Planning problems in a multi-tethered-robot system

Xiao Peng

N°d'ordre NNT : 2024ISAL0018

THESE de DOCTORAT DE L'INSA LYON, membre de l'Université de Lyon

Ecole Doctorale N° 512
Informatique et Mathématiques

Spécialité/ discipline de doctorat :

Informatique

Soutenue publiquement le 19/02/2024, par:
Xiao Peng

Planning Problems in a Multi-Tethered-Robot System

Devant le jury composé de :

Mme Aurélie Beynier	Maître de conférences HDR Sorbonne Université	Rapportrice
M. Yves Deville	Professeur des universités Université Catholique de Louvain	Rapporteur
M. David Coeurjolly	Directeur de recherche CNRS Université Lyon 1	Examinateur
M. Cédric Pradalier	Professeur des universités GeorgiaTech Lorraine	Examinateur
M. Cédric Pralet	Directeur de Recherche HDR, ONERA	Examinateur
M. Olivier Simonin	Professeur des universités, INSA Lyon	Directeur de thèse
Mme Christine Solnon	Professeur des universités, INSA Lyon	Co-directrice de thèse

Département FEDORA – INSA Lyon - Ecoles Doctorales

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
ED 206 CHIMIE	**CHIMIE DE LYON** https://www.edchimie-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage secretariat@edchimie-lyon.fr	**M. Stéphane DANIELE** C2P2-CPE LYON-UMR 5265 Bâtiment F308, BP 2077 43 Boulevard du 11 novembre 1918 69616 Villeurbanne directeur@edchimie-lyon.fr
ED 341 E2M2	**ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION** http://e2m2.universite-lyon.fr Sec. : Bénédicte LANZA Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.e2m2@univ-lyon1.fr	**Mme Sandrine CHARLES** Université Claude Bernard Lyon 1 UFR Biosciences Bâtiment Mendel 43, boulevard du 11 Novembre 1918 69622 Villeurbanne CEDEX e2m2.codir@listes.univ-lyon1.fr
ED 205 EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTÉ** http://ediss.universite-lyon.fr Sec. : Bénédicte LANZA Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.ediss@univ-lyon1.fr	**Mme Sylvie RICARD-BLUM** Laboratoire ICBMS - UMR 5246 CNRS - Université Lyon 1 Bâtiment Raulin - 2ème étage Nord 43 Boulevard du 11 novembre 1918 69622 Villeurbanne Cedex Tél : +33(0)4 72 44 82 32 sylvie.ricard-blum@univ-lyon1.fr
ED 34 EDML	**MATÉRIAUX DE LYON** http://ed34.universite-lyon.fr Sec. : Yann DE ORDENANA Tél : 04.72.18.62.44 yann.de-ordenana@ec-lyon.fr	**M. Stéphane BENAYOUN** Ecole Centrale de Lyon Laboratoire LTDS 36 avenue Guy de Collongue 69134 Ecully CEDEX Tél : 04.72.18.64.37 stephane.benayoun@ec-lyon.fr
ED 160 EEA	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** https://edeea.universite-lyon.fr Sec. : Philomène TRECOURT Bâtiment Direction INSA Lyon Tél : 04.72.43.71.70 secretariat.edeea@insa-lyon.fr	**M. Philippe DELACHARTRE** INSA LYON Laboratoire CREATIS Bâtiment Blaise Pascal, 7 avenue Jean Capelle 69621 Villeurbanne CEDEX Tél : 04.72.43.88.63 philippe.delachartre@insa-lyon.fr
ED 512 INFOMATHS	**INFORMATIQUE ET MATHÉMATIQUES** http://edinfomaths.universite-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage Tél : 04.72.43.80.46 infomaths@univ-lyon1.fr	**M. Hamamache KHEDDOUCI** Université Claude Bernard Lyon 1 Bât. Nautibus 43, Boulevard du 11 novembre 1918 69 622 Villeurbanne Cedex France Tél : 04.72.44.83.69 direction.infomaths@listes.univ-lyon1.fr
ED 162 MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** http://edmega.universite-lyon.fr Sec. : Philomène TRECOURT Tél : 04.72.43.71.70 Bâtiment Direction INSA Lyon mega@insa-lyon.fr	**M. Etienne PARIZET** INSA Lyon Laboratoire LVA Bâtiment St. Exupéry 25 bis av. Jean Capelle 69621 Villeurbanne CEDEX etienne.parizet@insa-lyon.fr
ED 483 ScSo	**ScSo**[1] https://edsciencessociales.universite-lyon.fr Sec. : Mélina FAVETON Tél : 04.78.69.77.79 melina.faveton@univ-lyon2.fr	**M. Bruno MILLY** (INSA : J.Y. TOUSSAINT) Univ. Lyon 2 Campus Berges du Rhône 18, quai Claude Bernard 69365 LYON CEDEX 07 Bureau BEL 319 bruno.milly@univ-lyon2.fr

[1] ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Abstract

Multiple robot systems are widely applied in our real life. As a special type of mobile system, tethered robots play a crucial role in special contexts, particularly in challenging conditions, where cables provide stable access to power and network connectivity. However, constraints imposed by cables also introduce new challenges for motion planning in these applications. This thesis focuses on planning problems for a team of tethered robots, addressing two major problems: non-crossing Anonymous Multi-Agent Path Finding (AMAPF) and Multiple Tethered Coverage Path Planning (MTCPP).

The goal of non-crossing AMAPF is to find a set of non-crossing paths such that the makespan is minimal. The study is carried out in two cases, considering whether the robots are treated as point-sized or not. This hypothesis significantly influences the computation of makespan. The problem is abstracted to the Euclidean Bipartite Assignment problem and we show that bounds can be efficiently computed by solving linear assignment problems. We introduce a variable neighborhood search method to improve upper bounds, and a Constraint Programming model to compute optimal solutions. The approach is experimentally evaluated on three different kinds of instances.

The MTCPP problem is addressed by initially partitioning the workspace into equitable connected subregions, enabling each robot to operate independently in its assigned area. We propose an approach based on the additively weighted Voronoi diagram, ensuring an equitable partitioning that enforces relative star-convexity of each subregion to the associated anchor point, thereby avoiding cable entanglement. For the coverage path planning of each robot, the Spanning Tree Coverage method is shown to effectively solve the problem while respecting cable constraints.

Résumé

Les systèmes de multiples robots ont été largement appliqué dans notre vie. En tant que type particulier de système mobile, les robots à câble jouent un rôle crucial dans des contextes spécifiques et des conditions difficiles, où le câble offre un accès stable à l'énergie et à la connectivité réseau. Cependant, les contraintes imposées par le câble introduisent également de nouveaux défis pour la planification des mouvements dans ces applications. Cette thèse se concentre sur les problèmes de planification pour une équipe de robots à câble, abordant deux problèmes majeurs: le Anonymous Multi-Agent Path Finding (AMAPF) sans croisement et le Multiple Tethered Coverage Path Planning (MTCPP).

L'objectif du AMAPF sans croisement est de trouver un ensemble de trajectoires non croisées de manière à minimiser la longueur du plus long chemin (makespan). L'étude est divisée en deux cas, selon que l'on néglige la taille du robot ou non. Cette hypothèse influence significativement le calcul du makespan. Le problème est abstrait sous la forme d'un couplage bipartite euclidien, et nous montrons que des bornes peuvent être efficacement calculées en résolvant des problèmes d'affectation linéaires. Nous introduisons une méthode de recherche à voisinage variable pour améliorer les bornes supérieures, et un modèle de programmation par contraintes pour calculer des solutions optimales. L'approche est évaluée expérimentalement sur trois types différents d'instances.

Le problème MTCPP est abordé en partitionnant initialement l'espace de travail en sous-régions équitables connectées, permettant à chaque robot de fonctionner indépendamment dans sa zone assignée. Nous proposons une approche basée sur le diagramme de Voronoi pondéré de manière additive, assurant une partition équitable qui impose la étoile-convexité relative de chaque sous-région par rapport au point d'ancrage associé, évitant ainsi l'emmêlement des cables. Pour la planification de la trajectoire de couverture de chaque robot, la méthode Spanning Tree Coverage permet de résoudre efficacement le problème tout en respectant les contraintes du câble.

Acknowledgements

I would like to firstly express my deepest gratitude to my supervisor Christine. I consider the mentorship you provided me as the most precious asset of my PhD experience, offering not only academic guidance but also serving as a professional and personal example. It reminds me the words of a Chinese philosopher from over a millennium ago who stated that teachers are those who propagate doctrines, impart professional knowledge, and resolve doubts. Your expertise, scientific rigor and continuous creativity in research have been instrumental in shaping my academic growth. Thank you for your kindness, patience and respect when I went through hard times. Without your support, I could have never finished this thesis.

I would also like to thank Oliver, my thesis supervisor, who made it possible for me to start this journey. Thank you for your support every time I made a decision and for giving me the freedom to explore the subjects I was passionate about. Thanks to your good management of the project, I can concentrate fully on the research work.

Meeting other researchers and fellows has been among the most precious moments of my PhD experience. I extend my thanks to François for his collaboration on the coverage path problem, bringing invaluable expertise in computational theory to enhance the quality of our work. I also want to acknowledge Florian for our joint efforts on the minimum-turn Hamiltonian path problem. Though not included in this thesis, I am very pleased with the progress we made. Special thanks to Agathe for her help with optimal transport theory and to Alessandro for his interesting discussions on the multi-robot partitioning problem—your insights have significantly contributed to improving the manuscript. Gratitude is also extended to my thesis committee and jury for their advice and time invested in reviewing my thesis.

The experience would not have been the same without the camaraderie of my colleagues: Benoît, Lélio, Romain, Loïc, Alix, Guillaume, Aurélien, David, Maxime, Benjamin, Idham, Anotine, Théotime and Simon. Our shared moments in the coffee corner and various afterwork activities created a convivial atmosphere that made me feel accompanied. A big hug to Pierre, with whom I trained for 8 months to prepare for a marathon, and finally we succeeded. This special and unforgettable experience, and all the discussions we had, helped me through the most stressful period of my PhD.

Finally, I extend my heartfelt thanks to my friends, in Paris, Beijing, Lyon, Shenzhen, Nice, Hong Kong, and beyond. While it is impossible to list all their names here, their support throughout this journey is deeply appreciated. To my parents and other family members, I express all my gratitude and love for their dedication, care, and continuous support, which gave me the strength to complete this degree.

Contents

Abstract iii

Résumé v

Acknowledgements vii

1 Introduction 1
- 1.1 Motivation and Challenges . . . 1
  - 1.1.1 Multiple Tethered Robots Path Finding . . . 2
  - 1.1.2 Multiple Tethered Robots Coverage Path Planning . . . 3
- 1.2 Thesis Overview . . . 4
  - 1.2.1 Thesis Contribution . . . 4
  - 1.2.2 Thesis Organization . . . 5
- 1.3 Publications . . . 5

2 Context 7
- 2.1 Geometry Basis . . . 7
  - 2.1.1 Notations and definitions . . . 7
  - 2.1.2 Homotopy class . . . 9
  - 2.1.3 Homotopic shortest path . . . 11
  - 2.1.4 Detecting crossing paths . . . 13
- 2.2 Tethered Robot Path Planning Problem . . . 14
  - 2.2.1 The shortest path problem . . . 14
    - The geometric methods . . . 14
    - The graph search based methods . . . 16
  - 2.2.2 Enumeration of non-homotopic paths . . . 17
  - 2.2.3 Multiple tethered robot path planning . . . 20
    - One-to-many settings . . . 20
    - Motion coordination . . . 21
  - 2.2.4 Coverage path planning . . . 21
    - STC algorithm . . . 22
    - Tethered coverage problem . . . 22
- 2.3 Discussion . . . 23

3 Mathematical Programming **25**
3.1 Constraint Programming . . . 25
3.1.1 Basic Notions . . . 25
3.1.2 CSP Solving . . . 26
Ordering heuristics . . . 27
Propagate . . . 27
3.1.3 Implementation of Global Constraints with Choco . . . 29
3.2 Integer Linear Programming . . . 29
3.3 Discussion . . . 31

4 Non-Crossing AMAPF for point-sized robots **33**
4.1 Definition of the NC-AMAPF Problem . . . 34
4.2 NC-AMAPF Problem without Obstacles . . . 35
4.2.1 Computation of Bounds by Solving Assignment Problems . . . 35
4.2.2 Variable Neighbourhood Search . . . 36
4.2.3 Constraint Programming Model . . . 37
4.2.4 Experimental Evaluation . . . 37
4.3 NC-AMAPF Problem with Obstacles . . . 39
4.3.1 Computation of Bounds . . . 40
4.3.2 Relevant Paths Enumeration . . . 43
4.3.3 Constraint Programming Model . . . 43
4.3.4 Description of Benchmarks . . . 44
4.3.5 Experimental Evaluation . . . 45
4.4 Conclusion . . . 48

5 Non-Crossing AMAPF for non point-sized robots **51**
5.1 Preliminaries . . . 53
5.1.1 Removing crossings . . . 53
5.1.2 Detecting deadlocks . . . 53
5.1.3 Computing the makespan . . . 54
5.2 Problem Statement . . . 55
5.3 Computation of bounds for the makespan . . . 56
5.3.1 Algorithm for removing deadlocks . . . 56
5.3.2 Computation of Makespan Bounds from LSAP and LBAP solutions . . . 59
5.3.3 Experimental evaluation . . . 60
5.4 Improving the upper bound with VNS . . . 62
5.4.1 Experimental evaluation . . . 64
5.5 Computation of the optimal solution with CP . . . 66
5.5.1 Relaxed CP model . . . 66
5.5.2 Lazy constraint generation . . . 67
5.5.3 Dichotomous Approach . . . 68
5.5.4 Experimental results . . . 69

5.6 Conclusion . . . 71

6 Single Tethered Robot Coverage Path Planning **73**
6.1 Problem Statement . . . 75
6.1.1 Notations and Definitions . . . 75
6.1.2 Computing the Cable Configuration . . . 75
6.1.3 Definition of the CPP Problem for Tethered Robots . . . 76
6.2 Complexity Study . . . 78
6.2.1 MCT Problem without Forbidden Areas . . . 78
6.2.2 MCT Problem with Forbidden Areas . . . 79
6.3 Configuration Enumeration Based Approach (Model $\mathcal{M}_0$) . . . 83
6.4 Approximation Algorithms . . . 86
6.4.1 Model $\mathcal{M}_1$ . . . 86
6.4.2 A Fixed-Parameter Tractable (FPT) Algorithm (Model $\mathcal{M}_2$) . . . 87
6.4.3 Model $\mathcal{M}_3$ and $\mathcal{M}_3h$ . . . 90
6.5 Experimental results . . . 92
6.5.1 Description of Benchmarks . . . 93
6.5.2 Computational Performance . . . 94
6.5.3 Scalability Analysis . . . 95
6.6 Discussion . . . 99
6.7 Conclusion . . . 100

7 Multiple Tethered Robots Coverage Path Planning **103**
7.1 Problem Statement . . . 104
7.2 Methodologies . . . 105
7.2.1 Ham-Sandwich Based Polygon Division . . . 106
7.2.2 DARP . . . 107
7.2.3 Voronoi-Based Partition . . . 108
7.3 Simulation Results . . . 111
7.4 Equitable division in nonconvex polygon . . . 116
7.5 Conclusion . . . 116
7.6 Open Problems . . . 117
7.6.1 Diameter Constraint . . . 117
7.6.2 Reachability Constraint . . . 118
7.6.3 Anchor points placement . . . 119

8 Conclusion **121**
8.1 Summary . . . 121
8.2 Future Work . . . 122
8.2.1 Avoiding Paths Enumeration . . . 122
8.2.2 Initial Locations Being not at Anchor Points . . . 123
8.2.3 Coverage Path with Minimum Turns . . . 123
8.2.4 Partitioning with Forbidden Areas . . . 124

8.2.5 Modeling Workspace with 3D Meshes 125

A Global Constraint for Detecting Deadlocks **127**

A.1 Incremental Topological Sorting 127

A.2 Incremental Deadlock Check 129

A.3 Global Constraint of Makespan 131

A.4 Experimental Results 131

List of Figures

1.1 The H2020 project BugWright2 (a) and the crawlers used to clean industrial surfaces (b). . . . 1

2.1 Left: example of workspace. Right: the associated visibility graph. . . . 8
2.2 Illustration of homotopy. . . . 10
2.3 The geometric illustration of cable configurations and homotopic shortest path. . . . 11
2.4 An example of the funnel algorithm. . . . 12
2.5 Different cases of two paths with a non empty intersection (angle sizes are represented by green arrows). . . . 13
2.6 An illustration of Algorithm SHORTESTPATH. . . . 15
2.7 An example of the homotopy augmented visibility graph G_{hvis} associated with the workspace in Fig. 2.1. . . . 17
2.8 Example of discarding non-k-optimal paths. . . . 19
2.9 Illustration of the STC algorithm. . . . 22

4.1 Example of a NC-AMAPF instance. . . . 34
4.2 Evolution of the optimal makespan (Opt), the lower bound (LB), the upper bounds (UB_i with $i \in \{1,3,5,7\}$) and the gap to optimality (in percentage) when increasing the number n of robots. . . . 38
4.3 Example of an optimal solution computed by non-shortest paths. . . . 40
4.4 Illustration of the proof for Theorem 4.3.1. . . . 41
4.5 Workspaces $\mathcal{W}_o$ with $o \in \{5,10,15,20\}$ (obstacles are displayed in green). . . . 44
4.6 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. . . . 45
4.7 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. . . . 46

5.1 Illustration of the precedence constraints on robots' motion. . . . 52
5.2 Deadlock example. . . . 54
5.3 Instance examples with $n = 30$, $o = 20$, and $d = U$ (left), B (middle), and A (right). . . . 56
5.4 Illustration of Algorithm 4. . . . 58

5.5 Gap in percentage (y-axis) between the optimal makespan and lb_{LBAP}, ub_{LSAP}, and ub_{LBAP}. 61
5.6 Comparison of ub_{LSAP}, and ub_{LBAP}. 62
5.7 Evolution of the gap to optimality in percentage with respect to time (in seconds) for OLDVNS and NEWVNS when $k_{max} \in \{1,3,5,7\}$. 65
5.8 Evolution of the gap to optimality in percentage with respect to time for COMBINEDVNS. 66
5.9 Percentage of solved instances with respect to time when $d = U$ (left), B (middle), and A (right). 69
5.10 Evolution of the gap to optimality (in percentage) with respect to time for $n = 30$, $o \in \{5,10,15,20\}$ and $d = U$ (Top), B (Middle), or A (Bottom). 70

6.1 Example forbidden areas. 74
6.2 Examples of cable configuration in a grid graph. (a): $X_s = q_0q_1q_2u$, $X_t = q_0q_1q_2q_3q_4v$. (b): $X_s = q_0q_1q_2u$, $X_t = q_0q_1v$. 76
6.3 Examples of Hamiltonian cycle in graph g 77
6.4 Example of paths and cable positions. 79
6.5 Gadgets. 81
6.6 Proof by a reduction from a Separable Planar 3-SAT instance. 82
6.7 Example of G_{cfg}. 84
6.8 An example showing that the search of a maximum tree in $\hat{G}_{cfg}$ is a suboptimal solution. 86
6.9 Example of a partition (incomplete) of the workspace. 88
6.10 Illustration of two types of connectivity between vertices in G_q. 90
6.11 Illustration of $\hat{G}_q$, corresponding to the example in Fig. 6.9. 91
6.12 An example of the prefix configuration graph G_p associated with the workspace in Fig. 6.9. 92
6.13 The 6 simulated workspaces tested in our experimental analysis. . . . 93
6.14 Comparison of the 5 ILP models' solving times with regard to solution quality and instance hardness. 96
6.15 Examples of workspace partitions resulting from the FPT methods ($\mathcal{M}_3$ and $\mathcal{M}_3h$). 97
6.16 Evolution of graph sizes and solving times (y-axis with a log scale) with respect to the cable length (x-axis). 98
6.17 Discretization issues related to the shape of obstacles and forbidden zones. 100

7.1 Different cases of reachability. 105
7.2 Examples of different $d(x,x_i)$. 110
7.3 Comparison between the three partitioning methods in case without obstacles in the workspace. 112
7.4 The case where the anchor points are densely located. 113

7.5 Example of partitioning in the presence of obstacles. 113
7.6 Example of partitioning in the three different environments introduced in [Pal+19]. 114
7.7 The results computed by the Voronoi-base method when the non-uniform density function is applied. 115
7.8 Illustration of Voronoi solution where the subregions are not compact. 115
7.9 2-partition for a concave polygon. 118

8.1 Illustrations of triangle intersections and corresponding coordinated robot motions. 124
8.2 Minimum turns problem with a coverage path generated by STC. . . . 125

A.1 An example from [PK07] 128
A.2 The illustration for Example A.2.1. 130
A.3 Comparison of solving time by global constraint and lazy constraint generation approach. 132

List of Tables

2.1 Summary of tethered robots applications in the literature. 24

4.1 Scale-up properties with respect to the number n of robots. 39
4.2 Results of UB$_i$CP with $i \in \{1,3,5,7\}$ for U instances with $n = 40$ and $o \in \{5,10,15,20\}$ (average on 30 instances). 47
4.3 Results of UB$_i$CP with $i \in \{1,3,5,7\}$ for B instances with $n = 20$ and $o \in \{5,10,15,20\}$. 47
4.4 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 $o = 20$. 48

