



HAL
open science

Solving the Non-Crossing MAPF with CP

Xiao Peng, Christine Solnon, Olivier Simonin

► **To cite this version:**

Xiao Peng, Christine Solnon, Olivier Simonin. Solving the Non-Crossing MAPF with CP. CP 2021 - 27th International Conference on Principles and Practice of Constraint Programming, Oct 2021, Montpellier (online), France. pp.1-17, <10.4230/LIPIcs.CP.2021.20>. <hal-03320987>

HAL Id: hal-03320987

<https://hal.science/hal-03320987v1>

Submitted on 16 Aug 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

1 Solving the Non-Crossing MAPF with CP

2 **Xiao Peng**

3 CITI, INRIA, INSA Lyon, F-69621, Villeurbanne, France

4 **Christine Solnon**

5 CITI, INRIA, INSA Lyon, F-69621, Villeurbanne, France

6 **Olivier Simonin**

7 CITI, INRIA, INSA Lyon, F-69621, Villeurbanne, France

8 — Abstract —

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

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

19 **2012 ACM Subject Classification** Computing methodologies

20 **Keywords and phrases** Constraint Programming (CP), Multi-Agent Path Finding (MAPF), Assign-
21 ment Problems

22 **Digital Object Identifier** 10.4230/LIPIcs.CP.2021.20

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

26 **1** Introduction

27 Multi-agent path finding (MAPF) is a very active research topic which has important appli-
28 cations for robotics in industrial contexts (*e.g.*, transport in fulfillment centers, autonomous
29 tug robots). In this paper we consider an extension of MAPF for tethered robots, *i.e.*, robots
30 attached with flexible cables to anchor points, allowing them to have continuous access to
31 fluids such as energy or water, for example. This is the case for our industrial partner in a
32 European project¹ where a fleet of mobile robots is used for inspecting and cleaning large
33 structures. Each robot has a cable which is kept taut between its anchor point and its
34 current position by a system that pulls on the cable when the robot moves back. The main
35 difficulty with these tethered robots comes from the fact that robots are not able to cross
36 cables. Hence, this paper introduces the Non-Crossing MAPF (NC-MAPF) problem which
37 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



38 Related work

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

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

50 MAPF problems are usually solved by using *Conflict Based Search (CBS)* approaches
 51 [15] which are two-level approaches: at the low level, paths are searched (while satisfying
 52 constraints added at the high level); and at the high level, path conflicts are resolved. CBS
 53 has been extended to agents with a specific geometric shape and volume (*e.g.*, [9, 17]) and to
 54 convoys (agents that occupy a sequence of nodes and their connecting edges) [16]. These
 55 MAPF variants share some similarities with NC-MAPF as a tethered robot may be viewed
 56 as a robot which has a very long body corresponding to its cable.

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

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

75 Contributions and outline of the paper

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

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

84 compute the optimal solution.

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

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

92 2 Definition of the workspace and notations

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

100 Given two points $u, v \in W$, we denote \overline{uv} the straight line segment that joins u to v , and
 101 $|uv|$ the Euclidean distance between u and v (i.e., $|uv|$ is the length of \overline{uv}). We say that a
 102 segment crosses an obstacle if $\overline{uv} \not\subset W$. Given two segments \overline{uv} and $\overline{u'v'}$, we say that they
 103 are incident if they have one common endpoint (i.e., $|\{u, v\} \cap \{u', v'\}| = 1$), and we say that
 104 they cross if they share one point (called the crossing point) which is not an endpoint (i.e.,
 105 $\{u, v\} \cap \{u', v'\} = \emptyset$ and $\overline{uv} \cap \overline{u'v'} \neq \emptyset$).

106 A chain of incident segments $\overline{u_0u_1}, \overline{u_1u_2}, \dots, \overline{u_{i-1}u_i}$ is represented by the sequence $\pi =$
 107 $\langle u_0, u_1, u_2, \dots, u_i \rangle$. The length of this chain of segments is denoted $|\pi|$ and is the sum of the
 108 lengths of its segments, i.e., $|\pi| = \sum_{j=1}^i |u_{j-1}u_j|$.

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

110 3 Definition of the NC-MAPF Problem

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

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

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

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

127 correspond to segments that do not cross obstacles, i.e., $E = \{(u, v) \in (A \cup V_O) \times (D \cup V_O) \mid \overline{uv} \subset$
 128 $W\}$. The graph is directed because edges from destinations to anchor points are forbidden.

