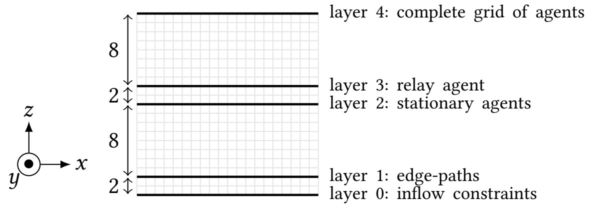

layer 4: complete grid of agents
8
layer 3: relay agent
2
layer 2: stationary agents
8
layer 1: edge-paths
2
layer 0: inflow constraints
z
x
y

**Figure 5: Sectional view of the organisation of the 3D topological graph $G$. Layers contain gadgets. Each layer is given with a summary of its contents.**

*Encoding of the AND/OR graph* $\mathfrak{G}$. We consider an embedding of $\mathfrak{G}$ in a 2D grid in which edges do not cross each other and follow the lines of the grid; such an embedding always exists and is computable in poly-time [12].

In layer 1, we place *flipping agents* that will simulate the directions of the edges on $\mathfrak{G}$. This layer has the same shape as the embedding of the AND/OR graph $\mathfrak{G}$ and each node in $\mathfrak{G}$ is replaced by a gadget of the form given in Figure 6. The edge $e_i$ in Figure 4 is symbolized by the path between nodes $u_i$ and $v_i$, called an *edge-path*. For simplicity, in Figure 4, the edge-path is depicted as a straight line, but in general it follows the shape of that edge in the planar AND/OR graph embedding. The flip of $e_i$ corresponds to the agent moving from $u_i$ to $v_i$, or from $v_i$ to $u_i$. We say that a flipping agent is near $u_i$ (resp. $v_i$) when it is placed at a distance of at most 6 from $u_i$ (resp. $v_i$) on the layer 1. When the agent is placed near $u_i$ (resp. near $v_i$), it means that the edge $e_i$ is going in (resp. out) the AND/OR node (in Figure 4). A flipping agent can in general be anywhere on its edge-path between $u_i$ and $v_i$. But when they are all near extremities $u_i$ or $v_i$, this corresponds to an AND/OR graph configuration.

In order to impose the dynamics of NCL, we introduce layer 4 which is far from layer 1 and thus is not connected to it directly. Layer 4 contains just a complete grid of stationary agents (see Figure 9) that are connected. We will introduce layer 2 and 3 that respectively impose that a configuration of the agents should represent a AND/OR configuration, and that a single flip occurs each time.

*Connectivity for AND/OR configurations.* Layer 2, shown in Fig. 7, provides connectivity to flipping agents who are near some edge-path extremities $u_i$ or $v_i$. It consists of stationary agents placed at the vertices $u'_i$ and $v'_i$: vertex $u'_i$ (resp. $v'_i$) is placed exactly above $u_i$ (resp. $v_i$). Layer 2 is placed at a distance of 8 from layer 1. Thereby, a stationary agent only communicates with the flipping agent that is just below them (or almost below but near the extremity). More precisely, the flipping agent is near the extremity if and only if it communicates directly with the stationary agent above it, as $\sqrt{6^2+8^2} = 10$. For instance, the stationary agent at $u'_i$ communicates with the agent near $u_i$ if there is one. Note that the stationary agents are all connected to layer 4.

*Flips.* An elementary step $\gamma \xrightarrow{\text{flip}} \gamma'$ consists in flipping *exactly one* edge. In other words, we should ensure that at most one flipping agent should move from an extremity to another. To do that, we introduce layer 3 made of a fully connected grid with a relay agent on it (Figure 8). The idea is that the relay agent follows the flipping agent while she moves from $u_i$ to $v_i$ (or conversely). The layer 3 is at a distance of 10 from layer 1 to ensure that the relay agent provides the connectivity of layer 4 to at most one flipping agent that attempts to move.

*NCL configurations should be valid.* Last, we add layer 0 to ensure that the incoming flow in each AND/OR node is at least 2 (Figure 10).

