Solving the Non-Crossing MAPF with CP

Xiao Peng, Christine Solnon, Olivier Simonin

Solving the Non-Crossing MAPF with CP

Xiao Peng
CITI, INRIA, INSA Lyon, F-69621, Villeurbanne, France

Christine Solnon
CITI, INRIA, INSA Lyon, F-69621, Villeurbanne, France

Olivier Simonin
CITI, INRIA, INSA Lyon, F-69621, Villeurbanne, France

Abstract

We introduce a new Multi-Agent Path Finding (MAPF) problem which is motivated by an industrial application. Given a fleet of robots that move on a workspace that may contain static obstacles, we must find paths from their current positions to a set of destinations, and the goal is to minimise the length of the longest path. The originality of our problem comes from the fact that each robot is attached with a cable to an anchor point, and that robots are not able to cross these cables.

We formally define the Non-Crossing MAPF (NC-MAPF) problem and show how to compute lower and upper bounds by solving well known assignment problems. We introduce a Variable Neighbourhood Search (VNS) approach for improving the upper bound, and a Constraint Programming (CP) model for solving the problem to optimality. We experimentally evaluate these approaches on randomly generated instances.

2012 ACM Subject Classification Computing methodologies

Keywords and phrases Constraint Programming (CP), Multi-Agent Path Finding (MAPF), Assignment Problems

Digital Object Identifier 10.4230/LIPIcs.CP.2021.20

Acknowledgements This work was supported by the European Commission under the H2020 project BugWright2 (871260): Autonomous Robotic Inspection and Maintenance on Ship Hulls and Storage Tanks.

1 Introduction

Multi-agent path finding (MAPF) is a very active research topic which has important applications for robotics in industrial contexts (*e.g.*, transport in fulfillment centers, autonomous tug robots). In this paper we consider an extension of MAPF for tethered robots, *i.e.*, robots attached with flexible cables to anchor points, allowing them to have continuous access to fluids such as energy or water, for example. This is the case for our industrial partner in a European project[1] where a fleet of mobile robots is used for inspecting and cleaning large structures. Each robot has a cable which is kept taut between its anchor point and its current position by a system that pulls on the cable when the robot moves back. The main difficulty with these tethered robots comes from the fact that robots are not able to cross cables. Hence, this paper introduces the Non-Crossing MAPF (NC-MAPF) problem which aims at finding paths such that robots never have to cross cables.

[1] H2020 project BugWright2 : Autonomous Robotic Inspection and Maintenance on Ship Hulls and Storage Tanks, 2020-24

Related work

In classic MAPF, agents move in a discretized environment (a grid or a graph). The goal is to find a plan for moving all agents from their initial locations to target locations so that no two agents share a same location (grid cell, graph node, or graph edge) at a same moment. Typically, a plan is a sequence of actions for each agent, where an action is either "move to an adjacent location" or "wait at the current location".

There are two main MAPF variants depending on whether each agent has a known target, or there is a set of targets and each agent must be first assigned to a target before searching for a plan. This latter variant, called *anonymous MAPF*, is more general and also more difficult as the search space is increased. There are two main objective functions, *i.e.*, minimise the *makespan*, corresponding to the latest arrival time of an agent to its target, or minimise the sum of all travel times. In both cases, the problem is $\mathcal{NP}$-hard [18].

MAPF problems are usually solved by using *Conflict Based Search (CBS)* approaches [15] which are two-level approaches: at the low level, paths are searched (while satisfying constraints added at the high level); and at the high level, path conflicts are resolved. CBS has been extended to agents with a specific geometric shape and volume (*e.g.*, [9, 17]) and to convoys (agents that occupy a sequence of nodes and their connecting edges) [16]. These MAPF variants share some similarities with NC-MAPF as a tethered robot may be viewed as a robot which has a very long body corresponding to its cable.

However, CBS is not suited to solve NC-MAPF because this approach is efficient when conflicts are easily resolved by applying small changes to paths (*e.g.*, waiting for a location to be freed or getting around an occupied location). This is not the case for NC-MAPF. For example, let us consider the case of two paths π_1 (from an anchor point a_1 to a target t_1) and π_2 (from a_2 to t_2) such that the cables cross at some point x. To solve this conflict, a first possibility is to ask the first robot to wait just before reaching x while the second robot continues its path from x to d_2, achieves its task on d_2, and returns back to x, thus removing the cable from x and allowing the first robot to continue its path from x to d_1. As robots usually have to achieve long duration tasks, this way of resolving conflicts dramatically increases the makespan. A second possibility is to search for new paths such that cables do not cross, but this cannot be done by applying small changes to the paths and this problem may have no solution in some cases.

In the robotics literature, few works have investigated path planning for tethered robots. In most cases, cables may be pushed and bent by robots (*e.g.*, [6, 19]), which is not possible in our industrial context. As far as we know, none has considered a case similar to our problem where (i) robots cannot cross neither push or bent cables, (ii) paths cannot be sequentialized (*i.e.*, a robot cannot wait for another robot to have achieved its task and returned back to its anchor point), and (iii) robots do not have assigned targets (anonymous MAPF).

Contributions and outline of the paper

In Section 2, we introduce notations and define the workspace on which robots evolve. This workspace is continuous, and we show in Section 3 how to reformulate our problem in a discrete visibility graph.

In Section 4, we first consider the case where the workspace has no obstacle. We show that the NC-MAPF problem without obstacle is a special kind of assignment problem in a bipartite graph, and we show how to efficiently compute lower and upper bounds by solving well known assignment problems. We also introduce a *Variable Neighbourhood Search (VNS)* approach, to improve the upper bound, and a *Constraint Programming (CP)* model, to

compute the optimal solution.

In Section 5, we consider the case where the workspace has obstacles. We prove that optimal solutions of assignment problems still provide bounds in this case. We also show that the optimal solution of the NC-MAPF problem may contain some paths that are not shortest paths. Hence, we introduce an approach for enumerating all relevant paths and, finally, we introduce a CP model for computing the optimal solution.

In Sections 4 and 5, we report experimental results on randomly generated instances and show that our approach scales well enough to solve realistic instances within a few seconds.

2 Definition of the workspace and notations

Robots move on a 2 dimensional workspace $W \subset \mathbb{R}^2$. This workspace is defined by a bounding polygon B and a set O of obstacles: every obstacle in O is a polygon within B, and W is composed of every point in B that does not belong to an obstacle in O. Without loss of generality, we assume that B is convex: if the bounding polygon is not convex, then we can compute its convex hull B and add to O the obstacle corresponding to the difference between the bounding polygon and B. We denote V_O the set of vertices of obstacles in O, and we assume that these vertices belong to W (and, therefore, obstacle boundaries belong to W).

Given two points $u, v \in W$, we denote $\overline{uv}$ the straight line segment that joins u to v, and $|uv|$ the Euclidean distance between u and v (*i.e.*, $|uv|$ is the length of $\overline{uv}$). We say that a segment crosses an obstacle if $\overline{uv} \not\subset W$. Given two segments $\overline{uv}$ and $\overline{u'v'}$, we say that they are incident if they have one common endpoint (*i.e.*, $|\{u, v\} \cap \{u', v'\}| = 1$), and we say that they cross if they share one point (called the crossing point) which is not an endpoint (*i.e.*, $\{u, v\} \cap \{u', v'\} = \emptyset$ and $\overline{uv} \cap \overline{u'v'} \neq \emptyset$).

