Solving the Non-Crossing MAPF for non point-sized robots

Xiao Peng[1]
Univ. Lyon, INSA Lyon, INRIA, CITI lab., F-69621, Villeurbanne, France

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

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

Abstract

This paper deals with the multi-agent path finding (MAPF) problem for a team of tethered robots. When considering point-sized robots, paths may share a same subpath provided that they do not cross, and we have shown in a previous work that this case basically involves solving an assignment problem: The objective is to find a set of non-crossing paths, and the makespan is equal to the length of the longest path. In this work, we extend it to the case of non-point-sized robots where robot paths must be synchronized when they share a same subpath and waiting times are considered when computing the makespan. We prove that the upper bound can be computed by solving the linear sum assignment problem. We introduce a new variable neighborhood search method to improve the upper bound and show that it is robust to different instances. We also introduce a Constraint Programming model for solving the problem to optimality.

2012 ACM Subject Classification Computing methodologies

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

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

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). The goal of MAPF is to find a set of paths from starting points to target points while minimizing the *makespan* (length of the longest path). In [6], we introduced an extension of MAPF for tethered robots, where robots are attached with flexible cables to anchor points (which are starting points), and the main difficulty comes from the fact that robots must not cross cables. We have shown that this problem is related to an Euclidean bipartite matching problem, and that we can compute feasible solutions that provide upper bounds in polynomial time, by solving the *Linear Sum Assignment Problem (LSAP)*. An approach based on the sequential combination of *Variable Neighborhood Search* (VNS) and *Constraint Programming* (CP) was introduced to solve the problem to optimality.

In [6], we considered point-sized robots, so that two paths may share a same subpath provided that they do not cross. This simplifying assumption on the physical form of the robots is far from reality. In this paper, we extend this work by studying the effect of robots' size on the makespan in a real-world application case. In this case, a robot cannot

[1] The corresponding author

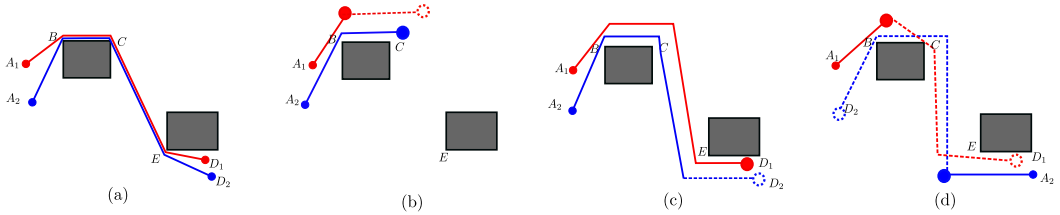

Figure 1 (a): A_i and D_i with $i = 1, 2$ represent the anchor points and the destinations. The dark grey rectangles are obstacles. In previous work, the makespan is defined as the maximum length between the red path π_1 and the blue path π_2. When robots' physical shape is considered, we impose a time delay dt when two robots share a same subpath. In this example, the blue path has higher priority at point B and C, while the red path is prioritized at E. To avoid getting crossed, the red robot should wait the blue one to pass B and C first, and the blue wait at E until the red robot has passed it, as showed in (b) and (c). Hence the *makespan* can be noted as $max\{|A_1B| + dt, |A_2B|\} + |BC| + |CE| + max\{|ED_2| + dt, |ED_1|\}$. We assume that the robots move at a constant velocity, therefore dt here in fact refers to a distance as $dt \times velocity$. (d): a deadlock occurs if we exchange the positions of A_2 and D_2: the red robot should wait the blue one to pass B first, while the blue one cannot pass E before the red robot.

travel between an obstacle and a cable if the distance between the cable and the obstacle is smaller than its size. As a consequence, we must ensure that the robot which is closer to the obstacle traverses the subpath first, and a minimal safety delay must be ensured with the other robot, as shown in Fig. 1.