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

134 A path in the visibility graph G is a sequence of vertices $\langle u_0, \dots, u_i \rangle$ such that $(u_{j-1}, u_j) \in$
 135 $E, \forall j \in [1, i]$. This path also corresponds to a chain of segments and its length is the sum of
 136 the lengths of its segments. We only consider elementary paths, i.e., a vertex cannot occur
 137 more than once in a path. Indeed, if a path is not elementary, then it can be replaced by a
 138 shorter elementary path obtained by removing its cycles.

139 Two paths are homotopic if there exists a continuous deformation between them without
 140 crossing obstacles [2], and a taut path is the shortest path of a homotopy class. For example,
 141 in the workspace of Fig. 1, all paths starting from the anchor point a_1 (point 1 in blue),
 142 passing between O_1 and O_2 and then between O_1 and O_3 , and finally reaching the destination
 143 d_1 (point 1 in red) are homotopic. Let x be the bottom-left vertex of O_1 , y its bottom-
 144 right vertex, z the top-left vertex of O_3 , and t the top-right vertex of O_2 . The paths
 145 $\pi = \langle a_1, x, y, z, d_1 \rangle$ and $\pi' = \langle a_1, x, t, z, d_1 \rangle$ are homotopic. π is taut because it is the shortest
 146 path of its homotopy class. π' is not taut because it is longer than π .

147 We say that a path is self-crossing if it contains two crossing segments. We say that two
 148 paths π and π' are crossing either if π contains a segment that crosses a segment of π' , or if
 149 π contains two incident segments \overline{uv} and \overline{vw} and π' contains two incident segments $\overline{u'v'}$ and
 150 $\overline{v'w'}$ such that $v = v'$ and \overline{uv} crosses $\overline{u'w'}$. However, two non crossing paths may share some
 151 vertices or some segments, as illustrated in Fig. 1(c). Indeed, as robots are small and cables
 152 are thin, a robot can slightly push the cable of another robot without crossing its cable. For
 153 example, if the black robot (starting from 3) in Fig. 1(c) arrives on the vertex of obstacle O_4
 154 before the blue robot (starting from 4) then, when the blue robot arrives on this vertex, it
 155 can slightly push the black cable to continue its path between O_4 and the black cable.

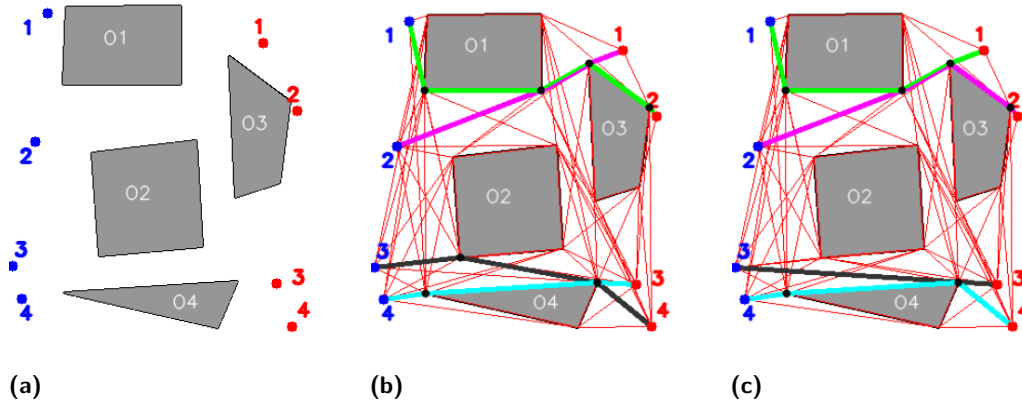
156 Let us now formally define our problem.

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

163 **4 NC-MAPF problem without obstacles**

164 In this section, we consider the case where the set O of obstacles is empty. In this case,
 165 $V_O = \emptyset$ and the visibility graph G is the complete bipartite graph such that $V = A \cup D$ and
 166 $E = A \times D$ (every edge of E is included in W as the bounding polygon is convex).

167 In Section 4.1, we show how to compute lower and upper bounds by solving well known
 168 assignment problems. In Section 4.2, we show how to improve the upper bound by performing
 169 variable neighbourhood search. In Section 4.3, we introduce a CP model and, in Section 4.4,
 170 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.

171 4.1 Computation of bounds by solving assignment problems

172 An assignment problem aims at finding a one-to-one matching between tasks and agents [3, 13].
 173 In our context, tasks correspond to destinations and agents to robots, and a matching is a
 174 bijection $m : A \rightarrow D$. We say that an edge (a, d) of the visibility graph G is selected whenever
 175 $m(a) = d$. The NC-MAPF problem without obstacles is a special case of assignment problem:
 176 ■ there is an additional constraint that ensures that no two selected edges cross, *i.e.*,
 177 $\forall \{a_i, a_j\} \subseteq A, \overline{a_i m(a_i)} \cap \overline{a_j m(a_j)} = \emptyset$;
 178 ■ there is an objective function that aims at minimising the maximal cost of a selected
 179 edge, *i.e.*, $\max_{a_i \in A} |a_i m(a_i)|$.