A chain of incident segments $\overline{u_0u_1}, \overline{u_1u_2}, \ldots, \overline{u_{i-1}u_i}$ is represented by the sequence $\pi = \langle u_0, u_1, u_2, \ldots, u_i \rangle$. The length of this chain of segments is denoted $|\pi|$ and is the sum of the lengths of its segments, *i.e.*, $|\pi| = \sum_{j=1}^{i} |u_{j-1}u_j|$.

We denote $[x, y]$ the set of all integer values ranging between x and y.

3 Definition of the NC-MAPF Problem

We consider an anonymous MAPF problem with a set of n robots such that each robot is attached with a flexible cable to an anchor point in W, and a set of n destinations. The goal is to find a path in W for each robot from its anchor point to a different destination so that the longest path is minimised and robots never have to cross cables.

As the workspace W is continuous, there exists an infinite number of paths from an anchor point a to a destination d. However, as each cable is kept taut, the number of different cable positions that start from a and end on d is finite (provided that we forbid infinite loops). More precisely, the cable position associated with a robot path from a to d is a chain of incident segments $\langle u_0, u_1, \ldots, u_i \rangle$ such that (i) $u_0 = a$ and $u_i = d$, (ii) no segment crosses an obstacle, and (iii) every internal point is an obstacle vertex, *i.e.*, $\forall j \in [1, i-1], u_j \in V_O$.

As the length of a robot path cannot be smaller than the length of its cable position, we can simplify our problem by assuming that the path of a robot is its cable position. Hence, we search for paths in a visibility graph [10] defined in Def. 1 and illustrated in Fig. 1.

▶ **Definition 1** (Visibility graph [10]). *The visibility graph associated with a workspace W, a set of anchor points A and a set of destinations D is the directed graph $G = (V, E)$ such that vertices are either points of A and D or obstacle vertices,* i.e., *$V = A \cup D \cup V_O$, and edges*

correspond to segments that do not cross obstacles, i.e., $E = \{(u, v) \in (A \cup V_O) \times (D \cup V_O) | \overline{uv} \subset W\}$. *The graph is directed because edges from destinations to anchor points are forbidden.*

In Def. 1, we implicitly assume that robots are points, which is an acceptable approximation when the actual size of robots is very small compared to the size of obstacles (which is the case in our industrial application). This definition may be extended to the case where robot shapes are approximated by circles with non null radius in a straightforward way by growing obstacles (see [10] for details).

A path in the visibility graph G is a sequence of vertices $\langle u_0, \ldots, u_i \rangle$ such that $(u_{j-1}, u_j) \in E, \forall j \in [1, i]$. This path also corresponds to a chain of segments and its length is the sum of the lengths of its segments. We only consider elementary paths, *i.e.*, a vertex cannot occur more than once in a path. Indeed, if a path is not elementary, then it can be replaced by a shorter elementary path obtained by removing its cycles.

Two paths are homotopic is there exists a continuous deformation between them without crossing obstacles [2], and a taut path is the shortest path of a homotopy class. For example, in the workspace of Fig. 1, all paths starting from the anchor point a_1 (point 1 in blue), passing between O_1 and O_2 and then between O_1 and O_3, and finally reaching the destination d_1 (point 1 in red) are homotopic. Let x be the bottom-left vertex of O_1, y its bottom-right vertex, z the top-left vertex of O_3, and t the top-right vertex of O_2. The paths $\pi = \langle a_1, x, y, z, d_1 \rangle$ and $\pi' = \langle a_1, x, t, z, d_1 \rangle$ are homotopic. π is taut because it is the shortest path of its homotopy class. π' is not taut because it is longer than π.

We say that a path is self-crossing if it contains two crossing segments. We say that two paths π and π' are crossing either if π contains a segment that crosses a segment of π', or if π contains two incident segments $\overline{uv}$ and $\overline{vw}$ and π' contains two incident segments $\overline{u'v'}$ and $\overline{v'w'}$ such that $v = v'$ and $\overline{uw}$ crosses $\overline{u'w'}$. However, two non crossing paths may share some vertices or some segments, as illustrated in Fig. 1(c). Indeed, as robots are small and cables are thin, a robot can slightly push the cable of another robot without crossing its cable. For example, if the black robot (starting from 3) in Fig. 1(c) arrives on the vertex of obstacle O_4 before the blue robot (starting from 4) then, when the blue robot arrives on this vertex, it can slightly push the black cable to continue its path between O_4 and the black cable.

Let us now formally define our problem.

▶ **Definition 2** (NC-MAPF problem). *Given a workspace W, a set A of n anchor points and a set D of n destinations such that every point in $A \cup D$ belongs to W, the goal of the NC-MAPF problem is to find n paths in the visibility graph G associated with W, A, and D such that (i) every path is taut, (ii) every path starts on a different anchor point of A, (iii) every path ends on a different destination of D, (iv) no path is self-crossing, (v) no two paths are crossing, and (vi) the length of the longest path is minimal.*

4 NC-MAPF problem without obstacles

In this section, we consider the case where the set O of obstacles is empty. In this case, $V_O = \emptyset$ and the visibility graph G is the complete bipartite graph such that $V = A \cup D$ and $E = A \times D$ (every edge of E is included in W as the bounding polygon is convex).

In Section 4.1, we show how to compute lower and upper bounds by solving well known assignment problems. In Section 4.2, we show how to improve the upper bound by performing variable neighbourhood search. In Section 4.3, we introduce a CP model and, in Section 4.4, we experimentally evaluate these approaches.

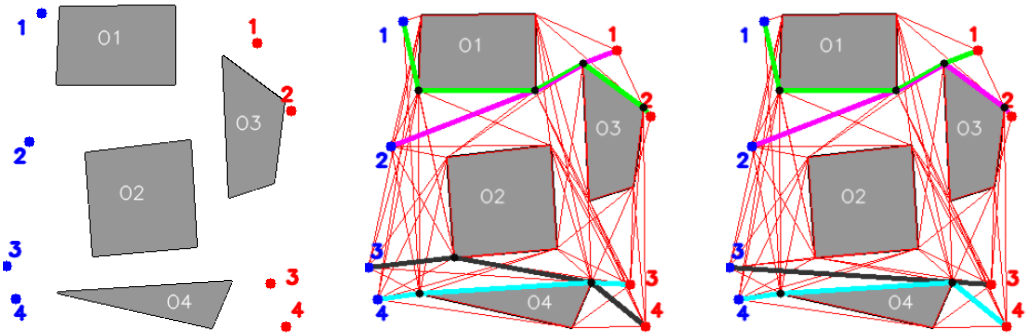

Figure 1 (a): Example of workspace W with four anchor points (in blue) and four destinations (in red). (b): Visibility graph with paths that are not solution of the NC-MAPF because the green path crosses the pink path and the black path crosses the blue path. Besides, the black path is not taut. (c): Visibility graph with paths that are solution of the NC-MAPF, even though the green and pink paths share a segment, and the black and blue paths share a vertex.