Related work and Contributions

Although the classic MAPF problem has been widely studied [4, 8, 10, 11], the case of tethered robots has been less studied. For the motion planning of a single tethered robot, recent studies mainly focus on finding the shortest path given the initial and final configurations [7], and planning paths for coverage and exploration tasks [9]. When considering multiple tethered robots, the challenge is that cables can easily get tangled, and the constraint to avoid such crossing makes the planning harder. In the work of Hert and Lumelsky [1, 2, 3], a problem of motion planning for circular robots moving on polygonal planar surface is considered, where cables in the workspace can be pushed and bent by robots. The objective is to find a set of non-intersecting curves connecting a set of point pairs in the plane. The authors design efficient algorithms to detect if two paths are intersecting or not, from a purely geometric standpoint. Given a target configuration, robots are prioritized and scheduled to move sequentially in straight lines to avoid crossing. However, there is no guarantee that a non-cyclic ordering can be always found. In [12], the authors investigate the feasibility of a target configuration by different motion modes. They propose algorithms to detect deadlocks in a target configuration and iteratively remove them by restricting the motion of robots involved in a deadlock situation until a valid solution is found.

In this work, we also impose a priority relationship between robots, assuming that robots with lower priority must wait until the ones with higher priorities have passed. Compared to the point-sized case studied in [6], this changes the definition of *makespan*, as robot's motion should be synchronized with respect to their priority. As a first contribution, we study the topological relationship of two polygonal lines in a 2D Euclidean plane in order to decide whether two paths cross as well as their priority from a geometrical point of view. This problem was also investigated in [3, 12] and it is remarked that deadlocks might appear

if we adopt a sequential motion mode between robots (as illustrated on the right part of Fig. 1). We use a topological sort and post a *NoCircuit* constraint on the constructed graph to avoid such deadlocks. As a second contribution, we prove that the optimal solution of *LSAP* cannot contain deadlocks and thus provides an upper bound for the problem. As a third contribution, we introduce a VNS algorithm to improve the LSAP solution and a CP model to improve the VNS solution and prove optimality.

2 Preliminaries

In this section, we formally describe the non-crossing MAPF problem for non point-sized robots. Robots move on a 2 dimensional workspace $W \subset \mathbb{R}^2$ defined by a bounding polygon B and a set of convex obstacles O: 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. There are n robots such that each robot is attached with a flexible cable to an anchor point in W and we denote A the set of n anchor points. We denote D the set of n destinations. 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 which has a vertex for each point of W which is either a polygon vertex or a point in $A \cup D$, and an edge between every pair of vertices $\{i, j\}$ such that the straight line segment between i and j fully belongs to W [5].

We use geometrical properties to define priority relations between paths that share a same vertex. We do not detail this computation due to lack of space but simply illustrate it on the left part of Figure 1. The paths $\pi_1 = \langle A1, B, C, E, D1\rangle$ and $\pi_2 = \langle A_2, B, C, E, D1\rangle$ share vertices B, C, and E. For B and C, π_2 has a higher priority than π_1 because $\langle A_2, B, C, E\rangle$ is closer to the upper-left obstacle than $\langle A_1, B, C, E\rangle$. This is denoted $\pi_1(B) \prec \pi_2(B)$ and $\pi_1(C) \prec \pi_2(C)$. For E, π_1 has a higher priority than π_2 because $\langle C, E, D_1\rangle$ is closer to the lower-right obstacle than $\langle C, E, D_2\rangle$. This is denoted $\pi_2(E) \prec \pi_1(E)$.

Priority relations are used to define a priority graph.