(a) An OR node must be pointed to by at least one edge, since all edges $e_1, e_2$ and $e_3$ are of weight 2. In our construction, we need thus to ensure that at least one flipping agent must be near $u_1, u_2$ or $u_3$. To this aim, we introduce a line of agents $a_1, a_2, a_3$ that are all connected (see Figure 10a). For them to be connected with all the other agents, one flipping agent must be near $u_1, u_2$ or $u_3$.
For instance, a flipping agent in $u_1$ provides direct connectivity to $a_1$: the distance between $u_1$ and the position $a_1$ is $\sqrt{6^2+6^2+2^2} = \sqrt{76} < 10$. Note that agents $a_1 - a_2 - a_3$ will not be connected otherwise: agents in layer 2 are too far from $a_1 - a_2 - a_3$ to provide connectivity to $a_1 - a_2 - a_3$.

(b) For an AND node, recall that the edges $e_1$, $e_2$ and $e_3$ are of weights 1, 2 and 1 respectively. The constraint of having an in-flow of at least 2 can be reformulated by:
(i) either $e_1$ or $e_2$ is pointing in (or both);
(ii) and either $e_2$ or $e_3$ is pointing in (or both).
We therefore add the line of agents $a'_1 - a'_2$ (resp. $a''_2 - a''_3$) for handling condition (i) (resp. (ii)). If $e_1$ and $e_2$ are pointing out (condition (i) is unsatisfied), then it means no agent is in $u_1$ (or nearby) and no agent is in $u_2$ (or nearby), then the agents $a'_1 - a'_2$ are disconnected from the rest of the group.

We defined the vertices of $tr(\mathfrak{G}, \gamma_0, \gamma_f) = (G, s, t)$ as well as their positions. It remains to define $s$ and $t$. We show how to map and AND/OR graph configuration $\gamma$ to a configuration of agents $c_\gamma$. We define $s := c_{\gamma_0}$ and $t := c_{\gamma_f}$. For an AND/OR configuration $\gamma$, $c_\gamma$ is the configuration in which, at each AND or OR node, in the corresponding gadget of Figure 6, if $e_i$ enters that node, then the corresponding agent is in $u_i$ (and if $e_i$ leaves that node, she is near $v_i$). The relay agent is, say, at the bottom-left corner as in Figure 8. The other agents are stationary so their positions are obvious.

## 3.3 Properties of the reduction

Proposition 3.6. *$tr(\mathfrak{G}, \gamma_0, \gamma_f) = (G, pos, \rho, s, t)$ is computable in time polynomial in the size of $\mathfrak{G}, \gamma_0, \gamma_f$.*

Proof. The computation starts by computing an embedding for the graph $\mathfrak{G}$ in a grid in $poly(|\mathfrak{G}|)$ (this is doable via the result of

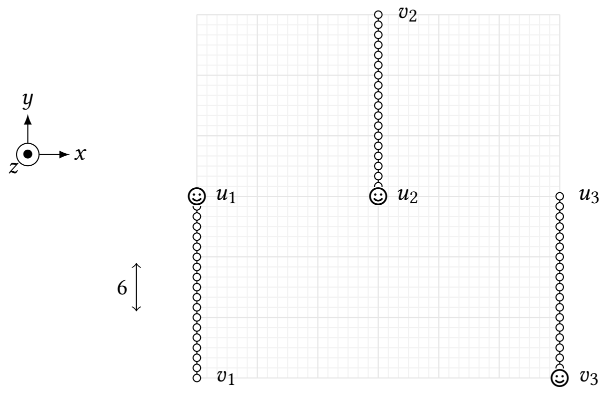


**Figure 6: Layer 1: Three edge-paths that simulate an AND or OR vertex in the AND/OR graph. Each path $u_i - v_i$, called an edge-path, corresponds to an edge in the AND/OR graph. For instance, the agent at $u_1$ can move along its edge-path until $v_1$.**