5.1 Time needed to compute all shortest paths (t_{path}), and the two bounds ub_{LSAP} and ub_{LBAP}. 62

6.1 The scenarios tested in the experimental evaluation. 94

7.1 Comparision between three partitioning methods 117

List of Abbreviations

AMAPF	**A**nonymous **M**ulti-**A**gent **P**ath **F**inding
COP	**C**onstraint **O**ptimization **P**roblem
CP	**C**onstraint **P**rogramming
CPP	**C**overage **P**ath **P**lanning
CSP	**C**onstraint **S**atisfaction **P**roblem
DAG	**D**irected **A**cyclic **G**raph
FPT	**F**ixed **P**arameter **T**ractable
GAC	**G**eneralized **A**rc **C**onsistency
GMSTP	**G**eneralized **M**inimum **S**panning **T**ree **P**roblem
HST	**H**am-**S**andwich **T**heorem
ILP	**I**nteger **L**inear **P**rogramming
LBAP	**L**inear **B**ottleneck **A**ssignment **P**roblem
LSAP	**L**inear **S**um **A**ssignment **P**roblem
MAPF	**M**ulti-**A**gent **P**ath **F**inding
MCCT	**M**aximum **C**luster **C**overage **T**ree Problem
MCT	**M**aximum **C**overage **T**ree Problem
MRS	**M**ulti-**R**obot **S**ystem
MTCPP	**M**ultiple **T**ethered robot **C**overage **P**ath **P**lanning
NC-AMAPF	**N**on-**C**rossing **A**nonymous **M**ulti-**A**gent **P**ath **F**inding
SP	**S**hortest **P**ath Search
STC	**S**panning **T**ree **C**overage
TCPP	**T**ethered robot **C**overage **P**ath **P**lanning
VNS	**V**ariable **N**eighborhood **S**earch

List of Notations

$\mathcal{W}$	The workspace
$\mathcal{B}$	The bounding polygon of $\mathcal{W}$
$\mathcal{O}$	The set of obstacles
$\mathcal{H}$	The set of holes
$\mathcal{V}_{\mathcal{O}}$	The obstacle vertices
X_s, X_t	The initial and target cable configuration
$\mathcal{A}$	The set of anchor points
$\mathcal{T}$	The set of target points
$h(\pi)$	The homotopy class of π
$H(\pi)$	The homotopy shortest path of π
g	A discretized representation of $\mathcal{W}$
$G_4 = (\mathcal{V}_4, \mathcal{E}_4)$	Reduced from g that groups 2×2 cells as a new vertex
$G_{vis} = (\mathcal{V}_{vis}, \mathcal{E}_{vis})$	The visibility graph
$G_{cfg} = (\mathcal{V}_{cfg}, \mathcal{E}_{cfg})$	The configuration graph
$\hat{G}_{cfg} = (\hat{\mathcal{V}}_{cfg}, \hat{\mathcal{E}}_{cfg})$	The directed configuration graph
$G_p = (\mathcal{V}_p, \mathcal{E}_p)$	The prefix configuration graph
$\hat{G}_p = (\hat{\mathcal{V}}_p, \hat{\mathcal{E}}_p)$	The directed prefix configuration graph
$G_q = (\mathcal{V}_q, \mathcal{E}_q)$	The quotient graph of G_{cfg}
$\hat{G}_q = (\hat{\mathcal{V}}_q, \hat{\mathcal{E}}_q)$	The directed quotient graph
$T = (\mathcal{V}_T, \mathcal{E}_T)$	The coverage tree found in G_4
$T_{cfg} = (\mathcal{V}_{T_{cfg}}, \mathcal{E}_{T_{cfg}})$	The coverage tree found in G_{cfg}
$\hat{T}_{cfg} = (\mathcal{V}_{\hat{T}_{cfg}}, \mathcal{E}_{\hat{T}_{cfg}})$	The coverage tree found in $\hat{G}_{cfg}$
$T_q = (\mathcal{V}_{T_q}, \mathcal{E}_{T_q})$	The coverage tree found in G_q
$\hat{T}_q = (\mathcal{V}_{\hat{T}_q}, \mathcal{E}_{\hat{T}_q})$	The coverage tree found in $\hat{G}_q$

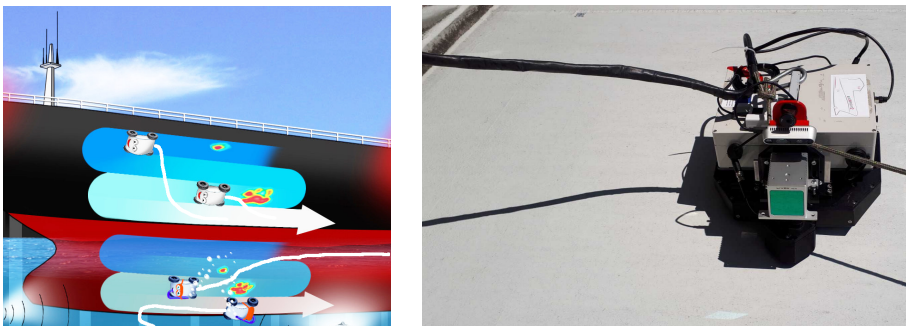
Dedicated to my parents

Chapter 1

Introduction

1.1 Motivation and Challenges

In recent years, multi-robot systems (MRS) have become increasingly prevalent in real-world applications. Depending on the type of vehicles involved, whether aerial, ground-based, or a combination of both, a MRS typically comprises Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs). UAVs are extensively used in surveillance, search and rescue operations, and agricultural tasks, while UGVs are primarily deployed in various service scenarios, including industrial manufacturing and warehouse transportation. The implementation of a cooperative MRS enhances the overall efficiency of these vehicles during missions. A crucial aspect in achieving safe and coordinated navigation within a MRS is path planning. This involves generating paths and movements that enable mobile vehicles to navigate from one state to another while avoiding static and dynamic obstacles in their environment.



(a) (b)

FIGURE 1.1: The H2020 project BugWright2 (a) and the crawlers used to clean industrial surfaces (b).

This thesis is implemented under the context of the European H2020 project BugWright2[1], which aims to deploy autonomous robots to conduct inspection tasks on hulls of large ships. In this thesis, we work on a fleet of crawlers operating on hull surfaces, which are mobile, wheel-mounted robots powered by a cable, as in illustrated in Fig. 1.1. Tethered robots are widely applied in the industrial applications,

[1]H2020 project BugWright2: Autonomous Robotic Inspection and Maintenance on Ship Hulls and Storage Tanks, 2020-24 (see https://www.bugwright2.eu/)

where robots are attached to anchor points by flexible cables, allowing them continuous access to essential resources such as energy, fluids, and network connectivity.

Traditional path planning in robotics deals with mobile robots that can be abstracted as points, circles, or other regular polygons. However, planning paths for tethered robots introduces additional complexities due to the constraints associated with the cables. The approach to handling these constraints depends on the assumptions made about the cables. If the cable's thickness cannot be ignored, the robot is typically modeled as a chain of cylinder-like bodies. In this scenario, the core challenge lies in computing valid configurations for the movement of joints. Alternatively, if the cable can be simplified to a line, computational geometry techniques can be applied to address cable-related constraints, such as avoiding cable entanglements. In our context, we adopt the second assumption, and we suppose also that the cables are kept taut by a recoiling system that pulls on cables when robots move back. In addition to the geometrical constraints related to the cables, the characteristics of the environment can also bring new constraints. For example, in some scenarios, the workspace contains areas that cannot be accessible by the cables, since the cables might be vulnerable to some tough conditions, like high temperature and electromagnetic interference. Consequently, for a feasible solution, the robot can only take a path ensuring that its cable never passes through these forbidden areas.

In a multi-robot system, while the cooperation between robots can be efficient in some complex tasks, the constraint to avoid the conflicts among the robots makes the planning harder. The multi-agent path finding problem (MAPF) is abstracted to address the challenge [Ma+17]. Both theoretical and engineering foundations have been established in this field. However, the MAPF problem for tethered robots is still a new challenge to be explored. The main difference is that in a classical MAPF problem, the collision between two robots take place temporarily and can be avoided by planning a new path or executing the "waiting" action. While for tethered robots, the cable's entanglements can not be simply solved, as the cable of one robot is treated as an obstacle for others. Base on this context, two principal problems are addressed in this thesis.

1.1.1 Multiple Tethered Robots Path Finding

In this problem, we consider a team of n tethered robots initially located at their anchor positions and n target points. These target points are assumed to be reached by the robots, but they are not specifically assigned to a particular robot in advance. The objective is to compute a set of collision-free paths for the robots and minimize the makespan, *i.e.*, the arrival time of the last robot. This definition extends the classic Anonymous Multi-Agent Path Finding (AMAPF) problem [Ste+19].

When the cable is viewed as a line, the problem transforms into a geometric problem: in a 2D plane, given two sets of points, how to establish a one-to-one matching and connect the paired points by a segment in a way that avoids any intersections

between segments. In the presence of obstacles in the workspace, segments are replaced by polylines. An optimal solution involves finding a bipartite matching, and a set of non-crossing segments (polylines), such that the length of the longest segment (polyline) is minimized. In this problem, we assume that the cable length is unlimited, otherwise the objective no longer makes sense.

In scenarios with obstacles, an additional consideration is the physical size of the robots. Specifically, in the simplified problem formulation, the definition of makespan, which considers the length of the longest path, is underestimated. In real-world applications, when two robots pass through the same point or along the same edge of an obstacle, motion coordination is necessary to prevent cable entanglement. In such cases, a practical solution is to impose an ordering on their motions [ZP19] and incorporate a waiting time to ensure a safety distance between two robots.

1.1.2 Multiple Tethered Robots Coverage Path Planning

As a specific application case, the coverage path planning (CPP) for a multi-tethered-robot system is studied in this thesis. CPP is a fundamental problem in robotics, with applications ranging from automatic floor cleaning, area patrol, and rescue search. While extensive research works have been conducted on classical robots, our focus is on exploring the dynamics and challenges posed by tethered robots, offering novel insights to the field. Our research interests in this problem are twofold: firstly, in the single tethered robot planning problem, adapting traditional coverage path planning approaches to meet cable constraints; secondly, in the scope of multiple tethered robots, figuring out how to avoid cable entanglement among different robots while searching for an optimal solution that minimizes the makespan.

The spanning tree coverage algorithm (STC) is widely applied when solving robots coverage problem [SR14]. Its main principle is to construct a Hamiltonian cycle circumnavigating a spanning tree in a structured grid graph. However, applying this method to tethered robots could violate the cable length constraint in some cases. Additionally, in the presence of forbidden areas in the workspace, finding a non-repetitive path that allows coverage of the maximum area remains a $\mathcal{NP}$-complete problem.

In the context of multiple robots tackling a coverage task, a common strategy involves initially partitioning the surface into a set of equitable connected subregions, allowing each robot to operate independently on its assigned area. This approach minimizes collisions among robots and optimizes the makespan. However, for tethered robots, an equitable subdivision alone is not sufficient. In nonconvex subregions, even if a robot moves inside, its cable could still cross the subregion of other robots. While addressing this issue in a scheduling problem is a solution, it tends to be computationally expensive. Our objective is to find a partition that enforces certain shape constraints on these subregions, thereby preventing cable entanglement.

1.2 Thesis Overview

1.2.1 Thesis Contribution

All the results reported in this thesis are based on perfect perception of the environment. We assume that the robots' initial positions are known and that the map of the environment is available a priori. The proposed algorithms serve as global planners, and are implemented offline, in a centralized manner. The main contributions of this thesis comprise the following items.

- The thesis introduces a new variant of MAPF problem for tethered robots, as Non-Crossing AMAPF problem. The problem structure and constraints depend on the physical size of robots. It is firstly addressed by assuming that the robots are all point-sized. This problem is related to an Euclidean bipartite matching problem, and a feasible upper bound can be computed in polynomial time, by solving the *Linear Sum Assignment Problem (LSAP)*. Additionally, an approach based on the sequential combination of *Variable Neighborhood Search* (VNS) and *Constraint Programming* (CP) is introduced for optimally solving the problem.

- We extend the Non-Crossing AMAPF problem to non point-sized robots. In this case, motion synchronization is emphasized to ensure a safety distance that prevents collision and cable entanglements. Motion constraints are translated into precedence constraints, which imply waiting times when computing the makespan. The solution of LSAP is proven to avoid deadlocks and always provides a valid upper bound. When computing the optimal solution, the VNS procedure is improved by considering non-shortest paths as neighbors, and a lazy constraint generation approach is proposed for solving the CP model.

- The thesis proposes the adoption of the STC algorithm for tethered robot coverage path planning (TCPP). The STC method is shown to meet these cable constraints, by applying Dijkstra's algorithm during the spanning tree search. A new constraint is introduced that considers forbidden areas within the workspace where the cable cannot pass. The presence of forbidden areas complicates the problem's complexity, and it is demonstrated that the TCPP problem incorporating forbidden areas is $\mathcal{NP}$-complete. Various approximate methods are introduced and compared to efficiently compute a lower bound for this problem.

- A solution is proposed for the multiple tethered robot coverage path planning problem (MTCPP). The key principle is to design a polygon partitioning algorithm that enforces each subregion to be relatively star-convex to the corresponding anchor point, in order to avoid cable entanglement. A survey

on existing polygon partitioning methods is conducted, showing that the approach based on the *additively weighted Voronoi diagram* can efficiently solve this problem.

1.2.2 Thesis Organization

The content of this thesis is organized as follows:

Chapter 2 provides an overview of basic geometric concepts and notions commonly applied in the context of tethered robots. Additionally, it surveys various planning problems associated with tethered robots found in the existing literature.

In Chapter 3, a general introduction to two mathematical programming methods used for solving combinatorial problems is presented: Constraint Programming (CP) and Integer Linear Programming (ILP). The chapter also outlines basic solution procedures for each method.

Chapter 4 formulates the Non-Crossing AMAPF problem. The results presented in this chapter are based on the assumption that the robots are point-sized.

Chapter 5 extends the work in Chapter 4 by considering the physical size of robots. Within this new setting, the problem is reformulated to incorporate a precedence constraint on robots' motion.

Chapter 6 focuses on the CPP problem for a single tethered robot. The problem is exploited from two perspectives. Firstly, a fundamental solution is introduced to address the cable length constraint by adapting the STC algorithm. Secondly, the constraint of forbidden areas within the workspace is considered.

Chapter 7 extends the coverage path planning work to the scope of multiple tethered robots. A qualitative comparison of different approaches is provided based on workspace characteristics.

Finally, Chapter 8 summarizes the main results obtained in the thesis and suggests potential directions for further research.

1.3 Publications

Most of the content of Chapters 4 and 5 has been published in:

- Xiao Peng, Olivier Simonin, and Christine Solnon. "Solving the Non-Crossing MAPF with CP". In: CP 2021-27th International Conference on Principles and Practice of Constraint Programming. 2021, pp. 1–17.
- Xiao Peng, Olivier Simonin, and Christine Solnon. "Non-Crossing Anonymous MAPF for Tethered Robots". In: Journal of Artificial Intelligence Research (JAIR) 78 (2023).

We plan to submit the content of Chapters 6 and 7 to two journals in the next months. In addition, the following publication was produced as a side work not directly related to the thesis project and not described in this document.

- Xiao Peng and Christine Solnon. "Using Canonical Codes to Efficiently Solve the Benzenoid Generation Problem with Constraint Programming". In: 29th International Conference on Principles and Practice of Constraint Programming 2023, pp. 1–17.

Chapter 2

Context

Contents

2.1 Geometry Basis 7
2.1.1 Notations and definitions 7
2.1.2 Homotopy class 9
2.1.3 Homotopic shortest path 11
2.1.4 Detecting crossing paths 13
2.2 Tethered Robot Path Planning Problem 14
2.2.1 The shortest path problem 14
2.2.2 Enumeration of non-homotopic paths 17
2.2.3 Multiple tethered robot path planning 20
2.2.4 Coverage path planning 21
2.3 Discussion 23

The path planning problem for tethered robots has gained considerable attention in the literature. Compared to the classical path finding problem, the main challenge with tethered robot comes from the constraints associated with the cable. For example, the robots' movement is constrained by the cable length and the cable is prohibited to get crossed. In this Chapter, we introduce firstly in Section 2.1 the key geometric concepts commonly used in these applications, along with relevant computational geometry algorithms. In Section 2.2, we categorize existing works in the literature in terms of their application scenarios as well as types of robots. For each category, we provide an overview of the methodologies employed to address each specific problem.

2.1 Geometry Basis

2.1.1 Notations and definitions

We consider robots move on a 2 dimensional workspace $\mathcal{W} \subset \mathbb{R}^2$. This workspace is defined by a bounding polygon $\mathcal{B}$ and a set $\mathcal{O}$ of obstacles: every obstacle in $\mathcal{O}$ is a polygon within $\mathcal{B}$, and $\mathcal{W}$ is connected, composed of every point in $\mathcal{B}$ that does not belong to an obstacle in $\mathcal{O}$. Without loss of generality, we assume that $\mathcal{B}$ is convex: if the bounding polygon is not convex, then we can compute its convex hull $\mathcal{B}$ and add

to $\mathcal{O}$ the obstacles corresponding to the difference between the bounding polygon and $\mathcal{B}$. We denote $\mathcal{V}_\mathcal{O}$ the set of vertices of obstacles in $\mathcal{O}$, and we assume that these vertices belong to $\mathcal{W}$ (and therefore, obstacle boundaries belong to $\mathcal{W}$). An example of workspace is displayed in Fig. 2.1(left).

In the applications of tethered robots, the robot is typically attached to a fixed anchor point. In most cases, the cable is assumed to be kept taut by a recoiling mechanism [SR14]. However, there are also works that consider scenarios with a slack cable [MH17], and some studies focus on robots with a snake-like structure [WBB21]. In our project, we assume that the cable remains taut while the robot is moving.

Given two points $u, v \in \mathcal{W}$, 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 $\overline{uv} \not\subset \mathcal{W}$. 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 $\overline{uv} \cap \overline{u'v'} \neq \emptyset$).

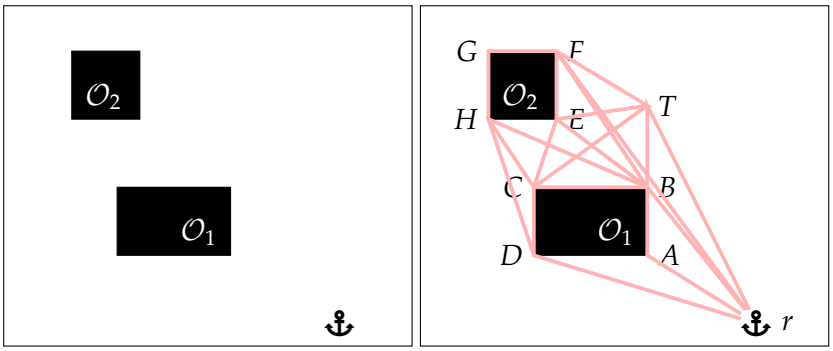


FIGURE 2.1: Left: example of workspace. Right: the associated visibility graph.

Given three points $u, v, w \in \mathcal{W}$, we denote $\angle uvw$ the angle between $\overline{vu}$ and $\overline{vw}$, and we denote $\measuredangle uvw$ the size of this angle measured in degrees when considering a counterclockwise order from $\overline{vu}$ to $\overline{vw}$.

Definition 2.1.1 (Path)**.** A path in $\mathcal{W}$ is composed of a chain of incident segments $\overline{u_0u_1}, \overline{u_1u_2}, \ldots, \overline{u_{i-1}u_i}$, which is represented by the vertex sequence $\pi = \langle u_0, u_1, u_2, \ldots, u_i \rangle$. The length of a path π is denoted as $|\pi|$ and is the sum of the lengths of its segments, *i.e.*, $|\pi| = \sum_{j=1}^{i} |u_{j-1}u_j|$.

Given two paths π_i and π_j, we denote $\pi_i.\pi_j$ the path obtained by concatenating π_j at the end of π_i. We use set operators to denote vertex membership and path inclusion: $u \in \pi$ denotes the fact that path π contains vertex u (*i.e.*, $\exists \pi_i, \pi_j, \pi = \pi_i.\langle u \rangle.\pi_j$) and $\pi_i \subseteq \pi$ denotes the fact that path π contains the subpath π_i (*i.e.*, $\exists \pi_j, \pi_k, \pi = \pi_j.\pi_i.\pi_k$). Given a set Π of paths and a vertex u, we denote Π_u the set of all paths of Π that contain u, *i.e.*, $\Pi_u = \{\pi \in \Pi \mid u \in \pi\}$.

Definition 2.1.2 (Cable configuration)**.** Let r be the anchor point where the cable is attached. The configuration of the cable as the robot moves to point t can be represented as a sequence of vertices $q = \langle r, u_1, u_2, ..., u_i, t\rangle$. Each internal point $u_j, j \in [1, i]$ corresponds to an obstacle vertex $\mathcal{V}_\mathcal{O}$, and no segment of the cable crosses an obstacle. We say that q is valid if (i) q is not self-crossing, (ii) its length does not exceed cable length, (iii) the cable is taut.

We suppose that the robot initially locates at the anchor point. As the length of a robot path cannot be smaller than the length of its cable configuration, we can simplify our problem by assuming that the path of a robot is its cable configuration, namely, the robot always moves along a taut path. Hence, we search for paths in a visibility graph [LW79] defined below.

Definition 2.1.3 (Visibility graph [LW79])**.** Given a workspace $\mathcal{W}$, a set of n anchor points $\mathcal{A}$, and a set of n targets $\mathcal{T}$, the visibility graph is the directed graph $G_{vis} = (\mathcal{V}_{vis}, \mathcal{E}_{vis})$ such that vertices are either points of $\mathcal{A}$ and $\mathcal{T}$ or obstacle vertices, *i.e.*, $\mathcal{V} = \mathcal{A} \cup \mathcal{T} \cup \mathcal{V}_\mathcal{O}$, and edges correspond to segments that do not cross obstacles and that do not contain any other vertex, *i.e.*,

$$\mathcal{E} = \{(u, v) \in (\mathcal{A} \cup \mathcal{V}_\mathcal{O}) \times (\mathcal{T} \cup \mathcal{V}_\mathcal{O}) \mid \overline{uv} \subset \mathcal{W} \ \wedge \ \forall w \in \mathcal{V} \setminus \{u, v\}, w \notin \overline{uv}\}$$

The graph is directed because edges starting from targets or ending on anchor points are forbidden.

An example of visibility graph is illustrated in Fig. 2.1(right). In [LW79], it is shown that visibility graphs can still be used when robots have a non-negligible size as obstacles may be expanded to compensate for robot sizes. In our problem, the robot size has no significant impact on the geometric properties of the cable. Even though the location of the cable does not perfectly overlap the trajectory along which the robots move, the cable can remain taut and the topological relationship between the cables (crossing or not) is not affected.

A path in the visibility graph G_{vis} is a sequence of vertices $\langle u_0, \ldots, u_i\rangle$ such that $(u_{j-1}, u_j) \in \mathcal{E}_{vis}, \forall j \in [1, i]$. 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.