4.1 Computation of bounds by solving assignment problems

An assignment problem aims at finding a one-to-one matching between tasks and agents [3, 13]. In our context, tasks correspond to destinations and agents to robots, and a matching is a bijection $m : A \to D$. We say that an edge (a, d) of the visibility graph G is selected whenever $m(a) = d$. The NC-MAPF problem without obstacles is a special case of assignment problem:

- there is an additional constraint that ensures that no two selected edges cross, *i.e.*, $\forall\{a_i, a_j\} \subseteq A, \overline{a_i m(a_i)} \cap \overline{a_j m(a_j)} = \emptyset$;
- there is an objective function that aims at minimising the maximal cost of a selected edge, *i.e.*, $\max_{a_i \in A} |a_i m(a_i)|$.

There exists many other assignment problems [3, 13]. The most well known one is the *Linear Sum Assignment Problem (LSAP)* that aims at minimising the sum of the costs of the selected edges. The LSAP can be solved in polynomial time (*e.g.*, by the Hungarian algorithm [7]). Interestingly, the solution of the LSAP cannot have crossing edges whenever edge costs are defined by Euclidean distances [14]. Indeed, if two selected edges cross, then we can obtain a better assignment by swapping their destinations so that the two edges no longer cross. Hence, the solution of the LSAP provides an upper bound to the NC-MAPF problem without obstacles.

The assignment problem that aims at minimising the maximal cost of a selected edge is known as the *Linear Bottleneck Assignment Problem (LBAP)*, and this problem can also be solved in polynomial time (*e.g.*, by adapting the Hungarian algorithm). However, when adding the constraint that the selected edges must not cross, the problem becomes $\mathcal{NP}$-hard [4]. Hence, the solution of the LBAP provides a lower bound to the NC-MAPF problem without obstacles.

4.2 Variable Neighbourhood Search

The upper bound computed by solving a LSAP may be tightened by performing local search. We consider a basic VNS framework [11] described below.

- The neighbourhood of a matching m contains every non crossing matching obtained by permuting the destinations of k anchor points, and it is explored in $\mathcal{O}(\binom{n-1}{k-1} \cdot k!)$: we first search for the longest edge $(a, m(a))$; then, we enumerate subsets of $A \setminus \{a\}$ that contain $k-1$ anchor points and, for each subset (to which a is added), we consider every permutation of the destinations without crossing edges, until finding a permutation whose longest edge is smaller than $(a, m(a))$.
- k is initialised to 2, and the search is started from the matching computed by solving the LSAP. We iteratively perform improving moves, by replacing the current matching with one of its neighbours that has a shorter longest edge. When we reach a locally optimal matching (that cannot be improved by permuting the destinations associated with k anchor points), we increase k. When an improving move is performed, k is reset to 2.
- The search is stopped either when a given time limit l is reached or when k becomes greater than a given upper bound k_{max}. (In the classical VNS framework, the current solution is perturbated and k is reset to its lowest possible value when k becomes greater than its upper bound k_{max}. We do not consider this perturbation phase here.)

4.3 Constraint Programming Model

Finally, let us introduce a CP model for the NC-MAPF problem without obstacles. Without loss of generality, we assume that all edge lengths have integer values: if this is not the case, then we can multiply every length by a given constant factor $c > 1$ and then round it to the closest integer value so that for each couple of edges $((u, v), (u', v'))$ such that $|uv| < |u'v'|$, we have $round(c * |uv|) < round(c * |u'v'|)$. In this case, the optimal solution of the integer problem is also an optimal solution of the original problem.

Let ub be an upper bound to the optimal solution. The variables are:

- an integer variable x_i is associated with every anchor point $a_i \in A$, and the domain of this variable contains every destination that is within a distance of ub from a_i, *i.e.*, $D(x_i) = \{d \in D : |a_i d| < ub\}$;
- an integer variable y represents the maximal length of a selected edge.

The constraints are:

- for each pair of anchor points $\{a_i, a_j\} \subseteq A$, we post a table constraint $(x_i, x_j) \in T_{ij}$ where T_{ij} is the table that contains every couple $(d, d') \in D(x_i) \times D(x_j)$ such that $d \neq d'$ and the segment $\overline{a_i d}$ does not cross the segment $\overline{a_j d'}$;
- for each anchor point $a_i \in A$, we post the constraint $y \geq |a_i x_i|$;
- we post an *allDifferent*$(\{x_i : a_i \in A\})$ constraint. This constraint is redundant as table constraints prevent assigning a same value to two different x_i variables. However, preliminary experiments have shown us that this improves the solution process for a wide majority of instances.

The goal is to minimise y.

4.4 Experimental evaluation

We evaluate our algorithms on randomly generated instances. For all instances, the bounding polygon is the square $B = [0, 200]^2$. To generate an instance with n robots, we randomly generate n anchor points and n destinations that all belong to B and such that the distance between two points is always larger than 4. For each value of n, we generate 50 different instances and report average results on these instances for all figures and tables.

We consider the following approaches:

- LB refers to the computation of a lower bound by solving an LBAP (see Section 4.1).

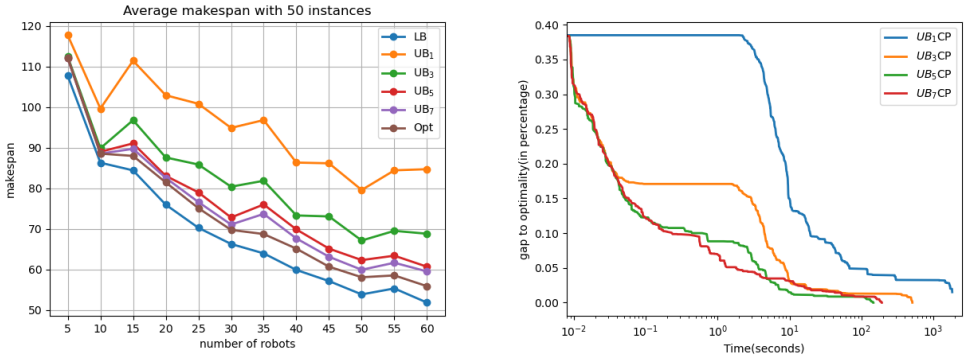

Figure 2 Left: Evolution of the optimal makespan (Opt), the lower bound (LB) and upper bounds (UB_i with $i \in \{1,3,5,7\}$) when increasing the number n of robots. Right: Evolution of the gap to optimality (in percentage) with respect to time for UB_iCP with $i \in \{1,3,5,7\}$, on average for the 50 instances with $n = 50$ robots.

- UB_i with $i \in \{1,3,5,7\}$ refers to the computation of an upper bound by first solving an LSAP (see Section 4.1) and then improving it by VNS with $l = 60$ seconds and $k_{max} = i$ (see Section 4.2). Note that when $i = 1$, VNS is immediately stopped as k is initialised to 2 and the search is stopped when k becomes greater than k_{max}.
- UB_iCP refers to the sequential combination of UB_i, for computing an upper bound ub, and CP (with the model described in Section 4.3) for computing the optimal solution.

LB and UB_i are implemented in Python. The CP model is implemented in MiniZinc [12] and solved with Chuffed [5]. All experiments are run on an Intel Core Intel Xeon E5-2623v3 of 3.0GHz×16 with 32GB of RAM.

On the left part of Fig. 2, we compare the optimal makespan with the lower bound computed by LB, and upper bounds computed by UB_i with $i \in \{1,3,5,7\}$. We observe that the optimal makespan decreases as the number n of robots increases. Indeed, when n gets larger, anchor and destination points tend to be located more densely and this makes it easier to assign anchor points to closer destinations. LB is always strictly smaller than the optimal makespan, *i.e.*, the solution of the LBAP always contains crossing segments.