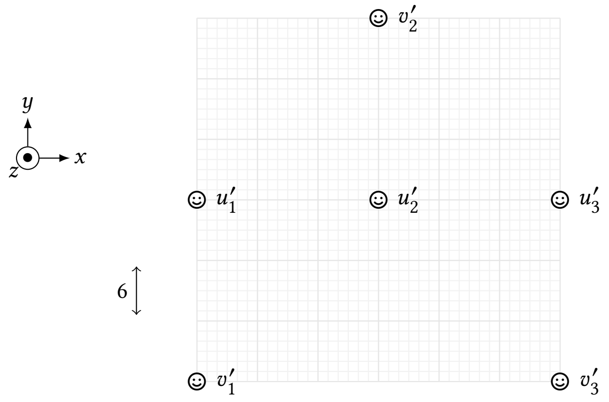


**Figure 7: Layer 2: Stationary isolated agents placed above the extremities of the edge-paths.**

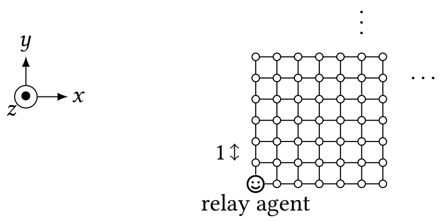


**Figure 8: Layer 3: a complete connected grid with a single so-called relay agent being moving on it.**

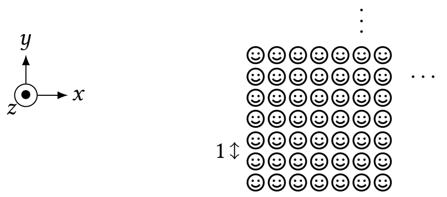


**Figure 9: Layer 4: a complete grid of isolated agents.**

[12]). That embedding will then give the positions of the gadget on layer 0 and 1. □

Proposition 3.7. *With $tr(\mathfrak{G}, \gamma_0, \gamma_f) = (G, s, t)$, we have: $\gamma_0 \xrightarrow{flip} \dots \xrightarrow{flip} \gamma_f$ iff there is a connected execution from $s$ to $t$.*

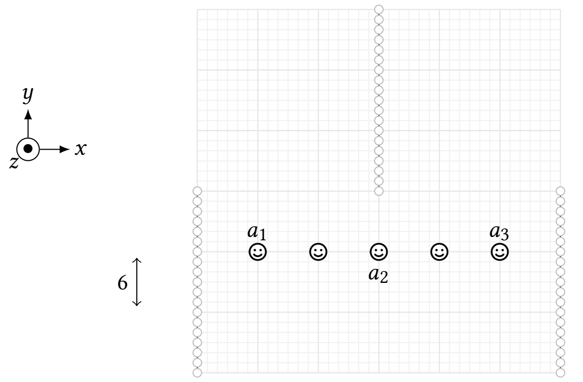
Proof. $\boxed{\Rightarrow}$ Suppose $\gamma_0 \xrightarrow{\text{flip}} \dots \xrightarrow{\text{flip}} \gamma_f$. We construct an execution from that sequence of flips as follows. For each flip of a given edge, we move the corresponding flipping agent to the other extremities ($u_i$ to $v_i$, or $v_i$ to $u_i$) with the relay agent moving above that flipping agent on the same way in its own layer 3. The obtained execution is connected. On the one hand, when flipping agents are at some $u_i/v_i$, the connectivity is guaranteed by the agents of layer 2. On the other hand, when a flipping agent is moving, its connectivity is guaranteed by the relay agent. Agents $a_1 - a_2 - a_3$ and $a_1' - a_2' - a_2'' - a_3''$ of Figure 10 remain connected to the rest of the group because the in-flow at each AND/OR node is at least 2.

$\boxed{\Leftarrow}$ Conversely, consider a connected execution from $s$ to $t$. We extract a sequence of flips as follows. When a flipping agent moves far from an edge-path extremity, it must be accompanied by the relay agent in order to remain connected to the rest of the agents. During its move, it starts near some $u_i$ (or $v_i$). If it goes back near the same extremity ($u_i$ to $u_i$, or $v_i$ to $v_i$), the move is simply ignored. Otherwise, if the flipping agent goes near the other extremity ($u_i$ to $v_i$, or $v_i$ to $u_i$), that move is interpreted as a flip of the corresponding edge. The sequence of flips extracted in that way transform $\gamma_0$ into $\gamma_f$. Again, as the agents of layer 0 are always connected, it guarantees that the in-flow at each AND/OR node during the extracted sequence of flips is at least 2. □

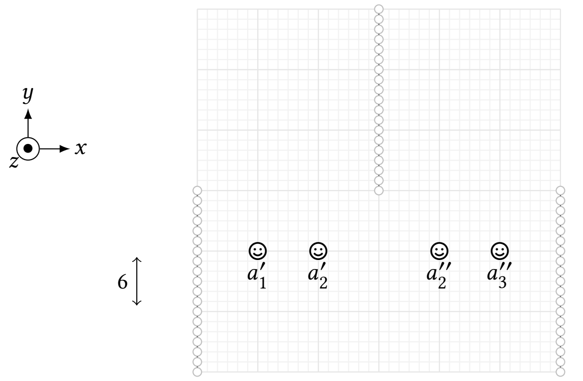
## 4 ALGORITHM

In this section, we will present a simple algorithm to solve the connected MAPF problem. We first review previously published algorithms from [16], which will be the baseline for comparison.

