Multi-Agent Path Finding in Configurable Environments

Matteo Bellusci
Politecnico di Milano
Milano, Italy
matteo.bellusci@mail.polimi.it

Nicola Basilico
Università degli Studi di Milano
Milano, Italy
nicola.basilico@unimi.it

Francesco Amigoni
Politecnico di Milano
Milano, Italy
francesco.amigoni@polimi.it

ABSTRACT

Multi-Agent Path Finding (MAPF) plays an important role in many real-life applications where autonomous agents must coordinate to reach their goals without collisions. MAPF problems often take place in structured environments that are usually assumed to be static and known in advance. In this paper, we introduce C-MAPF, i.e., MAPF in Configurable environments, a novel variant of the MAPF problem in which the environment is configurable, namely its structure and topology can be controlled within some given constraints. Consider, for instance, a warehouse logistics application: the environment can be changed (at least to some degree) by the managers of the warehouse, for example by re-arranging the positions of the shelves or by removing or adding temporary walls. We study the properties of the C-MAPF problem and we devise two algorithms for solving it, both based on Conflict-Based Search (CBS), a state-of-the-art MAPF algorithm. First, we present Parallel CBS (P-CBS), that searches for a solution by simultaneously considering all the possible configurations of the environment. We then present Abstract CBS (A-CBS), an extended version of the CBS algorithm that solves C-MAPF problems by introducing a new type of conflict on the allowable configurations of the environment. We prove that our solvers are both complete and optimal and we experimentally assess their performance in different settings.

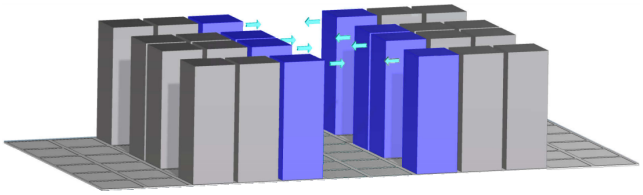
KEYWORDS

MAPF; multi-agent path finding; path planning; configurable environments; multi-robot systems

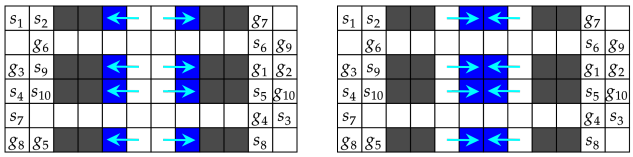
ACM Reference Format:
Matteo Bellusci, Nicola Basilico, and Francesco Amigoni. 2020. Multi-Agent Path Finding in Configurable Environments. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

Multi-Agent Path Finding (MAPF) is a challenging problem encountered in several applications that require coordinated motion of multiple agents, like warehouse logistics [47], office robots [45], quadrotor swarms [19], aircraft-towing vehicles [31], and digital entertainment [29]. In its basic formulation, the MAPF problem requires to find a set of non-interfering paths, one for each agent, such that agents can move from their start to their goal locations while minimizing some global cost. The MAPF problem has been widely studied [41], both in its abstract form, in which the environment is represented as a graph and full determinism is assumed [12],



(a) 3D representation of a configurable environment



(b) 2D representation of a corresponding C-MAPF problem

Figure 1: A warehouse where blue shelves can be moved to the center (a) and a corresponding C-MAPF problem (b), in which 10 agents have to move from s_i to g_i (with $i = 1, \ldots, 10$), with two possible configurations, G_1 (left) and G_2 (right).

and in more practical forms [28, 44], in which, for example, the kinematics [18, 23] and the physical shape [3, 27] of the robots are considered. To the best of our knowledge, almost all existing formulations of the MAPF problem assume that the environment is known and static. Situations that do not match this assumption, like the presence of dynamic obstacles, are usually reactively tackled during execution [13, 14].

In this paper, we introduce *MAPF in Configurable environments* (C-MAPF), a novel variant of the MAPF problem where the environment is configurable, namely its structure and topology can be controlled within some given constraints. The C-MAPF problem envisions settings where not only the agents, but also parts of the environment can be controlled. The potential of these settings has not yet been recognized and exploited by the available MAPF algorithms, which usually assume the environment as part of the problem and not of the solution. Consider, for instance, a warehouse logistics application in which the environment can be changed (at least to some degree) by the managers of the warehouse, for example by re-arranging the positions of the shelves or by removing or adding temporary walls. In some cases, the positions of the shelves can be dynamically controlled (e.g., mobile shelves [39, 47] controlled by other agents or remotely). This setting can be modeled as a C-MAPF problem in which the different arrangements of the environment are called *configurations* (an example is shown in Figure 1). Other application settings that can be modeled using C-MAPF include search and rescue (e.g., finding the best passages that can be opened by extinguishing fires [21]), city planning (e.g., selecting

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

an optimal configuration of one-way routes [9]), military operations (e.g., intelligence preparation of the battlefield [24]), digital entertainment (e.g., videogames design [20]), and hardware circuit design. We study the properties of the C-MAPF problem and we devise two algorithms for its resolution, based on Conflict-Based Search (CBS) [35], a state-of-the-art MAPF algorithm. Specifically, we introduce Parallel CBS (P-CBS), that searches for a solution by simultaneously considering all the possible configurations of the environment, and Abstract CBS (A-CBS), an extended version of the CBS algorithm that solves C-MAPF problems by introducing a new type of conflict on the allowable configurations of the environment. We prove that both our algorithms are complete and optimal and we experimentally evaluate them in different settings in order to assess their performance in terms of success rate and computing time required to get solutions for increasingly complex instances.

2 RELATED WORK

The basic formulation of the MAPF problem asks for finding non-interfering paths for multiple agents, each one with its start and goal locations in a given discretized environment represented as a graph [41]. The solution is often required to minimize some global cost function, such as the sum of the individual path costs (flowtime) or the maximum of the individual path costs (makespan).

Different algorithms are available for solving MAPF problems. Here, we just overview the main families, referring the reader to [22, 41] for further details. Solving algorithms can be complete, optimal, both, or neither. *Complete* solvers guarantee to find a solution if the problem is solvable. Note that complete solvers do not guarantee to find the optimal solution (according to the global cost function), although sometimes they can guarantee to return a solution whose cost is at a bounded distance from that of an optimal solution. *Optimal* solvers are usually slower, but they guarantee to find only optimal solutions. Switching to the techniques used by solving algorithms, search-based and reduction-based families can be identified. *Search-based* solvers search the solution in a given search space and are typically designed to deal with a flowtime global cost function [15, 16, 35, 37, 40, 46]. *Reduction-based* solvers translate the problem to an equivalent problem in another domain for which efficient algorithms are available and are typically designed to deal with a makespan global cost function [10, 43, 48].

Since it is relevant for presenting our contribution, we describe in some more detail a prominent complete and optimal search-based solving algorithm called *Conflict-Based Search* (CBS) [35]. It runs searches at two levels. The low-level searches for an optimal path for each agent individually, typically using an A*-based procedure [16], taking into account constraints imposed by the high-level. When a conflict between individual paths is found, the high-level expands a constraint tree via a split action, keeping all information about the conflict, and imposes new constraints to the agents in order to avoid the conflict the next time the low-level search for those agents is performed. A constraint prohibits an agent from being at a certain position at a given time. The algorithm is exponential in the number of conflicts since it tries to solve the detected conflicts in all the possible ways. Several improvements of CBS are available [6, 11, 25], as well as sub-optimal variants [5].