▶ **Definition 1** (Priority graph). *The priority graph associated with a set of non-crossing paths $(\pi_1, .., \pi_n)$ is the directed graph $G = (V, E)$ such that vertices are the nodes labeled by the path to which they belong,* i.e., *$V = \{(w_i, \pi_j) | w_i \in \pi_j\}$ and edges correspond to the node pairs with known priority order,* i.e., *$E = \{(u, v) \in V \times V | u \prec v\}$. The graph is directed because edges are directed from nodes with lower priority to those with higher priority.*

This definition is similiar to the the Pair Interaction Graph in [12]. For two paths π_1 and π_2, there might be different priority order at different nodes, this is to say, there is not necessarily a consistent priority order between two paths. We can thus infer that given a target configuration composed by robots' paths $\{\pi_1, \pi_1, .., \pi_n\}$ mutually nonintersecting, there might be a deadlock for robots' motion. A deadlock exists if the priority graph contains cycles, as shown in Fig. 2.

The goal of NC-MAPF problem is to find a path for each robot, starting from its anchor point and reaching a different destination such that (i) paths do not cross each other, (ii) the associated priority graph does not contain cycles, and (iii) the arrival time of the last robot is minimized given that, when two robots R_i and R_j must pass through a same vertex v such that $R_i(v) \prec R_j(v)$, the arrival time of R_i on v must be larger than or equal to the arrival time of R_j on v plus δt (where δt is a fixed amount of time which depends on robots' size).

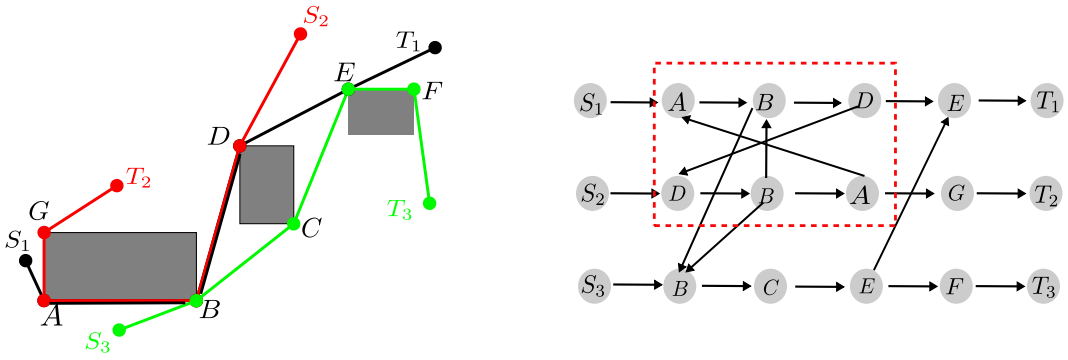

Figure 2 An example of deadlock situation. Left: R_1 should pass B after R_2, meanwhile, R_2 can't pass D until R_1 has passed it. Right, the corresponding directed graph for robots' motion priority order, for example, $A \to B$ means node A is traversed before node B; when two robots pass a common node, this priory order is deduced with the definition. $B(R_1) \to D(R_1) \to D(R_2) \to B(R_2) \to B(R_1)$ forms a loop and it implies that a deadlock exists when robots move.

3 Methodology

We solve our problem in three steps: we first compute an upper bound in polynomial time by solving an LSAP problem; then we improve this bound by VNS; and finally we compute the optimal solution with CP. We briefly describe these steps in this section, without detailing proofs or algorithms due to lack of space.

3.1 Computation of an initial upper bound

For each anchor/destination couple $(a, d) \in A \times D$, we compute the shortest path from a to d in the visibility graph, and we denote $c_{a,d}$ the length of this path.

▶ **Theorem 2.** *The matching $m : A \to D$ that minimizes $\sum_{a \in A} c_{a,m(a)}$ contains no deadlock.*

Indeed, given a set of paths, deadlocks can be always removed and the total cost decreases accordingly. Hence, the minimal matching cannot contain deadlocks. As we have shown in [6] that this minimal matching does not contain crossing paths, it provides an initial upper bound to our problem. It is computed in polynomial time by solving an LSAP problem, using the Hungarian algorithm.