One of the main difficulties in multi-agent path finding is combinatorial explosion due to the number of agents. In fact, from each configuration, there is, in general, an exponential number of successor configurations despite the connectivity constraints. Searching for a plan in an exponentially-branching state space is infeasible, and even choosing a good successor configuration is a nontrivial task. To deal with the large branching factor, a greedy approach was presented [16] in which a heuristic A* search is used to select a successor configuration step by step to minimize the remaining distance to the target configuration. This becomes theoretically complete with an additional backtracking mechanism. The resulting algorithm is similar to a DFS over the configuration space guided by a heuristic; so it will be named simply DFS in our experiments. We use here the authors' own implementation modified to take collision constraints into account; this was a straightforward adaptation. The authors also present sampling-based algorithms



(a) Simulating an OR node: agents from $a_1 - a_2 - a_3$ must be connected.



(b) Simulating an AND node: agents from $a'_1 - a'_2$ and $a''_2 - a''_3$ must be connected.

**Figure 10: Layer 0: handling the inflow of at least 2 at each node. Agents $a_1 - a_2 - a_3$, $a'1, a'_2, a''_2, a''_3$ are all isolated and thus stationary. Vertices shown in gray are those in Layer 1.**

in which the successors are selected with a randomization scheme. These scale up to 10 agents [16] but become inefficient above 20 agents. We will only consider the deterministic DFS algorithm as a representative of this approach. An algorithm is described in [6] but within a different setting requiring only periodic connectivity.

## 4.1 Our suggestion

We show here that a straightforward application of the windowed hierarchical cooperative A* (WHCA*) [14] with a simple randomized conflict resolution mechanism performs much better than the above algorithms from the literature. The idea of CA*, explained in our setting, is the following. In order to compute a plan for $n$ agents, we select a random total order of the agents. We first compute a shortest path $\rho^1$ for the first agent from their source to target vertices. At iteration $i$, we compute a shortest path $\rho^i$ for the $i$-th agent from their source to target, but subject to the constraints of the previous agents: the $i$-th agent cannot occupy the same vertex as a previous agent at the same time step (that is, $\rho^i_t \neq \rho^j_t$ for all $1 \leq j < i$ and all $t$), and moreover they must be connected to one of the previous agents at all moments (that is, for each $t$, there exists $1 \leq j < i$ such that $(\rho^j_t, \rho^i_t) \in E_C$).

The algorithm retries different orderings until an execution is found. The *window* optimization consists in changing the order randomly after a certain number of steps. Furthermore, the *hierarchical* version uses heuristic values that are shortest path distances computed by ignoring all interactions between agents. With both optimizations, this defines the WHCA* algorithm. We call each iteration that starts from the source a *trial*.

It is known however that WHCA* is not complete [14]: on some instances that a solution, WHCA* does not find one. The source of incompleteness is due to the fact that the algorithm assumes that one of the agents follows a shortest path. Such a situation is depicted in Fig. 11. In the figure, the target of the upper agent is $t_1$, and their shortest path moves upwards, and disconnects from the bottom agent. The bottom agent's situation is symmetric: they want to reach the vertex $t_2$, and their shortest path moves down, disconnecting from the other agent. The only solution is to pass through the path at the middle; but then none of the agents use a shortest path, and WHCA* fails. WHCA* can also fail due to collision conflicts if the execution requires agents to idle, for instance, to let other agents pass.