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

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

194 4.2 Variable Neighbourhood Search

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

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

212 4.3 Constraint Programming Model

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

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

- 220 ■ an integer variable x_i is associated with every anchor point $a_i \in A$, and the domain
 221 of this variable contains every destination that is within a distance of ub from a_i , *i.e.*,
 222 $D(x_i) = \{d \in D : |a_i d| < ub\}$;
- 223 ■ an integer variable y represents the maximal length of a selected edge.

224 The constraints are:

- 225 ■ 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
 226 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
 227 the segment $\overline{a_i d}$ does not cross the segment $\overline{a_j d'}$;
- 228 ■ for each anchor point $a_i \in A$, we post the constraint $y \geq |a_i x_i|$;
- 229 ■ we post an *allDifferent* $(\{x_i : a_i \in A\})$ constraint. This constraint is redundant as
 230 table constraints prevent assigning a same value to two different x_i variables. However,
 231 preliminary experiments have shown us that this improves the solution process for a wide
 232 majority of instances.

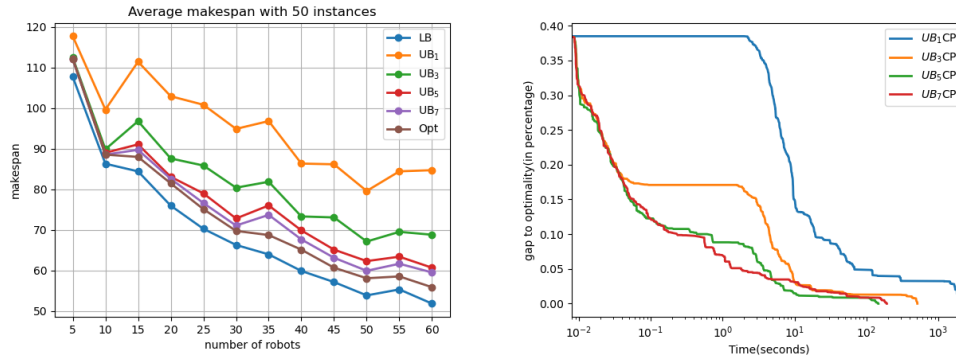
233 The goal is to minimise y .

234 4.4 Experimental evaluation

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

240 We consider the following approaches:

- 241 ■ 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_{*i*}CP with $i \in \{1, 3, 5, 7\}$, on average for the 50 instances with $n = 50$ robots.

- 242 ■ UB_{*i*} with $i \in \{1, 3, 5, 7\}$ refers to the computation of an upper bound by first solving an
 243 LSAP (see Section 4.1) and then improving it by VNS with $l = 60$ seconds and $k_{max} = i$
 244 (see Section 4.2). Note that when $i = 1$, VNS is immediately stopped as k is initialised to
 245 2 and the search is stopped when k becomes greater than k_{max} .
- 246 ■ UB_{*i*}CP refers to the sequential combination of UB_{*i*}, for computing an upper bound ub ,
 247 and CP (with the model described in Section 4.3) for computing the optimal solution.
- 248 LB and UB_{*i*} are implemented in Python. The CP model is implemented in MiniZinc [12]
 249 and solved with Chuffed [5]. All experiments are run on an Intel Core Intel Xeon E5-2623v3
 250 of 3.0GHz×16 with 32GB of RAM.

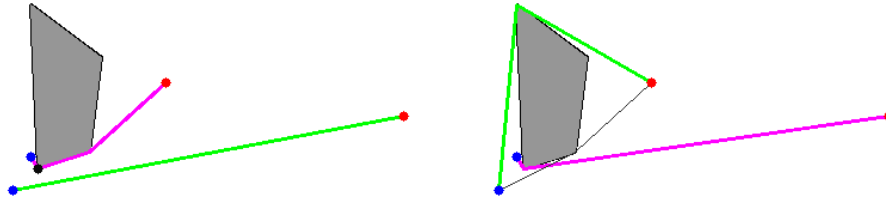
251 On the left part of Fig. 2, we compare the optimal makespan with the lower bound
 252 computed by LB, and upper bounds computed by UB_{*i*} with $i \in \{1, 3, 5, 7\}$. We observe that
 253 the optimal makespan decreases as the number n of robots increases. Indeed, when n gets
 254 larger, anchor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

257 UB₁ corresponds to the solution of the LSAP, and this upper bound is much larger than
 258 the optimal makespan. VNS strongly decreases this upper bound, and the larger k_{max} the
 259 smaller the bound. Note that when $k_{max} \geq n$, VNS actually finds the optimal makespan as
 260 it explores all possible permutations of the n destinations (provided that we do not limit
 261 time, *i.e.*, $l = \infty$). Hence, when $n = 5$, the solution of UB₅ is equal to the optimal makespan.