In the classical formulation of the MAPF problem, the graph representing the environment is usually given and fixed. To the best of our knowledge, the resolution of MAPF problems in environments that can be purposefully changed, that we call *configurable*, has not yet been investigated. Nevertheless, there are some results concerning MAPF in non-static environments. MAPF with Dynamic Obstacles (MAPF-DO) [30, 32] considers the presence of dynamic obstacles in the map, overcoming the assumption of static environments of classical MAPF. However, in MAPF-DO problems, dynamic obstacles follow given, potentially random, movement rules and there is no possibility to control them, nor to modify the environment.

One attempt to overcome the assumption of unchangeable environments in single-agent path planning comes from the computational geometry community, in which the problem of computing the best path for a single agent in environments with removable polygonal obstacles has been recently addressed [1]. We borrow a similar idea, but move from a single-agent to a multi-agent setting. A recent attempt to deal with MAPF and removable obstacles uses a Theta*-based [33] algorithm [2]. As discussed by the authors, their algorithm cannot be directly applied to MAPF but it can be modified to become base block of MAPF solves such as CBS. The approach of [2] does not compute optimal solutions. Our goal is to go beyond and provide CBS-based resolution methods for optimally solving C-MAPF problems, which, differently from [1, 2], involve a more general class of configurable environments.

We finally note that the MAPF problem with use of highways [7] and the MAPF problem with optimal task assignment [17] exhibit some configurable aspects. In the former, sub-optimal MAPF solvers push agents to follow some human-generated highways, that can be configured in various ways, in the attempt of avoiding collisions. In the latter, the agents' goal locations can be configured (keeping the environment unchanged), while in the C-MAPF problem it is possible to configure the environment while keeping the start and goal locations of the agents unchanged.

3 C-MAPF

In this section, we introduce and analyze a general formulation of the C-MAPF problem built as a natural extension of the classical MAPF problem. In the next section, we devise an alternative (equivalent) formulation that, although less straightforward, will help to develop one of the resolution algorithms we propose.

3.1 Problem Formulation

The MAPF problem in Configurable environments, *C-MAPF problem*, assumes a non-empty family $\mathcal{G} = \{G_1, \ldots, G_n\}$ of graphs $G_j = (V_j, E_j)$ that represent all the possible *configurations* of the environment and a set of k labeled agents $A = \{a_1, \ldots, a_k\}$. We assume that a given physical location is associated to a single vertex. Hence, if the location corresponding to vertex v is present in two configurations, $G = (V, E)$ and $G' = (V', E')$, then $v \in V$ and $v \in V'$. Similarly for physical connections between locations and the edges that represent them. Define $V^{\cap} = \bigcap_{j=1,\ldots,n} V_j$, $E^{\cap} = \bigcap_{j=1,\ldots,n} E_j$, $V^{\cup} = \bigcup_{j=1,\ldots,n} V_j$, and $E^{\cup} = \bigcup_{j=1,\ldots,n} E_j$. Each agent $a_i \in A$ has an associated unique start vertex $s_i \in V^{\cap}$ and goal vertex $g_i \in V^{\cap}$. Hence, we assume that start and goal vertices s_i and g_i are present

in every configuration. Two agents can never occupy the same vertex at the same time. As a consequence, meaningful instances of the problem satisfy $|V^{\cap}| \geq k$. From now on we shall assume that this always holds. At each (discrete) time step t, each agent a_i moves between two vertices or stays at its current vertex. A *path* π_i of cost $\overline{\pi_i} = T_i$ for an agent a_i is a sequence of vertices $\pi_i = (v_i^0, \ldots, v_i^{T_i}, v_i^{T_i+1}, \ldots)$ such that $v_i^0 = s_i$ and $v_i^t = g_i, \forall t \geq T_i$ (i.e., the path π_i, when followed by a_i, brings the agent from s_i to g_i). That is, T_i represents the earliest time at which agent a_i reaches its goal vertex g_i and stops moving. A path π_i is *applicable* in a graph $G = (V, E)$ if, for any t, $v_i^t \in V$ and $v_i^t \neq v_i^{t+1}$ implies that $(v_i^t, v_i^{t+1}) \in E$ (i.e., a path π_i is applicable in a graph G if, intuitively, it possible to simulate π_i in G). For the moment, we require paths to be applicable in $G^{\cup} = (V^{\cup}, E^{\cup})$. A set of k paths $\pi = \{\pi_1, \ldots, \pi_k\}$, one for each agent, is called a *plan*. Namely, a plan is composed of paths that *could be* applicable in some configuration of the environment (a graph in $\mathcal{G}$). A *conflict* between paths π_i and π_j is a tuple $\langle a_i, a_j, v, t \rangle$ meaning that agents a_i and a_j are occupying the same vertex $v \in V^{\cup}$ at time step t. A *solution* $\langle \pi, G \rangle$ for the C-MAPF problem is a conflict-free plan π and a configuration $G \in \mathcal{G}$ such that all the paths in π are applicable in G. When $\mathcal{G}$ is composed of a single configuration, the C-MAPF problem trivially reduces to the MAPF problem.

If a solution $\langle \pi, G \rangle$ minimizes some given global cost function, then it is an *optimal* solution. Most common global cost functions for MAPF problems are the *flowtime*, namely the sum of individual path costs (SIC), defined as $\sum_{\pi_i \in \pi} \overline{\pi_i}$, and the *makespan*, namely the cost of the most expensive path, defined as $\max_{\pi_i \in \pi} \overline{\pi_i}$. In this paper, we focus on solving the C-MAPF problem optimally and we consider the flowtime as the global cost function.

Example 3.1. Consider $k = 10$ agents that have to move from s_i to g_i (with $i = 1, \ldots, 10$) in a warehouse. The environment can be set in two different configurations, G_1 and G_2, reported in Figure 1. Free areas are white cells, while obstacles are colored cells. Although not strictly necessary, the two configurations have the same number (24) of obstacle cells arranged in different patterns and, in accordance with the definition above, start and goal vertices of the agents are the same in both configurations. A C-MAPF instance can represent this setting and an optimal solution returns a set of conflict-free paths for the agents that minimizes the flowtime and the corresponding configuration. Finally, note that one may discover (e.g., by generating and solving various C-MAPF instances) that, on average, one configuration is better than the other.

3.2 Problem Analysis

The following analysis is based on the C-MAPF problem in its general form where edge conflicts[1] can be present too.

The definition of the C-MAPF problem poses almost no restriction on how the configurations in $\mathcal{G}$ are defined. A way to simplify the process of finding an optimal solution to such problem is to identify and eliminate the inefficiencies caused by redundancies among configurations and by areas that are unreachable in some configurations. For instance, consider a C-MAPF problem and two configurations $G, G' \in \mathcal{G}$. If G is a subgraph[2] of G', then, when solving the C-MAPF problem, we can discard G from the set of configurations $\mathcal{G}$ since all the plans that are applicable in G are also applicable in G'.

It is interesting to study the possibility of using solvers for the MAPF problem in order to optimally solve the C-MAPF problem.