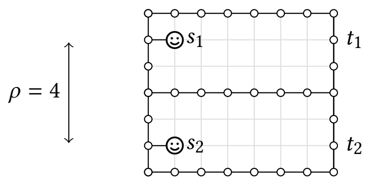


**Figure 11: An instance that is difficult for WHCA*. Agent at $s_1$ should reach $t_1$, agent at $s_2$ should reach $t_2$.**

*Randomized conflict resolution.* We focus here on conflicts due to connectivity constraints, and introduce a simple mechanism to resolve such conflicts, well adapted to connectivity constraints. While running WHCA*, if no solution is found after $\theta$ trials, then we start each subsequent trial by randomly selecting a direction, and moving all agents from the source towards that direction for a number $\ell$ of steps using WHCA*, and continuing the trial from this new configuration to target. We also apply randomization inside each trial: if the execution has not been extended in the last $\theta'$ windows, then we move towards a random configuration, and continue our way towards target. We slowly increment $\ell$ after each trial so that if a longer execution towards a particular direction is necessary, this will be tried eventually. In Fig. 11, moving the agents towards left (including upper or bottom left), for a few steps suffices to unblock the situation, a solution is found eventually.

The resulting algorithm is probabilistically complete. In fact, when $\ell$ is sufficiently large, if there is a connected execution from the current configuration to goal configuration, then, there is a nonzero probability to pick precisely that execution. This holds at each step with a uniform lower bound on the probability; thus, it will eventually be picked with probability 1. Of course, no useful bound on the expected time can be obtained from this reasoning. Thus, we rather evaluate the algorithm empirically.

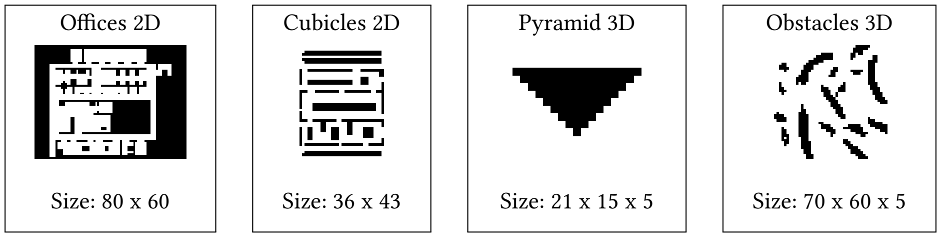


**Figure 12: The four maps used to obtain topological graphs. Obstacles are black pixels. The communication range is 1 pixel for Offices and Obstacles, and 3 pixels for Cubicles and Pyramid.**

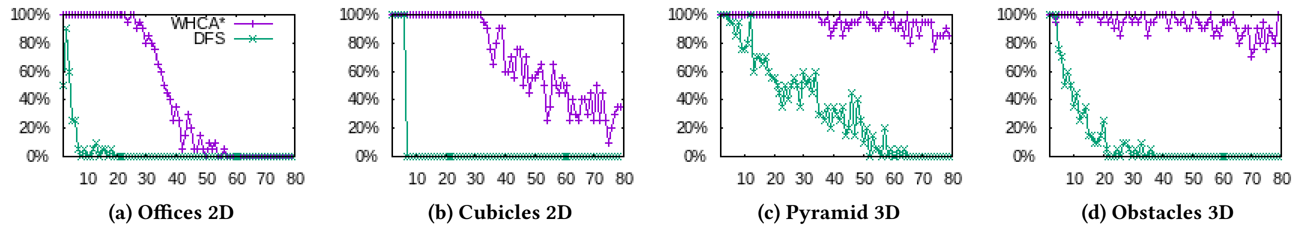


**Figure 13: Average success rate of the algorithms as a function of the number of agents. Each point $(x, y)$ means that a percentage of $y$ among the 20 instances with $x$ agents were solved each within the timeout (5 minutes).**