262 However, if UB_{*i*} finds smaller bounds when increasing i , it also needs more time. This
 263 is shown on the right part of Fig. 2, for instances that have $n = 50$ robots. We display the
 264 evolution of the average gap to optimality in percentage (*i.e.*, $\frac{s-s^*}{s^*}$ where s^* is the optimal
 265 makespan and s is the current makespan) with respect to CPU time. For UB₁CP, the upper
 266 bound ub is very quickly computed by solving the LSAP, but it is 38% as large as the optimal
 267 makespan. ub is used to filter variable domains of x_i variables. However, as ub is not very
 268 tight, the construction of the table T_{ij} for every couple of variables (x_i, x_j) is time consuming.
 269 This construction phase corresponds to the horizontal part of the curve. Once the CP model
 270 has been constructed, Chuffed finds better solutions and finally proves optimality. When
 271 increasing k_{max} , the time spent by VNS to improve ub increases but, as a counterpart, the
 272 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 $n \in \{20, \dots, 60\}$, we report CPU times of UB_iCP (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



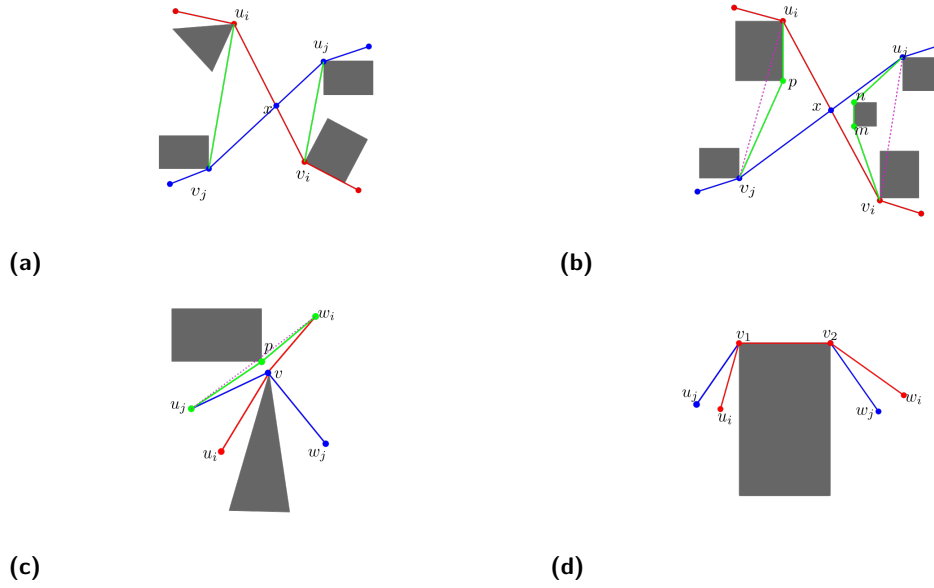
■ **Figure 3** The solution displayed on the left only uses shortest paths, and its makespan is larger than the solution displayed on the right (the green right path is longer than the black path).

273 Table 1 allows us to study scale-up properties when increasing the number n of robots.
 274 The time spent by UB_i (t_1) strongly increases when i increases: from 0.008s when $i = 1$ to
 275 more than 16s when $i = 7$ for $n = 60$. This was expected as the time complexity of VNS
 276 is exponential with respect to k_{max} . The time limit $l = 60s$ is never reached by VNS when
 277 $i \leq 5$ whereas it is reached when $i = 7$: for 7 (resp. 1 and 1) instances when $n = 60$ (resp.
 278 50 and 40). However, when increasing i , UB_i computes better bounds and this reduces the
 279 time needed to generate the model (t_2) and to solve it (t_3). When $i = 1$, the time limit of
 280 3600s is reached by Chuffed for 6 (resp. 1 and 1) instances when $n = 60$ (resp. 50 and 30).
 281 It is also reached once when $i = 3$ and $n = 60$. A good compromise is observed with UB_5CP .