UB_1 corresponds to the solution of the LSAP, and this upper bound is much larger than the optimal makespan. VNS strongly decreases this upper bound, and the larger k_{max} the smaller the bound. Note that when $k_{max} \geq n$, VNS actually finds the optimal makespan as it explores all possible permutations of the n destinations (provided that we do not limit time, *i.e.*, $l = \infty$). Hence, when $n = 5$, the solution of UB_5 is equal to the optimal makespan.

However, if UB_i finds smaller bounds when increasing i, it also needs more time. This is shown on the right part of Fig. 2, for instances that have $n = 50$ robots. We display the evolution of the average gap to optimality in percentage (*i.e.*, $\frac{s-s^*}{s^*}$ where s^* is the optimal makespan and s is the current makespan) with respect to CPU time. For UB_1CP, the upper bound ub is very quickly computed by solving the LSAP, but it is 38% as large as the optimal makespan. ub is used to filter variable domains of x_i variables. However, as ub is not very tight, the construction of the table T_{ij} for every couple of variables (x_i, x_j) is time consuming. This construction phase corresponds to the horizontal part of the curve. Once the CP model has been constructed, Chuffed finds better solutions and finally proves optimality. When increasing k_{max}, the time spent by VNS to improve ub increases but, as a counterpart, the time spent to build the CP model and the time spent by Chuffed to solve it also decreases.

Table 1 Scale-up properties with respect to the number n of robots. For each $n \in \{20, \ldots, 60\}$, we report CPU times of UB$_i$CP (in seconds), for $i \in \{1, 3, 5, 7\}$: t_1 is the time spent to solve the LSAP and improve the upper bound with VNS when $k_{max} = i$ and $l = 60s$; t_2 is the time to generate the MiniZinc model; t_3 is the time spent by Chuffed; $t_{tot} = t_1 + t_2 + t_3$ is the total time (in blue when minimal). Chuffed is limited to 3600s and the time of a run is set to 3600 when this limit is reached. In this case, t_3 is a lower bound of the actual time (and we display $\geq$ before the time).

n	UB$_1$CP				UB$_3$CP				UB$_5$CP				UB$_7$CP			
	t$_1$	t$_2$	t$_3$	t$_{tot}$	t$_1$	t$_2$	t$_3$	t$_{tot}$	t$_1$	t$_2$	t$_3$	t$_{tot}$	t$_1$	t$_2$	t$_3$	t$_{tot}$
20	0.001	0.4	0.1	0.5	0.01	0.2	0.1	0.3	0.0	0.2	0.1	0.3	0.8	0.2	0.1	1.1
30	0.002	1.4	≥35.4	36.9	0.01	0.9	0.4	1.3	0.1	0.6	0.1	0.9	2.3	0.6	0.2	3.1
40	0.004	3.4	12.4	15.8	0.02	2.1	1.4	3.5	0.3	1.8	0.6	2.6	7.2	1.6	0.5	9.2
50	0.003	6.7	≥127.2	133.9	0.03	4.1	13.6	17.7	0.5	3.1	7.5	11.1	7.6	2.8	7.7	18.2
60	0.008	16.8	≥529.3	546.1	0.06	9.4	≥197.6	207.4	1.3	6.1	27.0	34.4	16.8	5.7	25.5	48.1

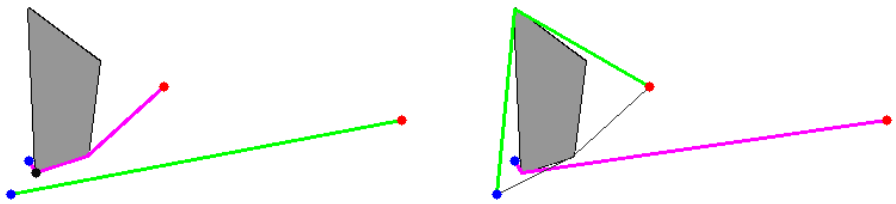

Figure 3 The solution displayed on the left only uses shortest paths, and its makespan is larger than the solution displayed on the right (the green right path is longer than the black path).

Table 1 allows us to study scale-up properties when increasing the number n of robots. The time spent by UB$_i$ (t_1) strongly increases when i increases: from 0.008s when $i = 1$ to more than 16s when $i = 7$ for $n = 60$. This was expected as the time complexity of VNS is exponential with respect to k_{max}. The time limit $l = 60s$ is never reached by VNS when $i \leq 5$ whereas it is reached when $i = 7$: for 7 (resp. 1 and 1) instances when $n = 60$ (resp. 50 and 40). However, when increasing i, UB$_i$ computes better bounds and this reduces the time needed to generate the model (t_2) and to solve it (t_3). When $i = 1$, the time limit of $3600s$ is reached by Chuffed for 6 (resp. 1 and 1) instances when $n = 60$ (resp. 50 and 30). It is also reached once when $i = 3$ and $n = 60$. A good compromise is observed with UB$_5$CP.

5 NC-MAPF problem with obstacles

Let us now consider the case where the workspace contains obstacles. In this case, the visibility graph is no longer a bipartite graph, and a path from an anchor point to a destination may contain more than one edge. Besides, with the existence of obstacles, there might exist more than one possible path, even when restricting our attention to paths in the visibility graph, and an optimal solution may contain paths that are not shortest paths, as illustrated in Fig. 3. As a consequence, our problem is no longer a simple bipartite matching problem: we must not only choose a different destination for each anchor point, but also choose paths.

The number of paths between two points grows exponentially with respect to the number of obstacles. However, if we have an upper bound on the maximal length of a path, we can reduce the number of paths. Hence, we show how to compute upper bounds on the makespan

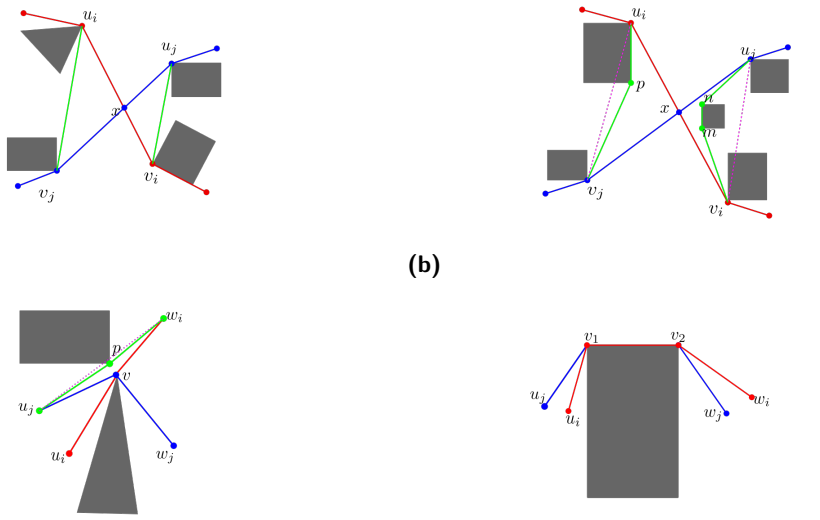

Figure 4 Top (Case 1): π_i (in red) and π_j (in blue) contain two crossing segments $\overline{u_iv_i}$ and $\overline{u_jv_j}$. (a): $\overline{u_iv_j}$ and $\overline{u_jv_i}$ (in green) do not cross obstacles and $|u_iv_j| + |u_jv_i| < [u_iv_i| + |u_jv_j|$. (b): $\overline{u_iv_j}$ and $\overline{u_jv_i}$ (dotted lines) cross obstacles but $\pi_{ij} = \langle u_i, p, v_j\rangle$ and $\pi_{ji} = \langle u_j, n, m, v_i\rangle$ (in green) do not cross obstacles and $|\pi_{ij}| + |\pi_{ji}| < |u_iv_i| + |u_jv_j|$. Bottom (Case 2): π_i (in red) and π_j (in blue) cross at a common vertex. (c): By swapping w_i and w_j we obtain non crossing paths which are not shortest paths ($|\langle u_j, p, w_i\rangle| < |u_j, v, w_i\rangle|$). (d): By swapping w_i and w_j we obtain non crossing paths that have the same length.

in Section 5.1. In Section 5.2, we show how to compute all relevant paths. In Section 5.3, we describe a CP model and in Section 5.4 we experimentally evaluate our approach.