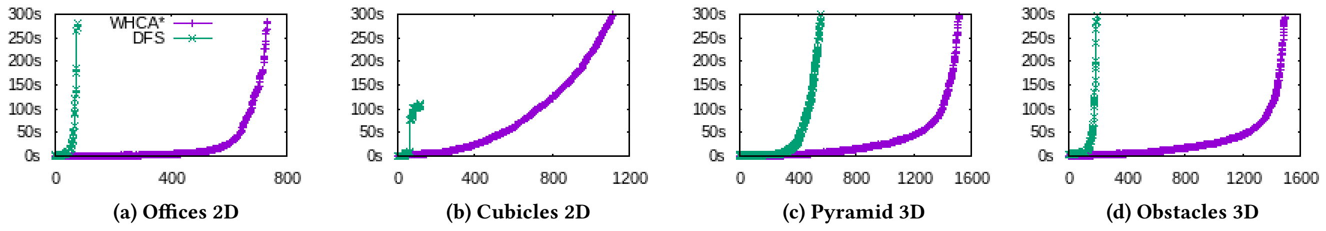


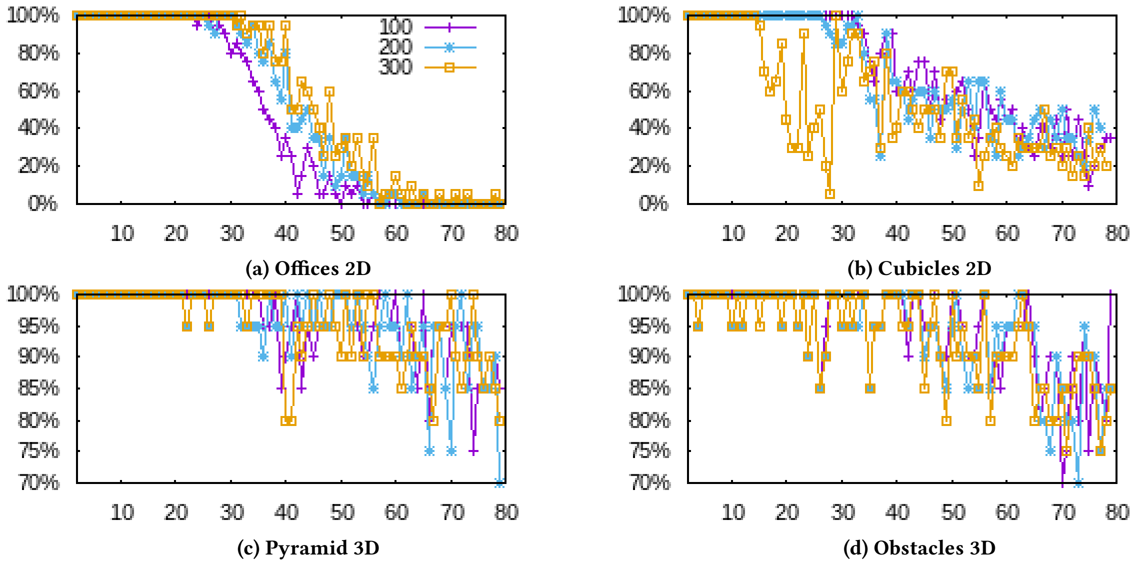
**Figure 14: Cactus plots of the execution times: point $(x, y)$ means that $x$ instances were solved each within $y$ seconds.**

The approach of WHCA* is particularly well adapted for connected MAPF. In fact, due to connectivity, agents are tightly dependent on each other since none of them can travel far from the group. This makes the problem quite different than the disconnected case where interactions between agents can be sparse and local (only a few collisions might have to be addressed at distinct instants). In WHCA*, once an execution is found for a few agents, this can create a lot of room for the rest of the agents, and the search can find an execution quickly, or make progress towards the goal. Blocking situations are overcome by conflict resolution.

## 4.2 Experiments

We evaluate the WHCA* approach in comparison with the DFS algorithm from [16] on four benchmarks: the maps Offices and Cubicles are 2D grids; Obstacles and Pyramid are in 3D. The bitmap images corresponding to each map are shown in Fig. 12. The 3D maps were obtained from the bitmap image by copying it 5 times towards a third dimension, and adding obstacles at the free cells with a density of 15%. Similar benchmarks were considered in [6, 16]; Offices appears in [6]. For each $2 \leq n \leq 80$, we created 20 random instances with $n$ agents, on each map. The instances on the Cubicles were particular: we placed two agents at a position similar to that of Fig. 11 while others were assigned random positions.

In our implementation, at each trial, we compute the longest possible execution before changing the order of the agents. So trials do not necessarily change the ordering of the agents at predetermined window sizes, but only when the current ordering does not allow to make further progress. The program attempts to extend each trial 100 times with a random order; let us call these *extension trials*. This parameter was observed to work well on our benchmarks. A too small value meant that trials ended without making enough progress so solutions were found less often; and a too large value meant that too much time was spent in each trial so less time could be spent for remaining trials. We show the performance with 300 extension trials in Fig. 15 where a performance drop was observed. We allowed the solver 5 minutes per instance. The parameters used in conflict resolution was $\theta = 5$, $\ell = 10$. A small (arbitrary) value for $\theta$ was convenient here because WHCA* alone could not make any progress from the source configuration in Cubicles, a small value meant that this initial conflict could be resolved quickly. Moving towards a random direction for a small number of steps does not affect the feasibility of the instance; it can only slightly increase the total length of the computed plan.