282 5 NC-MAPF problem with obstacles

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

290 The number of paths between two points grows exponentially with respect to the number
 291 of obstacles. However, if we have an upper bound on the maximal length of a path, we can
 292 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 $\overline{u_i v_i}$ and $\overline{u_j v_j}$. (a): $\overline{u_i v_j}$ and $\overline{u_j v_i}$ (in green) do not cross obstacles and $|u_i v_j| + |u_j v_i| < [u_i v_i] + |u_j v_j|$. (b): $\overline{u_i v_j}$ and $\overline{u_j v_i}$ (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 $|\pi_{ij}| + |\pi_{ji}| < |u_i v_i| + |u_j v_j|$. 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 ($|\langle u_j, p, w_i \rangle| < |u_j, v, w_i|$). (d): By swapping w_i and w_j we obtain non crossing paths that have the same length.

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

295 5.1 Computation of bounds

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

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

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

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

308 **Case 1:** π_i and π_j contain two crossing segments $\overline{u_i v_i}$ and $\overline{u_j v_j}$. Let us show that this implies
 309 that m does not minimise the sum of the selected edge costs. There are two sub-cases to
 310 consider.

311 **Subcase a:** $\overline{u_i v_j}$ and $\overline{u_j v_i}$ do not cross obstacles, as illustrated in Fig. 4a.

312 Let π_i^p (resp. π_i^s) be the prefix (resp. suffix) of π_i that precedes (resp. succeeds)
 313 $\overline{u_i v_i}$, *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
 314 $\pi_j = \pi_j^p \cdot \langle u_j, v_j \rangle \cdot \pi_j^s$. Let x be the crossing point between $\overline{u_i v_i}$ and $\overline{u_j v_j}$. We have:

$$315 \quad |u_i v_i| = |u_i x| + |x v_i| \text{ and } |u_j v_j| = |u_j x| + |x v_j|. \quad (1)$$

316 The triangle inequality implies that

$$317 \quad |u_i v_j| < |u_i x| + |x v_j| \text{ and } |u_j v_i| < |u_j x| + |x v_i|. \quad (2)$$

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

$$319 \quad |u_i v_j| + |u_j v_i| < |u_i v_i| + |u_j v_j|. \quad (3)$$

320 When swapping v_i and v_j , π_i and π_j are replaced by the two paths $\pi'_i = \pi_i^p \cdot \langle u_i, v_j \rangle \cdot \pi_i^s$
 321 and $\pi'_j = \pi_j^p \cdot \langle u_j, v_i \rangle \cdot \pi_j^s$. From Eq. (3), we have $|\pi'_i| + |\pi'_j| < |\pi_i| + |\pi_j|$. This is in
 322 contradiction with the fact that m minimises the sum of the costs of the selected edges
 323 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
 324 to $|\pi'_i|$ and $|\pi'_j|$, respectively (they may be strictly smaller if π'_i or π'_j are not shortest
 325 paths in G).

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

327 In this case, we cannot simply exchange the two crossing segments to obtain two non
 328 crossing paths. However, let π_{ij} be the path from u_i to v_j corresponding to the convex
 329 hull of all vertices that belong to the triangle defined by u_i , v_j and x . This path is
 330 displayed in green in Fig. 4b. We can show that $|\pi_{ij}| < |u_i x| + |x v_j|$ by recursively
 331 exploiting the triangle inequality (see [1]). Similarly, there exists a path π_{ji} between
 332 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
 333 in Subcase a, this is in contradiction with the fact that m minimises the sum of the
 334 costs of the selected edges in G' .

335 **Case 2:** π_i and π_j do not contain crossing segments but they cross at some vertex v . Let π be
 336 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 $\pi_j = \pi_j^p \cdot \pi \cdot \pi_j^s$.
 337 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 $\pi'_j = \pi_j^p \cdot \pi \cdot \pi_i^s$.
 338 There are two sub-cases to consider.

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

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

348 ◀

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

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

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

358 5.2 Relevant paths enumeration

359 The non crossing assignment in G' that minimises the makespan may not be the optimal
 360 solution of the original problem as edges of G' correspond to shortest paths, and as the
 361 optimal solution may use non shortest paths. To find the optimal solution, for each couple
 362 $(a, d) \in A \times D$, we must consider all relevant paths from a to d in the visibility graph G ,
 363 where a path π is relevant if it satisfies the three following constraints:

- 364 **(C1)** Given an upper bound ub on the optimal makespan (or on the maximal length of the
 365 cable anchored at a), π must be shorter than ub , *i.e.*, $|\pi| < ub$;
- 366 **(C2)** π must be elementary and not self-crossing;
- 367 **(C3)** π must be a taut path (as defined in Section 3).