Proposition 3.2. *The C-MAPF problem can be optimally solved for flowtime and makespan minimization by exploiting any MAPF solver.*

Proof. The C-MAPF problem can be optimally solved in a brute-force manner by invoking any MAPF solver to solve the MAPF problems corresponding to all configurations $G \in \mathcal{G}$. Note that it is possible to preliminary check whether each of these MAPF problems is solvable [50] if the adopted MAPF solver is unable to identify unsolvable MAPF instances. Intuitively, given the MAPF problems corresponding to all configurations, if there is at least one solvable problem and all solvable problems are solved by any optimal MAPF solver, the C-MAPF problem can be optimally solved simply by selecting the best MAPF solution and the corresponding configuration. If, for each configuration, there is no solution to the corresponding MAPF problem, then the C-MAPF problem admits no solution. □

We now turn to analyze the complexity of finding a solution and of finding an optimal solution for the C-MAPF problem.

Proposition 3.3. *Solvability of the C-MAPF problem can be determined in polynomial time.*

Proof. For a MAPF problem on a graph $G = (V, E)$, solvability can be determined in linear time [50]. We may need to check the solvability of n MAPF problems, one per each configuration $G \in \mathcal{G}$. The entire process has polynomial time worst-case complexity. □

Proposition 3.4. *A (non necessarily optimal) solution for the C-MAPF problem can be found in polynomial time.*

Proof. A MAPF problem on a graph $G = (V, E)$ can be solved (not optimally) in polynomial time [50]. In order to (not optimally) solve the C-MAPF problem, it is sufficient to solve one MAPF problem among those that can be defined starting from the set of configurations $\mathcal{G}$. The entire process, which include the solvability test for the MAPF problems since they are not necessarily solvable, has polynomial time worst-case complexity. □

Theorem 3.5. *The C-MAPF problem is NP-hard to solve optimally for flowtime and makespan minimization, even on grid environments.*

Proof. The C-MAPF problem is trivially a generalization of the MAPF problem, which is NP-hard to solve optimally for flowtime and makespan minimization [49], even on grid environments [4]. □

4 ALGORITHMS

According to the previous section, the C-MAPF problem can be optimally solved in a brute-force manner by invoking the CBS algorithm [35] (or any other optimal MAPF solver) for each MAPF problem that can be defined starting from $\mathcal{G}$. Clearly, this approach is impractical for complex instances. We propose two more efficient algorithms, both based on the CBS algorithm, for solving

[1] An *edge conflict* between paths π_i and π_j is a tuple $\langle a_i, a_j, v, v', t \rangle$ meaning that from time step t to $t+1$ agent a_i is traveling from $v \in V^{\cup}$ to $v' \in V^{\cup}$ while agent a_j is traveling from v' to v, i.e., they are traveling along an edge in opposite directions.

[2] A graph $G = (V, E)$ is a *subgraph* of $G' = (V', E')$ when $V \subseteq V'$ and $E \subseteq E'$.

the C-MAPF problem. We choose CBS because it is a search-based algorithm and it matches well with the combinatorial complexity introduced by the selection of the configuration of the environment.

We quickly overview some aspects (expanding those mentioned in Section 2) of CBS that are useful to understand our algorithms. Please refer to the original paper [35] for full details. CBS imposes constraints to agents. A *constraint* for an agent a_i is a triplet $\langle a_i, v, t\rangle$ meaning that a_i cannot occupy vertex v at time step t. If a path π_i for agent a_i satisfies all the constraints for a_i, then π_i is said to be a *consistent* path. A consistent plan is a plan composed only of consistent paths, one for each agent. The CBS algorithm operates at two levels. The high-level performs a best-first search on a binary *constraint tree* (CT). Each CT node N contains a set of constraints ($N.constraints$), a plan consistent with its constraints ($N.plan$, calculated by the low-level algorithm), and its associated flowtime cost ($N.cost$). The root R of the CT has $R.constraints = \emptyset$. When a node N is selected, a validation process searches for any conflict in $N.plan$. If $N.plan$ is conflict-free, then it is a solution for the MAPF problem and the node N is a goal node (the search terminates at that point). If a conflict $\langle a_i, a_j, v, t\rangle$ is found, then a split action is performed, generating two child nodes, N_1 and N_2. The constraint $\langle a_i, v, t\rangle$ is added to $N_1.constraints$ and the constraint $\langle a_j, v, t\rangle$ is added to $N_2.constraints$. For the two child nodes, a new plan is then computed invoking the low-level algorithm, which replans the path only for the agent involved in the added constraint. Given an agent a_i and its constraints, the low-level plans a path π_i on the graph representing the environment ignoring the presence of the other agents and taking into account only the constraints of a_i. Generally, the low-level employs A*-based procedures [16]; however, is possible to use any single-agent path finding algorithm to find paths. Finally, N_1 and N_2 are added to an OPEN list. CBS performs a best-first search in OPEN based on the cost of the nodes ($N.cost$).

Improved CBS (ICBS) [6] improves CBS by resolving first the *cardinal* conflicts in $N.plan$, that are the ones that lead to the generation of two child nodes with a cost higher than $N.cost$. CBSH [11] improves it further by adding an admissible heuristic to guide the high-level search. The idea behind the heuristic is that if a CT node N has a cardinal conflict, then all goal nodes descending from N have a cost of at least $N.cost+1$. Consequently, if N has $N.h$ disjoint (i.e., between disjoint pairs of agents) cardinal conflicts, $N.h$ is an admissible heuristic for N. Both our algorithms are based on CBSH and perform a best-first search considering $N.cost + N.h$.

4.1 P-CBS

Our first algorithm is called *Parallel CBS* (P-CBS) and is a simple optimized version of the brute-force approach mentioned above. While the brute-force approach sequentially solves all the MAPF instances that can be generated from a C-MAPF instance (one for each configuration in $\mathcal{G}$), P-CBS carries out a best-first search similar to the high-level of CBS, but simultaneously considering all the MAPF instances. The idea is to perform a forest search, similarly to what done for solving the MAPF problem with optimal task assignment [17].

Specifically, in P-CBS, each CT node N has a new field called $N.configuration$ that stores a configuration $G \in \mathcal{G}$, which remains the same for its child nodes. The low-level algorithm, when invoked by the high-level for updating $N.plan$, is bound to look for paths that are applicable in $N.configuration$. At the beginning of the search, P-CBS generates a CT node N_j, equivalent to the root node of plain CBS, for each configuration $G_j \in \mathcal{G}$, and sets $N_j.configuration = G_j$. It is possible to speed up the generation of these initial nodes by reusing individual agent paths from previously generated nodes whenever possible (e.g., when they are still optimal and applicable in the new configuration). These n nodes are all added to OPEN and the search continues as in CBS.

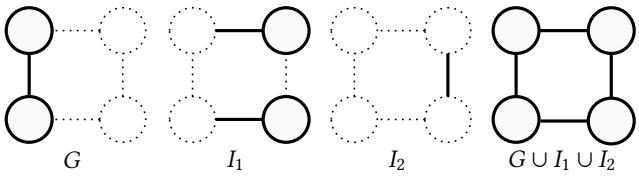


Figure 2: A graph G and two graph improvements for it, I_1 and I_2. Adding both I_1 and I_2 results in the graph on the right.

In practice, P-CBS searches in parallel over multiple "virtual" trees (one per configuration) and, as soon as a goal node is found by the best-first search strategy, the search is stopped for all the trees. The approach of P-CBS is straightforward and has some limitations. For instance, consider a conflict whose presence is independent of the configuration of the environment. It may be necessary to solve this conflict in several "virtual" trees. In the next section, we present an algorithm which, by using an aggregation technique, is able to partially address this limitation.

We now show a result about completeness and optimality.