(a) Offices 2D

(b) Cubicles 2D

(c) Pyramid 3D

(d) Obstacles 3D

**Figure 15: Success rates of WHCA$^*$ with 100, 200, and 300 extension trials with a 5 minute timeout per instance.**

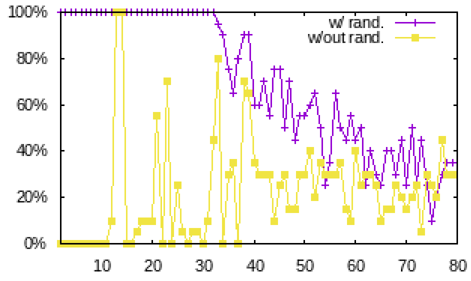


**Figure 16: Success rates of WHCA$^*$ with and without randomized conflict resolution (for 100 extension trials).**

The results are shown in Fig. 13 which gives the average success rates for the instances per number of agents. The success rate of WHCA$^*$ was systematically higher than that of DFS. The former scaled up to 80 agents on Pyramid and Obstacles maps, and up to about 30-40 on Offices and Cubicles. The performance of DFS dropped quickly, often after 10 agents. Figure 14 compares the execution times; the WHCA$^*$ solved a large number of instances; for instance, in Obstacles, the algorithm solved about 1200 instances (out of 1580) each under a minute, while the DFS algorithm could only solve about 400 instances each under a minute.

Figure 15 compares the success rates of the WHCA$^*$ algorithm with 100, 200, and 300 extension trials. A larger number means that the algorithm insists in trying to extend each trial. The performance in Offices improved as this number increased, presumably because a larger number of ordering changes is often required in this map, but that in Cubicles was better with a smaller number. A smaller number has the advantage of inducing more fresh trials within the given deadline. There was no substantial difference in performance in Pyramid and Obstacles maps.

In Fig. 16, we show the effect of randomized conflict resolution on performance, particularly in the Cubicles instances which contain at least two agents placed in a position depicted in Fig. 11. The success rate without randomization is 0 for small instances. For larger numbers of agents, this varies, in a rather unpredictable way but stays clearly lower than the success rate with randomization. This is due to randomly generated instances in which some agents can be placed near the problematic situation and render the instance solvable by WHCA$^*$. Overall, randomization helped increase and stabilize the success rate. It did not have a large impact on *performance* on other maps (not shown here) where instances were generated completely randomly. However, on several instances of Offices and Obstacles we noticed that situations similar to Figure 11 did occur, and randomization did help solve the instances; these were however not frequent enough to be noticed on average plots because instances were generated randomly.

## 5 CONCLUSION

In this work, we proved the PSPACE-hardness of the CMAPF problem when the topological graph is a subgraph of a 3D grid under range-based communication, establishing the computational complexity in a more realistic setting than in the literature. The hardness for subgraphs of the 2D grid is open. Our experiments contained both 2D and 3D maps, but the shape of the map seemed to have more impact on performance than its dimension. An important question is automatizing the choice of the parameters used in WHCA$^*$ (such as $\ell, \theta, \theta'$). We chose them empirically and specifically for our benchmarks, but other values might be preferable for maps of different shapes and sizes.

The connected coverage problem consists in finding an execution that visits all vertices and is similar to CMAPF [4]. In particular, it is also PSPACE-complete. We conjecture that our reduction could be adapted to prove the PSPACE-hardness of the coverage problem in 3D.

## ACKNOWLEDGMENTS

This work was supported by the ANR EpiRL project ANR-22-CE23-0029.