Given an anchor point $a \in \mathcal{A}$ and a target $t \in \mathcal{T}$, we denote $sp(a, t)$ a shortest path from a to t in the visibility graph.

2.1.2 Homotopy class

As the workspace $\mathcal{W}$ is continuous, there exists an infinite number of cable configurations from an anchor point r to a target t. We use the notion of homotopy class to distinguish how a curve moves around the different obstacles (see Fig. 2.2).

Definition 2.1.4 (Homotopy class [Bha10]). Two curves are *homotopic* if there exists a continuous deformation between them without crossing obstacles, and a taut path is the shortest path of an homotopy class.

The homotopy class of a curve is represented by its h-signature [SH15]. The idea is that we set a reference point $p_i \in \mathcal{O}_i$ for each obstacle $\mathcal{O}_i$, and extend it as a vertical ray r_i towards $y = +\infty$. For each curve γ in the workspace, its homotopy class is expressed by a word $h(\gamma)$ composed of a sequence of letters. Each letter of $h(\gamma)$ encodes a crossing between the curve γ and one of these rays. Since the curve γ is directed, intersecting a ray from left to right is different from the right to left. The word $h(\gamma)$ is defined as follows. We start with the empty string Λ; then each time γ intersects a ray r_i from left to right (resp. right to left), we concatenate the letter t_i (resp. t_i^{-1}). We finally reduce the word by deleting factors of the form $t_i t_i^{-1}$ or $t_i^{-1} t_i$ in it.

Example 2.1.1 (Example of h-signature). Consider the curve γ_1 in Fig. 2.2. As it crosses the extended ray of $\mathcal{O}_1$ from right to left and crosses it again from left to right, and then crosses the extended ray of $\mathcal{O}_2$ from right to left, the resulted word is $t_1 t_1^{-1} t_2^{-1}$. After reduction, we obtain t_2^{-1}. Similarly the signature of γ_2 and γ_2' is Λ; the signature of γ_3 is $t_1^{-1} t_2^{-1}$.

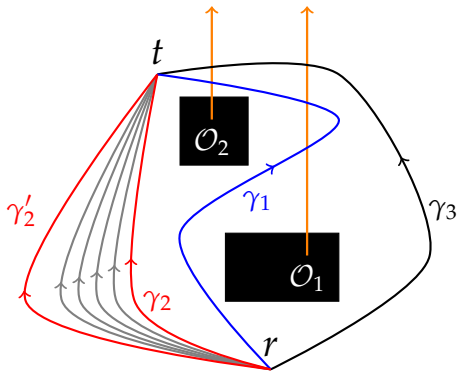


FIGURE 2.2: Illustration of homotopy. Orange arrows are the rays used to compute homotopy classes. Curves γ_2 and γ_2' are homotopic while γ_1, γ_2 and γ_3 are mutually not.

The h-signature is a homotopy invariant of a curve and two curves are homotopic if they connect the same endpoints and have identical h-signatures. Besides, we can also learn that the concatenation rule is preserved for h. If a curve is a concatenation of two curves, as $\gamma = \gamma_1 \circ \gamma_2$, then $h(\gamma) = h(\gamma_1) \circ h(\gamma_2)$ holds, where $\circ$ is the concatenation on words followed by the reduction operation.

Following this notion, suppose that the initial configuration of the cable is q_s, when the robot moves from s to t along the path $\pi_{s\to t}$, the final configuration of the cable at t can be considered as the homotopic taut path of $h(q_s) \circ h(\pi_{s\to t})$. In Section 2.1.3, we talk about algorithms that shorten a path to a taut path of the same homotopy class.

2.1.3 Homotopic shortest path

In our settings, the cable is considered to keep taut. Interestingly, the shortest path between any two points in the workspace is in fact a taut path. A taut path is also referred to as the homotopic shortest path, and we use $H(\pi)$ to denote the homotopic shortest path for a given path π. Let X be a taut path from s to u (corresponding to the initial cable configuration when the robot is at point u), Y be a taut path from u to v, and Z be the cable configuration when the robot arrives at point v by following path Y (see Fig. 2.3). It can be observed that X,Y and Z are all polylines. The following properties have been proved in [BVX15].

(a). X, Y and Z are all homotopic shortest paths.

(b). The polygon formed by their subpaths does not contain any points from $\mathcal{A} \cup \mathcal{T} \cup \mathcal{V}_{\mathcal{O}}$ in its interior. Specially, it can be expressed as a pseudotriangle $\triangle abv$, where ab, bv and av are all convex chains with respect to the exterior of $\triangle abv$.

(c). Following a, for any point $t \in ab$, let vt be the shortest path contained in $\triangle abv$, then $|vt| \leq |av|$ and $|vt| \leq |bv|$.

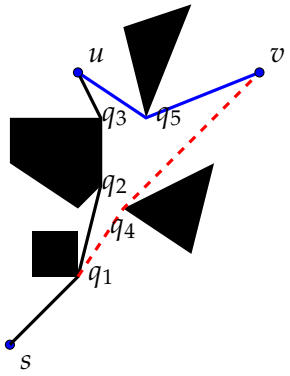


FIGURE 2.3: The geometric illustration of cable configurations and homotopic shortest path. $X = sq_1q_2q_3u$, $Y = uq_5v$, and $Z = sq_1q_4v$.

Specifically, the property (c) is interesting when considering the constraint that the tether cannot exceed its maximum length ℓ_{max}. It suffices to ensure $|X| \leq \ell_{max}$ and $|Z| \leq \ell_{max}$, then Y is an admissible path. The geometric properties are all inferred based on the the fact that the workspace and the obstacles are all polygonal. In [Yan+22], this hypothesis is generalized to any convex workspace.

Moreover, the common tasks in the path planning problem for tethered robots listed below can be regarded as the same.

- Given the initial and final cable configuration, compute the shortest path;
- Given the initial cable configuration and the path, compute the target cable configuration;
- Given a non taut path, shorten it to the homotopic shortest path.

As for the third task, we can imagine a path $av.vb$ that connects X and Y and is homotopic to Y. The computation of Y is exactly a shortening process of a non taut path to a taut one. It can be solved by using the funnel algorithm proposed by Hershberger and Snoeyink [HS94].

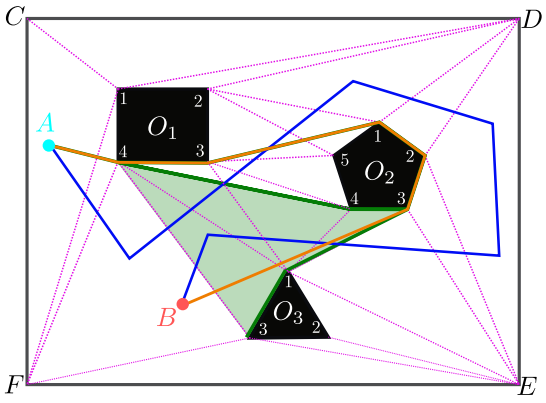


FIGURE 2.4: An example of the funnel algorithm. We have a given path π (in blue), which can be reduced to its homotopic shortest path $\langle A, O_{1-4}, O_{1-3}, O_{2-1}, O_{2-2}, O_{2-3}, B\rangle$ (in orange). The resulting funnel is displayed in green, where its boundary is composed by a tail $\langle A, O_{1-4}, O_{1-3}, O_{2-1}, O_{2-2}, O_{2-3}\rangle$, a left side boundary $\langle O_{2-3}, O_{2-4}, O_{1-4}\rangle$ and a right side boundary $\langle O_{2-3}, O_{3-1}, O_{3-3}\rangle$.

The funnel algorithm Given a path π, the funnel algorithm computes the shortest path homotopic to π, denoted as $H(\pi)$. The general idea is to find a convex hull, also called funnel that contains the shortest path. An example [1] is shown in Fig. 2.4. We briefly summarize the algorithm. It processes in two steps. Firstly, the workspace is decomposed into a set of triangles by a triangulation operation. The boundary of the funnel is constituted by the edges from these triangles, and its left side and right side correspond respectively to two homotopic shortest paths. Then, we start from the first vertex of π, and check the intersection of each line segment with these triangles. If π intersects with a triangle and leaves it immediately through the same edge, then the boundary can be tightened. The objective is to construct the tightest boundary that contains the homotopic shortest path of π. Once this funnel is found, we can simply connect the last vertex of π with the vertices on the tail of the funnel in order to build the shortest path. In terms of complexity, let $|\mathcal{V}_\mathcal{O}|$ be the number of vertices of the polygons, k be the length of π, and N_π be the number of times that π crosses a triangulation edge. The triangulation process requires $\mathcal{O}(|\mathcal{V}_\mathcal{O}| \log |\mathcal{V}_\mathcal{O}|)$ time by the method of [HM85]. The preprocessing steps, including workspace triangulation and visibility graph computation, are executed just once, and the computation of $H(\pi)$ can be completed in $\mathcal{O}(N_\pi + k)$ time [HS94].

[1] We reuse the example from https://medium.com/@reza.teshnizi/the-funnel-algorithm-explained-visually-41e374172d2d, where a more detailed explanation with animation can be found.

The method mentioned above is an exact approach. In some cases we only need to know the upper bound of the length of $H(\pi)$, or when the workspace contains circular obstacles, it becomes too expensive to calculate an accurate solution. In such cases, the approximated methods are proposed. For example, in [KBK14], it executes a function *CurveShorten* to shorten a path and leads to an approximation of $H(\pi)$. The general idea is to sample the points on the path and sequentially join the vertices using straight line segments until the line segment intersects an obstacle. If an intersection occurs, we then try to connect the last traced point to subsequent points until we reach the final point. The resulting solution has a longer euclidean distance than $H(\pi)$.

2.1.4 Detecting crossing paths

Whenever two paths π_1 and π_2 have a non-empty intersection, we need to decide whether they are crossing or not. The different cases of non-empty intersection paths are illustrated in Fig. 2.5. The case in Fig. 2.5(a) is trivial: when two paths share a same point that does not belong to any obstacle, then they are crossing.

When the intersection corresponds to an obstacle vertex, the two paths may be crossing or not, as illustrated in Fig. 2.5(b) and 2.5(c). To decide whether they are crossing or not, we consider the following lemma which is an adaptation of [HL97] to our context.

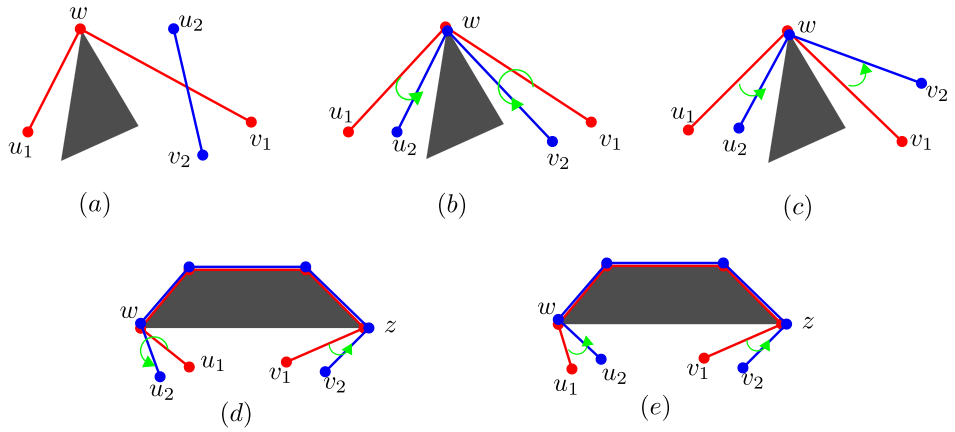


FIGURE 2.5: Different cases of two paths with a non empty intersection (angle sizes are represented by green arrows). (a): The paths are trivially crossing because the intersection is a point that does not belong to an obstacle. (b) and (c): the intersection is a single obstacle vertex w. (b) corresponds to non-crossing paths ($\measuredangle u_1wu_2 < 180°$ and $\measuredangle v_1wv_2 > 180°$), whereas (c) corresponds to crossing paths (both angles are lower than 180°). (d) and (e): the intersection is a sequence of segments from w to z. (d) corresponds to non-crossing paths ($\measuredangle u_1wu_2 > 180°$ and $\measuredangle v_1zv_2 < 180°$), whereas (e) corresponds to crossing paths (both angles are lower than 180°).

Definition 2.1.5 (Intersecting paths)**.** Let us consider two paths $\pi_1 = \langle u_1, w, v_1\rangle$ and $\pi_2 = \langle u_2, w, v_2\rangle$ such that $\{u_1, v_1\} \cap \{u_2, v_2\} = \emptyset$ (*i.e.*, the intersection of π_1 and

π_2 contains the single vertex w). π_1 and π_2 are crossing if and only if $\measuredangle u_1wu_2$ and $\measuredangle v_1wv_2$ are either both lower than 180° or both greater than 180°.

This may be extended in a straightforward way to the case where the intersection is a sequence of segments π_3 instead of a single vertex, *i.e.*, $\pi_1 = \langle u_1\rangle.\pi_3.\langle v_1\rangle$ and $\pi_2 = \langle u_2\rangle.\pi_3.\langle v_2\rangle$ and $\{u_1, v_1\} \cap \{u_2, v_2\} = \emptyset$. Let w and z be the first and last vertices of π_3, respectively. In this case, the two paths are crossing if and only if $\measuredangle u_1wu_2$ and $\measuredangle v_1zv_2$ are either both lower than 180° or both greater than 180°, as illustrated in Fig. 2.5(d) and 2.5(e).

Hence, we can decide whether two paths are crossing or not in $\mathcal{O}(k^2)$ where k is the maximum number of segments in a path. Indeed, we first check in $\mathcal{O}(k^2)$ that there is no crossing segments (there are $\mathcal{O}(k^2)$ pairs of segments and we check if two segments are crossing in constant time). Then, we search for all common subpaths in $\mathcal{O}(k)$, and for each common subpath we check that there is no crossing by measuring two angles in constant time.

2.2 Tethered Robot Path Planning Problem

2.2.1 The shortest path problem

The single tethered robot planning problem focuses on navigating a robot through an obstacle-filled workspace while satisfying cable constraints such as length limitations and avoiding entanglements. Numerous studies have tackled this problem by aiming to find the shortest path given the initial cable configuration and the target point [Xav99; SH15; BVX15] (where the target cable configuration is undefined). More precisely, given an anchor point r, a starting point s, an initial cable configuration which is a homotopic shortest path from r to s, and a target point t, the goal is to find a shortest path from s to t such that the cable length limit ℓ_{max} is not exceeded. These methods can be categorized into the two following classes.

The geometric methods

The first family is based on the geometric simplification. When the workspace is convex and the obstacles are polygonal, the properties presented in Section 2.1 can be applied to find the optimal solution. In [Xav99], Xavier introduced an algorithm that allows one to compute an optimal solution by searching for all admissible paths with respect to the limited cable length. The pseudocode of this algorithm is displayed in Algorithm 1 and is illustrated in Fig. 2.6.

Let X be the initial cable configuration. We denote $X_{u,v}$ the section of the path lying on X between points u and v, where u and v denote two distinct points on X. Additionally, we refer to $SP(a, b)$ as the untethered shortest path between any a and b within the workspace, determined using common shortest path algorithms such as Dijkstra's. As a preliminary step, this algorithm requires a full knowledge of the workspace's visibility details and takes the visibility graph G_{vis} as an input.

Algorithm 1: SHORTESTPATH($\mathcal{G}_{vis}, X, t, \ell_{max}$) [XAV99]

Input: The visibility graph $\mathcal{G}_{vis} = (\mathcal{V}, \mathcal{E})$ where the workspace is represented, the initial cable configuration X: a polyline that connects r (the anchor point) and s (the start point), the target point t, and the maximum cable length ℓ_{max}

Output: The shortest path from s to t

```
Traverse from s to r, find all sections {S_i}
for each section S_i = X_{x_{i-1},x_i} do
    Find V_i corresponding to S_i
    for each vertex v ∈ V_i do
        Compute the taut path H(X_{r,x_i} ∘ x_i v ∘ SP(v,t))
        if |H(X_{r,x_i} ∘ x_i v ∘ SP(v,t))| ≤ ℓ_max then
            Compute a candidate path π_cand ← H(X_{s,x_i} ∘ x_i v ∘ SP(v,t))
            if |π_cand| < |π_opt| then update π_opt ← π_cand;
        end
    end
end
Return π_opt
```

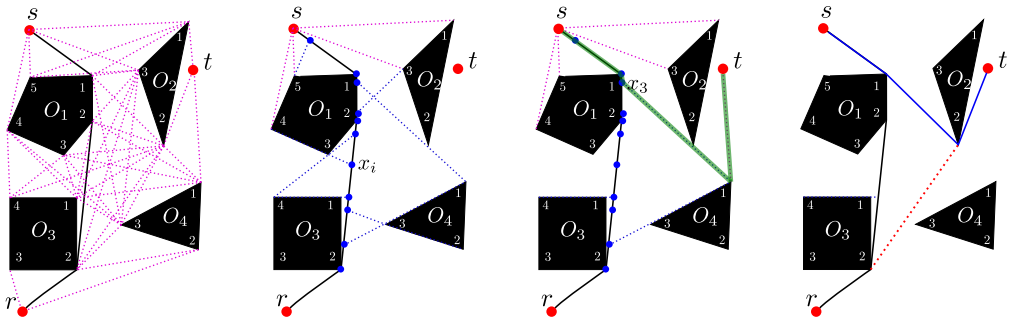


FIGURE 2.6: An illustration of Algorithm SHORTESTPATH. (a) A problem instance and the associated visibility graph. (b) X is divided into a set of sections $\{S_i\}$. (c) At the event point c, the terminal O_{4-1} becomes visible, the green path refers to $X_{s,x_3} \circ \overline{x_3 O_{4-1}} \circ SP(O_{4-1}, t)$. (d) The optimal path in $\langle s, O_{1-1}, O_{2-2}, t \rangle$ (in blue).

To outline the procedure, we initiate traversal of X from point s to r in a reverse manner. For each point $u \in X$, G_{vis} aids in identifying the set of vertices that are visible from u. For example, the vertices set at s is $\{O_{1-1}, O_{1-4}, O_{1-5}, O_{2-1}, O_{2-3}\}$. The key concept involves partitioning X into a set of sections $\{S_i\}$ such that for every point on $\{S_i\}$, this visible vertex set remains consistent, as depicted in Fig. 2.6(b). The distinct event points, denoted by the blue points x_i, mark the instances of change, with $x_0 = s$ and $x_{\#x_i} = r$. Each section S_i is represented as X_{x_{i-1},x_i}. Within each section S_i, we establish a set of vertices V_i, which includes (i) every visible vertex if $i = 0$; and (ii) any visible vertex that was not visible in the preceding section. For instance, in Fig. 2.6(c), $V_3 = O_{4-1}$ since O_{4-1} becomes visible after traversing x_3. For each such vertex v, we connect it to t via the shortest path $SP(v, t)$. Subsequently, a candidate path can be generated as $H(X_{s,x_i} \circ \overline{x_i v} \circ SP(v, t))$, which is admissible

if the corresponding target cable configuration $H(X_{r,x_i} \circ \overline{x_i v} \circ SP(v,t))$ does not exceed ℓ_{max} (see lines 5-7). The optimal solution is the shortest one among all these candidate paths.

The complexity of this algorithm is $O(k_l k n^3)$, where k_l is the number of loops in X, $k = \#X$ and $n = |\mathcal{V}|$ (for a detailed analysis see Section 3.3 of [Xav99]). This complexity has been improved to $\mathcal{O}(kn^2 \log n)$ in [BVX15] by (i) using a binary search to reduce the number of computed event points; (ii) preprocessing X with a special data structure to speed up the taut path computation.

The graph search based methods

Motion planning using homotopy class has been frequently applied to solve the tether robots path planning problem. In a traditional shortest path finding problem, we have a graph G that represents the accessibility between the adjacent locations, like *Region Adjacency Graph*, and graph search based methods, such as A* and Dijkstra's can be applied to compute the optimal solution. The homotopy-augmented graph is an important notion that incorporates the homotopy information into the graph representation [IS10], and it can be defined as follows.

Definition 2.2.1 (Homotopy-augmented graph [KBK14])**.** Let $G = (\mathcal{V}, \mathcal{E})$ be a discrete representation of the workspace, r be a fixed initial point (in our context, it refers to the anchor point), the homotopy-augmented graph $G_h = (\mathcal{V}_h, \mathcal{E}_h)$ of G is defined such that every vertex in the form of $v = (q, w) \in \mathcal{V}_h$, where $q \in \mathcal{V}$ and w is the homotopy of the path taken from r to reach v, and every edge e connects two adjacent vertices (q_1, w_1) and (q_2, w_2) if $(q_1, q_2) \in \mathcal{E}$ with $w_2 = w_1 \circ h(\overline{q_1 q_2})$.

The interest of G_h lies in its ability to expand the representation of the configuration space by introducing a new dimension that tracks the topology of paths leading to a particular state. This expansion facilitates the fulfillment of cable-related constraints, such as limited cable length and the prohibition of cable crossings, by checking the validity of each state as well as the connectivity of two states. Depending on the structure of G, G_h can be constructed based on the grid graph [KBK14; Bha10], the visibility graph [SH15], or the Probabilistic Roadmap (PRM)[KF11; MH17]. Moreover, the utilization of the homotopy-augmented graph extends beyond just tethered robots. It has been also used to generate topological heuristics for some other robot path planning challenges [WSB23; Ran+20].

The homotopy-augmented graph can be constructed using Dijkstra's algorithm [KBK14]. Starting from (r, Λ), we explore all other reachable nodes that satisfy the cable constraints. For example, let (v_i, w_i) a node in G_h, then for every adjacent node v_j with $(v_i, v_j) \in \mathcal{E}$, (v_j, w_j) can be added to G_h if the cable configuration at v_j with homotopy class $v_j = w_i \circ h(\overline{v_i v_j})$ is less than ℓ_{max} and not self-crossing.