Theorem 4.1. *P-CBS terminates if a solution exists (completeness). If P-CBS terminates, it returns the optimal solution according to the flowtime (optimality).*

Proof is trivially derived from the fact that P-CBS performs (at worst) an exhaustive search for plans over all configurations.

4.2 A-CBS

In order to introduce our second algorithm, called *Abstract CBS* (A-CBS), we present an alternative and more operative formulation for the C-MAPF problem defined in Section 3.

Operative formulation of the C-MAPF problem. The idea of the operative formulation of the C-MAPF problem is to start from a graph $G = (V, E)$, representing a primitive skeleton, to which structures of vertices and edges, called graph improvements, can be added.

Definition 4.2. A *graph improvement* $I = (V', E')$ for a graph $G = (V, E)$ is a structure composed of a set of vertices V' and a set of edges E' (it is not required that $E' \subseteq (V \cup V') \times (V \cup V')$) such that $V \cap V' = \emptyset$ and $E \cap E' = \emptyset$.

In general, adding a graph improvement $I = (V', E')$ to a graph $G = (V, E)$ does not result in a well-formed graph $G' = G \cup I = (V \cup V', E \cup E')$ (e.g., some edges can be dangling). In the operative formulation, a configuration of the original formulation of Section 3 can be obtained as an appropriate combination of some graph

improvements. An example is shown in Figure 2 in which the left graph G and the right graph $G \cup I_1 \cup I_2$ represent two configurations. The operative formulation of the C-MAPF problem is obtained as follows. Consider a graph $G = (V, E)$, a set of disjoint[3] graph improvements $\mathcal{I} = \{I_1, \ldots, I_m\}$ for G, a validation function $f : \mathcal{P}(\mathcal{I}) \rightarrow \{0, 1\}$, and a set of k labeled agents $A = \{a_1, \ldots, a_k\}$. Each agent $a_i \in A$ has an associated unique start vertex $s_i \in V$ and goal vertex $g_i \in V$. Thus, we assume that $|V| \geq k$. $f()$ tells, by returning 1, which combinations of graph improvements produce configurations in $\mathcal{G}$ and is defined as ($\mathcal{I}' \subseteq \mathcal{I}$):

$$f(\mathcal{I}') = \begin{cases} 1 & \text{if } G \cup (\bigcup_{I \in \mathcal{I}'} I) \in \mathcal{G} \\ 0 & \text{otherwise} \end{cases}.$$

We require that there exist at least a set $\mathcal{I}' \subseteq \mathcal{I}$ such that $f(\mathcal{I}') = 1$.

The above operative formulation resembles that for the (single-agent) path finding problem with removable obstacles in the plane of [1], in which some disjoint polygonal-shaped obstacles can be removed, paying a cost, from an initial environment. In the formulation of [1], a (given) limited budget can be spent for removing obstacles. In our case, instead of the budget, which implicitly defines the combinations of obstacles that can be removed, we have a more general validation function $f()$ with a similar purpose. Moreover, while the approach of [1] operates by subtracting polygonal obstacles, we operate by adding parts of the graph.

The operative formulation of the C-MAPF problem does not require to explicitly list all the configurations of the environment $\mathcal{G}$, but to list the graph improvements that can be used to obtain such configurations.

Proposition 4.3. *It is possible to translate a C-MAPF problem formulated as in Section 3 into the operative formulation and vice versa.*

Proof. We immediately note that we can easily obtain the original C-MAPF formulation of Section 3 from the operative formulation since we can always define $\mathcal{G}$ by inspecting $f()$. We now prove that we can obtain the operative formulation from the original C-MAPF formulation. Define $\Delta\mathcal{G} = \{\Delta G_1, \ldots, \Delta G_n\}$ and $\neg\Delta\mathcal{G} = \{\neg\Delta G_1, \ldots, \neg\Delta G_n\}$ (n is the number of configurations in $\mathcal{G}$), where[4] $\Delta G_j = G_j - G^\cap$ and $\neg\Delta G_j = G^\cup - G_j$ (where $G^\cap = (V^\cap, E^\cap)$ and $G^\cup = (V^\cup, E^\cup)$). $\mathcal{I}$ is defined starting from $\Delta\mathcal{G}$ and $\neg\Delta\mathcal{G}$. For each $\{i, \ldots, j\} \subset \{1, \ldots, n\}$, we define a graph improvement $I_{i,\ldots,j}$ for $G^\cap$ as:

$$I_{i,\ldots,j} = \left(\bigcap_{k \in \{i,\ldots,j\}} \Delta G_k \right) \cap \left(\bigcap_{k \in \{1,\ldots,n\} - \{i,\ldots,j\}} \neg\Delta G_k \right).$$

At this point, we discard all the empty graph improvements. The set of graph improvements $\mathcal{I}$ obtained as above is composed of disjoint graph improvements for $G^\cap$. In fact, given two different graph improvements I_x and I_y, $x \neq y$ implies that there exist at least one index $h \in \{1, \ldots, n\}$ such that $h \in x \wedge h \notin y$ or $h \notin x \wedge h \in y$ and then[5] $I_x \subseteq \Delta G_h \wedge I_y \subseteq \neg\Delta G_h$ or $I_y \subseteq \Delta G_h \wedge I_x \subseteq \neg\Delta G_h$: it follows that $I_x \cap I_y = (\bigcap_{k \in x} \Delta G_k) \cap (\bigcap_{k \in \{1,\ldots,n\} - x} \neg\Delta G_k) \cap (\bigcap_{k \in y} \Delta G_k) \cap (\bigcap_{k \in \{1,\ldots,n\} - y} \neg\Delta G_k) = \Delta G_h \cap \ldots \cap \neg\Delta G_h = \emptyset$, since $\Delta G_h \cap \neg\Delta G_h = \emptyset$. Finally, we define $G = G^\cap$ and $f()$ such that, given $\mathcal{I}' \subseteq \mathcal{I}$, it returns 1 iff $\bigcup_{I \in \mathcal{I}'} I \in \Delta\mathcal{G}$. □

Observation 4.4. *Given n configurations in $\mathcal{G}$, by using the translation method shown in the proof of Proposition 4.3, we can obtain at most $m = 2^n - 2$ (disjoint) graph improvements in $\mathcal{I}$.*

Observation 4.5. *Given m disjoint graph improvements in $\mathcal{I}$, we can obtain at most $n = \binom{m}{\lfloor m/2 \rfloor}$ (i.e., $n \approx 2^m/\sqrt{m}$) configurations $\mathcal{G}$ such that there are no configurations $G, G' \in \mathcal{G}$ in which G is a subgraph of G'.*

Proof is trivially derived from the Sperner's theorem [38].

Algorithm behavior. A-CBS extends CBS in considering not only constraints for the agents, but also constraints for the environment. The idea behind A-CBS is that, unless otherwise imposed by environment constraints, the low-level algorithm considers *all* the graph improvements as added to G (where G is equal to $G^\cap$). At high-level, each CT node N has an additional field called $N.improvements$, which contains all the graph improvements used by $N.plan$, i.e., given the improvement $I \in \mathcal{I}$, $I \in N.improvements$ iff $N.plan$ is not applicable in the largest graph embedded in $G^\cup - I$. Note that computing $N.improvements$ from $N.plan$ is trivial. The root R of the CT has $R.constraints = \emptyset$ (as in CBS). Consequently, the plan of the root node is calculated considering all the graph improvements added to G, ignoring (for the moment) the fact that this abstract search space (which is initially equivalent to $G^\cup$) may be illegal.