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

373 5.3 Constraint Programming Model

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

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

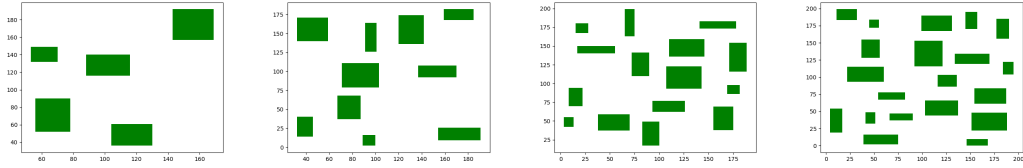
383 The constraints are:

- 384 ■ 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
 385 T_{ij} is the table that contains every couple $(\pi, \pi') \in D(z_i) \times D(z'_i)$ such that $d(\pi) \neq d(\pi')$
 386 and path π does not cross path π' ;
- 387 ■ for each anchor point $a_i \in A$, we post the constraint $y \geq l(z_i)$;
- 388 ■ we channel x_i and z_i variables by posting $x_i = d(z_i)$ and we post an *allDifferent* $(\{x_i : a_i \in A\})$
 389 constraint. This constraint is redundant as table constraints prevent selecting two paths
 390 that have a same destination. However, preliminary experiments have shown us that this
 391 improves the solution process for a wide majority of instances.

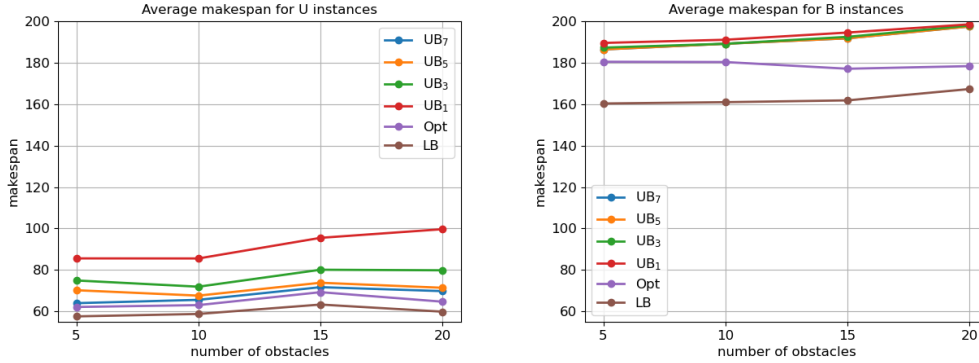
392 The goal is to minimise y .

393 5.4 Experimental evaluation

394 Like in the case where there is no obstacle, we consider a bounding polygon $B = [0, 200]^2$.
 395 We introduce a parameter m to set the number of obstacles. For each obstacle, we randomly
 396 generate the coordinates of its lower left corner $(x, y) \in [0, 160]^2$ and the coordinates of its
 397 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 $n = 20$).

398 ensuring that the distance between two obstacles is larger than 10. We consider 4 maps with
 399 $m = 5, 10, 15, 20$ which are displayed in Fig. 5.

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

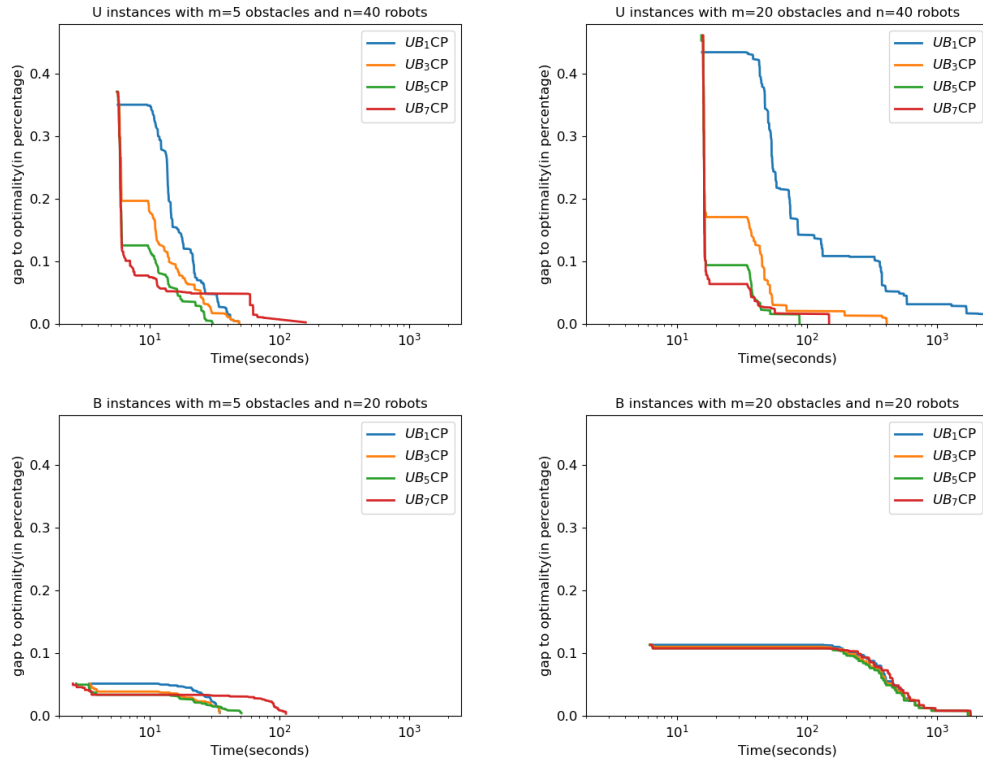
402 **Uniform (U)**: anchor points and destinations are randomly generated in W according to a
 403 uniform distribution;