However, a limitation of the homotopy-augmented graph is its exponential growth with the increase in obstacles. The graph size generally scales as $\mathcal{O}(2^{\#\mathcal{O}}|\mathcal{V}|)$, influenced by factors like cable length and obstacle positions. Since it contains only

valid cable configurations, its size is also restricted by the non-crossing constraint. In Fig. 2.7, we demonstrate an example of homotopy-augmented visibility graph, denoted as G_{hvis}. Each node in G_{hvis} corresponds to a feasible cable configuration. In this example, two obstacles exist in the workspace, yielding a visibility graph with 10 vertices and a corresponding G_{hvis} containing 44 vertices. To address this challenge, heuristics play a crucial role in designing effective path finding algorithms.

The search for the shortest path from a start point to a target point involves running an A* algorithm from the source node representing the initial cable configuration. The search terminates upon reaching a node encoding the target point. While the Euclidean distance to the target is a common admissible heuristic, it can lead to local minima traps. To counteract this, a solution proposed by [KL15] introduces the Topology-based Multi-Heuristic A* as a variant of Multi-Heuristic A* (MHA*) [Ain+16]. The core idea of MHA* is to apply multiple arbitrarily inadmissible heuristics independently to explore the different regions of the search space, while preserving guarantees of completeness and sub-optimality bounds via a consistent heuristic. The topology-based MHA* can be efficient when dedicatedly solving the shortest path planning problems for tethered robots. In their work, the single admissible heuristic is chosen as the Euclidean distance to the target, and the other additional heuristics are designed based on different topological paths whose number is proportional to the number of obstacles.

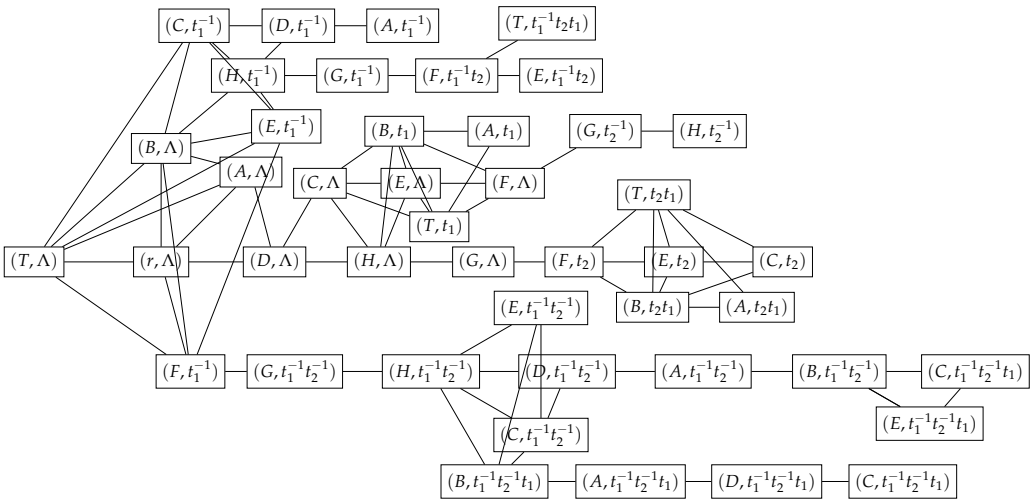


FIGURE 2.7: An example of the homotopy augmented visibility graph G_{hvis} associated with the workspace in Fig. 2.1.

.

2.2.2 Enumeration of non-homotopic paths

Another type of common task for tethered robots is to enumerate the non homotopic paths given the initial and target points, *i.e.*, to search for a set of paths each of which is the shortest path within a different homotopy class. An exhaust search frame, like *depth-first-search (DFS)* can be applied to enumerate the paths. In Algorithm 2,

we display an algorithm based on DFS that is adjusted to enumerate the different homotopic shortest paths subject to these cable constraints. Especially, in line 6, we define a function CHECKVALID() to check the validity of the current path after the addition of a new vertex. This function will check firstly whether the newly added edge intersects with other segments, and it also ensures that it is kept taut. Taking the example in Fig. 2.5(a), for a taut path, the convex angle formed by three successive vertices u_1, w, u_2 must contain the two segments on the obstacle and incident to w. Since the path p is extended incrementally, we check only the last three vertices of p each time it is called. In this algorithm, we also introduce a variable N_{max} to control the number of paths to be enumerated. In some cases, calculating all the paths may be unnecessary and costly, and only the shortest ones are of interest in real applications.

Algorithm 2: KNONHOMOTOPICPATHS($G_{vis}, s, t, \ell_{max}, N_{max}$)

Input: The visibility graph $G_{vis} = (\mathcal{V}_{vis}, \mathcal{E}_{vis})$, the start point s, the target point t, the maximum cable length ℓ_{max}, and N_{max} the number of paths to be generated
Output: The generated path list Π
Global Variables: A list of paths: Π, initialized to be empty; a set of $|\mathcal{V}_{vis}|$ Boolean variables *isVisited*, initialized to false.

```
1  Let p be an empty path
2  DFSENUMERATEPATHS(p, G_vis, s, t, ℓ_max, N_max)
3
4  Function DFSENUMERATEPATHS(π, G_vis, u, v, ℓ_max, N_max):
5      isVisited[u] ← true, add u to π
6      if CHECKVALID(π) and |π| ≤ ℓ_max then
7          if u == v then
8              add π to Π
9              if #Π > N_max then update Π and ℓ_max
10         else
11             for each w from the successors of u and isVisited[w] == false do
12                 DFSENUMERATEPATHS(π, G_vis, w, v, ℓ_max, N_max)
13             end
14         end
15     else
16         π.pop(), isVisited[u] ← false, return
17     end
```

In [Yan+22] the author introduced a method that allows one to enumerate k shortest non-homotopic paths. It is a kind of online planner, since no prior environment information, neither topological nor geometric, is known. In this work, a hierarchical topological tree is proposed as a novel representation of the workspace, which has the following two properties: (i) it exhaustively explores all topologies in the workspace, (ii) the concatenation of edges in this tree is proven to be a homotopically shortest path. A noteworthy contribution of this work is the incorporation

of a branch pruning mechanism. This mechanism effectively identifies partially constructed paths that cannot be extended to yield optimal k-shortest paths. By safely terminating path finding along these non-viable paths, the approach avoids the need to compute all possible paths and subsequently selects the k-shortest ones, therefore the computation time can be significantly reduced. Moreover, the approach can be generalized to handle a wide types of obstacles, not just polygonal ones. Additionally, the method demonstrates its capability to take into account the size of robots. However, in such cases, the path length is approximately computed.

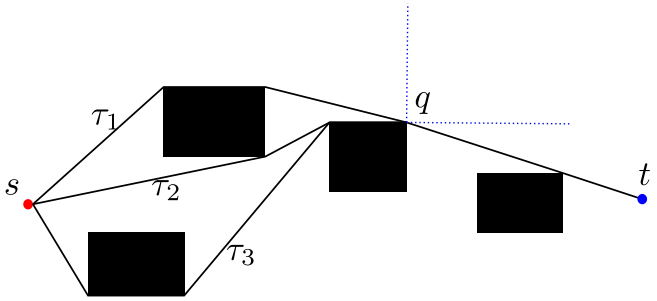


FIGURE 2.8: Example of discarding non-k-optimal paths.

It is worth noting that Algorithm 2 can be improved by integrating insights gleaned from [Yan+22]. In simple terms, we can safely eliminate some nodes during the exhaustive search process. As illustrated in Fig. 2.8, the aim is to find the two shortest non-homotopic paths from s to t. Here, we encounter three partially constructed, distinct, and non-homotopic paths denoted as τ_1, τ_2, τ_3 all converging at point q, with $|\tau_1| < |\tau_2| < |\tau_3|$. In this case, we can safely discard the intermediate path node τ_3, as it cannot be extended to form the 2 optimal paths. It is crucial to recognize that this conclusion rests on the geometric insight that these three paths must navigate around point q and converge in a specific direction. Therefore, while distance optimality is a significant factor, the geometric characteristics, particularly when considering the non-crossing constraints, must be equally taken into account when sorting these intermediate states. Besides, like A^*, for these unexpanded nodes, a cost can be assigned as the sum of the cost to the current node and a heuristic cost to the target point based on the Euclidean distance. If the cost of a node exceeds the value of the k paths already found, then we can safely prune this node.

In Chapters 4 and 5, when searching for an optimal solution to our specific problems, the enumeration of all paths becomes essential. Section 4.3.5 will showcase a specific application case where only a subset of candidate paths needs enumeration. While the improved version promises better theoretical efficiency, it involves additional computations on the heuristics, especially when N_{max} tends to be large number. In both cases, the vanilla implementation of Algorithm 2 proves sufficient.

2.2.3 Multiple tethered robot path planning

Multi-Agent Path Finding (MAPF) is a very active research topic which has important applications for robotics in industrial contexts such as transport in fulfillment centers or autonomous tug robots, for example. The goal of MAPF is to find a set of collision-free paths from starting points to target points. Usually, there is an objective function to optimize such as the duration of the longest path (called *makespan*), the sum of all path durations, or the number of agents that cannot reach their targets within a given makespan [Ma+18]. The problem is $\mathcal{NP}$-hard in the general case [YL13a]. Classical MAPF problems are often solved by using a two-level approach called *Conflict Based Search (CBS)* [Sha+15]: path searching at the low level and resolution of path conflicts at the high level.

In the case of *anonymous* MAPF (AMAPF), the target of each agent is not known, *i.e.*, there is a set of targets and each agent must be first assigned to a target before searching for paths [Ste+19]. Regardless of whether the goal is to minimize the makespan or the sum of all paths' costs, AMAPF can be generally reduced to a Network Flow problem, and solved in polynomial time by applying the maximum flow algorithms [For56; YL13b]. However, both CBS and the Network Flow-based method are limited in handling agents' collisions occurring at an instant or over a short period of time and cannot effectively address the topological constraints associated with cables.

When considering the case of multiple tethered robots, the challenge is that cables can easily get tangled, and avoiding cable crossings makes the planning harder.

One-to-many settings

[Sin90] introduces the tethered robot problem where robots may have to visit several points in a workspace that does not contain any obstacle. In their settings, there are n robots and m stations (equivalent to targets and $m \geq n$), and each robot must visit the m stations. Each time a station is assigned to a robot, and within each assignment, the robots move simultaneously from their anchor points to the assigned target points, then back to their anchor points along the tether position. The objective is to find a set of m assignments that each assign the n robots one by one to n of the stations, subject to the following two constraints: (1) in each assignment, no cable crosses itself, (2) among the m assignments, each robot is assigned to each station exactly once. This problem has been approached by formulating it as a bipartite drawing problem, where the goal is to find rectilinear drawings of a complete bipartite graph $K_{n,m}$ whose edges are colored with m colors so that no two edges of the same color cross or have a common vertex. As such a solution does not always exist, depending on the location of anchor points and stations, the author has studied some special cases where solutions do exist.

Motion coordination

In the work of [HL96; HL97], the workspace does not contain obstacles and robots cannot cross cables but they may push and bend the cables of other robots. Every robot is assigned to a unique target point. Since the interaction between robot and cable is allowed, the path (or the cable location) is no longer a simple straight line, but rather a sequence of segments. In this case, the primary objective involves searching for a set of feasible target cable configurations, denoted as $\mathcal{C} = \{C_1, ..., C_n\}$, with one for each robot, in other words, a set of non-intersecting curves connecting a set of point pairs in the plane. In order to reach $\mathcal{C}$, it is necessary to coordinate the robots' movements: (i) find the paths corresponding to $\mathcal{C}$, (ii) order the robots' movements to avoid collisions if two paths share a common vertex. This planning task is formulated as a computational geometry problem, and an algorithm with a time complexity of $\mathcal{O}(n^3 \log n)$ has been proposed. In addition, Hert & Lumelsky describe efficient algorithms for detecting whether two paths are crossing or not, from a purely geometric standpoint. The procedure that we use for detecting path crossings (described in Section 2.1.4) is adapted from this work. When the objective is to minimize the sum of all path lengths, it is proved to be $\mathcal{NP}$-hard. A similar methodology has been also applied to address planning challenges within a 3D workspace [HL99], where the initial and target points lie in a common 2D surface, and the anchor points are located in another parallel 2D surface. When the robot moves from one point to another, the cable configuration is a 3D straight line segment, and the surface swept by the cable is a triangle. An ordering between robots' movements allows to avoid entanglement.

[ZP19] considers the same problem as [HL96], but the workspace may contain obstacles and there is no objective function to optimize: they simply search for a valid schedule such that robots do not cross cables. To avoid crossings, precedence constraints and waiting times are introduced, and a precedence graph is used to detect deadlocks. Zhang & Pham propose algorithms for iteratively removing deadlocks. However, there is no guarantee that all deadlocks can be suppressed, and robots are constrained to diverge from straightline motions whenever there are non resolved deadlocks.

2.2.4 Coverage path planning

Coverage Path Planning (CPP) is a fundamental problem in robotics, and it has many applications such as automatic floor cleaning, area patrol, or rescue search. CPP can be formulated as follows. Given a planar 2D workspace with convex polygonal obstacles, and the size of a mobile robot, find a shortest coverage cycle that fully covers the workspace and returns to its starting point. To solve this problem, we usually discretize the workspace into a 4-connected grid graph g such that each cell of the grid has the same size as the robot. In this case, if there exists a Hamiltonian cycle in g, we actually have a shortest coverage path as each cell is traversed exactly

once. The complexity of deciding of the existence of a Hamiltonian cycle depends on the properties of g: it is in $\mathcal{O}(1)$ for rectangular grids with no obstacles whereas it becomes $\mathcal{NP}$-complete in case of obstacles [IPS82].

STC algorithm

Spanning Tree Coverage (STC) may be used to compute an approximate solution in polynomial time [GR01]: we construct a graph G_4 such that each vertex of G_4 corresponds to a non-overlapping group of 2×2 adjacent cells in g, and edges of G_4 correspond to adjacency relations between these 2×2 cell groups; if each cell of g belongs to exactly one 2×2 cell group, then there exists a Hamiltonian cycle in g if and only if G_4 is connected and, given a spanning tree T of G_4, a Hamiltonian cycle in g can be constructed by circumnavigating T, as illustrated in Fig. 2.9.

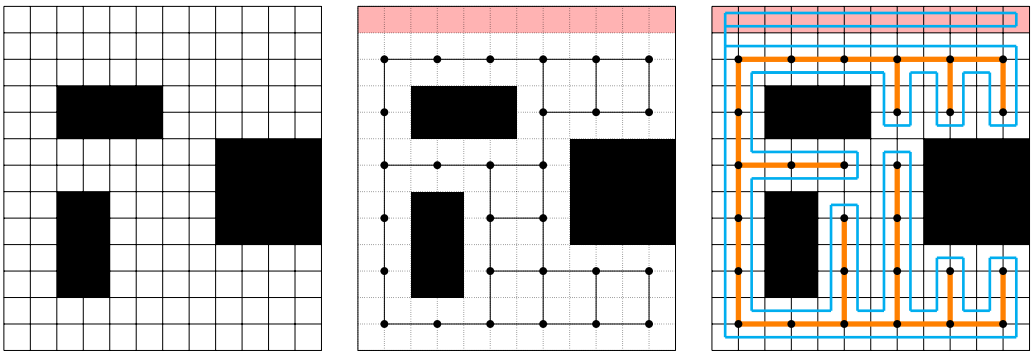


FIGURE 2.9: Illustration of the STC algorithm. (a) An example of a workspace discretized in a grid g. (b): The vertices of G_4 are represented by the black circles, and each of them is at the center of the 4 cells surrounding it. In this example, G_4 does not cover the top row of grid cells. (c) A spanning tree of G_4 is displayed in orange and the path in blue is an approximate solution that visits all cells but those in the top row exactly once: the red cells are visited twice by a forth and back path.

Tethered coverage problem

In the state of the art of Tethered Coverage (TC) problem, existing algorithms can be broadly categorized as online or offline, depending on the level of knowledge about the environment. The paper [SR14] presents an offline TC algorithm that addresses this problem when the robot has a complete knowledge of the environment. Their approach involves decomposing the environment into "split cells" and "corridor" structures. These split cells are stored in a stack, and whenever the robot reaches a new split cell, it conducts a complete coverage of the corresponding corridor. Once the coverage is finished, the robot proceeds to the next split cell, repeating the process for the remaining corridors. Ultimately, the robot returns to the anchor point with its cable fully retracted.

The work in [Mec+17] presents a solution to the offline TC problem using a specific decomposition of the environment derived from the "Morse-based Cellular Decomposition" [AC02]. The grid map is divided into interconnected rectangular shapes around the obstacles, and each shape is covered using a zig-zag motion pattern. However, it is important to note that their primary objective is to cover all the accessible areas, and there is no specific focus on minimizing the overall length of the coverage path.

The online version of TC problem has been also studied in [Sha+19; SR14] where the robot has no prior knowledge of the environment. Under this settings, the environment is incrementally explored while a tree map is simultaneously built to keep track of the frontiers of uncovered area. In [SR14], an approximation algorithm is proposed with a factor $\frac{2L}{D}$ compared to the minimum path length for the coverage, and this result is improved to $2(1 - \frac{1}{N})$ in [Sha+19], where L is the cable length, D is the cell size, and N is the total number of accessible obstacle-free cells.

2.3 Discussion

In this chapter, we provided an extensive review of the geometry foundations in the tethered robots path planning problem, as well as its diverse application scenarios in the literature. Table. 2.1 summaries the different applications, categorized based on the task type, such as shortest path search (SP), coverage path planning (CPP) or travelling salesman problem (TSP), and the types of robots employed in these applications, ranging from a single robot to multiple robots, drones and axel rovers.

The path planning problem for tethered robots generally presents inherent challenges, making it notably complex to solve. In addition to traditional graph search based and geometric methods, Chapter 3 will introduce some mathematical programming techniques used to tackle these complexities.

	SP	CPP	TSP	Others
single robot	[Xav99], [SH15], [BVX15], [SP20], [KBK14], [Bha10], [SH15], [KL15], [TS14], [YXW22], [IS10], [TBN13]	[SR14], [Sha+19], [Mec+17]	[MH17], [YXW22]	[WB18]
duo robots	[TS21]			[Bha+15], [Su+22]
multiple robots	[PSS21], [PSS23]		[Sin90]	[HL96], [HL97], [HL99], [ZP19]
single drone			[PD22]	[Xia+18]
multiple drones				[Cao+23a], [Cao+23b]
axel rover	[ANB11]			

TABLE 2.1: Summary of tethered robots applications in the literature. SP: shortest path search; CPP: coverage path planning; TSP: traveling salesman problem.

Chapter 3

Mathematical Programming

Contents

3.1 Constraint Programming 25
3.1.1 Basic Notions 25
3.1.2 CSP Solving 26
3.1.3 Implementation of Global Constraints with Choco 29
3.2 Integer Linear Programming 29
3.3 Discussion 31

In this chapter, we explore two widely used declarative approaches for solving combinatorial problems: Constraint Programming (**CP**) and Integer Linear Programming (**ILP**). Declarative modeling involves describing problems through a set of variables and a set of constraints defining relations between variables. We will provide a comprehensive overview of the fundamental concepts and the solving procedures associated with each approach.

3.1 Constraint Programming

3.1.1 Basic Notions

Definition 3.1.1 (Constraint [RVW06])**.** Let $X = (x_1, x_2, ..., x_k)$ be a k-tuple of variables, and for each variable x_i, let $D(x_i)$ be the domain of x_i, *i.e.*, the set of values that may be assigned to x_i. A constraint c on X is a subset of the Cartesian product of the domains of the variables in X, i.e., $c \subseteq D(x_1) \times ... \times D(x_k)$. X is called the scope of c.

In this chapter, we consider only the constraints where each involved variable has a finite domain. A constraint c may be defined in extension, by listing all tuples in c, or in intention, by using mathematical operators. For example, when $D(x_1) = D(x_2) = D(x_3) = \{0, 1, 2\}$, to constraint x_3 to be equal to the sum of x_1 and x_2, we may either define a constraint in extension

$$(x_1, x_2, x_3) \in \{(0, 0, 0), (0, 1, 1), (0, 2, 2), (1, 0, 1), (1, 1, 2), (2, 0, 2)\}$$

or a constraint in intention $x_1 + x_2 = x_3$.

Definition 3.1.2 (Variable assignment)**.** A variable x_1 is said to be assigned whenever its domain is reduced to a singleton, *i.e.*, $D(x_i) = \{v_i\}$. In this case, we note $v(x_i)$ the single value in $D(x_i)$.

Definition 3.1.3 (Constraint satisfaction)**.** A constraint c defined on a tuple $(x_1, ...x_k)$ is satisfied if all its variables are assigned and the tuple $(v(x_1), ..., v(x_k))$ belongs to c.

Definition 3.1.4 (Constraint Satisfaction Problem (CSP) [RVW06])**.** A *constraint satisfaction problem (CSP)* is defined by a triple $\mathcal{P} = \langle X, D, C \rangle$ where

- $X = \{x_1, x_2, ..., x_n\}$ is a set of n variables.
- For each variable $x_i \in X$, $D(x_i)$ is its domain.
- C is a set of constraints.

A solution of a CSP is an assignment of all variables that satisfies all the constraints in C simultaneously. If no solution exists, the CSP is said to be inconsistent.

Example 3.1.1 (8-queens)**.** Let us consider the 8-queens problem: how to place the 8 queens on 8×8 chessboard so that no two queens share the same row, column or diagonal. It can be formulated as the following CSP. There are 8 variables $X = \{x_1, x_2, ..., x_8\}$, for each variable x_i, its domain is $D(x_i) = \{1, 2, ..., 8\}$. Assigning x_i to a value $j \in D(x_i)$ means placing a queen in row j and column i. For any two variables x_i and x_j, we post a constraint that ensures that the queens at columns i and j are not in a same row or diagonal, *i.e.*,

$$x_i \neq x_j \wedge x_i + i \neq x_j + j \wedge x_i - i \neq x_j - j$$

One possible solution is given by $\{x_1 = 1, x_2 = 7, x_3 = 5, x_4 = 8, x_5 = 2, x_6 = 4, x_7 = 6, x_8 = 3\}$.

Definition 3.1.5 (Constrained Optimization Problem (COP) [RVW06])**.** A *constrained optimization problem (COP)* is a CSP $\mathcal{P}$ defined on a set of of variables $X = \{x_1, x_2, ..., x_n\}$, together with an *objective function* f:$D(x_1) \times ... \times D(x_n) \rightarrow \mathbb{R}$ that assigns a value to each assignment of values to the variables. An *optimal solution* to a minimization (maximization) COP is a solution d to $\mathcal{P}$ that minimizes (maximizes) the value of $f(d)$.