When a node N is selected from OPEN, a new validation process is immediately carried out, before checking for conflicts in $N.plan$. A-CBS first checks if there exists a configuration in which $N.plan$ is applicable. This is done by exploiting the operative formulation and the following function.

Definition 4.6. The *extended validation function* $F : \mathcal{P}(\mathcal{I}) \rightarrow \{0, 1\}$ determines if there is a configuration that contains given graph improvements and is defined as ($\mathcal{I}' \subseteq \mathcal{I}$):

$$F(\mathcal{I}') = \begin{cases} 1 & \exists \mathcal{I}'' \subseteq \mathcal{I} - \mathcal{I}' \text{ s.t. } f(\mathcal{I}' \cup \mathcal{I}'') = 1 \\ 0 & \text{otherwise} \end{cases}.$$

$F()$ is similar to the validation function $f()$ but less restrictive (note that $f(\mathcal{I}') = 1 \Rightarrow F(\mathcal{I}') = 1$) and can be computed from $f()$. If a configuration in which the plan is applicable exists, A-CBS proceeds following the behavior of CBS, otherwise, A-CBS imposes environment constraints, limiting the search space. The idea is to perform a disjoint splitting, as in [26], but applied to the environment. We define a new type of conflict and two new types of constraints.

Definition 4.7. An *environment conflict* is a tuple $\langle I, \ldots, I' \rangle$ meaning that the plan uses all the graph improvements in $\{I, \ldots, I'\} \subseteq \mathcal{I}$ but $F(\{I, \ldots, I'\}) = 0$.

Definition 4.8. There are two types of *environment constraints*. The constraint $\langle I \rangle$ imposes the presence of all vertices and edges in $I \in \mathcal{I}$ within the search space, while the constraint $\overline{\langle I \rangle}$ imposes the removal (absence) of all vertices and edges in $I \in \mathcal{I}$ from the search space.

[3] Two graph improvements $I = (V, E)$ and $I' = (V', E')$ are disjoint if $I \cap I' = (V \cap V', E \cap E')$ is empty (i.e., $V \cap V' = \emptyset$ and $E \cap E' = \emptyset$).

[4] Given two graphs $G = (V, E)$ and $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$, $G - G' = (V - V', E - E')$. Note that $G - G'$ is a graph improvement for G' but it is not necessarily a graph.

[5] Given two graph improvements $I = (V, E)$ and $I' = (V', E')$, $I \subseteq I'$ if $V \subseteq V'$ and $E \subseteq E'$.

Algorithm 1: A-CBS algorithm (simplified version).

```
1  Main(C-MAPF instance)
2    R.constraints ← ∅;
3    R.plan ← find individual paths using the low-level;
4    R.cost ← flowtime(R.plan);
5    R.improvements ← graph improvements in R.plan;
6    insert R in OPEN;
7    while OPEN not empty do
8      P ← best node from OPEN;
9      if F(P.improvements) is 0 then
10       I ← select-one(P.improvements);
11       Generate Child(P, ⟨I⟩) //left child;
12       Generate Child(P, ⟨Ī⟩) //right child;
13       continue
14     validate the paths in P;
15     if P has no conflict then
16       return P //P is goal node;
17     C ← first conflict ⟨a_i, a_j, v, t⟩ in P;
18     Generate Child(P, ⟨a_i, v, t⟩) //left child;
19     Generate Child(P, ⟨a_j, v, t⟩) //right child;
20 Generate Child(Node N, Constraint(s) C)
21   A.constraints ← N.constraints ∪ C;
22   A.plan ← N.plan;
23   foreach agent a_i involved in C do
24     update A.plan by invoking the low-level for a_i;
25   A.cost ← flowtime(A.plan);
26   A.improvements ← graph improvements in A.plan;
27   insert A in OPEN;
```

Given a CT node N, if an environment conflict $\langle I, \ldots, I' \rangle$ is found, then a split action is performed, generating two child nodes, N_1 and N_2. A-CBS selects a graph improvement $I_j \in \{I, \ldots, I'\}$ (such that $\langle I_j \rangle$ is not in $N.constraints$) and adds the constraint $\langle I_j \rangle$ to $N_1.constraints$ and the constraint $\overline{\langle I_j \rangle}$ to $N_2.constraints$. We choose the one that is most used by the agents in $N.plan$, but other policies are possible. A node N is consistent with its environment constraints if $N.plan$ is applicable in the largest graph embedded in the (largest) search space consistent with $N.constraints$. As a consequence, when adding an environment constraint of type $\overline{\langle I \rangle}$ (with $I \in \mathcal{I}$), we must re-compute the individual paths of all the agents that use the graph improvement I in order to find a consistent plan. A node N is pruned if the environment can not satisfy the environment constraints in $N.constraints$[6].

Example 4.9. Suppose that the plan in the root node of the CT has an environment conflict $\langle I_1, I_2, I_3 \rangle$ and that the split action generates N_1 with the constraint $\langle I_1 \rangle$ and N_2 with the constraint $\overline{\langle I_1 \rangle}$. Moreover, suppose that $N_1.cost = N_2.cost$ and $N_2.improvements = \{I_2, I_4\}$. At this point N_2 could be selected from OPEN and $\pi = N_2.plan$ could be conflict-free. If $f(\{I_1, I_2, I_4\}) = 1$, then $F(\{I_2, I_4\}) = 1$ (because of I_1). It does not matter if $\overline{\langle I_1 \rangle} \in N_2.constraints$ because the constraint $\overline{\langle I_1 \rangle}$ imposes the absence of the vertices and edges in I_1 from the search space and not from the configuration that is part of the solution: the best-first search ensures that $N_2.cost$ is the optimal cost. In fact, it is trivial to prove that, in this case, the sub-tree below N_1 will necessarily contain a goal node with the same cost of π. Therefore, π can be selected as part of the optimal solution $(\langle \pi, G \cup (\bigcup_{j \in \{1,2,4\}} I_j) \rangle)$.

The pseudocode (salient steps) of the A-CBS algorithm is reported in Algorithm 1. The main changes wrt CBS are highlighted.

In practically implementing A-CBS, multiple environment constraints can be considered at each step. For instance, consider a case in which there is no configuration in $\mathcal{G}$ that contains the vertices and the edges of both $I \in \mathcal{I}$ and $I' \in \mathcal{I}$. In this case, the addition of the environment constraint $\langle I \rangle$ to $N_1.constraints$ can be done jointly with the addition of the further environment constraint $\overline{\langle I' \rangle}$. In other words, given a CT node N and a graph improvement $I \in \mathcal{I}$ (such that $\langle I \rangle$ and $\overline{\langle I \rangle}$ are not in $N.constraints$), if the addition of $\langle I \rangle$ (or $\overline{\langle I \rangle}$) to $N.constraints$ leads in pruning N, then $\overline{\langle I \rangle}$ (or $\langle I \rangle$) can be added to $N.constraints$.

A-CBS maintains the nodes that reported an environment conflict on top of the CT. Let $EC(N)$ be the higher ancestor node of N in which a (non-environment) conflict $\langle a_i, a_j, v, t \rangle$ has been detected. When considering the CT node N, if an environment conflict $\langle I, \ldots, I' \rangle$ is detected, then the sub-tree below $EC(N)$ is discarded and a new split action at $EC(N)$ is performed assuming the presence of the conflict $\langle I, \ldots, I' \rangle$ and ignoring the conflict $\langle a_i, a_j, v, t \rangle$. Note that the algorithm could be further optimized in order to somehow reuse the sub-tree below $EC(N)$, for example by possibly hanging it on the new left child node of $EC(N)$.