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

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

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

416 For U instances, UB_1 is much larger than UB_3 which is always larger than UB_5 . UB_5
 417 and UB_7 have close values, and UB_7 is also close to the optimal solution. Results are
 418 quite different for B instances, where UB_1 and UB_7 have very close values. In other words,
 419 VNS does not improve much the upper bound for B instances, whatever the value of k_{max} .
 420 However, the optimal solution is much smaller than the upper bounds computed by UB_i .
 421 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 $m = 20$.

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

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

425 For U instances, LSAP is rather long to solve (see row t_1 in the tables): around 3s when
426 $m = 5$, and 13s when $m = 20$. This comes from the fact that the function that decides
427 whether two paths are crossing or not has a quadratic time complexity with respect to the
428 number of vertices in the paths, and this number increases when increasing the number
429 of obstacles. UB_3CP , UB_5CP , and UB_7CP improve the upper bound computed by LSAP
430 with VNS, and we observe a quick drop of the curves. Then, we observe an horizontal part
431 which corresponds to the time needed to enumerate all relevant paths and to generate the
432 CP model. The time needed to enumerate all paths (t_3) strongly increases when increasing
433 the number of obstacles. This was expected as the number of paths grows with respect to
434 the number of obstacles. t_3 slightly decreases when increasing k_{max} because the smaller
435 the bound computed with VNS, the less relevant paths (see row RP). The time needed
436 to generate the CP model (t_4) decreases when increasing k_{max} (because this decreases the
437 number of relevant paths) and it increases when increasing m (because this increases the
438 number of vertices in a path and, therefore, the time needed to decide whether two paths are
439 crossing). Finally, after the horizontal part (corresponding to t_3 and t_4), the curves drop
440 again because CP improves the bound. As expected, the time needed by CP to compute the
441 optimal solution (t_5) decreases when increasing k_{max} (because the initial bound is smaller,
442 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.

m	UB ₁ CP				UB ₃ CP				UB ₅ CP				UB ₇ CP			
	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 $m \in \{5, 10, 15, 20\}$.

m	UB ₁ CP				UB ₃ CP				UB ₅ CP				UB ₇ CP			
	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

443 (because this increases the number of relevant paths).

444 Now, let us look at B instances. These instances only have $n = 20$ robots (instead of 40
445 for U instances) because they are harder. This comes from the fact that the bound computed
446 by UB_i is much larger, as seen in Fig. 6. This increases the number of relevant paths, as seen
447 when looking at row RP: when $m = 20$, this number is larger than 20 for B instances whereas
448 it is smaller than 5 for U instances. Also the number of vertices in a path increases. Hence,
449 the time needed to enumerate all relevant paths (t_3) is much larger for B instances than for U
450 instances (*e.g.*, when $m = 20$ and $k_{max} = 7$, 94s for B and 21s for U). Also, the time needed
451 to generate the CP model (t_4) is much larger (*e.g.*, when $m = 20$ and $k_{max} = 7$, 318s for B
452 and 7s for U). However, the time spent by VNS (t_2) is much smaller (*e.g.*, when $m = 20$
453 and $k_{max} = 7$, 4s for B instead of 13s for U) because n is twice as small for B than for U.
454 Finally, the time needed to solve the CP model increases when increasing m , but it does
455 not decrease when increasing k_{max} . This comes from the fact that VNS does not improve
456 much the upper bound, whatever the value of k_{max} (as seen in Fig. 6). Row IM displays the
457 number of improving moves performed by VNS, and we observe that this number is close to
458 0 for B instances.