3.1.2 CSP Solving

The constraint satisfaction problem is $\mathcal{NP}$-complete in the general case. Solving a CSP requires an exhaustive search, where all potential assignments should be enumerated. This search stops either upon finding a feasible solution, or when it concludes that no solution exists after a complete search. This blind enumeration technique can be improved by applying two orthogonal techniques: ordering heuristics and constraint propagation.

Ordering heuristics

The search space is usually explored with a depth first search (DFS). More precisely, at each node of the search:

- if some constraints are not satisfied, then we backtrack;
- otherwise, if all variables are assigned, then we have found a solution;
- otherwise, we choose an unassigned variable x_i, and a value v_i in $D(x_i)$, and we recursively explore two nodes, corresponding to $x_i = v_i$ and $x_i \neq v_i$, respectively (alternatively, we may create one branch for each possible value in $D(x_i)$).

Variable ordering heuristics are used to choose the next variable to assign. For example, classical variable ordering heuristics are *minDomain* or *wdeg* [Bou+04] which associates weights with constraints in order to identify critical variables.

Value ordering heuristics are used to choose the value v_i in the domain of x_i. For example, if the goal is to minimize a sum of variables, we may first try to assign small values to variables involved in the sum.

Propagate

In the search phase, when a variable is instantiated, inferences can be made to tighten the domains of other variables involved in the same constraints as this variable. This process is known as *constraint propagation*.

Constraint propagation removes values which are locally inconsistent. For example, when a variable x is assigned to 1, we can propagate the constraint $x \neq y$ by removing from the domain of y the value 1. Whenever constraint propagation wipes out a domain, this implies that the current node cannot be extended to a solution and the search can backtrack.

Different levels of constraint propagation may be considered, each of them ensuring a different level of local consistency. A classical local consistency is the Generalized Arc Consistency.

Definition 3.1.6 (Generalized Arc Consistency (GAC) [RVW06])**.** A constraint C is ***generalized arc consistent*** if for every variable x_i and every value $v \in D(x_i)$, there exists a tuple $(d_1, d_2, ..., d_k) \in D(x_1) \times D(x_2) \times \cdots \times D(x_k) \cap C$ such that $d_i = v$. A CSP is GAC if each of its constraints is GAC.

There exist also some other levels of consistency, such as node consistency and path consistency [Mac77; Lec96]. The choice of consistency level is a trade-off between the filtering quality and efficiency, and it also depends on the nature of the constraint.

There are different methods to establish **(G)AC**, such as AC3 and AC2001 [RVW06]. The general idea of AC3 is to examine each constraint once, and re-examine a constraint if a variable in its scope has changed. The complexity of AC3 is in $\mathcal{O}(ed^3)$ time

and $\mathcal{O}(e)$ space, where e is the number of arcs [1] (linear to the number of constraints in case of binary constraints) and d is the size of the largest domain.

Example 3.1.2 (AC Example)**.** Consider a CSP example, where $X = \{x_1, x_2, x_3\}$, $D(x_1) = D(x_2) = D(x_3) = \{1, 2, 3, 4\}$, with two constraints $c_1 : x_1 < x_2$ and $c_2 : x_2 < x_3$. This CSP problem can achieve **AC** by executing the following steps.

1. Remove 4 from $D(x_1)$ since no value v_2 exists from $D(x_2)$ such that $(4, v_2)$ satisfies the constraint C_1.
2. Remove 4 from $D(x_2)$, similarly.
3. Remove 1 from $D(x_2)$ to make $x_1 < x_2$ consistent.
4. Remove $\{1, 2\}$ from $D(x_3)$, similarly.
5. Remove 3 from $D(x_1)$, as a consequence of step 2.

As a result, these domains can be reduced to $D(x_1) = \{1, 2\}$, $D(x_2) = \{2, 3\}$, $D(x_3) = \{3, 4\}$.

In some applications, the utilization of global constraints can better address the structure of a problem, especially when specifying a complex relationship over a set of variables. For instance, Example 3.1.1 can also be modeled using the global constraint **AllDifferent**$(x_1, x_2, ..., x_8)$. The application of global constraints becomes particularly interesting if it can improve the filtering process. Consider the AllDifferent constraint, which can be decomposed into a set of binary constraints in polynomial time, and we can filter domains to ensure GAC for each of these constraints. However, this does not ensure the generalized arc consistency of the global constraint. For example, if $D(x_1) = D(x_2) = (a, b)$ and $D(x_3) = (a, b, c)$, the three difference constraints $x_1 \neq x2$, $x_2 \neq x_3$, and $x_1 \neq x_3$ are already arc consistent. However the global alldifferent constraint is not GAC as the tuples that satisfy the alldifferent constraint are (a, b, c) and (b, a, c), therefore, we should remove a and b from $D(x3)$.

The complexity of enforcing GAC for global constraints varies based on the nature of the constraint. In [Rég94], an efficient algorithm is outlined to enforce GAC on **AllDifferent** by linking it to the problem of finding maximal matchings in a bipartite graph, solvable in polynomial time. Additionally, there are specialized algorithms like STR2 [Lec11] dedicated for enforcing GAC on specific constraints like *table constraint*. However, enforcing GAC some constraints (such as **NValue**[2]) is $\mathcal{NP}$-hard [Bes+07]. In these cases, while enforcing full GAC might be computationally expensive, enforcing a weaker level of consistency remains feasible.

[1] An arc refers to a variable-constraint pair (x_i, c) where $x_i \in X(c)$.

[2] **NValue**$(N, X = \langle x_1, x_2, ..., x_n \rangle)$ counts the number of different values assigned to the variables in X and sets it equal to N.

3.1.3 Implementation of Global Constraints with Choco

Choco[PFL16] is a java library for CSP, which is widely used in the community of constraint programming. This section provides a brief introduction about how to implement a global constraint in Choco.

When designing a global constraint, two fundamental methods must be implemented.

- **void propagate (int evtmask)**: This method serves as a primary filtering algorithm of the propagator, starting from scratch. It gets invoked once during the initial propagation.
- **ESat isEntailed()**: It is used to check the state of the constraint according to the current status of the variables. It returns *ESat.TRUE* if the constraint is known to be always satisfied whatever happen on the variables from now on, *ESat.FALSE* if it is violated, and *ESat.UNDEFINED* if no definitive conclusions can be drawn.

In addition, for some constraints, we can also implement an incremental propagator, reacting to changes in specific variables, if necessary.

- **void propagate (int varIdx, int mask)**: The modification of a certain variable (denoted by varIdx) allows to filtering the inconsistent values in other variable domains. It is called if a particular variable "varIdx" is changed since the last call.

Besides the variables directly constituting the constraint, in some cases, it is necessary to define and maintain several intermediate variables with a storable data structure. Choco provides objects, like *IStateInt* and *IStateIntVector*, which are defined by a primitive value and an associated timestamp. These objects help in backtracking the variables to their previous states.

3.2 Integer Linear Programming

Integer Linear Programming (ILP) is a specical case of COP where all constraints are linear inequalities and the objective function is linear. The standard form of an ILP model is defined as follows.

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \qquad \text{integer} \end{aligned}$$

where $\mathbf{b}, \mathbf{c} \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$.

The ILP model represents a specialized case of linear programming, the difference being that variables in an ILP model must take integer values. In many applications, fractional solutions are not desirable; for example, the 0-1 programming

approach is commonly used to solve problems such as the knapsack problem. In addition, when dealing with graph constraints, the ILP formulation often exhibits inherent simplicity.

Example 3.2.1 (GMSTP on an undirected graph [MLT95])**.** Consider the generalized minimum spanning tree problem (GMSTP) (we will see its application in Chapter 7) on an undirected graph $G = (V, E)$, where V is partitioned into m mutually exclusive node sets, as $V = V_1 \cup V_2 \cup ... \cup V_m$. Each edge $e = (i, j) \in E$ is assigned with a positive cost c_e. The objective of GMSTP is to find a minimum-cost tree T that spans a subset of nodes, including exactly one node from each node set. We define a variable x_e for each edge e, such that $x_e = 1$ if edge e is included in T and 0 if not. Additionally, $y_i = 1$ indicates that node i is selected and 0 otherwise. For any subset of nodes $S \subset V$, we denote $\delta(S) = \{e \in E \mid \#e \cap S = 1\}$ as the collection of edges incident to exactly one node in S. The GMSTP can be formulated as the following ILP.

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e & & \\
\text{s.t.} \quad & \sum_{i \in V_k} y_i = 1 & & \text{all } 1 \leq k \leq m \\
& \text{// To ensure that } T \text{ is connected} & & \\
& \sum_{e \in \delta(S)} x_e \geq y_i + y_j - 1 & & i \in S \text{ and } j \notin S, \forall S \subset V \\
& \sum_{e \in E} x_e = m - 1 & & \\
& y_i, x_e \in \{0, 1\} & & \text{all } e \in E, i \in V
\end{aligned}
\tag{3.1}
$$

Solving **ILP** is $\mathcal{NP}$-complete in the general case. The two most common methods for **ILP** solution are Branch and Bound [LW66] and the cutting-plane method [Gom58]. The Branch and bound is based on the principle that the feasible solutions sets can be divided into small subsets of solutions represented as a tree structure. Systematic evaluation is performed on these subsets until the best solution is found. The branching step divides the subsets and the bound step aims at pruning useless branches by solving relaxations of the problem.

The cutting-plane method is derived from the simplex algorithm [Dan90]. The general idea is to create linear relaxations on the integer constraints, and solve it on $\mathbb{R}^n$. If this solution is not an integer point, a hyperplane (cutting plane) is added as an additional linear constraint, separating this solution from all feasible integer points. The new problem is then solved and the process is repeated until an integer solution is found.

3.3 Discussion

In this chapter, we have introduced the fundamental concepts of Constraint Satisfaction Problems and Integer Linear Programming, both of which are widely used in solving combinatorial optimization problems. In this thesis, our primary focus lies on the non-crossing AMAPF problem for tethered robots, that will be elaborated upon in Section 4 and Section 5. We will show how this problem can be addressed by CP models. We also explore the potential utilization of global constraints to express the computation of makespan in Chapter 5. Furthermore, in Chapter 6, within the context of searching a coverage path based on a spanning tree, we will formulate the tree constraint using an ILP model.

Chapter 4

Non-Crossing AMAPF for point-sized robots

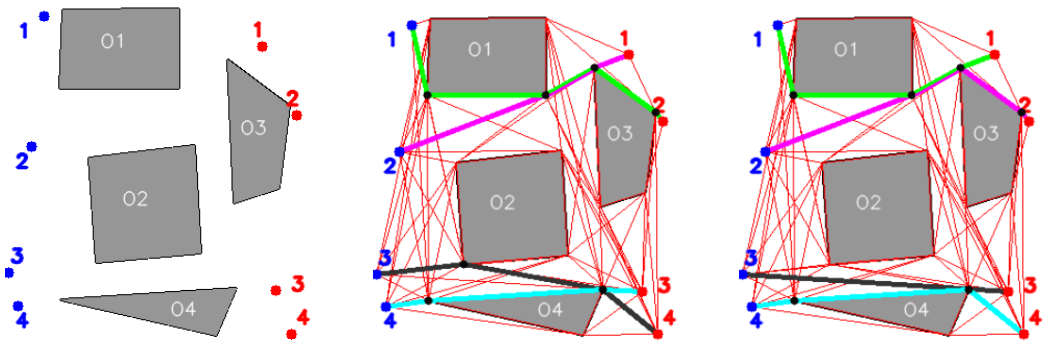
Contents

4.1 Definition of the NC-AMAPF Problem 34
4.2 NC-AMAPF Problem without Obstacles 35
4.2.1 Computation of Bounds by Solving Assignment Problems . 35
4.2.2 Variable Neighbourhood Search 36
4.2.3 Constraint Programming Model 37
4.2.4 Experimental Evaluation 37
4.3 NC-AMAPF Problem with Obstacles 39
4.3.1 Computation of Bounds 40
4.3.2 Relevant Paths Enumeration 43
4.3.3 Constraint Programming Model 43
4.3.4 Description of Benchmarks 44
4.3.5 Experimental Evaluation 45
4.4 Conclusion 48

In Chapter 2, we have presented the geometric basis for the tethered robot path planning problem. In this chapter, we introduce a new MAPF problem for tethered robots, as NC-AMPAF problem, which aims to find paths for multiple tethered robots while ensuring theirs cable do not cross. In this work, we assume that these robots are considered as point-sized, and our problem is formulated in a discrete visibility graph (see in Section 4.1).

In Section 4.2, we first consider the case where the workspace has no obstacle. We show that the NC-AMAPF 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 4.3, we consider the case where the workspace has obstacles. We prove that optimal solutions of assignment problems still provide bounds in this case. We



(a) (b) (c)

FIGURE 4.1: (a): Example of workspace $\mathcal{W}$ with four anchor points (in blue) and four targets (in red). (b): Visibility graph with paths that are not solution of the NC-AMAPF 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-AMAPF, even though the green and pink paths share a segment, and the black and blue paths share a vertex.

also show that the optimal solution of the NC-AMAPF 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 4.2 and 4.3, we report experimental results on randomly generated instances and show that our approach scales well enough to solve realistic instances within a few seconds.

4.1 Definition of the NC-AMAPF 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 a workspace $\mathcal{W} \subset \mathbb{R}^2$, and a set of n targets. The goal is to find a path in $\mathcal{W}$ for each robot from its anchor point to a different target so that the longest path is minimised and robots never have to cross cables.

In this chapter, we assume we have point-sized robots. In other words, robots are small enough and cables are thin enough to allow a robot to slightly push the cable of another robot when it has to pass between this cable and an obstacle, provided that cables do not cross. For example, if the black robot (starting from 3) in Fig. 4.1(c) arrives on the vertex of obstacle $\mathcal{O}_4$ 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 $\mathcal{O}_4$ and the black cable.

Let us now formally define our problem.

Definition 4.1.1 (NC-AMAPF problem)**.** An instance of the NC-AMAPF problem for tethered robots is defined by:

- a workspace $\mathcal{W}$;
- a set $\mathcal{A} \subseteq \mathcal{W}$ of n different anchor points (also corresponding to starting points);
- a set $\mathcal{T} \subseteq \mathcal{W}$ of n different targets such that $\mathcal{A} \cap \mathcal{T} = \emptyset$;

A solution is a couple (m, Π) such that:

- $m : \mathcal{A} \to \mathcal{T}$ is a bijection that assigns a different target to each anchor point;
- Π is a set of n paths such that (i) for each anchor point $a \in \mathcal{A}$, Π contains a path from a to its assigned target $m(a)$; (ii) paths in Π do not cross.

Given a solution $s = (m, \Pi)$, we denote $makespan(s)$ the makespan the length of the longest path in s, *i.e.*, $makespan(m, \Pi) = max_{\pi \in \Pi} |\pi|$. An optimal solution is a solution $s = (m, \Pi)$ such that $makespan(s)$ is minimal.

When the workspace $\mathcal{W}$ has no obstacle, we will show that our problem is equivalent to the bottleneck matching problem with edge-crossing constraints which has been shown to be $\mathcal{NP}$-hard by [Car+15]. Hence, our problem is also $\mathcal{NP}$-hard in the more general case where $\mathcal{W}$ contains obstacles.

4.2 NC-AMAPF Problem without Obstacles

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

In Section 4.2.1, we show how to compute lower and upper bounds by solving well known assignment problems. In Section 4.2.2, we show how to improve the upper bound by performing variable neighbourhood search. In Section 4.2.3, we introduce a CP model and, in Section 4.2.4, we experimentally evaluate these approaches.

4.2.1 Computation of Bounds by Solving Assignment Problems

An assignment problem aims at finding a one-to-one matching between tasks and agents [BÇ99; Pen07]. In our context, tasks correspond to targets and agents to robots, and a matching is a bijection $m : \mathcal{A} \to \mathcal{T}$. We say that an edge (a, t) of the visibility graph G_{vis} is selected whenever $m(a) = t$. The NC-AMAPF 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.*, $\forall \{a_i, a_j\} \subseteq \mathcal{A}, \overline{a_i m(a_i)} \cap \overline{a_j m(a_j)} = \emptyset$;

- there is an objective function that aims at minimising the maximal cost of a selected edge, *i.e.*, $\max_{a_i \in \mathcal{A}} |a_i m(a_i)|$.

There exist many other assignment problems [BÇ99; Pen07]. 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 [Kuh55]). Interestingly, the solution of the LSAP cannot have crossing edges whenever edge costs are defined by Euclidean distances [Put79]. Indeed, if two selected edges cross, then we can obtain a better assignment by swapping their targets so that the two edges no longer cross. Hence, the solution of the LSAP provides an upper bound to the NC-AMAPF 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 [Car+15]. Hence, the solution of the LBAP provides a lower bound to the NC-AMAPF problem without obstacles.

4.2.2 Variable Neighbourhood Search

The upper bound computed by solving an LSAP may be tightened by performing local search. We consider a basic VNS framework [MH97] described below.

- The neighbourhood of a matching m contains every non crossing matching obtained by permuting the targets 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 $\mathcal{A} \setminus \{a\}$ that contain $k-1$ anchor points and, for each subset (to which a is added), we consider every permutation of the targets 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 targets 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.)

4.2.3 Constraint Programming Model

Finally, let us introduce a CP model for the NC-AMAPF 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 target that is within a distance of ub from a_i, *i.e.*, $D(x_i) = \{t \in \mathcal{T} : |a_i t| < ub\}$;
- an integer variable y represents the maximal length of a selected edge, and its domain is $[0, ub]$.

The constraints are:

- for each pair of anchor points $\{a_i, a_j\} \subseteq \mathcal{A}$, we post a table constraint $(x_i, x_j) \in T_{ij}$ where T_{ij} is the table that contains every couple $(t,t') \in D(x_i) \times D(x_j)$ such that $t \neq t'$ and the segment $\overline{a_i t}$ does not cross the segment $\overline{a_j t'}$;
- for each anchor point $a_i \in \mathcal{A}$, we post the constraint $y \geq |a_i x_i|$;
- we post an *allDifferent*$(\{x_i : a_i \in \mathcal{A}\})$ 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.

4.2.4 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 targets 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 4.2.1).
- UB$_i$ with $i \in \{1,3,5,7\}$ refers to the computation of an upper bound by first solving an LSAP (see Section 4.2.1) and then improving it by VNS with $l =$

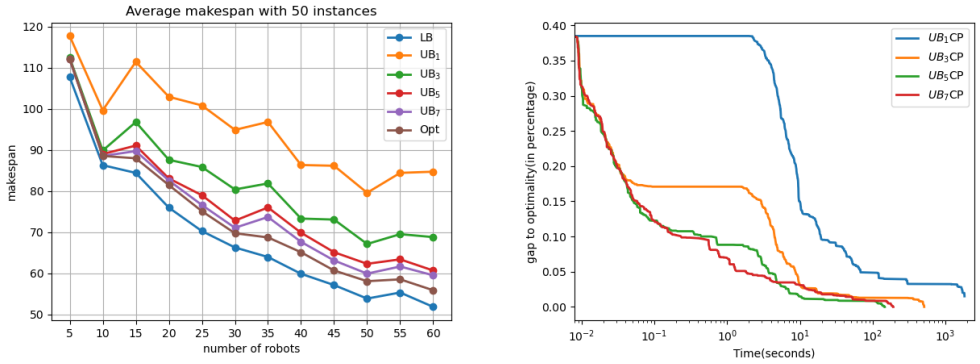


FIGURE 4.2: Left: Evolution of the optimal makespan (Opt), the lower bound (LB) and upper bounds (UB_i with $i \in \{1,3,5,7\}$) when increasing the number n of robots. Right: Evolution of the gap to optimality (in percentage) with respect to time for UB_iCP with $i \in \{1,3,5,7\}$, on average for the 50 instances with $n = 50$ robots.

60 seconds and $k_{max} = i$ (see Section 4.2.2). 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 4.2.3) for computing the optimal solution.

LB and UB_i are implemented in Python. The CP model is implemented in MiniZinc [Net+07] and solved with Chuffed [CS14]. 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. 4.2, 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 target points tend to be located more densely and this makes it easier to assign anchor points to closer targets. 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 targets (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. 4.2, 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

TABLE 4.1: 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	$\geq$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	$\geq$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	$\geq$529.3	546.1	0.06	9.4	$\geq$197.6	207.4	1.3	6.1	27.0	34.4	16.8	5.7	25.5	48.1

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 4.1 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.

4.3 NC-AMAPF 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 target 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. 4.3. As a consequence, our problem is no longer a simple bipartite matching problem: we must not only choose a different target for each anchor point, but also choose paths.

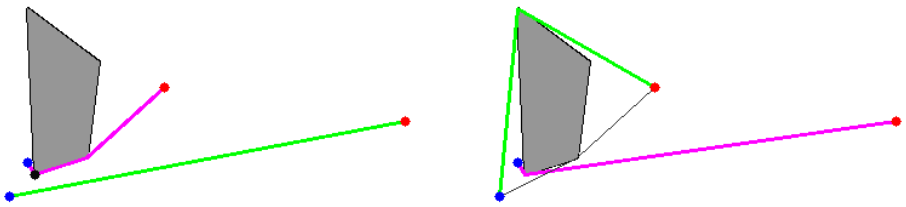


FIGURE 4.3: The solution displayed on the left only uses shortest paths, and its makespan is larger than the solution displayed on the right (the green right path is longer than the black path).

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 in Section 4.3.1. In Section 4.3.2, we show how to compute all relevant paths. In Section 4.3.3, we describe a CP model. In Section 4.3.4, we describe our benchmark and in Section 4.3.5 we experimentally evaluate our approach.

4.3.1 Computation of Bounds

When there are obstacles, the visibility graph G_{vis} associated with $\mathcal{W}$, $\mathcal{A}$ and $\mathcal{T}$ is no longer a bipartite graph. However, we can build a bipartite graph $G'_{vis} = (\mathcal{V}'_{vis}, \mathcal{E}'_{vis})$ such that $\mathcal{V}'_{vis} = \mathcal{A} \cup \mathcal{T}$ and $\mathcal{E}'_{vis} = \mathcal{A} \times \mathcal{T}$, and define the cost of an edge $(a, t) \in \mathcal{E}'_{vis}$ as the length of the shortest path from a to t in G_{vis}. In this case, we can compute a lower bound by solving the LBAP in G'_{vis}.