These adjustments drastically reduce the number of environment conflicts encountered during the high-level search. In particular, the following proposition can be trivially proved.

Proposition 4.10. *The number n of configurations in $\mathcal{G}$ represents a strict upper bound on the number of environment conflicts encountered during the high-level search of A-CBS.*

For example, adding (possibly multiple) environment constraints to the child nodes of the root node R means, in a sense, dividing the set of configurations into two subsets associated with the child nodes of R. The above result is reassuring in cases in which the C-MAPF instance presents many more graph improvements than configurations (i.e., when $m >> n$).

We now prove completeness and optimality of A-CBS.

Theorem 4.11. *A-CBS terminates if a solution exists (completeness). If A-CBS terminates, it returns the optimal solution according to flowtime (optimality).*

Proof. The completeness proof of CBS shown in [36] is still valid for A-CBS for the same reasons (consider also Proposition 3.3). The optimality proof for A-CBS is an extension of that for CBS shown in [35], which is presented as a consequence of two lemmas (please refer to [35] for details). Hence, we demonstrate that these two lemmas hold also for A-CBS. Lemma 1 states that the cost of a node N in the CT is a lower bound on $\min_{\langle \pi, G' \rangle \in CV(N)} flowtime(\pi)$,

[6] That is, if there is no configuration that includes all the vertices and edges in I, $\forall \langle I \rangle \in N.constraints$, and that not includes any of the vertices and edges in I', $\forall \overline{\langle I' \rangle} \in N.constraints$.

where $CV(N)$ is the set of all the (non necessarily optimal) solutions consistent with N. The proof in [35] can be extended by considering that π must be applicable in the (same) largest graph embedded in the (largest) search space consistent with $N.constraints$, $\forall\langle\pi, G'\rangle \in CV(N)$. In our case, Lemma 2 states that, for any solution $\langle\pi, G'\rangle$, at all time steps, there exists a CT node N in OPEN s.t. $\langle\pi, G'\rangle \in CV(N)$. Proof by induction provided in [35] is still valid even considering environment constraints. $CV(R)$ (where R is the root node) contains all the solutions $\langle\pi, G'\rangle$ since the search space is the widest ($G^{\cup}$). Secondly, given a solution $\langle\pi, G'\rangle$, if $\langle\pi, G'\rangle \in CV(N)$ at a given time step for a certain CT node N, and a split action (due to an environment conflict) is performed generating N_1 and N_2, then $\langle\pi, G'\rangle$ must satisfy at least one of the new environment constraints, i.e., $\langle\pi, G'\rangle \in CV(N_1)$ or $\langle\pi, G'\rangle \in CV(N_2)$. □

Low-level search. Given a CT node N and an agent $a_i \in A$, the low-level algorithm computes an optimal path π_i that is consistent with $N.constraints$, including environment constraints. Furthermore, when possible, the low-level prefers paths that use fewer graph improvements without counting those involved in environment constraints (i.e., all $I \in \bar{I}$ s.t. $\langle I\rangle \in N.constraints$). In CBS, a common choice for the low-level algorithm is A* [16] with a perfect heuristic. This choice may be impractical for C-MAPF since it is necessary to pre-compute a perfect heuristic for all the configurations of the environment. Instead, we use a simpler consistent heuristic, such as the Manhattan distance for 4-connected grid environments or the Euclidean distance for Euclidean graphs [34]. We also use EPEA* [15], a fast solver for single agents when the heuristic is less informative. Obviously, there are various alternatives, as M* [46].

5 EXPERIMENTS

All the experiments are conducted on a computer equipped with an AMD Ryzen™ 5 1500X quad-core processor at 3.79 GHz and with 16 GB of RAM. We implemented our two algorithms in Java. We also implemented an *ideal* solver (labeled IDEAL) that knows in advance, from an oracle, the configuration $G \in \mathcal{G}$ of an optimal solution $\langle\pi, G\rangle$ and has only to solve a MAPF problem on the graph G by invoking CBS. The IDEAL algorithm is of course not realistic, but represents a lower bound for the computing time needed to solve a C-MAPF problem. To the best of our knowledge, in the literature, there is no other MAPF algorithm that can be used in configurable environments against which our algorithms can compare. As mentioned above, our algorithms use the conflict prioritization and the high-level heuristic values of CBSH [11].

We know, from theoretical analysis (Proposition 4.10), that A-CBS is able to efficiently limit the size of the CT when $m >> n$ (where m is the number of graph improvements and n is the number of configurations) and that, by definition, P-CBS generates n "virtual" trees (regardless of the value of m). Therefore, in our experiments, we mainly consider instances defined starting from a set of graph improvements (where typically $n > m$). We run experiments in two classes of grid environments[7], which are discussed in the following sections. In order to have a controllable way for generating configurations of the environments, we adopt the following approach. Some obstacle cells can be removed by selecting a given percentage of them or, alternatively, by paying obstacle-removal costs within a limited total budget. The configurations $\mathcal{G}$ of the environment are thus determined by all the possible combinations of removed obstacles.

[7] Given a 2D grid composed of cells, which can be either free or occupied by obstacles, the corresponding graph is obtained by associating vertices to the centers of free cells and edges to pairs of adjacent free cells (assuming 4-connection).

m (1 × 1 obstacles)	5	10	15	20	25	30
n (configurations)	5	120	1365	38760	480700	14307150

Table 1: Number of configurations wrt to m for $\beta = 30\%$.

				Runtime (ms)			Expanded CT nodes		
			#	P-CBS	A-CBS	IDEAL	P-CBS	A-CBS	IDEAL
15 × 15 grid maps	k	10	50	34	5	1	310	34	7
		15	49	62	10	1	726	107	13
		20	49	731	194	18	3030	828	87
		25	47	846	81	36	3798	396	220
		30	44	2405	930	272	9776	4395	1618
		35	39	2676	859	107	10192	2895	396
	m	5	50	1069	742	363	6281	4484	2482
		10	47	846	81	36	3798	396	220
		15	44	2082	78	3	15381	462	51
		20	35	5755	663	12	20730	1814	75
		25	46*	NA	944*	7*	NA	5884*	55*
		30	46*	NA	1096*	2*	NA	5005*	35*
den900d	k	10	49	5874	121	89	299	1	1
		20	45	12975	1517	1064	231	7	6
		30	37	29742	2611	1503	186	19	11
		40	25	29661	7030	3063	115	38	21
		50	25	38357	10043	5331	378	80	75
		60	16	47144	15240	10845	203	50	36

Table 2: Runtime and number of expanded CT nodes wrt k and m. Column "#" is the number of instances solved by both P-CBS and A-CBS or only by A-CBS (*) within the timeout.

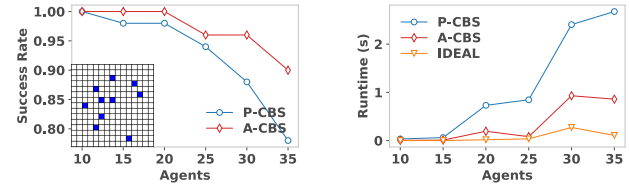
For each setting (a setting is an environment and a configuration of parameters), we solve 50 instances with different, randomly generated, start and goal vertices. We set a timeout of 30 seconds for small maps (as in Section 5.1) and a timeout of 5 minutes for large maps (as in Section 5.2). We measure the success rate, defined as the percentage of instances solved within the timeout, the runtime of the algorithms, and the number of nodes that the algorithms expand in the CT. The data points representing these values are averaged over the instances solved by all the three algorithms.