5.1 Computation of bounds

When there are obstacles, the visibility graph G associated with W, A and D is no longer a bipartite graph. However, we can build a bipartite graph $G' = (V', E')$ such that $V' = A \cup D$ and $E' = A \times D$, and define the cost of an edge $(a, d) \in E'$ as the length of the shortest path from a to d in G. In this case, we can compute a lower bound by solving the LBAP in G'.

Let us now show that we can also compute an upper bound by solving the LSAP in G', as a straightforward consequence of the following theorem.

▶ **Theorem 3.** *Let $m : A \to D$ be an optimal solution of the LSAP in G' and, for each anchor point $a_i \in A$, let π_i be the shortest path that connects a_i to $m(a_i)$ in the visibility graph. For each pair of different anchor points $\{a_i, a_j\} \subseteq A$, either π_i and π_j are not crossing, or they can be replaced by two non crossing paths π'_i and π'_j such that $|\pi_i| + |\pi_j| = |\pi'_i| + |\pi'_j|$.*

Proof. Let us suppose that there exist two crossing paths π_i and π_j. There are two cases to consider, depending on whether π_i and π_j contain two crossing segments or not.

Case 1: π_i and π_j contain two crossing segments $\overline{u_iv_i}$ and $\overline{u_jv_j}$. Let us show that this implies that m does not minimise the sum of the selected edge costs. There are two sub-cases to consider.

Subcase a: $\overline{u_iv_j}$ and $\overline{u_jv_i}$ do not cross obstacles, as illustrated in Fig. 4a.

Let π_i^p (resp. π_i^s) be the prefix (resp. suffix) of π_i that precedes (resp. succeeds) $\overline{u_i v_i}$, *i.e.*, $\pi_i = \pi_i^p \cdot \langle u_i, v_i \rangle \cdot \pi_i^s$ where $\cdot$ denotes path concatenation. Similarly, let $\pi_j = \pi_j^p \cdot \langle u_j, v_j \rangle \cdot \pi_j^s$. Let x be the crossing point between $\overline{u_i v_i}$ and $\overline{u_j v_j}$. We have:

$$|u_i v_i| = |u_i x| + |x v_i| \text{ and } |u_j v_j| = |u_j x| + |x v_j|. \tag{1}$$

The triangle inequality implies that

$$|u_i v_j| < |u_i x| + |x v_j| \text{ and } |u_j v_i| < |u_j x| + |x v_i|. \tag{2}$$

From Eq. (1) and (2), we infer that

$$|u_i v_j| + |u_j v_i| < |u_i v_i| + |u_j v_j|. \tag{3}$$

When swapping v_i and v_j, π_i and π_j are replaced by the two paths $\pi'_i = \pi_i^p \cdot \langle u_i, v_j \rangle \cdot \pi_j^s$ and $\pi'_j = \pi_j^p \cdot \langle u_j, v_i \rangle \cdot \pi_i^s$. From Eq. (3), we have $|\pi'_i| + |\pi'_j| < |\pi_i| + |\pi_j|$. This is in contradiction with the fact that m minimises the sum of the costs of the selected edges in G' as the costs of edges $(a_i, m(a_j))$ and $(a_j, m(a_i))$ in G' are smaller than or equal to $|\pi'_i|$ and $|\pi'_j|$, respectively (they may be strictly smaller if π'_i or π'_j are not shortest paths in G).

Subcase b: $\overline{u_i v_j}$ and $\overline{u_j v_i}$ cross obstacles, as illustrated in Fig. 4b.

In this case, we cannot simply exchange the two crossing segments to obtain two non crossing paths. However, let π_{ij} be the path from u_i to v_j corresponding to the convex hull of all vertices that belong to the triangle defined by u_i, v_j and x. This path is displayed in green in Fig. 4b. We can show that $|\pi_{ij}| < |u_i x| + |x v_j|$ by recursively exploiting the triangle inequality (see [1]). Similarly, there exists a path π_{ji} between u_j and v_i such that $|\pi_{ji}| < |u_j x| + |x v_i|$. Therefore, $|\pi_{ij}| + |\pi_{ji}| < |u_i v_i| + |u_j v_j|$. Like in Subcase a, this is in contradiction with the fact that m minimises the sum of the costs of the selected edges in G'.

Case 2: π_i and π_j do not contain crossing segments but they cross at some vertex v. Let π be the longest path that is common to both π_i and π_j, *i.e.*, $\pi_i = \pi_i^p \cdot \pi \cdot \pi_i^s$ and $\pi_j = \pi_j^p \cdot \pi \cdot \pi_j^s$. We can exchange π_i^s and π_j^s to obtain two paths $\pi'_i = \pi_i^p \cdot \pi \cdot \pi_j^s$ and $\pi'_j = \pi_j^p \cdot \pi \cdot \pi_i^s$. There are two sub-cases to consider.

Subcase c: π'_i and/or π'_j are not shortest paths, as illustrated in Fig. 4c. In this case, we can obtain a better assignment by matching a_i with $m(a_j)$ and a_j with $m(a_i)$. This is in contradiction with the fact that m is the optimal assignment.

Subcase d: π'_i and π'_j are shortest paths, as illustrated in Fig. 4d. In this case, we can obtain an assignment which has the same cost as m by matching a_i with $m(a_j)$ and a_j with $m(a_i)$, and π'_i and π'_j no longer cross at vertex v. If they cross at some other vertex, we can recursively apply the same reasoning to either show that π'_i and π'_j are not shortest paths and exhibit a contradiction (Subcase c), or show that there exist two non crossing paths that have the same length as π'_i and π'_j (Subcase d).

◀

Hence, we can compute an upper bound by solving the LSAP in the bipartite graph G'. If some paths are crossing in the optimal solution, then we can exchange sub-paths in the crossing paths in order to obtain a solution with no crossing paths (and the same objective function value), as explained in Subcase d of Theo. 3.

Like for the NC-MAPF without obstacles, this upper bound may be improved by VNS, as explained in Section 4.2. We only have to adapt the procedure that explores the neighbourhood

of a matching, in order to check that permutations do not contain crossing paths (instead of crossing edges). Note that this test is done in quadratic time with respect to the number of edges in a path (whereas it is done in constant time when there is no obstacle).