3.2 Improving the upper bound with VNS

As proposed in [6], we can improve the matching that minimizes the sum of costs by performing a VNS: the neighborhood of a matching contains every non crossing matching obtained by permuting the destinations of k anchor points; k is initialized to 2 and it is incremented each time the current matching is locally optimal; k is reset to 2 each time an improving move has been found; the search is stopped when k exceeds a given upper bound k_{max} or when a time limit is reached. To adapt this VNS to non-point-sized robots, we simply have to modify the computation of the makespan to integrate waiting times in case of shared vertices.

In [6], we show that VNS is able to quickly improve the upper bound for some instances whereas it does not improve much the upper bound for some other instances, depending on the distribution of anchor and destination points. This comes from the fact that VNS only

considers shortest paths while it is necessary to consider non-shortest paths to improve the upper bound for some instances. For this reason, we extend our approach: once the search is stopped (because k_{max} or a first time limit is reached), we perform a second VNS where the neighborhood is enlarged by taking into account non-shortest paths. As it would be too long to compute all possible paths, we focus on anchor points which are situated in a circle with a radius r from the anchor of the longest path, and we search for every taut path between one of these anchor points and a destination point whose length is smaller than the current upper bound.

For the sake of clarity, we denote the basic VNS method as oldVNS, and the extended approach as newVNS.

3.3 Computation of the optimal solution with CP

As pointed out in [6], the optimal solution may use non-shortest paths. Given an upper bound ub, We denote P_{ub} the set of all relevant paths that may belong to the optimal solution (see [6] for more details on how to compute this set). For each path $\pi \in P_{ub}$, $o(\pi)$ and $d(\pi)$ denote the origin and the destination, respectively. Our CP model uses the same variables as the one introduced in [6]: for each anchor point $a_i \in A$, the integer variable x_i represents the destination associated with a_i and the integer variable z_i represents the path that links a_i to x_i. The initial domains of these variables are $D(x_i) = \{d(\pi) : \pi \in P_{ub} \wedge o(\pi) = a_i\}$ and $D(z_i) = \{\pi \in P : o(\pi) = a_i\}$. The integer variable m represents the makespan and its domain is $[0, ub]$. We introduce new variables for taking into account waiting times due to shared vertices: for each pair of anchor points $\{a_i, a_j\} \subseteq A$, the integer variable $y_{i,j}$ represents the maximum duration of the two paths z_i and z_j, including the waiting times due to shared vertices.

Like in [6], we channel x_i and z_i variables by posting $x_i = d(z_i)$ and we post a redundant $allDifferent(\{x_i : a_i \in A\})$ constraint.

z and y variables are related thanks to a table constraint. For each couple of anchor points $(a_i, a_j) \in A^2$, we pre-compute the table $T_{i,j}$ that contains every triple $(\pi_i, \pi_j, t) \in D(z_i) \times D(z_j) \times [0, ub]$ such that (i) $d(\pi_i) \neq d(\pi_j)$, (ii) path π_i does not cross path π_j, (iii) paths π_i and π_j do not have deadlocks, and (iv) t is the maximum duration of the two paths π_i and π_j, including waiting times due to shared vertices. For each pair of anchor points $\{a_i, a_j\} \subseteq A$, we post a table constraint $(z_i, z_j, y_{i,j}) \in T_{ij}$, and we post the constraint $m \geq y_{i,j}$.

The solution that minimises m is not necessarily the optimal solution of our problem. Indeed, table constraints ensure that (i) there is no crossing path; (ii) there is no deadlock between 2 paths; and (iii) m takes into account waiting times due to pairs of paths that have common vertices. However, deadlocks may be due to a circular dependency between more than 2 paths (as illustrated in Figure 2). Also, in the case of dependency chains of more than 2 paths (*e.g.*, when R_1 must wait for R_2 at some vertex, and R_2 must wait for R_3 at some other vertex), the makespan may be larger than $y_{1,2}$ and $y_{2,3}$. In Sections 3.3.1 and 3.3.2, we describe two different approaches to integrate these constraints.