459 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%

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

462 As observed on row RP of Tables 2 and 3, the number of relevant paths being searched
463 for each anchor/destination pair increases as m gets larger. Theoretically, this number
464 exponentially grows with the number of obstacles. When the optimal makespan is small and
465 the upper bound computed by VNS is close enough to it, the actual number of relevant paths
466 is rather small (*e.g.*, smaller than 3 for U instances when $k_{max} \geq 5$). However, for B instances,
467 this number is greater than 20 when $m = 20$, and the time needed to enumerate these paths
468 and generate the CP model becomes greater than 400s. To overcome this problem, we can
469 introduce a parameter p and limit the number of relevant paths to p (keeping the p best
470 ones whenever the number of relevant paths is greater than p). Of course, in this case we no
471 longer guarantee optimality as it may happen that the optimal solution uses a path that
472 has been discarded. In table 4 we display the results of UB₅CP for different values of p on
473 B instances when $m = 20$. Not surprisingly t_2 , t_3 , t_4 are all reduced as p decreases, while
474 the average gap to optimality increases up to more than 10% for $p = 1$. In our experiment,
475 $p = 8$ ensures that an optimal solution can always be found, and divides by 2 the total time.

476 6 Conclusion

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

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

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

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

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

505 — References —

- 506 1 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational*
 507 *Geometry: Algorithms and Applications*. Springer-Verlag, 3rd ed. edition, 2008.
- 508 2 S. Bhattacharya, M. Likhachev, and V. Kumar. Topological constraints in search-based robot
 509 path planning. *Autonomous Robots*, 33(3):273–290, 2012.
- 510 3 Rainer E. Burkard and Eranda Çela. Linear assignment problems and extensions. *Handbook*
 511 *of Combinatorial Optimization*, pages 75–149, 1999.
- 512 4 J. Carlsson, B. Armbruster, Saladi Rahul, and Haritha Bellam. A bottleneck matching problem
 513 with edge-crossing constraints. *Int. J. Comput. Geom. Appl.*, 25:245–262, 2015.
- 514 5 Geoffrey Chu and Peter J. Stuckey. Chuffed solver description, 2014. Available at http://www.minizinc.org/challenge2014/description_chuffed.txt.
 515
- 516 6 Susan Hert and Vladimir J. Lumelsky. The ties that bind: Motion planning for multiple
 517 tethered robots. *Robotics Auton. Syst.*, 17(3):187–215, 1996.
- 518 7 H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics*
 519 *Quarterly*, 2(1-2):83–97, 1955.
- 520 8 Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.
- 521 9 Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, T. K. Satish Kumar, and Sven Koenig.
 522 Multi-agent path finding for large agents. *Proceedings of the AAAI Conference on Artificial*
 523 *Intelligence*, 33(01):7627–7634, Jul. 2019. doi:10.1609/aaai.v33i01.33017627.
- 524 10 Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths
 525 among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, 1979.
- 526 11 Nenad Mladenovic and Pierre Hansen. Variable neighborhood search. *Comput. Oper. Res.*,
 527 24(11):1097–1100, 1997.
- 528 12 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and
 529 Guido Tack. Minizinc: Towards a standard CP modelling language. In *Principles and Practice*
 530 *of Constraint Programming - CP 2007*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.
- 531 13 David W. Pentico. Assignment problems: A golden anniversary survey. *Eur. J. Oper. Res.*,
 532 176(2):774–793, 2007.
- 533 14 Putnam. Problem a4. 1979.
- 534 15 Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search
 535 for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015. doi:<https://doi.org/10.1016/j.artint.2014.11.006>.
 536
- 537 16 S. Thomas, Dipti Deodhare, and M. N. Murty. Extended conflict-based search for the convoy
 538 movement problem. *IEEE Intelligent Systems*, 30:60–70, 2015.
- 539 17 Thayne T. Walker, Nathan R. Sturtevant, and Ariel Felner. Extended increasing cost tree
 540 search for non-unit cost domains. In *Proceedings of the 27th International Joint Conference*
 541 *on Artificial Intelligence, IJCAI'18*, page 534–540. AAAI Press, 2018.
- 542 18 J. Yu and S. LaValle. Structure and intractability of optimal multi-robot path planning on
 543 graphs. In *In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages
 544 1444–1449, 2013.

- 545 19 Xu Zhang and Quang-Cuong Pham. Planning coordinated motions for tethered planar mobile
546 robots. *Robotics and Autonomous Systems*, 118:189–203, 2019. doi:[https://doi.org/10.](https://doi.org/10.1016/j.robot.2019.05.008)
547 [1016/j.robot.2019.05.008](https://doi.org/10.1016/j.robot.2019.05.008).