5.2 Relevant paths enumeration

The non crossing assignment in G' that minimises the makespan may not be the optimal solution of the original problem as edges of G' correspond to shortest paths, and as the optimal solution may use non shortest paths. To find the optimal solution, for each couple $(a, d) \in A \times D$, we must consider all relevant paths from a to d in the visibility graph G, where a path π is relevant if it satisfies the three following constraints:

(C1) Given an upper bound ub on the optimal makespan (or on the maximal length of the cable anchored at a), π must be shorter than ub, *i.e.*, $|\pi| < ub$;

(C2) π must be elementary and not self-crossing;

(C3) π must be a taut path (as defined in Section 3).

Before enumerating all relevant paths, we remove from the visibility graph every edge that cannot belong to a taut path, thus obtaining the reduced visibility graph [8]. Then, all relevant paths starting from an anchor point a are enumerated by performing a depth first search starting from a, and pruning branches whenever a constraint is violated. To check constraint (C3), we perform a local geometric test in constant time.

5.3 Constraint Programming Model

Let ub be an upper bound to the optimal solution, and let P be the set of relevant paths as defined in the previous section (paths in P are numbered from 1 to $\#P$). For each path $\pi \in P$, $o(\pi)$, $d(\pi)$, and $l(\pi)$ denote the origin, the destination, and the length of π, respectively. The CP model has the following variables:

- an integer variable x_i is associated with every anchor point $a_i \in A$, and its domain contains every destination that may be reached from a_i, *i.e.*, $D(x_i) = \{d(\pi) : \pi \in P \wedge o(\pi) = a_i\}$;
- an integer variable z_i is associated with every anchor point $a_i \in A$, and its domain is the set of all paths starting from a_i, *i.e.*, $D(z_i) = \{\pi \in P : o(\pi) = a_i\}$;
- an integer variable y represents the maximal length of a selected path.

The constraints are:

- for each pair of anchor points $\{a_i, a_j\} \subseteq A$, we post a table constraint $(z_i, z_j) \in T_{ij}$ where T_{ij} is the table that contains every couple $(\pi, \pi') \in D(z_i) \times D(z'_i)$ such that $d(\pi) \neq d(\pi')$ and path π does not cross path π';
- for each anchor point $a_i \in A$, we post the constraint $y \geq l(z_i)$;
- we channel x_i and z_i variables by posting $x_i = d(z_i)$ and we post an *allDifferent*$(\{x_i : a_i \in A\})$ constraint. This constraint is redundant as table constraints prevent selecting two paths that have a same destination. However, preliminary experiments have shown us that this improves the solution process for a wide majority of instances.

The goal is to minimise y.

5.4 Experimental evaluation

Like in the case where there is no obstacle, we consider a bounding polygon $B = [0, 200]^2$. We introduce a parameter m to set the number of obstacles. For each obstacle, we randomly generate the coordinates of its lower left corner $(x, y) \in [0, 160]^2$ and the coordinates of its upper right corner (x', y') such that $x + 1 \leq x' \leq x + 40$ and $y + 1 \leq y' \leq y + 40$, while

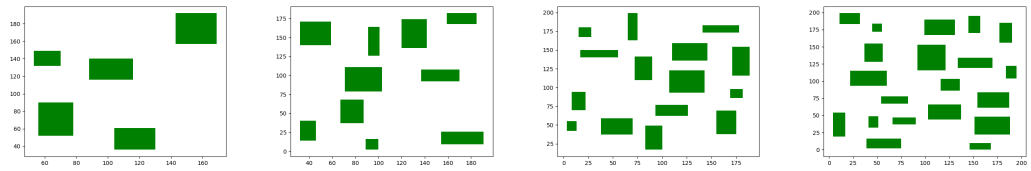

Figure 5 Workspace when $m \in \{5, 10, 15, 20\}$ (obstacles are displayed in green).

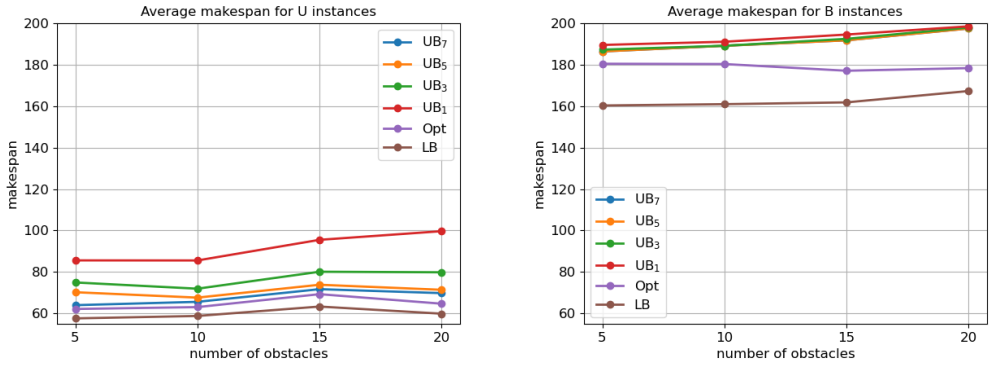

Figure 6 Evolution of the optimal makespan (Opt), the lower bound (LB) and upper bounds (UB, with $i \in \{1, 3, 5, 7\}$) when increasing the number of obstacles from 5 to 20. Left: U instances (with $n = 40$). Right: B instances (with $n = 20$).

ensuring that the distance between two obstacles is larger than 10. We consider 4 maps with $m = 5, 10, 15, 20$ which are displayed in Fig. 5.

We consider two different kinds of distributions for generating anchor points and destinations, in order to study the impact of this distribution on solution hardness:

Uniform (U): anchor points and destinations are randomly generated in W according to a uniform distribution;

Bipartite (B): anchor points (resp. destinations) are randomly generated on the left (resp. right) part of W, by constraining their abscissa to be smaller than 60 (resp. greater than 140).

For U instances, we set the number of robots n to 40, whereas for B instances it is set to 20 because these instances are harder, as explained later. For each value of m and each kind of distribution, we have generated 30 instances.

In Fig. 6, we display the optimal makespan, the lower bound computed by LB, and upper bounds computed by UB_i with $i \in \{1, 3, 5, 7\}$, for U and B instances. In both cases, we observe that the number of obstacles has no significant effect on the optimal makespan. However, the optimal makespan is much smaller for U instances than for B instances: For U instances, it is smaller than 80 whereas for B instances it is close to 180. This was expected as anchor points are constrained to be far from destinations in B instances.