## REFERENCES

[1] F. Amigoni, J. Banfi, and N. Basilico. 2017. Multirobot Exploration of Communication-Restricted Environments: A Survey. *IEEE Intelli. Sys.* 32, 6 (2017), 48–57. https://doi.org/10.1109/MIS.2017.4531226

[2] Zahy Bnaya and Ariel Felner. 2014. Conflict-Oriented Windowed Hierarchical Cooperative A*. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 3743–3748. https://doi.org/10.1109/ICRA.2014.6907401

[3] Tristan Charrier, Arthur Queffelec, Ocan Sankur, and François Schwarzentruber. 2020. Complexity of planning for connected agents. *Auton. Agents Multi Agent Syst.* 34, 2 (2020), 44. https://doi.org/10.1007/s10458-020-09468-5

[4] Tristan Charrier, Arthur Queffelec, Ocan Sankur, and François Schwarzentruber. 2020. Complexity of planning for connected agents. *Auton. Agents Multi Agent Syst.* 34, 2 (2020), 44. https://doi.org/10.1007/s10458-020-09468-5

[5] Robert A. Hearn and Erik D. Demaine. 2005. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.* 343, 1-2 (2005), 72–96. https://doi.org/10.1016/j.tcs.2005.05.008

[6] Geoffrey A. Hollinger and Sanjiv Singh. 2012. Multirobot Coordination With Periodic Connectivity: Theory and Experiments. *IEEE Transactions on Robotics* 28, 4 (2012), 967–973. https://doi.org/10.1109/TRO.2012.2190178

[7] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W Durham, TK Satish Kumar, and Sven Koenig. 2021. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11272–11281.

[8] Hang Ma and Sven Koenig. 2017. AI Buzzwords Explained: Multi-Agent Path Finding (MAPF). *AI Matters* 3 (2017). https://doi.org/10.1145/3137574.3137579

[9] Bernhard Nebel, Thomas Bolander, Thorsten Engesser, and Robert Mattmüller. 2019. Implicitly Coordinated Multi-Agent Path Finding under Destination Uncertainty: Success Guarantees and Computational Complexity. *J. Artif. Intell. Res.* 64 (2019), 497–527. https://doi.org/10.1613/jair.1.11376

[10] Arthur Queffelec, Ocan Sankur, and François Schwarzentruber. 2020. Conflict-based search for connected multi-agent path finding. *arXiv preprint arXiv:2006.03280* (2020).

[11] Arthur Queffelec, Ocan Sankur, and François Schwarzentruber. 2021. Planning for Connected Agents in a Partially Known Environment. In *AI 2021 - 34th Canadian Conference on Artificial Intelligence*. Vancouver / Virtual, Canada, 1–23. https://hal.archives-ouvertes.fr/hal-03205744

[12] Walter Schnyder. 1990. Embedding Planar Graphs on the Grid. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms* (San Francisco, California), David S. Johnson (Ed.). SIAM, 138–148. http://dl.acm.org/citation.cfm?id=320176.320191

[13] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66.

[14] David Silver. 2005. Cooperative Pathfinding. In *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (Marina del Rey, California) *(AIIDE'05)*. AAAI Press, 117–122.

[15] Trevor Scott Standley and Richard Korf. 2011. Complete Algorithms for Cooperative Pathfinding Problems. In *Twenty-Second International Joint Conference on Artificial Intelligence.*

[16] Davide Tateo, Jacopo Banfi, Alessandro Riva, Francesco Amigoni, and Andrea Bonarini. 2018. Multiagent Connected Path Planning: PSPACE-Completeness and How to Deal With It. *Proceedings of the AAAI Conference on Artificial Intelligence* 32, 1 (Apr. 2018). https://doi.org/10.1609/aaai.v32i1.11587

[17] Glenn Wagner and Howie Choset. 2017. Path Planning for Multiple Agents under Uncertainty. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017*, Laura Barbulescu, Jeremy Frank, Mausam, and Stephen F. Smith (Eds.). AAAI Press, 577–585. https://aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15756

[18] Ko-Hsin Cindy Wang and Adi Botea. 2011. MAPP: A Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *J. Artif. Int. Res.* 42, 1 (sep 2011), 55–90.

[19] Ko-Hsin Cindy Wang, Adi Botea, et al. 2008. Fast and Memory-Efficient Multi-Agent Pathfinding.. In *ICAPS*. 380–387.

[20] Zheng Zhang, Qing Guo, Juan Chen, and Peijiang Yuan. 2018. Collision-Free Route Planning for Multiple AGVs in an Automated Warehouse Based on Collision Classification. *IEEE Access* 6 (2018), 26022–26035. https://doi.org/10.1109/ACCESS.2018.2819199