3.3.1 A posteriori approach

In this approach, we enumerate all solutions of the model described above and, each time a solution is found, we proceed as follows:

- If this solution implies a deadlock (*i.e.*, the associated priority graph contains a cycle), we search for the minimum set of anchor points whose associated paths cause this deadlock,

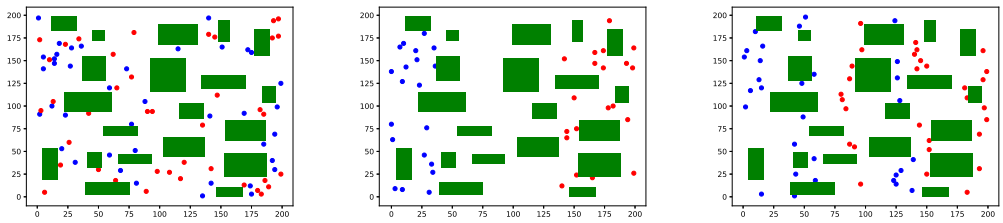

Figure 3 The illustration of different distribution for generating instances. Left: U-Instance, Middle: B-Instance, Right: A-Instance. For A-Instance, anchor points are randomly generated with their abscissas being equally constrained in $[0, 20] \cup [40, 60] \cup [120, 140]$, and destinations are equally distributed in $[80, 100] \cup [140, 160] \cup [180, 200]$.

and we add a nogood constraint to prevent the search from enumerating again the values of these anchor points;

- If there is no deadlock, we compute the makespan M of this solution;
  - If $M \geq ub$, we search for the minimum set of anchor points whose associated paths lead to this makespan, and we add a nogood constraint to prevent the search from enumerating again the values of these anchor points;
  - If $M < ub$, we set ub to M and post the constraint $m < ub$.

3.3.2 Global Constraint

In this approach, we introduce a global constraint $globalMakeSpan(\{z_i : a_i \in A\}, m)$ which ensures that the set of paths $\{z_i : a_i \in A\}$ does not contain any deadlock and that its associated makespan (*i.e.*, the largest arrival time when including waiting times) is smaller than or equal to m. This constraint is propagated each time a variable z_i is instantiated by (i) checking that z_i does not cross previously selected paths; (ii) checking that z_i does not create a cycle in the priority graph; and (iii) updating the lower bound of m if the makespan is increased due to z_i.

4 Experimental Results

We evaluate our algorithms on randomly generated instances. For all instances, the bounding polygon is the square $B = [0, 200]^2$. We introduce a parameter o to set the number of obstacles with $o \in \{5, 10, 15, 20\}$. To generate an instance with n robots, we randomly generate n anchor points and n destinations, and for each value of n, we generate 30 different instances. We consider three different kinds of distributions for generating anchor points and destinations, in order to study the impact of this distribution on solution hardness: for U instances, all points are uniformly distributed in the workspace; in B instances, anchor (resp. destination) points are uniformly distributed in the left (resp. right) part of the workspace; and in A instances, anchor and destination points are alternated (see Fig. 3 for an illustration). All experiments are run on an Intel Core Intel Xeon E5-2623v3 of 3.0GHz×16 with 32GB of RAM.

In Fig.4, we display the optimal makespan, the lower bound, and upper bounds computed by two different VNS methods as explained in section 3.2. In particular, the parameter of the neighbourhood's size (k_{max}) is set to 7 in the experiments, as a compromise between the efficiency and performance. The lower bound is computed by solving an LBAP instance in