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

Theorem 4.3.1. Let $m : \mathcal{A} \to \mathcal{T}$ be an optimal solution of the LSAP in G'_{vis} and, for each anchor point $a_i \in \mathcal{A}$, 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 $\{a_i, a_j\} \subseteq \mathcal{A}$, 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 $\overline{u_i v_i}$ and $\overline{u_j v_j}$. 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: $\overline{u_i v_j}$ and $\overline{u_j v_i}$ do not cross obstacles, as illustrated in Fig. 4.4a.

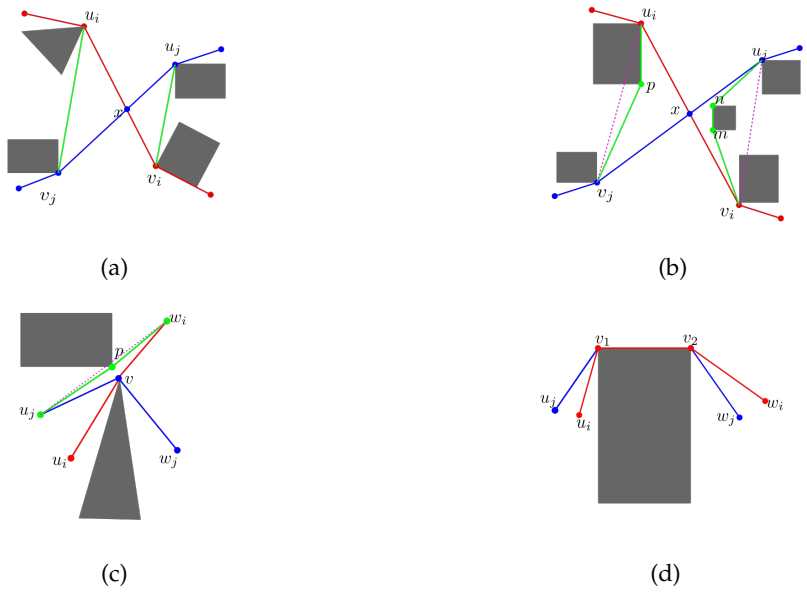


FIGURE 4.4: Top (Case 1): π_i (in red) and π_j (in blue) contain two crossing segments $\overline{u_iv_i}$ and $\overline{u_jv_j}$. (a): $\overline{u_iv_j}$ and $\overline{u_jv_i}$ (in green) do not cross obstacles and $|u_iv_j| + |u_jv_i| < [u_iv_i| + |u_jv_j|$. (b): $\overline{u_iv_j}$ and $\overline{u_jv_i}$ (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.

Let π_i^p (resp. π_i^s) be the prefix (resp. suffix) of π_i that precedes (resp. succeeds) $\overline{u_iv_i}$, *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 $\overline{u_iv_i}$ and $\overline{u_jv_j}$. We have:

$$|u_iv_i| = |u_ix| + |xv_i| \text{ and } |u_jv_j| = |u_jx| + |xv_j|. \tag{4.1}$$

The triangle inequality implies that

$$|u_iv_j| < |u_ix| + |xv_j| \text{ and } |u_jv_i| < |u_jx| + |xv_i|. \tag{4.2}$$

From Eq. (4.1) and (4.2), we infer that

$$|u_iv_j| + |u_jv_i| < |u_iv_i| + |u_jv_j|. \tag{4.3}$$

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. (4.3), 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. 4.4b.

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. 4.4b. We can show that $|\pi_{ij}| < |u_i x| + |x v_j|$ by recursively exploiting the triangle inequality (see [Ber+08]). 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. 4.4c. 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. 4.4d. 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'_{vis}. 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. 4.3.1.

Like for the NC-AMAPF without obstacles, this upper bound may be improved by VNS, as explained in Section 4.2.2. 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).

4.3.2 Relevant Paths Enumeration

The non crossing assignment in G'_{vis} that minimises the makespan may not be the optimal solution of the original problem as edges of G'_{vis} correspond to shortest paths, and as the optimal solution may use non shortest paths. To find the optimal solution, for each couple $(a, t) \in \mathcal{A} \times \mathcal{T}$, we must consider all relevant paths from a to t in the visibility graph G'_{vis}, 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 2.1.2).

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 [Lat12]. 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, as described in Algorithm 2. To check constraint (C3), we perform a local geometric test in constant time.

4.3.3 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 $\pi \in P$, $a(\pi)$, $t(\pi)$, and $l(\pi)$ denote the origin, the target, and the length of π, respectively. The CP model has the following variables:

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

The constraints are:

- for each pair of anchor points $\{a_i, a_j\} \subseteq \mathcal{A}$, 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 $t(\pi) \neq t(\pi')$ and path π does not cross path π';
- for each anchor point $a_i \in \mathcal{A}$, we post the constraint $y \geq l(z_i)$;

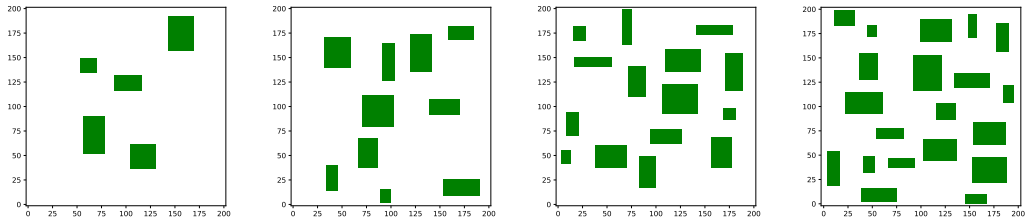


FIGURE 4.5: Workspaces $\mathcal{W}_o$ with $o \in \{5, 10, 15, 20\}$ (obstacles are displayed in green).

- we channel x_i and z_i variables by posting $x_i = d(z_i)$ and we post an $allDifferent(\{x_i : a_i \in \mathcal{A}\})$ constraint. This constraint is redundant as table constraints prevent selecting two paths that have a same target. However, preliminary experiments have shown us that this improves the solution process for a wide majority of instances.

The goal is to minimise y.

4.3.4 Description of Benchmarks

To study the sensibility of our algorithms to different configurations, we generate instances according to a random model that has three parameters o, n, and d which are described below. For all instances, the bounding polygon is the square $\mathcal{B} = [0, 200]^2$.

The first parameter o is used to set the number of obstacles. For each value of $o \in \{5, 10, 15, 20\}$, we have randomly generated one set $\mathcal{O}_o$ of o obstacles such that each obstacle is a rectangle[1] whose height and width belong to $[1, 40]$, and such that the distance between two obstacle vertices is larger than 10. For each value of o, the workspace is defined by $\mathcal{W}_o = \mathcal{B} \setminus \mathcal{O}_o$. These workspaces are displayed in Fig. 4.5.

The second parameter n is used to set the number of robots. 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.

The third parameter d is used to generate anchor points and targets in $\mathcal{W}_o$, and we consider three different kinds of distributions $d \in \{U, B\}$, in order to study the impact of this distribution on bound quality and instance hardness:

- when $d = U$ (Uniform), anchor points and targets are randomly generated in $\mathcal{W}_o$ according to a uniform distribution;
- when $d = B$ (Bipartite), anchor points (resp. targets) are randomly generated on the left (resp. right) part of $\mathcal{W}_o$, by constraining their abscissa to be smaller than 60 (resp. greater than 140).

[1] We have made experiments with other kinds of obstacles and obtained similar conclusions as with rectangular obstacles.

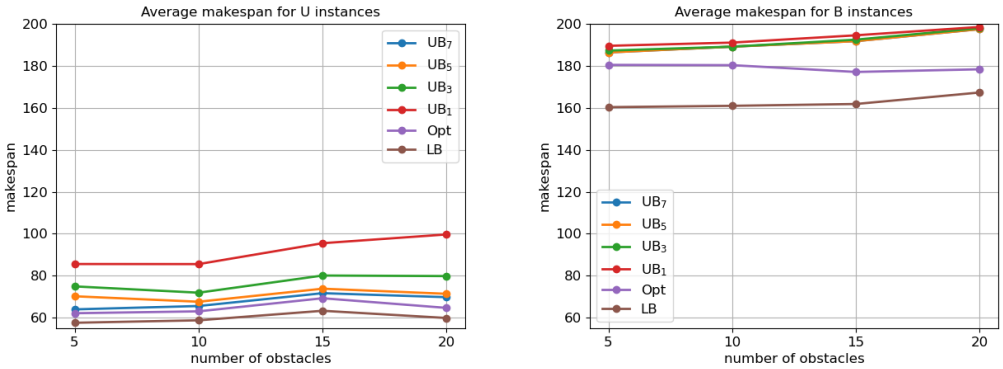


FIGURE 4.6: Evolution of the optimal makespan (Opt), the lower bound (LB) and upper bounds (UB, with $i \in \{1,3,5,7\}$) when increasing the number of obstacles from 5 to 20. Left: U instances (with $n = 40$). Right: B instances (with $n = 20$).

For the two distributions, we ensure that the distance between two points is always larger than 4 by rejecting any point that does not satisfy this constraint. We believe that setting this minimum distance is suitable considering the sizes of the workspace and the obstacles.

For each value of o and each kind of distribution, we have generated 30 instances.

4.3.5 Experimental Evaluation

In Fig. 4.6, 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 targets 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 the solution than for U instances (remember that VNS only considers shortest paths).

In Fig. 4.7, we display the evolution of the gap to optimality (in percentage) with respect to time, and in Tables 4.2 and 4.3 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 $o = 5$, and 13s when $o = 20$. This comes from the fact that the function that decides whether two paths are crossing or not has a quadratic time complexity

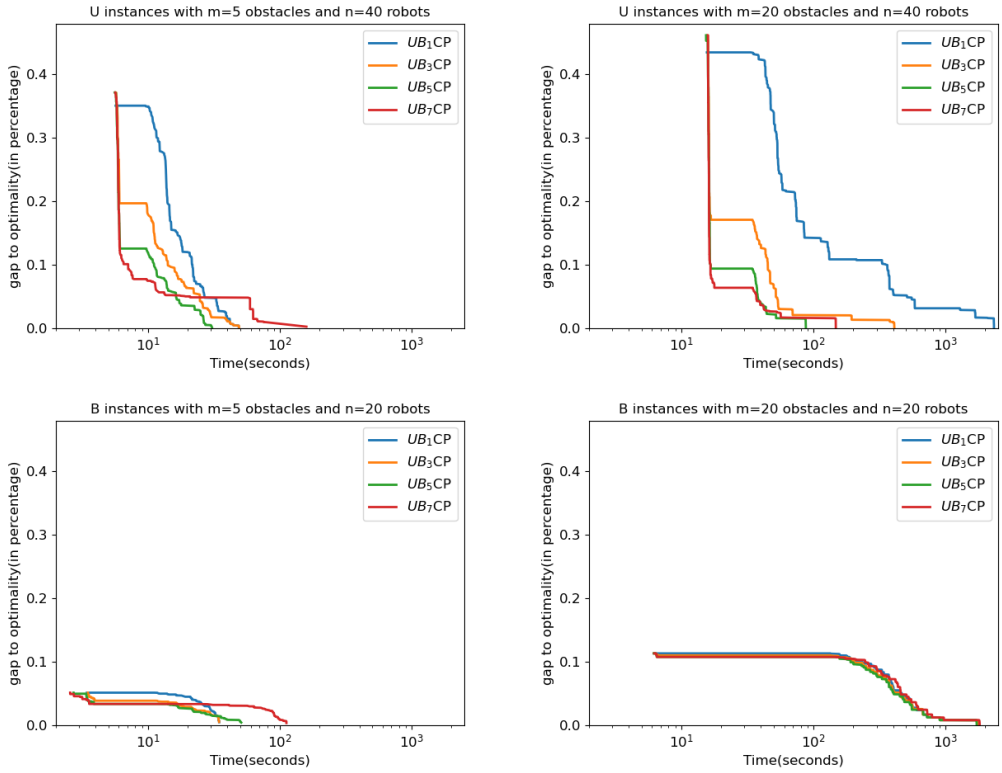


FIGURE 4.7: Evolution of the gap to optimality (in percentage) with respect to time for UB$_i$CP with $i \in \{1,3,5,7\}$, on average for 30 instances. Top left: U instances with $o = 5$. Top right: U instances with $o = 20$. Bottom left: B instances with $o = 5$. Bottom right: B instances with $o = 20$.

with respect to the number of vertices in the paths, and this number increases when increasing the number of obstacles. UB$_3$CP, UB$_5$CP, and UB$_7$CP 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 exponentially 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 (because this increases the number of relevant paths).

TABLE 4.2: Results of UB_iCP with $i \in \{1,3,5,7\}$ for U instances with $n = 40$ and $o \in \{5,10,15,20\}$ (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-target 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 target.

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

	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

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. 4.6. This increases the number of relevant paths, as seen when looking at row RP: when $o = 20$, 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 $o = 20$ 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 $o = 20$ 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 $o = 20$ 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. 4.6). 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

TABLE 4.4: 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 $o = 20$.

	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%

spent by 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 4.2 and 4.3, the number of relevant paths being searched for each anchor/target 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 $o = 20$, 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 4.4 we display the results of UB$_5$CP for different values of p on B instances when $o = 20$. 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.

4.4 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 targets, 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/target 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 target 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 targets 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.

In Chapter 5, we will extend the work to non-point-sized agents by considering robots with a body, generating complementary constraints on their motions and their cables.

Chapter 5

Non-Crossing AMAPF for non point-sized robots

Contents

5.1 Preliminaries . . . 53
5.1.1 Removing crossings . . . 53
5.1.2 Detecting deadlocks . . . 53
5.1.3 Computing the makespan . . . 54
5.2 Problem Statement . . . 55
5.3 Computation of bounds for the makespan . . . 56
5.3.1 Algorithm for removing deadlocks . . . 56
5.3.2 Computation of Makespan Bounds from LSAP and LBAP solutions . . . 59
5.3.3 Experimental evaluation . . . 60
5.4 Improving the upper bound with VNS . . . 62
5.4.1 Experimental evaluation . . . 64
5.5 Computation of the optimal solution with CP . . . 66
5.5.1 Relaxed CP model . . . 66
5.5.2 Lazy constraint generation . . . 67
5.5.3 Dichotomous Approach . . . 68
5.5.4 Experimental results . . . 69
5.6 Conclusion . . . 71

In Chapter 4, we introduced the NC-AMAPF problem. In this preliminary study, we considered bodiless robots so that two robots could share a same subpath without having to include waiting times. Based on this simplifying assumption, we showed that the NC-AMAPF can be formulated as Euclidean bipartite matching problem. In this chapter, we extend this work to non point-sized robots. In this case, we must synchronize robots in order to ensure a safety distance that prevents the collisions and entanglements of cables (usually, this safety distance is larger than the size of the robots). As a consequence, robots may have to wait at some points, as illustrated in Fig. 5.1(a-c), and these waiting times must be taken into account when computing the makespan. Also, deadlocks may occur, as illustrated in Fig. 5.1(d).

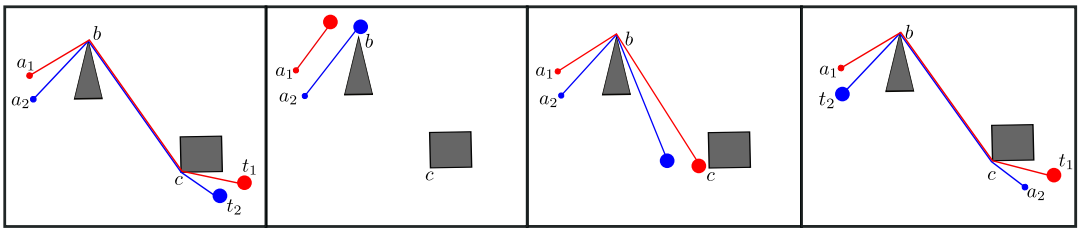


FIGURE 5.1: (a): a_1 and a_2 are the anchor points of 2 robots that both have to go to points b and c before reaching their final targets t_1 and t_2 (gray polygons are obstacles). (b): The red robot could arrive first on point b (because it is closer), but it has to wait for the blue robot to pass point b (because the blue robot cannot travel between the red cable and point b if the red robot has already passed point b). (c): Then, the blue robot could arrive first on point c (because it left point b first), but it has to wait for the red robot to pass point c. (d): When the anchor point and the target of the blue robot are exchanged, a deadlock occurs because the red robot must wait for the blue one to pass b first, while the blue one cannot pass c before the red robot.

In Sections 5.1 and 5.2, we present all the basic elements and formally define the NC-AMAPF for non point-sized tethered robots. The main difference with the NC-AMAPF introduced in Chapter 4 is that we add precedence constraints between robots that share a same subpath in order to take into account the fact that a safety distance must be maintained between them to avoid collisions and cable entanglements. This new setting definitely changes the definition of the makespan as we may have to introduce waiting times to satisfy precedence constraints.

In Section 5.3, we introduce an algorithm that exploits a precedence graph to suppress deadlocks by reassigning targets. We also prove that the optimal solution of LSAP cannot contain deadlocks. This allows us to compute two different upper bounds of the optimal solution: the first one is computed by solving the LSAP; the second one is computed by first solving the LBAP and then using our new algorithm to remove deadlocks, if any.

In Section 5.4, we introduce a VNS approach to improve the upper bounds computed in Section 3. This algorithm is similar to the VNS approach introduced in Chapter 4, but it enlarges the neighborhood by taking into account non-shortest paths, as the optimal solution may contain non-shortest paths in order to avoid crossings.

In Section 5.5, we extend the CP model of Chapter 4 to include waiting times due to interactions between pairs of robots when computing the makespan. This CP model relaxes constraints due to interactions of more than two robots, and we show how to lazily generate constraints to compute the optimal solution. We also introduce a dichotomous approach to avoid computing useless paths.

5.1 Preliminaries

5.1.1 Removing crossings

In Theorem 4.3.1, we have shown that whenever two paths π_i and π_j are crossing, we can always replace them by two non crossing paths π'_i and π'_j such that $|\pi_i| + |\pi_j| \geq |\pi'_i| + |\pi'_j|$. The basic idea is to exchange the end of π_i with the end of π_j, starting from the crossing point, and to replace the resulting paths with taut paths whenever they are not taut.

Given a set of paths with crossings, these pairwise exchanges of path ends may be iterated until all crossings have been removed. The resulting set of non-crossing paths has a total length smaller than or equal to the initial set of crossing paths.

5.1.2 Detecting deadlocks

In Chapter 4, we considered bodiless robots, *i.e.*, we assumed that a robot can always travel between the cable of another robot and an obstacle. With this simplifying assumption, a set of non-crossing paths is always a consistent solution. In this chapter, we take into account the fact that a safety distance must be maintained between two robots when they pass through the same sub-path, to avoid collisions and cable entanglements. This is done by constraining robots to pass shared obstacles in a given order, as illustrated in Fig. 5.1. This order depends on the relative path positions with respect to the obstacle: if π_i is closer to the obstacle than π_j, then the robot associated with π_i must reach the vertex before the robot associated with π_j. For example, in Fig. 5.1(a), the blue path is closer to the triangle than the red path whereas the red path is closer to the rectangle than the blue path. Given two paths that share a same obstacle vertex, we can decide in constant time which one is the closer to the obstacle, as described in [ZP19]. This may be extended to the case of two paths that share some sub-path, by ensuring that paths never cross. For example, in Fig. 5.2, once we have decided that $\pi_1 \prec_d \pi_2$, we can propagate the relative positions of π_1 and π_2 at vertex b, *i.e.*, π_2 is above π_1 and, therefore, $\pi_2 \prec_b \pi_1$. We use this to define total orders among paths that traverse a same obstacle vertex.

Definition 5.1.1 (Total order $\prec_u$)**.** Let Π be a set of non-crossing paths, and $u \in \mathcal{V}_\mathcal{O}$ be a vertex of an obstacle $o \in \mathcal{O}$. We denote $\prec_u$ the strict total order on Π_u such that $\forall\{\pi_i, \pi_j\} \subseteq \Pi_u, \pi_i \prec_u \pi_j$ if and only if π_i is closer to o than π_j (as defined in [ZP19]). In this case, the robot associated with π_i must visit u before the robot associated with π_j, and we say that π_i has a higher priority than π_j for vertex u.

These total orders are used to define a precedence graph which is similar to the Pair Interaction Graph of [ZP19]. This precedence graph models precedence constraints between path steps, where a path step is a couple (u, π_i) that represents the visit of a vertex u by the robot associated with a path π_i. Besides the precedence constraints induced by total orders, this graph also models precedence constraints due to the fact that a robot must visit vertices in the order defined by its path.

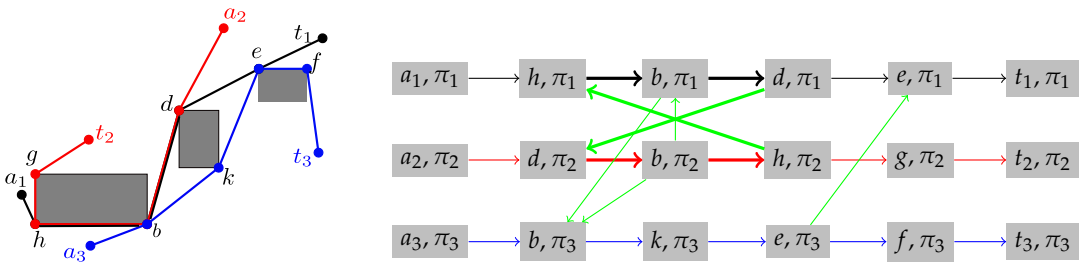


FIGURE 5.2: Deadlock example. Left: Three paths $\pi_1 = \langle a_1, h, b, d, e, t_1 \rangle$ in black, $\pi_2 = \langle a_2, d, b, h, g, t_2 \rangle$ in red, and $\pi_3 = \langle a_3, b, k, e, f, t_3 \rangle$ in blue. The robot associated with π_1 should pass h after the robot associated with π_2 because $\pi_2 \prec_h \pi_1$, meanwhile, the robot associated with π_2 cannot pass d until the robot associated with π_1 has passed it because $\pi_1 \prec_d \pi_2$. Right: The corresponding precedence graph (vertical edges are in green whereas horizontal edges have the same color as their corresponding path). This graph contains the cycle $c = \langle (h, \pi_1), (b, \pi_1), (d, \pi_1), (d, \pi_2), (b, \pi_2), (h, \pi_2), (h, \pi_1) \rangle$ (displayed in bold).