5.1 Small Grid Maps

We consider a 15 × 15 grid map. We place 1 × 1 obstacles in the map and we allow the solvers to remove a fraction $\beta = 30\%$ of the present obstacles. Although obstacles are randomly placed, we guarantee that they are detached from the outer border of the map and from each other. Note that, given the number m of removable obstacles, the number of configurations is $n = n(m) = \binom{m}{\lfloor \beta m \rfloor} \approx 2^{H(\beta)m}$, where $H(\beta)$ is the binary entropy function[8] [8]. Note that $n(25) > 10^5$ for $\beta = 30\%$ (Table 1). Results are reported in Figure 3 and in Table 2.

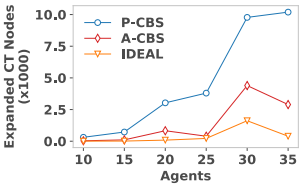
We first test this setting for a varying number of agents $k \in \{10, 15, \ldots, 35\}$ and $m = 10$ obstacles. In these small grid maps, A-CBS performs slightly better than P-CBS in terms of success rate and consistently better in terms of computing effort (runtime and number of expanded CT nodes), coming closer (wrt to P-CBS) to the lower bound represented by IDEAL. In particular, when the number

[8] $H(\beta) = -\beta\log(\beta) - (1-\beta)\log(1-\beta)$ for $\beta \in [0, 1]$.

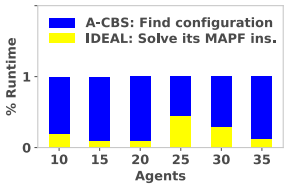


(a) Success rate wrt k

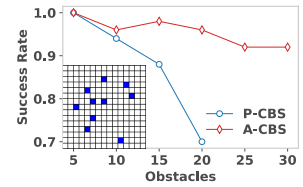
(b) Runtime wrt k



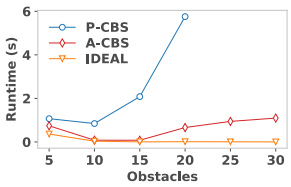
(c) Expanded CT nodes wrt k



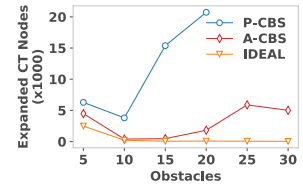
(d) Runtime partition wrt k



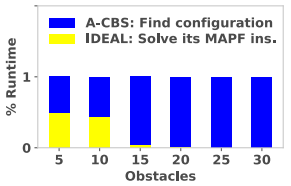
(e) Success rate wrt m



(f) Runtime wrt m



(g) Expanded CT nodes wrt m



(h) Runtime partition wrt m

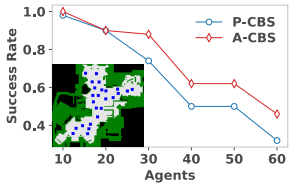
Figure 3: Results on small grid maps.

k of agents increases, A-CBS maintains a higher success rate and a shorter runtime. Runtime partition results suggest that, in this test, A-CBS spends more time in finding an optimal configuration than in solving the associated MAPF instance.

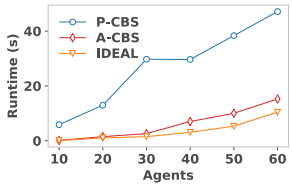
We further test the same setting for a varying number of removable obstacles $m \in \{5, 10, \ldots, 30\}$ and $k = 25$ agents. Results show that A-CBS behaves much better than P-CBS when the number of obstacles m increases. When m is large, the number of possible configurations of the environment is large and the combinatorial aspects of the corresponding C-MAPF problem becomes more relevant. Our results show that A-CBS dominates this increased complexity better than P-CBS, which seems unable to solve instances with $m = 25$ and $m = 30$ within the timeout.

5.2 Large Grid Maps

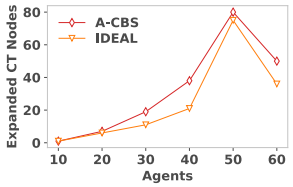
We consider a large 128×128 grid map, namely the benchmark *Dragon Age: Origins* (DAO) computer game map den900d from [42]. We also consider $m = 25$ 4×4 obstacles, each with an integer removal cost picked randomly from $\{1, 2, \ldots, 5\}$, and a budget of



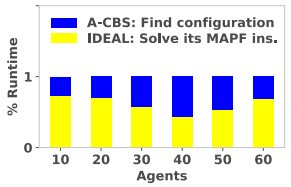
(a) Success rate wrt k



(b) Runtime wrt k



(c) Expanded CT nodes wrt k



(d) Runtime partition wrt k

Figure 4: Results on large grid maps.

10. In this way, the solvers will be likely forced to choose which obstacles should be removed, especially when the number of agents grows. We test this setting for a varying number of agents $k \in \{10, 20, \ldots, 60\}$. Results are reported in Figure 4 and in Table 2. A-CBS performs similarly to P-CBS for success rate and clearly outperforms P-CBS for runtime. Unlike P-CBS, the computing time of A-CBS is very close to the lower bound represented by the computing time of IDEAL, especially when k increases. In this case, runtime partition results show that the time spent by A-CBS to find an optimal configuration is slightly less than the time spent to solve the associated MAPF instance.

6 CONCLUSIONS

In this paper, we introduced C-MAPF, namely a version of the MAPF problem in configurable environments. From an algorithmic point of view, the C-MAPF problem adds another combinatorial dimension to MAPF problems, which is related to the need of selecting the most convenient configuration in order to optimize the flowtime, namely the sum of the costs of the individual paths. We also introduced two complete and optimal algorithms, P-CBS and A-CBS, that are experimentally demonstrated to perform well in different settings. In particular, A-CBS shows high success rates and short computing times even in challenging environments that admit several configurations. However, a deeper and more systematic comparison between the two algorithms is needed.

Future work will address the investigation of new algorithms for C-MAPF, for example sub-optimal solvers, in the attempt of trading off between efficiency and quality of solutions. Moreover, while in this paper we considered the flowtime as the only objective function, more complex objective functions could be defined, for example combining the cost of solutions (either flowtime or makespan) and the cost of modifying the environment (e.g., the cost of setting the environment in a configuration). Future work also includes the "online" variant of the problem, in which the environment can be dynamically reconfigured during execution, for example, to better exploit the advantages of mobile shelving.

REFERENCES

[1] P. Agarwal, N. Kumar, S. Sintos, and S. Suri. 2018. Computing shortest paths in the plane with removable obstacles. In *Proceedings of the Scandinavian Symposium and Workshops on Algorithm Theory*. 5:1–5:15.

[2] A. Andreychuk and K. Yakovlev. 2018. Path finding for the coalition of cooperative agents acting in the environment with destructible obstacles. In *Proceedings of the International Conference on Interactive Collaborative Robotics*. 13–22.

[3] D. Atzmon, A. Diei, and D. Rave. 2019. Multi-train path finding. In *Proceedings of the Annual Symposium on Combinatorial Search*. 125–129.