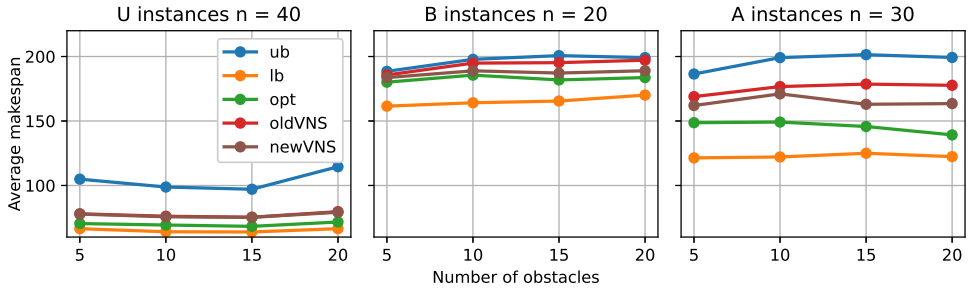

Figure 4 Evolution of the optimal makespan (Optimal), the lower bound (LB) and upper bounds (UB, with two different VNS methods) when increasing the number of obstacles from 5 to 20. From left to right are the U instance, B instance and A instance.

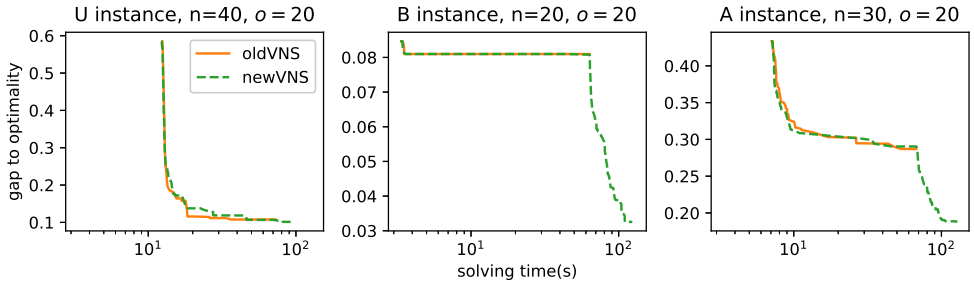

Figure 5 Evolution of gap to optimality for different VNS methods.

polynomial time, as shown in [6]. In all cases, we observe that the number of obstacles has no significant effect on the the optimal makespan. If comparing different type of instances, it can be learned the optimal makespan is the largest for the B instance, and then the A instance has an intermediate level of makespan and the U instance has the smallest. This was expected as the distance between anchor points and destinations points differs among when simulating these instances. Concerning the performance of two VNS methods on these instances. For U instance, we get almost the same upper bound with the two methods, while newVNS works better for B instance and A instance, namely, we more often need to use non shortest paths to improve the solution for such instances.

In Fig.5, we show the evolution of gap to optimality (in percentage) with repect to time. For U instance, we see that newVNS performs closely to oldVNS. As newVNS is oldVNS followed by a second stage search with an enlarged neighborhood, when oldVNS finishes, the follwed search doesn't continue to improve the bound much. For B instance, oldVNS is less efficient, and sometimes the time limit (60s) is reached, as we can clearly observe the curve of oldVNS for B instance staying constant after a little descent, which exactly overlaps the first stage of newVNS. We see that the bound is improved again when non-shortest paths are considered. This conclusion holds for the A instance, namely that newVNS has a clear advantage over oldVNS.

For each instance, we compute an upper bound by newVNS, and then resolve the CP problem to optimality respectively with the two methods. In Fig.6, we compare the time spent on the CP solving process by each method. There are a total of 360 instances to