For U instances, UB_1 is much larger than UB_3 which is always larger than UB_5. UB_5 and UB_7 have close values, and UB_7 is also close to the optimal solution. Results are quite different for B instances, where UB_1 and UB_7 have very close values. In other words, VNS does not improve much the upper bound for B instances, whatever the value of k_{max}. However, the optimal solution is much smaller than the upper bounds computed by UB_i. This means that for B instances we more often need to use non shortest paths to improve

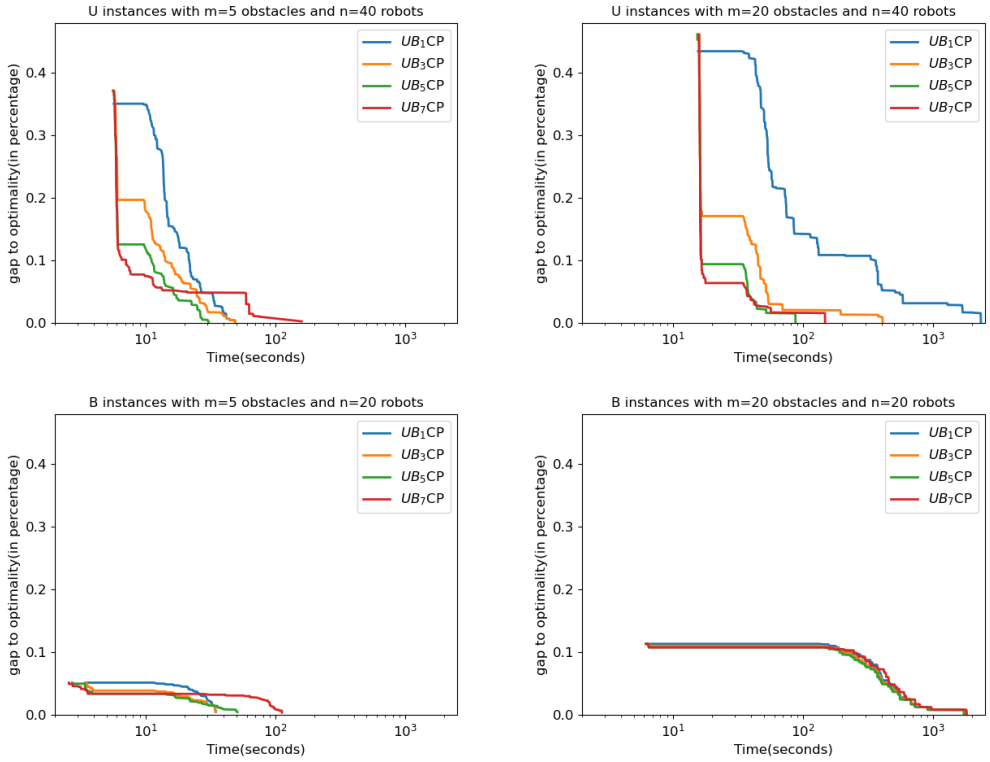

Figure 7 Evolution of the gap to optimality (in percentage) with respect to time for UB_iCP with $i \in \{1, 3, 5, 7\}$, on average for 30 instances. Top left: U instances with $m = 5$. Top right: U instances with $m = 20$. Bottom left: B instances with $m = 5$. Bottom right: B instances with $m = 20$.

the solution than for U instances (remember that VNS only considers shortest paths).

In Fig. 7, we display the evolution of the gap to optimality (in percentage) with respect to time, and in Tables 2 and 3 we display the time spent by each step of the solving process.

For U instances, LSAP is rather long to solve (see row t_1 in the tables): around 3s when $m = 5$, and 13s when $m = 20$. This comes from the fact that the function that decides whether two paths are crossing or not has a quadratic time complexity with respect to the number of vertices in the paths, and this number increases when increasing the number of obstacles. UB_3CP, UB_5CP, and UB_7CP improve the upper bound computed by LSAP with VNS, and we observe a quick drop of the curves. Then, we observe an horizontal part which corresponds to the time needed to enumerate all relevant paths and to generate the CP model. The time needed to enumerate all paths (t_3) strongly increases when increasing the number of obstacles. This was expected as the number of paths grows with respect to the number of obstacles. t_3 slightly decreases when increasing k_{max} because the smaller the bound computed with VNS, the less relevant paths (see row RP). The time needed to generate the CP model (t_4) decreases when increasing k_{max} (because this decreases the number of relevant paths) and it increases when increasing m (because this increases the number of vertices in a path and, therefore, the time needed to decide whether two paths are crossing). Finally, after the horizontal part (corresponding to t_3 and t_4), the curves drop again because CP improves the bound. As expected, the time needed by CP to compute the optimal solution (t_5) decreases when increasing k_{max} (because the initial bound is smaller, and therefore tables are smaller), and it increases when increasing the number of obstacles

Table 2 Results of UB_iCP with $i \in \{1, 3, 5, 7\}$ for U instances with $n = 40$ and $m \in \{5, 10, 15, 20\}$ (average on 30 instances). t_1 = time to solve the LSAP; t_2 = time of VNS when $k_{max} = i$; t_3 = time to enumerate all relevant paths for each anchor-destination pair; t_4 = time to generate the CP model; t_5 = time to solve the CP model; $t_{tot} = t_1 + t_2 + t_3 + t_4 + t_5$; IM = number of Improving Moves for VNS; RP = maximum number of Relevant Paths between an anchor point and a destination.

	UB_1CP				UB_3CP				UB_5CP				UB_7CP			
m	5	10	15	20	5	10	15	20	5	10	15	20	5	10	15	20
t_1	2.8	5.9	9.4	13.0	2.8	5.8	9.3	12.8	2.8	5.9	9.3	12.9	2.8	5.9	9.3	12.9
t_2	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.2	0.1	0.2	0.3	8.7	4.8	5.5	7.3
t_3	4.4	10.2	21.5	33.7	3.9	8.5	14.5	21.6	3.7	8.0	14.7	21.1	3.4	7.9	14.2	20.7
t_4	5.4	9.7	39.3	75.6	3.2	3.0	4.0	8.4	2.0	2.0	4.4	7.0	1.2	1.8	3.4	6.7
t_5	122.5	23.2	47.6	184.3	2.5	7.6	1.1	9.1	1.3	0.3	1.3	0.8	0.4	0.3	1.7	0.8
t_{tot}	135.1	49.0	117.8	306.5	12.3	24.9	29.1	51.8	10.0	16.3	30.0	42.0	16.6	20.6	34.2	48.3
IM	0	0	0	0	1.4	4.0	1.7	2.0	2.6	4.0	3.4	3.8	4.6	4.6	4.4	4.6
RP	2.5	2.8	3.9	4.7	2.2	2.4	3.1	3.0	2.0	2.2	2.8	2.7	1.9	2.6	2.6	2.5

Table 3 Results of UB_iCP with $i \in \{1, 3, 5, 7\}$ for B instances with $n = 20$ and $m \in \{5, 10, 15, 20\}$.

	UB_1CP				UB_3CP				UB_5CP				UB_7CP			
m	5	10	15	20	5	10	15	20	5	10	15	20	5	10	15	20
t_1	0.8	1.6	2.6	3.5	0.8	1.6	2.6	3.6	1.0	1.6	2.6	3.5	1.0	1.6	2.6	3.6
t_2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	0.5	0.5	0.6	45.7	41.7	37.6	50.6
t_3	3.5	12.2	42.0	96.2	3.4	11.8	40.2	95.5	4.2	12.0	40.4	93.6	4.3	12.0	40.5	93.6
t_4	15.6	32.4	94.6	339.6	13.8	27.6	81.2	329.0	16.4	28.3	79.1	315.9	16.7	28.2	78.9	317.6
t_5	0.4	0.7	1.6	5.6	0.4	0.6	1.3	5.3	0.5	0.6	1.3	5.0	0.4	0.6	1.3	5.1
t_{tot}	20.3	46.9	140.7	445.2	18.4	41.6	125.3	433.4	22.6	43.0	123.8	418.7	68.1	84.0	160.9	470.4
IM	0	0	0	0	0.3	0.2	0.3	0.2	0.4	0.3	0.3	0.2	0.4	0.3	0.3	0.2
RP	6.8	8.0	13.3	23.3	6.4	7.8	12.6	22.9	6.3	7.8	12.4	22.7	6.4	7.8	12.4	22.7

(because this increases the number of relevant paths).