Definition 5.1.2 (Precedence graph)**.** Let Π be a set of non-crossing paths. The precedence graph associated with Π is the directed graph $G_\Pi = (\mathcal{V}_\Pi, \mathcal{E}_\Pi)$ such that vertices correspond to path steps, *i.e.*, $\mathcal{V}_\Pi = \{(u, \pi_i) \mid \pi_i \in \Pi \wedge u \in \pi_i\}$, and edges correspond to precedence constraints between path steps, *i.e.*, $\mathcal{E}_\Pi = \{((u, \pi_i), (v, \pi_i)) \mid \pi_i \in \Pi \wedge \langle u, v \rangle \subseteq \pi_i\} \cup \{((u, \pi_i), (u, \pi_j)) \mid u \in \mathcal{V}_\mathcal{O} \wedge \{\pi_i, \pi_j\} \subseteq \Pi_u \wedge \pi_i \prec_u \pi_j\}$. Edges $((u, \pi_i), (v, \pi_i))$ between two path steps in a same path are called *horizontal edges* whereas edges $((u, \pi_i), (u, \pi_j))$ between two path steps in two different paths are called *vertical edges*.

Two paths that both visit two vertices may have different precedence constraints on these two vertices. For example, let us consider the paths π_1 and π_2 displayed in Fig. 5.2. Both paths visit vertices h and d. However, $\pi_2 \prec_h \pi_1$ (because π_2 is closer to the left most obstacle than π_1) whereas $\pi_1 \prec_d \pi_2$ (because π_1 is closer to the middle obstacle than π_2). As π_1 must visit h before d whereas π_2 must visit d before h, a deadlock occurs.

More generally, deadlocks occur if and only if the precedence graph contains cycles, as cyclic precedence constraints cannot be satisfied. These deadlocks may be detected in linear time with respect to the size of G_Π by performing a depth first search [Cor+09].

5.1.3 Computing the makespan

We aim at minimizing the makespan, *i.e.*, the arrival time of the latest robot including waiting times due to the fact that robots must pass shared obstacles in a given order. This may be computed by exploiting the precedence graph. To this aim, we define a cost function $c_\Pi : \mathcal{E}_\Pi \to \mathbb{R}^+$ such that:

- the cost of an horizontal edge associated with a path segment is the time needed to travel through this segment, *i.e.*, $\forall((u,\pi_i),(v,\pi_i)) \in \mathcal{E}_\Pi, c((u,\pi_i),(v,\pi_i)) = |uv|$;
- the cost of a vertical edge associated with a priority at a vertex is the time that a robot should wait to let another robot pass before it, *i.e.*, $\forall((u,\pi_i),(u,\pi_j)) \in \mathcal{E}_\Pi, c((u,\pi_i),(u,\pi_j)) = dt$ where dt is a parameter corresponding to the duration for traveling the safety distance (which depends on the size of robots, among other things).

The makespan is equal to the size of the longest path in G_Π when considering cost function c_Π. It may be computed in linear time with respect to the size of G_Π by topologically sorting vertices of $\mathcal{V}_\Pi$ and then relaxing edges of $\mathcal{E}_\Pi$ according to this topological order [Cor+09]. We do not define the exact locations of the waiting times as this does not affect the computation of the overall makespan. Obviously, robots must wait before starting to follow shared sub-paths and the choice of waiting locations should be considered when planning robot actions, which is not in the scope of this work.

5.2 Problem Statement

In case where the body size of robots is considered, we extend the formulation of NC-AMAPF problem defined in Def. 4.1.1 by adding the following elements.

- a positive value $dt \in \mathbb{R}^+$ corresponding to the time a robot must wait to let another robot pass before it at some shared point.
- for a solution (m,Π), an additional constraint that Π must not contain deadlocks.

In Section. 4.3.4, we randomly generated instances in terms of the parameter triple (o,n,d) where o represents the number of obstacles, n denotes the number of robots and d is used to describe the distribution of anchor points and targets. Based on this benchmark, we generate a new distribution termed "Alternate".

- when $d = A$ (Alternate) anchor points are randomly generated with their abscissa being equally constrained in $[0,20] \cup [40,60] \cup [120,140]$, and targets are equally distributed in $[80,100] \cup [140,160] \cup [180,200]$.

For each value of n, o, and d, we have randomly generated 30 instances (all instances with a same value of o share the same workspace). For all instances, the value of dt is set to 4 (see Section 5.6 for a discussion on the impact of this parameter on instance hardness). Examples of instances are displayed in Fig. 5.3.

All experiments reported in this chapter are run on Grid5000 [Bal+13] with an AMD EPYC 7642 with 512GB of RAM.

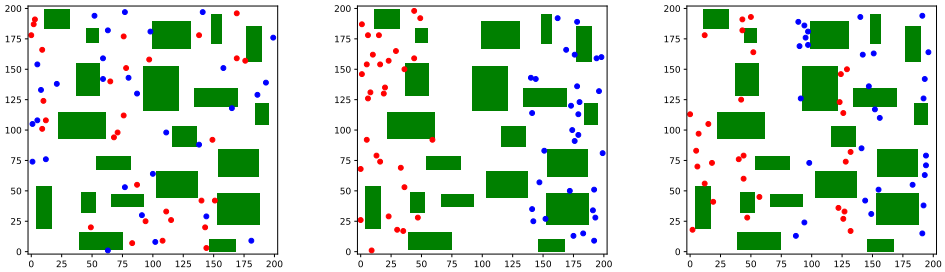


FIGURE 5.3: Instance examples with $n = 30$, $o = 20$, and $d = U$ (left), B (middle), and A (right).

5.3 Computation of bounds for the makespan

In Chapter 4, we considered the NC-AMAPF problem with bodiless robots and we showed how to efficiently compute an upper bound by solving an LSAP in in the complete bipartite graph $G_{\mathcal{A},\mathcal{T}} = (\mathcal{A}, \mathcal{T}, \mathcal{A} \times \mathcal{T})$, and a lower bound by solving an LBAP.

When taking into account the physical shape of robots, a set of non-crossing paths is not necessarily solution because it may imply deadlocks, as illustrated in both Fig. 5.1(d) and Fig. 5.2. In this section, we first introduce an algorithm that removes all deadlocks from a set of paths by iteratively exchanging targets, and we show that this algorithm decreases the path length sum. Then, we show how to compute a lower bound and two different upper bounds by exploiting LSAP and LBAP solutions. Finally, we experimentally compare these bounds.

5.3.1 Algorithm for removing deadlocks

To remove deadlocks, we remove cycles in the precedence graph G_Π by exchanging targets. Given a cycle c in G_Π, let Π_c denote the set of paths involved in c, where a path $\pi_i \in \Pi$ is involved in c if c contains a vertex (u, π_i). Given a path $\pi_i \in \Pi_c$, let $in_c(\pi_i)$ (resp. $out_c(\pi_i)$) denote the set of vertices of π_i that have an incoming (resp. outgoing) vertical edge in c, *i.e.*,

$$\begin{aligned} in_c(\pi_i) &= \{u \in \pi_i : \exists \pi_j \in \Pi_c \setminus \{\pi_i\}, \langle (u, \pi_j), (u, \pi_i) \rangle \subseteq c\} \\ out_c(\pi_i) &= \{u \in \pi_i : \exists \pi_j \in \Pi_c \setminus \{\pi_i\}, \langle (u, \pi_i), (u, \pi_j) \rangle \subseteq c\} \end{aligned}$$

For example, let us consider the precedence graph G_Π and the cycle c displayed in Fig. 5.2: $in_c(\pi_1) = \{h\}$ and $out_c(\pi_1) = \{d\}$.

Property 5.3.1. Let Π be a set of paths and c be an elementary cycle in G_Π. For each path $\pi_i \in \Pi_c$, the number of vertical edges of c that end on a vertex of π_i is equal to the number of vertical edges of c that start from a vertex of π_i, *i.e.*, $\#in_c(\pi_i) = \#out_c(\pi_i)$.

Algorithm 3: REMOVECYCLE(Π, c)

Input: A set Π of non-crossing paths and an elementary cycle c in G_Π such that, for each path π_i involved in c, $\#in_c(\pi_i) = \#out_c(\pi_i) = 1$

Output: A set of non-crossing paths Π' with the same sets of anchor and target points as Π and such that $G_{\Pi'}$ no longer contains cycle c

1 $\Pi' \leftarrow \Pi \setminus \Pi_c$
2 **for** *each path* $\pi_i \in \Pi_c$ **do**
3 Let $((u, \pi_j), (u, \pi_i)) \in c$ be the incoming vertical edge of π_i
4 Let π_{i1} and π_{i2} be the prefix and suffix of π_i such that $\pi_i = \pi_{i1}.\langle u \rangle.\pi_{i2}$
5 Let π_{j1} and π_{j2} be the prefix and suffix of π_j such that $\pi_j = \pi_{j1}.\langle u \rangle.\pi_{j2}$
6 Let $\pi'_i = \pi_{i1}.\langle u \rangle.\pi_{j2}$
7 **if** π'_i *is not a taut path* **then**
8 Replace π'_i with the shortest path from the first vertex of π'_i to the last vertex of π'_i and in the same homotopy class as π'_i
9 **end**
10 Add π'_i to Π'
11 **end**
12 Remove crossings from Π' as explained in Section 5.1.1

Proof. This is a straightforward consequence of the fact that (i) c is a cycle and (ii) every horizontal edge connects two vertices in a same path. Indeed, each time the cycle reaches a path $\pi_i \in \Pi_c$, using a vertical edge that ends on a vertex of $in_c(\pi_i)$, it must use a vertical edge that starts from a vertex of $out_c(\pi_i)$ to leave π_i. □

Property 5.3.1 ensures that, whenever $\#in_c(\pi_i) = \#out_c(\pi_i) = 1$ for each path $\pi_i \in \Pi_c$, there are exactly $\#\Pi_c$ vertical edges, and these vertical edges define a permutation on the paths of Π_c. In this case, we can remove cycle c by replacing the target of each path π_i with the target of the previous path π_j in the permutation, as described in Algorithm 3. More precisely, the new path π'_i is composed of the prefix of π_i that ends on the vertex u such that $in_c(\pi_i) = \{u\}$, and the suffix of π_j that starts from u (lines 3-6). This new path is valid as it is composed of two valid sub-paths that share a same vertex U. However, it may not be taut and, in this case, we have to replace it by its corresponding taut path (lines 7-8). It may be possible that the new taut paths contain crossings and, in this case, we use the procedure described in Section 5.1.1 to remove all crossings (line 10).

For example, let us consider the precedence graph G_Π displayed in Fig. 5.2. This graph contains two different elementary cycles.

- Let us suppose that REMOVECYCLE is called with $c = \langle (h, \pi_1), (b, \pi_1), (d, \pi_1), (d, \pi_2), (b, \pi_2), (h, \pi_2), (h, \pi_1) \rangle$. The vertical edge of c that reaches π_1 is $((h, \pi_2), (h, \pi_1))$. Hence, the new path π'_1 is composed of the prefix of π_1 that ends on h and the suffix of π_2 that starts from h, *i.e.*, $\pi'_1 = \langle a_1, h, g, t_2 \rangle$. The vertical edge of c that reaches π_2 is $((d, \pi_1), (d, \pi_2))$. Hence, the new path π'_2 is composed of the prefix of π_2 that ends on d and the suffix of π_1 that starts from d, *i.e.*,

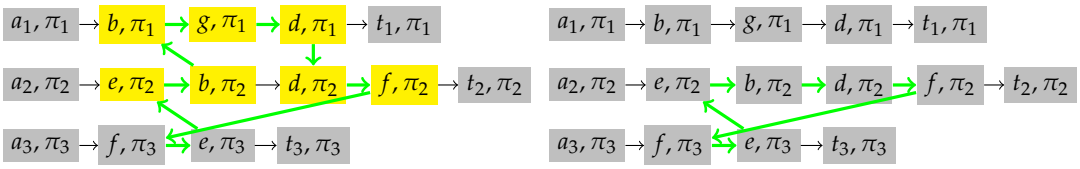


FIGURE 5.4: Illustration of Algorithm 4. Left: A precedence graph with an elementary cycle c displayed in green. $in_c(\pi_2) = \{e, d\}$, $out_c(\pi_2) = \{b, f\}$, $\pi_{2,3} = \langle e, b, d, f \rangle$. Right: The precedence graph obtained after replacing the sub-path c' from (e, π_2) to (f, π_2) (displayed in yellow on the left) with $\langle (e, \pi_2), (b, \pi_2), (d, \pi_2), (f, \pi_2) \rangle$.

$\pi'_2 = \langle a_2, d, e, t_1 \rangle$. These two paths are not taut, and their associated taut paths are $\langle a_1, g, t_2 \rangle$ and $\langle a_2, t_1 \rangle$.

- Let us suppose that REMOVECYCLE is called with $c = \langle (b, \pi_1), (d, \pi_1), (d, \pi_2), (b, \pi_2), (b, \pi_1) \rangle$. The vertical edge of c that reaches π_1 is $((b, \pi_2), (b, \pi_1))$. Hence, the new path π'_1 is $\langle a_1, h, b, h, g, t_2 \rangle$. The vertical edge of c that reaches π_2 is $((d, \pi_1), (d, \pi_2))$. Hence, the new path π'_2 is $\langle a_2, d, e, t_1 \rangle$. These two paths are not taut, and their associated taut paths are $\langle a_1, g, t_2 \rangle$ and $\langle a_2, t_1 \rangle$.

It may be possible that new deadlocks are introduced by Algorithm 3, either when paths are replaced with taut paths, or when crossings are removed. In this case, we have to call again Algorithm 3 in order to remove them. The following property ensures that such an iterative process eventually stops.

Property 5.3.2. Let $\Pi' = \text{REMOVECYCLE}(\Pi, c)$. We have: $\sum_{\pi_i \in \Pi} |\pi_i| > \sum_{\pi'_i \in \Pi'} |\pi'_i|$

Proof. Let us first note that the cycle c necessarily contains at least two horizontal edges because vertical edges correspond to total order relations that cannot contain cycles.

Now, let us consider Algorithm 3 without lines 7-8 (*i.e.*, non taut paths are not replaced with taut paths). In this case, every horizontal edge of c no longer appears in the new paths and we have $\sum_{\pi_i \in \Pi} |\pi_i| = \sum_{\pi'_i \in \Pi'} |\pi'_i| + \sum_{\langle (u, \pi_i), (v, \pi_i) \rangle \subset c} |uv|$. As c contains at least two horizontal edges, we have $\sum_{\pi_i \in \Pi} |\pi_i| > \sum_{\pi'_i \in \Pi'} |\pi'_i|$.

Finally, let us consider Algorithm 3 with lines 7-8. Replacing a non taut path with a taut path can only reduce the length of the path because a taut path is the shortest path within the same homotopy class. This may introduce some crossings, but we have shown in Section 5.1.1 that path length sums can only be reduced when removing crossings. Hence, we still have $\sum_{\pi_i \in \Pi} |\pi_i| > \sum_{\pi'_i \in \Pi'} |\pi'_i|$. □

Algorithm 4 exploits Algorithm 3 to build a valid solution given a set of paths that may contain crossing paths or deadlocks. It first removes crossing paths using the approach described in Section 5.1.1 (line 1). Then, while the precedence graph G_Π contains cycles, it searches for an elementary cycle c (with a simple depth-first-search, for example) and iteratively simplifies c until $\#in_c(\pi_i) = \#out_c(\pi_i) = 1$ for every path π_i involved in c (lines 4-9). To simplify c, we search for a path π_i such that $\#in_c(\pi_i) = \#out_c(\pi_i) > 1$, and we shortcut c by replacing its sub-path that goes

Algorithm 4: BUILDSOLUTION(m, Π)

Input: A bijection $m : \mathcal{A} \to \mathcal{T}$ and a set Π of n paths such that, $\forall a \in \mathcal{A}$, Π contains a path from A to $m(a)$
Output: A solution to the NC-AMAPF problem

1 Remove crossings from Π as explained in Section 5.1.1
2 **while** G_Π *contains cycles* **do**
3 Search for an elementary cycle c in G_Π
4 **while** *there exists a path* π_i *involved in* c *such that* $\#in_c(\pi_i) > 1$ **do**
5 Let $\pi_{i,1}$ be the longest prefix of π_i such that $\forall w \in \pi_{i,1}, w \notin in_c(\pi_i)$
6 Let $\pi_{i,2}$ be the longest suffix of π_i such that $\forall w \in \pi_{i,2}, w \notin out_c(\pi_i)$
7 Let $\pi_{i,3} = \langle u_1, \dots, u_k \rangle$ be the sub-path of π_i such that $\pi_i = \pi_{i,1}.\pi_{i,3}.\pi_{i,2}$
8 Let c' be the sub-path of c that starts on (u_1, π_i) and ends on (u_k, π_i)
9 Replace c' with $\langle (u_1, \pi_i), (u_2, \pi_i), \dots, (u_k, \pi_i) \rangle$
10 **end**
  /* For each path π_i involved in c, we have $\#in_c(\pi_i) = \#out_c(\pi_i) = 1$ */
11 $\Pi \leftarrow$ REMOVECYCLE(Π, c)
12 **end**
13 Update m with respect to Π and return (m, Π)

from the leftmost vertex $u_1 \in in_c(\pi_i)$ to the rightmost vertex $u_k \in out_c(\pi_i)$ with the sub-path of π_i that joins u_1 to u_k. This removes at least one vertex from both $in_c(\pi_i)$ and $out_c(\pi_i)$. An example is depicted in Fig. 5.4. In some cases, the number of paths involved in c is also decreased, but this number cannot become smaller than two as c still contains at least two vertical edges (one that ends on u_1 and one that starts from u_k) and a vertical edge necessarily involves two different paths. Hence, the loop lines 4-9 is ensured to reach a cycle c such that $\#in_c(\pi_i) = \#out_c(\pi_i) = 1$ for every path $\pi_i \in \Pi_c$ in at most $v/2 - 2$ iterations, where $v = \sum_{\pi_i \in \Pi_c} \#in_c(\pi_i)$ is the initial number of vertical edges in c. Finally, the cycle is removed by using Algorithm 3 (line 10). Property 5.3.2 ensures that the loop lines 2-9 is performed a finite number of times: As each cycle removal decreases the total path length, the process necessarily stops with a deadlock free situation.

5.3.2 Computation of Makespan Bounds from LSAP and LBAP solutions

We denote $G_{\mathcal{A},\mathcal{T}} = (\mathcal{A}, \mathcal{T}, \mathcal{A} \times \mathcal{T})$ the complete bipartite graph such that the cost of an edge $(a, t) \in \mathcal{A} \times \mathcal{T}$ is $|sp(a, t)|$ (*i.e.*, the length of the shortest path from a to t in the visibility graph). Let us now show how to compute lower and upper bounds for our NC-AMAPF problem by solving assignment problems in $G_{\mathcal{A},\mathcal{T}}$. Let $m_{LSAP} : \mathcal{A} \to \mathcal{T}$ be the optimal solution of LSAP without crossings, and Π_{LSAP} be the corresponding set of paths, *i.e.*, $\Pi_{LSAP} = \{sp(a, m_{LSAP}(a)) : a \in \mathcal{A}\}$. Similarly, let m_{LBAP} denote the optimal solution of LBAP and $\Pi_{LBAP} = \{sp(a, m_{LBAP}(a)) : a \in \mathcal{A}\}$.

The following property shows us that $s_{LSAP} = (m_{LSAP}, \Pi_{LSAP})$ may be used to build a solution of the NC-AMAPF problem.

Property 5.3.3. The precedence graph $G_{\Pi_{LSAP}}$ contains no cycle.

Proof. Let us suppose that $G_{\Pi_{LSAP}}$ contains cycles. In this case, we could use Algorithm 4 to remove these cycles. However, this would decrease the sum of all path lengths, which is in contradiction with the fact that m_{LSAP} minimizes the sum of all path lengths. □

Hence, s_{LSAP} is a solution of our NC-AMAPF problem and, therefore, $makespan(s_{LSAP})$ is an upper bound.

$s_{LBAP} = (m_{LBAP}, \Pi_{LBAP})$ provides a lower bound to our NC-AMAPF problem and it is the optimal solution whenever paths in Π_{LBAP} do not cross nor do they share vertices. However, s_{LBAP} is not a solution if Π_{LBAP} contains crossings or deadlocks. In this case, we may use Algorithm 4 to remove all crossings and deadlocks (in other words, BUILDSOLUTION(s_{LBAP}) is a solution) so that $makespan($BUILDSOLUTION$(s_{LBAP}))$ is an upper bound.

To summarize relations between bounds, if opt denotes the optimal makespan of our NC-AMAPF problem, we have:

$$\begin{array}{rcl} opt & \leq & makespan(s_{LSAP}) \\ \max\limits_{\pi_i \in \Pi_{LBAP}} |\pi_i| \leq opt & \leq & makespan(\text{BUILDSOLUTION}(s_{LBAP})) \end{array}$$

However, $makespan(s_{LSAP})$ and $makespan($BUILDSOLUTION$(s_{LBAP}))$ are not comparable.

5.3.3 Experimental evaluation

Algorithms have been implemented in Python.

In Fig. 5.5, we display the gap between the optimal makespan opt and the lower bound $lb_{LBAP} = \max_{\pi_i \in \Pi_{LBAP}} |\pi_i|$, the upper bound $ub_{LSAP} = makespan(s_{LSAP})$, and the upper bound $ub_{LBAP} = makespan($BUILDSOLUTION$(s_{LBAP}))$. This gap is computed as a percentage (*i.e.*, we display $100 * \frac{b-opt}{opt}$ for each bound b, where opt is computed with the exact methods introduced in Section 5.5), and on average for 30 instances per combination (n, o, d). lb_{LSAP} is usually rather close to the optimal solution, especially for U instances, and the bound tends to move away from the optimal solution when increasing n, especially for B and A instances. The number of obstacles o does not seem to have a strong influence on the quality of the lower bound.