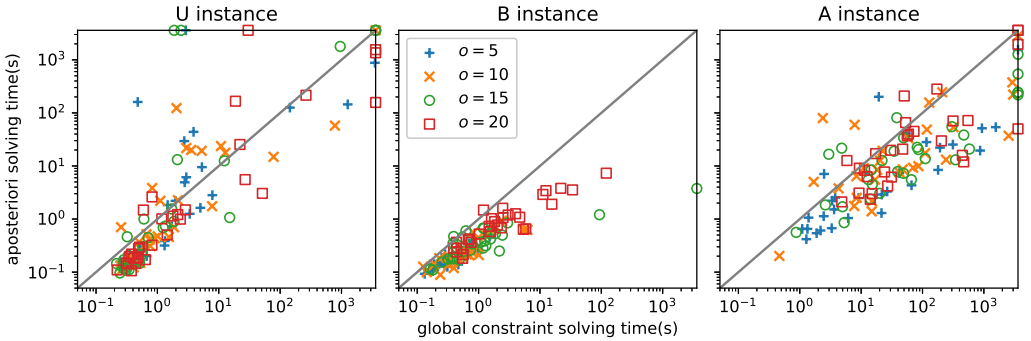

Figure 6 Compare the global constraint and the a posteriori approach for solving cp model. Each point (x, y) corresponds to an instance which is solved in x seconds by the global constraint and y seconds by the a posteriori approach. When an instance is not solved by the global constraint (resp. the a posteriori approach), it is displayed on $x = 3600$ (resp. $y = 3600$).

solve and 30 instances for each scenario($o = 5$, 10, 15, 20) and for each instance type. For U instance, When $t < 2s$, the a posteriori approach solves more instances than the global constraint. After 2s, the global constraint becomes more efficient. Under a time limit of 3600s, there are 112 instances solved by global constraint, and 113 instances by the a posteriori approach. For B instance, all the instances are solved within 3600s, however, the a posteriori approach performs more efficiently. As for A instance, the a posteriori approach can solve all the instances, while the global constraint solves 106 instances. We can come up with the conclusion that the a posteriori approach performs better than the global constraint.

5 Conclusion

In this work, we extended previous work on non-crossing MAPF by considering the impact of robots's size. Our findings show that a new definition of *makespan* should be introduced to cope with the real-world constraints. In these new settings, we follow the principe of combining the VNS method with CP model to resolve the problem to optimality on randomly generated instances. We introduced an improved VNS method that considers non-shortest paths as neighbors, which performs better than the basic VNS method for some type of instances. We proposed to sequentially combine them to improve the robustness in practice. We also provided two ways to solve the CP model optimally, and the result shows that the a posteriori approach generally outperforms the global constraint, while the latter may lead to a fast solution for some instances. For the future work, we envisage to exploit the strength of the two approaches and implement them as in parallel. In this problem, one of the main issues affecting the resolution effciency is related to the upper bound found to generate the CP model. The larger the upper bound, the greater the number of candidate paths and the heavier the CP model, which requires more time to solve optimally. To avoid this unnecessary computation, we plan to adopt a dichotomous method to implement the resolution procedure.

References

1 S. Hert and V. Lumelsky. The ties that bind: motion planning for multiple tethered robots. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 2734–2741 vol.4, 1994. doi:10.1109/ROBOT.1994.350923.

2 S. Hert and V. Lumelsky. Motion planning in r/sup 3/ for multiple tethered robots. *IEEE Transactions on Robotics and Automation*, 15(4):623–639, 1999. doi:10.1109/70.781966.

3 Susan Hert and Vladimir Lumelsky. Planar curve routing for tethered-robot motion planning. *International Journal of Computational Geometry & Applications*, 7(03):225–252, 1997.

4 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.

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

6 Xiao Peng, Christine Solnon, and Olivier Simonin. Solving the Non-Crossing MAPF with CP. In *CP 2021 - 27th International Conference on Principles and Practice of Constraint Programming*, LIPIcs: Leibniz International Proceedings in Informatics, pages 1–17, Montpellier (on line), France, October 2021. URL: https://hal.archives-ouvertes.fr/hal-03320987, doi:10.4230/LIPIcs.CP.2021.20.

7 Oren Salzman and Dan Halperin. Optimal motion planning for a tethered robot: Efficient preprocessing for fast shortest paths queries. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4161–4166. IEEE, 2015.

8 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.

9 Iddo Shnaps and Elon Rimon. Online coverage by a tethered autonomous mobile robot in planar unknown environments. *IEEE Transactions on Robotics*, 30(4):966–974, 2014.

10 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.

11 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.

12 Xu Zhang and Quang-Cuong Pham. Planning coordinated motions for tethered planar mobile robots. *Robotics and Autonomous Systems*, 118:189–203, 2019.