Now, let us look at B instances. These instances only have $n = 20$ robots (instead of 40 for U instances) because they are harder. This comes from the fact that the bound computed by UB_i is much larger, as seen in Fig. 6. This increases the number of relevant paths, as seen when looking at row RP: when $m = 20$, this number is larger than 20 for B instances whereas it is smaller than 5 for U instances. Also the number of vertices in a path increases. Hence, the time needed to enumerate all relevant paths (t_3) is much larger for B instances than for U instances (*e.g.*, when $m = 20$ and $k_{max} = 7$, 94s for B and 21s for U). Also, the time needed to generate the CP model (t_4) is much larger (*e.g.*, when $m = 20$ and $k_{max} = 7$, 318s for B and 7s for U). However, the time spent by VNS (t_2) is much smaller (*e.g.*, when $m = 20$ and $k_{max} = 7$, 4s for B instead of 13s for U) because n is twice as small for B than for U. Finally, the time needed to solve the CP model increases when increasing m, but it does not decrease when increasing k_{max}. This comes from the fact that VNS does not improve much the upper bound, whatever the value of k_{max} (as seen in Fig. 6). Row IM displays the number of improving moves performed by VNS, and we observe that this number is close to 0 for B instances.

For both B and U instances, we observe a good compromise between the time spent by

Table 4 Impact of the parameter p on the time needed to enumerate relevant paths (t_3), to generate the CP model (t_4), and to solve it (t_5), and on the gap to optimality (in percentage) for B instances when $k_{max} = 5$ and $m = 20$.

	p=1	p=2	p=4	p=8	p=16	no limit
t_3	0.0	65.5	76.6	87.1	93.2	93.6
t_4	1.9	8.2	34.4	121.7	265.5	315.9
t_5	0.1	0.2	0.6	2.2	4.1	5.0
$t_{tot} = t_1 + t_2 + t_3 + t_4 + t_5$	6.9	78.0	115.8	215.1	367.9	418.7
gap to optimality	10.8%	5.9%	0.9%	0.0%	0.0%	0.0%

VNS to improve the bound, and the time spent to enumerate relevant paths, build the CP model and solve it when $k_{max} \in \{3, 5\}$.

As observed on row RP of Tables 2 and 3, the number of relevant paths being searched for each anchor/destination pair increases as m gets larger. Theoretically, this number exponentially grows with the number of obstacles. When the optimal makespan is small and the upper bound computed by VNS is close enough to it, the actual number of relevant paths is rather small (*e.g.*, smaller than 3 for U instances when $k_{max} \geq 5$). However, for B instances, this number is greater than 20 when $m = 20$, and the time needed to enumerate these paths and generate the CP model becomes greater than 400s. To overcome this problem, we can introduce a parameter p and limit the number of relevant paths to p (keeping the p best ones whenever the number of relevant paths is greater than p). Of course, in this case we no longer guarantee optimality as it may happen that the optimal solution uses a path that has been discarded. In table 4 we display the results of UB_5CP for different values of p on B instances when $m = 20$. Not surprisingly t_2, t_3, t_4 are all reduced as p decreases, while the average gap to optimality increases up to more than 10% for $p = 1$. In our experiment, $p = 8$ ensures that an optimal solution can always be found, and divides by 2 the total time.

6 Conclusion

We have introduced a new MAPF problem which is motivated by an industrial application where tethered robots cannot cross cables. We have shown that we can compute feasible solutions that provide upper bounds in polynomial time, by solving LSAPs, even when the workspace has obstacles. We have also introduced a VNS approach that improves the feasible solution of LSAP by iteratively permuting k destinations, and a CP model that solves the problem to optimality. Finally, we have proposed to sequentially combine VNS and CP, thus allowing us to use the upper bound computed by VNS to filter domains.

Experimental results on randomly generated instances have shown us that the number of obstacles has a strong impact on the solving time. When there is no obstacle, there is exactly one path between every origin/destination pair of points, and this path is a straight line segment. When increasing the number of obstacles, the number of paths between two points grows exponentially, even when limiting our attention to taut paths. Hence, it is important to have good upper bounds on the optimal solution in order to reduce the number of candidate paths. Also, when increasing the number of obstacles, the number of vertices in a path increases linearly, and this has an impact on the time needed to decide whether two paths are crossing or not.

We have reported experiments on randomly generated instances that allow us to control the number of obstacles and the number of robots. We have considered two models for

generating anchor and destination points, and we have observed that the distribution of the points has a strong influence on the solution process. In particular, when anchor points and destinations are constrained to belong to two opposite sides of the workspace, this increases the hardness of the problem because this increases the makespan and, therefore, the number of relevant paths and the number of vertices in a path. We have introduced a parameter to control the number of paths and the solving time, at the price of the loss of optimality.

For future work, we plan to investigate other solving approaches, such as Tabu search or Integer Linear Programming. Also, we want to extend the work to non-point agents by considering robots with a body, generating complementary constraints on their motions and their cables. This will allow to deal with industrial and robotics applications.

References

1 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd ed. edition, 2008.

2 S. Bhattacharya, M. Likhachev, and V. Kumar. Topological constraints in search-based robot path planning. *Autonomous Robots*, 33(3):273–290, 2012.

3 Rainer E. Burkard and Eranda Çela. Linear assignment problems and extensions. *Handbook of Combinatorial Optimization*, pages 75–149, 1999.

4 J. Carlsson, B. Armbruster, Saladi Rahul, and Haritha Bellam. A bottleneck matching problem with edge-crossing constraints. *Int. J. Comput. Geom. Appl.*, 25:245–262, 2015.

5 Geoffrey Chu and Peter J. Stuckey. Chuffed solver description, 2014. Available at `http://www.minizinc.org/challenge2014/description_chuffed.txt`.

6 Susan Hert and Vladimir J. Lumelsky. The ties that bind: Motion planning for multiple tethered robots. *Robotics Auton. Syst.*, 17(3):187–215, 1996.

7 H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

8 Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

9 Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Multi-agent path finding for large agents. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7627–7634, Jul. 2019. `doi:10.1609/aaai.v33i01.33017627`.

10 Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.

11 Nenad Mladenovic and Pierre Hansen. Variable neighborhood search. *Comput. Oper. Res.*, 24(11):1097–1100, 1997.

12 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming - CP 2007*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.

13 David W. Pentico. Assignment problems: A golden anniversary survey. *Eur. J. Oper. Res.*, 176(2):774–793, 2007.

14 Putnam. Problem a4. 1979.

15 Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015. `doi:https://doi.org/10.1016/j.artint.2014.11.006`.

16 S. Thomas, Dipti Deodhare, and M. N. Murty. Extended conflict-based search for the convoy movement problem. *IEEE Intelligent Systems*, 30:60–70, 2015.

17 Thayne T. Walker, Nathan R. Sturtevant, and Ariel Felner. Extended increasing cost tree search for non-unit cost domains. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 534–540. AAAI Press, 2018.

18 J. Yu and S. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1444–1449, 2013.

19 Xu Zhang and Quang-Cuong Pham. Planning coordinated motions for tethered planar mobile robots. *Robotics and Autonomous Systems*, 118:189–203, 2019. doi:https://doi.org/10.1016/j.robot.2019.05.008.