[4] J. Banfi, N. Basilico, and F. Amigoni. 2017. Intractability of time-optimal multi-robot path planning on 2D grid graphs with holes. *IEEE Robotics and Automation Letters* 2, 4 (2017), 1941–1947.

[5] M. Barer, G. Sharon, R. Stern, and A. Felner. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of the Annual Symposium on Combinatorial Search*. 19–27.

[6] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and S. Shimony. 2015. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 740–746.

[7] L. Cohen, T. Uras, and S. Koenig. 2015. Feasibility study: Using highways for bounded-suboptimal multi-agent path finding. In *Proceedings of the Annual Symposium on Combinatorial Search*. 2–8.

[8] S. Das. 2016. A brief note on estimates of binomial coefficients. (2016). http://page.mi.fu-berlin.de/shagnik/notes/binomials.pdf.

[9] Z. Drezner and G. Wesolowsky. 1997. Selecting an optimum configuration of one-way and two-way routes. *Transportation Science* 31, 4 (1997), 386–394.

[10] E. Erdem, D. Kisa, U. Oztok, and P. Schüller. 2013. A general formal framework for pathfinding problems with multiple agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 290–296.

[11] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T. Kumar, and S. Koenig. 2018. Adding heuristics to conflict-based search for multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*. 83–87.

[12] A. Felner, R. Stern, S. Shimony, E. Boyarski, M. Goldenberg, G. Sharon, N. Sturtevant, G. Wagner, and P. Surynek. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Proceedings of the Annual Symposium on Combinatorial Search*. 29–37.

[13] D. Ferguson, M. Darms, C. Urmson, and S. Kolski. 2008. Detection, prediction, and avoidance of dynamic obstacles in urban environments. In *Proceedings of the IEEE Intelligent Vehicles Symposium*. 1149–1154.

[14] C. Fulgenzi, A. Spalanzani, and C. Laugier. 2007. Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 1610–1616.

[15] M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. Sturtevant, R. Holte, and J. Schaeffer. 2014. Enhanced partial expansion A*. *Journal of Artificial Intelligence Research* 50 (2014), 141–187.

[16] P. Hart, N. Nilsson, and B. Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[17] W. Hönig, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian. 2018. Conflict-based search with optimal task assignment. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*. 757–765.

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

[19] W. Hönig, J. Preiss, T. Kumar, G. Sukhatme, and N. Ayanian. 2018. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics* 34, 4 (2018), 856–869.

[20] D. Johnson and J. Wiles. 2001. Computer games with intelligence. In *Proceedings of the IEEE International Conference on Fuzzy Systems*. 1355–1358.

[21] S. Karma, E. Zorba, G. Pallis, G. Statheropoulos, I. Balta, K. Mikedi, J. Vamvakari, A. Pappa, M. Chalaris, G. Xanthopoulos, and M. Statheropoulos. 2015. Use of unmanned vehicles in search and rescue operations in forest fires: Advantages and limitations observed in a field trial. *International Journal of Disaster Risk Reduction* 13 (2015), 307–312.

[22] S. Koenig. 2019. Learn all about Multi-Agent Path Finding (MAPF). (2019). http://mapf.info

[23] N. Kou, C. Peng, X. Yan, Z. Yang, H. Liu, K. Zhou, H. Zhao, L. Zhu, and Y. Xu. 2019. Multi-agent path planning with non-constant velocity motion. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*. 2069–2071.

[24] J. Kozůbek, Z. Flasar, and I. Dumišinec. 2019. Military Factors Influencing Path Planning. In *Path Planning for Autonomous Vehicles - Ensuring Reliable Driverless Navigation and Control Maneuver*. IntechOpen.

[25] J. Li, E. Boyarski, A. Felner, H. Ma, and S. Koenig. 2019. Improved heuristics for multi-agent path finding with conflict-based search. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 442–449.

[26] J. Li, D. Harabor, P. Stuckey, H. Ma, and S. Koenig. 2019. Disjoint splitting for multi-agent path finding with conflict-based search. In *Proceedings of the International Conference on Automated Planning and Scheduling*. 279–283.

[27] J. Li, P. Surynek, A. Felner, H. Ma, T. Kumar, and S. Koenig. 2019. Multi-agent path finding for large agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 7627–7634.

[28] H. Ma, S. Koenig, N. Ayanian, L. Cohen, W. Hönig, T. Kumar, T. Uras, H. Xu, C. Tovey, and G. Sharon. 2016. Overview: Generalizations of multi-agent path finding to real-world scenarios. In *Proceedings of the IJCAI-16 Workshop on Multi-Agent Path Finding*.

[29] H. Ma, J. Yang, L. Cohen, T. Kumar, and S. Koenig. 2017. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 270–272.

[30] O. Majerech. 2017. *Solving algorithms for multi-agent path planning with dynamic obstacles*. Master's thesis. Charles University, Prague, Czech Republic.

[31] R. Morris, C. Păsăreanu, K. Luckow, W. Malik, H. Ma, T. Kumar, and S. Koenig. 2016. Planning, scheduling and monitoring for airport surface operations. In *Proceedings of the AAAI-16 Workshop on Planning for Hybrid Systems*.

[32] A. Murano, G. Perelli, and S. Rubin. 2015. Multi-agent path planning in known dynamic environments. In *Proceedings of the International Conference on Principles and Practice of Multi-Agent Systems*. 218–231.

[33] A. Nash, K. Daniel, S. Koenig, and A. Felner. 2007. Theta*: Any-angle path planning on grids. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 1177–1183.

[34] M. Ryan. 2008. Exploiting subgraph structure in multi-robot path planning. *Journal of Artificial Intelligence Research* 31 (2008), 497–542.

[35] G. Sharon, R. Stern, A. Felner, and N. Sturtevant. 2012. Conflict-based search for optimal multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 563–569.

[36] G. Sharon, R. Stern, A. Felner, and N. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Journal of Artificial Intelligence Research* 219 (2015), 40–66.

[37] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. 2011. The increasing cost tree search for optimal multi-agent pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 662–667.

[38] E. Sperner. 1928. Ein satz über untermengen einer endlichen menge. *Mathematische Zeitschrift* 27, 1 (1928), 544–548.

[39] M. Staller and M. Staller. 1972. Motorized mobile shelving units. (1972). US Patent 3,640,595.

[40] T. Standley. 2010. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 173–178.

[41] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar, E. Boyarski, and R. Barták. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the Annual Symposium on Combinatorial Search*. 151–158.

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

[43] P. Surynek. 2012. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *Proceedings of the Pacific Rim International Conference on Artificial Intelligence*. 564–576.

[44] J. Švancara. 2018. Bringing multi-agent path finding closer to reality. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*. 1784–1785.

[45] M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal. 2015. CoBots: Robust symbiotic autonomous mobile service robots. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 4423–4429.

[46] G. Wagner and H. Choset. 2015. Subdimensional expansion for multirobot path planning. *Journal of Artificial Intelligence* 219 (2015), 1–24.

[47] P. Wurman, R. D'Andrea, and M. Mountz. 2007. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 1752–1759.

[48] J. Yu and S. LaValle. 2013. Planning optimal paths for multiple robots on graphs. In *Proceedings of the International Conference on Robotics and Automation*. 3612–3617.

[49] J. Yu and S. LaValle. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 1443–1449.

[50] J. Yu and D. Rus. 2015. Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *Algorithmic Foundations of Robotics XI*. 729–746.