There is no clear winner between the two upper bounds. When $d = U$, ub_{LBAP} tends to be closer to opt than ub_{LSAP}, whereas when $d = B$ the two bounds are very close and, when $d = A$, ub_{LSAP} is better than ub_{LBAP} for 8 (o, n) combinations (among 12). In Fig. 5.6, we display scatter plots to compare the two upper bounds on a per instance basis when $n = 30$. For U instances, the gap of ub_{LBAP} is equal to 0% for 8 instances whereas it is greater than 125% for two instances, and the gap of ub_{LSAP} is equal to 0% for 9 instances whereas it is equal to 112% for one instance.

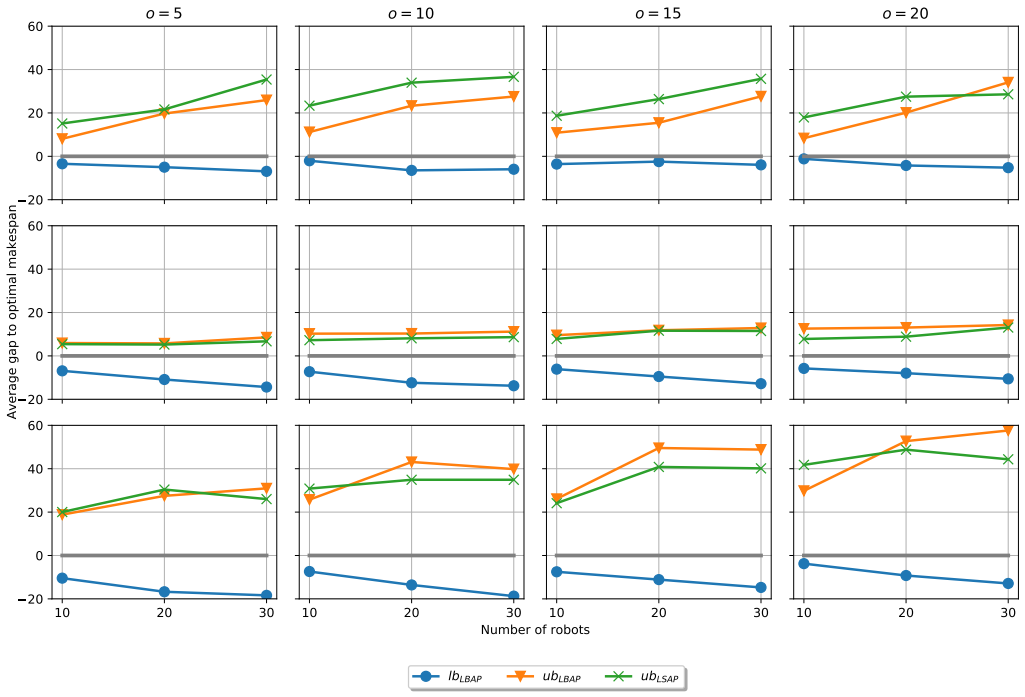


FIGURE 5.5: Gap in percentage (y-axis) between the optimal makespan and lb_{LBAP}, ub_{LSAP}, and ub_{LBAP} when $n \in \{10, 20, 30\}$ (x-axis), $o \in \{5, 10, 15, 20\}$ (from left to right), and $d = U$ (Top), B (Middle), or A (Bottom). Every point corresponds to an average value over 30 instances.

For B instances, gaps are always smaller than 25%. For both B and A instances, the difference between the two bounds is less important than for U instances.

In Table 5.1, we display the time needed to compute all shortest paths from anchor points to targets, and the time needed to compute the two upper bounds (when excluding the time needed to compute paths). For U instances, we always spend more time to compute all shortest paths than to compute a bound. Times increase when increasing the number of obstacles o or the number of robots n, but bounds are always computed within a few tenths of a second. For A instances, the time needed to compute ub_{LSAP} is larger than for U instances (*e.g.*, 0.4s instead of 0.02s when $n = 30$ and $o = 20$), and this time is even larger for B instances (*e.g.*, 1.4s when $n = 30$ and $o = 20$). This comes from the fact that the optimal solution of LSAP for U instances nearly never contains crossing paths, whereas it more often contains crossing paths for A and B instances. For example, when $n = 30$ and $o = 20$, the average number of crossing paths is equal to 0.6 (resp. 3.2 and 10) for U (resp. A and B) instances (and replacing these crossing paths by non-crossing paths is time-consuming). This conclusion also holds for ub_{LBAP}, as we can observe that the computation time also increases with m and o for all types of instances. However, compared to ub_{LSAP}, ub_{LBAP} takes more time, notably, when $n = 30$, this difference can go up to several seconds for A and B instances. We know that the optimal solution of LBAP is computed without considering the non-crossing constraints, so

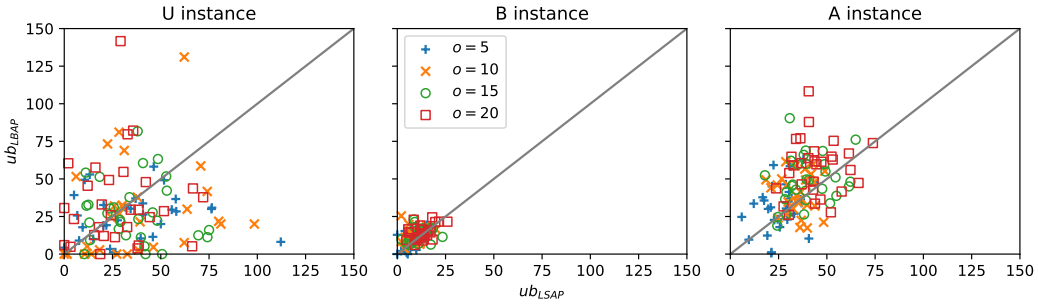


FIGURE 5.6: Comparison of ub_{LSAP}, and ub_{LBAP}: Each point (x, y) corresponds to an instance with $n = 30$ such that $x = 100 * \frac{ub_{LSAP} - opt}{opt}$ and $y = 100 * \frac{ub_{LBAP} - opt}{opt}$. The color of the point depends on o. U (resp. B and A) instances are displayed on the left (resp. middle, right).

TABLE 5.1: Time (in seconds) needed to compute all shortest paths (t_{path}), and the two bounds ub_{LSAP} and ub_{LBAP} (when excluding the time needed to compute shortest paths), on average over the 30 instances per combination (n, o, d).

		n=10				n=20				n=30			
		o=5	o=10	o=15	o=20	o=5	o=10	o=15	o=20	o=5	o=10	o=15	o=20
d=U	t_{path}	0.03	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.4
	ub_{LSAP}	0.0	0.0	0.01	0.01	0.0	0.01	0.01	0.01	0.01	0.01	0.01	0.02
	ub_{LBAP}	0.02	0.03	0.1	0.1	0.05	0.1	0.1	0.2	0.1	0.2	0.3	0.3
d=B	t_{path}	0.03	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.4
	ub_{LSAP}	0.01	0.04	0.1	0.2	0.04	0.2	0.5	0.7	0.1	0.4	1.0	1.4
	ub_{LBAP}	0.1	0.3	0.7	1.3	0.3	0.8	2.4	4.3	0.7	1.8	4.9	8.3
d=A	t_{path}	0.03	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.4
	ub_{LSAP}	0.0	0.0	0.03	0.02	0.01	0.05	0.1	0.1	0.04	0.1	0.3	0.4
	ub_{LBAP}	0.04	0.1	0.6	0.3	0.2	0.4	1.0	1.4	0.4	1.0	2.4	3.8

there are more crossings to remove. When $n = 30$ and $o = 20$, this average number is equal to 11.3 (resp. 44.2 and 58.1) for U (resp. A and B) instances. We do not report times needed to solve LBAP (and compute lb_{LBAP}): this time is always smaller than 0.03s and is negligible with respect to other times.

In conclusion, ub_{LSAP} is more efficiently computed than ub_{LBAP}, while the two bounds are rather comparable in quality. Therefore, we use the result of ub_{LSAP} as an initial upper bound.

5.4 Improving the upper bound with VNS

In Section 4.2.2, we have introduced a VNS approach for improving the solution s_{LSAP} that minimizes the sum of costs. To adapt this VNS to the case where robots have physical shapes, we simply have to forbid deadlocks and to modify the computation of the makespan by integrating waiting times in case of shared vertices. This VNS approach is denoted OLDVNS.

Algorithm 5: NEWVNS(m, Π, k_{max})

Input: an initial solution (m, Π), and a parameter $k_{max} \in \mathbb{N}$
Output: an improved solution (m, Π)

1 $k \leftarrow 2$
2 **while** $k \leq k_{max}$ **do**
3 let π_{max} be the path of Π with the latest arrival time and a_{max} its anchor point
4 $\mathcal{C} \leftarrow \{\pi \in \Pi : \pi$ is in the same connected component as π_{max} in $G_\Pi\}$
5 **if** $\#\mathcal{C} < k$ **then**
6 add to C the $k - \#\mathcal{C}$ paths whose anchors are the closest to a_{max}
7 **end**
8 **for** *each* $\mathcal{S} \subseteq \mathcal{C}$ *such that* $\#\mathcal{S} = k$ *and* $\pi_{max} \in \mathcal{S}$ **do**
9 let $\mathcal{A}_\mathcal{S}$ be the set of anchor points of paths in $\mathcal{S}$
10 **for** *each permutation* $\sigma : \mathcal{A}_\mathcal{S} \to \mathcal{A}_\mathcal{S}$ *such that* $\forall a \in \mathcal{A}_\mathcal{S}, |sp(a, m(\sigma(a)))| < makespan(m, \Pi)$ **do**
11 **for** *each* $a \in \mathcal{A}_\mathcal{S}$ **do**
12 compute the set Π_a of every taut path π from A to $m(\sigma(a))$ such that $|\pi| < makespan(m, \Pi)$, and π does not cross any path in $\Pi \setminus \mathcal{S}$
13 **end**
14 **for** *each set* $\mathcal{S}'$ *which contains exactly one path of* Π_a, $\forall a \in \mathcal{A}_\mathcal{S}$ **do**
15 **if** $\Pi' = (\Pi \setminus \mathcal{S}) \cup \mathcal{S}'$ *is valid and improving* **then**
16 replace Π with Π' and update m consequently
17 set k to 2, and go to line 2
18 **end**
19 **end**
20 **end**
21 **end**
22 increment k
23 **end**
24 return (m, Π)

In Fig. 4.3, we showed that optimal solutions may contain non-shortest paths. For some instances (in particular those generated with $d \neq U$), optimal solutions widely use non shortest paths and have much shorter makespans than solutions computed with shortest paths only (as done by OLDVNS). For this reason, we introduce a new VNS approach, denoted NEWVNS, which takes into account non-shortest paths as described in Algorithm 5.

NEWVNS starts the search from an initial solution (m, Π) (which may be s_{LSAP} or BUILDSOLUTION(s_{LBAP}), for example). At each iteration of the loop lines 2-16, it explores the neighborhood of the current solution (m, Π). This neighborhood contains couples (m', Π') such that Π' is obtained from Π by replacing a set $\mathcal{S}$ of k paths with a set $\mathcal{S}'$ of k new paths (which are not necessarily shortest paths). Obviously, $\mathcal{S}$ must contain the path π_{max} with the latest arrival time (corresponding to the makespan), as this is a mandatory condition to reduce the makespan. The search for $\mathcal{S}'$ is done in four steps:

Step 1 (lines 4-6): As it is time-consuming to compute new paths, the neighborhood is deliberately reduced by constraining paths of $\mathcal{S}$ to belong to a limited set of candidate paths $\mathcal{C}$ which contains all paths in the same connected component as π_{max} in G_Π. If there are less than k paths in $\mathcal{C}$, it is completed by selecting the paths whose anchor points are the closest to the anchor point of π_{max}.

Step 2 (lines 7-9): For each subset $\mathcal{S}$ of k paths in $\mathcal{C}$ (including π_{max}), we enumerate every permutation σ of the k anchor points of $\mathcal{S}$ such that the length of the shortest path from any of these k anchor points A to $m(\sigma(a))$ is smaller than the current makespan (otherwise the permutation cannot lead to a shorter makespan).

Step 3 (lines 10-11): For every anchor point A involved in $\mathcal{S}$, we compute the set Π_a of all paths from A to the new target $m(\sigma(a))$ associated with A, while limiting the search to taut paths that do not cross paths of $\Pi \setminus \mathcal{S}$ and that have a length smaller than the current makespan.

Step 4 (lines 12-15): We search for a new set $\mathcal{S}'$ of k paths in Π_a such that the k new paths are non-crossing and the makespan of $(\Pi \setminus \mathcal{S}) \cup \mathcal{S}'$ is smaller than the makespan of Π. If such an improving set of paths is found, the current solution is updated, and a new improving move is searched with $k = 2$.

If no improving neighbor is found in the loop lines 7-15, then k is increased to enlarge the neighborhood.

5.4.1 Experimental evaluation

Algorithms have been implemented in Python[1].

In Fig. 5.7, we display the evolution of the gap to optimality (in percentage) of bounds computed by OLDVNS and NEWVNS when $k_{max} \in \{1, 3, 5, 7\}$ and when the initial solution is s_{LSAP}. OLDVNS and NEWVNS both return s_{LSAP} when $k_{max} = 1$ as k is initialized to 2 and the search is stopped whenever $k > k_{max}$. Increasing k_{max} decreases the makespan, but we observe larger improvements from 1 to 3 than for 3 to 5 and then to 7. We have made experiments with values of k_{max} larger than 7 and observed that this does not allow us to significantly improve the makespan.

For U instances, we obtain a better upper bound with OLDVNS which only considers the shortest paths, while NEWVNS works better for B and A instances. This difference is due to two reasons. First, compared to OLDVNS, NEWVNS takes into account non-shortest paths. For A and B instances, we more often need to use non shortest paths to improve the solution than for U instances. Second, for the neighborhood construction, in NEWVNS, we restrained the number of anchor points, or the location ranges where they can be, as well as the location ranges of targets are fixed. For B instances, the anchor points and targets are located in a bipartite way,

[1]Our implementation of the work in Chapter 5 is publicly available at https://gitlab.inria.fr/xipeng/tethered-amapf-jair2023.git.

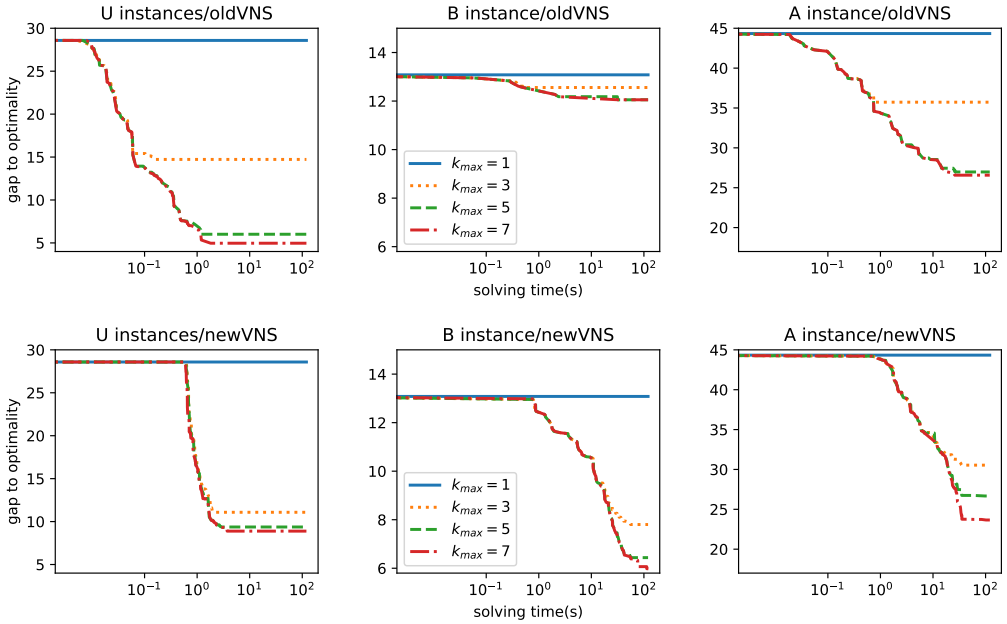


FIGURE 5.7: Evolution of the gap to optimality in percentage with respect to time (in seconds) for OLDVNS (top row) and NEWVNS (bottom row) when $k_{max} \in \{1,3,5,7\}$: average value over 30 instances when $n = 30, o = 20$, and $d = U$ (left), B (middle), and A (right). Scales on y-axis are different for U, B, and A instances (from the left to the right), whereas they are identical for OLDVNS and NEWVNS.

so the optimal solution could also follow a symmetric match along the Y axis. This means that our new neighborhood is smaller but more effective than the previous one. A instances also follow a bit of bipartite symmetry, so it works too. For U instances, as anchors and targets are uniformly distributed, exchanging with nearby points can easily violate the non-crossing constraint and it becomes harder to get a better solution: in this case, performing an exhaustive search among all anchor points (as OLDVNS does) is more effective.

Since OLDVNS and NEWVNS show complementary performance, we combine them by running OLDVNS first then followed by NEWVNS (each step is limited to 60s) in order to improve the robustness when tackling different instances, and we call this combined method COMBINEDVNS. The switching time of 60 seconds is chosen to find a compromise between the quality of the solution and the time required for resolution. From Fig. 5.7, we can observe that when oldVNS and newVNS are run individually, the optimality gap curves for each tend to converge around 60 seconds. We have tested other combination methods, and this sequential combination performs better in terms of robustness and efficiency.

In Fig. 5.8, we show the evolution of the gap to optimality (in percentage) of bounds computed by OLDVNS, NEWVNS, and COMBINEDVNS when $k_{max} = 7$. For U instances, we see that running NEWVNS after OLDVNS slightly improves the bound. For B instances, the curve drops slowly in the first phase (which corresponds to OLDVNS), until the 60s time limit is reached, then it continues to improve

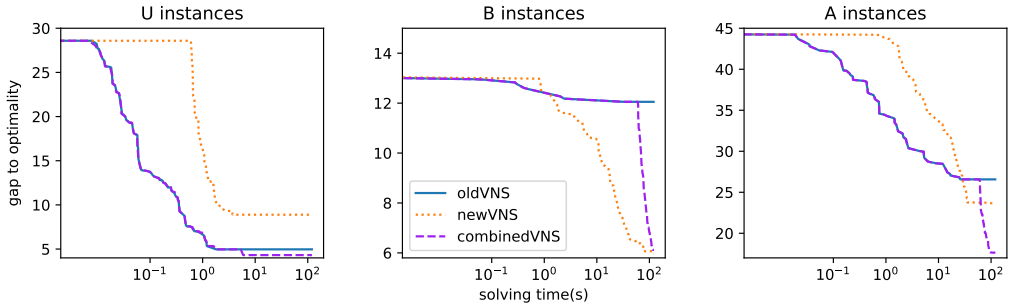


FIGURE 5.8: Evolution of the gap to optimality in percentage with respect to time (in seconds) for COMBINEDVNS (which runs OLDVNS for 60s and then NEWVNS for 60s) when $k_{max} = 7$: average value over 30 instances when $n = 30, o = 20, d = U$ (left), B (middle), and A (right).

the bound, and even reaches a level lower than the simple NEWVNS does. This conclusion also holds for A instances, and we can see that COMBINEDVNS has a clear advantage over OLDVNS and NEWVNS.

5.5 Computation of the optimal solution with CP

In this section, we first introduce a relaxed CP model, where interactions between more than two paths are ignored. Then, we show how to lazily generate constraints due to interactions between more than two paths in order to compute the optimal solution, and we introduce a dichotomous approach to reduce the number of candidate paths that must be pre-computed. Finally, we report experimental results.

5.5.1 Relaxed CP model

Since the optimal solution may use non-shortest paths, to find the optimal solution we must compute all relevant paths that may belong to the optimal solution. These paths must be taut, elementary, and non self-crossing. As there is an exponential number of paths with respect to the number of obstacle vertices, we add a limit l to the length of the paths. Obviously, we can set l to the best known upper bound, as the makespan cannot be smaller than the length of the longest selected path. Let ub denote this best known upper bound, computed with approaches described in the previous sections. By default, we assume that $l = ub$ (we shall describe in Section 5.5.3 a dichotomous approach where l is set to values smaller than ub). Given a length bound l, Π_l denotes the set of all taut, elementary, and non self-crossing paths of length smaller than l. Paths in Π_l are numbered from 1 to $\#\Pi_l$. For each path $\pi \in \Pi_l$, $a(\pi)$ and $t(\pi)$ denote the anchor point and the target of π, respectively.

In this CP model, we use the same variables as in Section 4.3.3: for each anchor point $a_i \in \mathcal{A}$, the integer variable x_i represents the target matched with a_i, the integer

Algorithm 6: LAZYAPPROACH(ub, Π_l)

Input: an initial upper bound ub, and a set Π_l of candidate paths
Output: the optimal makespan

```
let M be the CP model described in Section 5.5.1
while M has a solution s do
    if G_{s(z_i):a_i∈A} contains a cycle c then
        add to M a nogood constraint that forbids c
    else
        let π be the longest path in G_{s(z_i):a_i∈A}
        if |π| < ub then ub ← |π|; add to M the constraint Obj < ub;
        else add to M a nogood constraint that forbids π;
    end
end
return ub
```

variable z_i represents the path used to reach x_i from a_i, and the integer variable Obj [2] represents the makespan. Based on the previous model, we introduce new variables for taking into account waiting times due to vertices shared by two paths: for each pair of anchor points $\{a_i, a_j\} \subseteq \mathcal{A}$, the integer variable $M_{i,j}$ represents the makespan of the two paths z_i and z_j, including the waiting times due to vertices shared by z_i and z_j, *i.e.*, the length of the longest path in the precedence graph $G_{\{z_i,z_j\}}$. The domain of this variable is $D(M_{i,j}) = [0, ub[$.

z_i and $M_{i,j}$ variables are related thanks to table constraints. For each pair of anchor points $\{a_i, a_j\} \subseteq \mathcal{A}$, we pre-compute the table $\mathcal{T}_{i,j}$ that contains every triple $(\pi_i, \pi_j, ms) \in D(z_i) \times D(z_j) \times \mathbb{N}$ such that (i) $t(\pi_i) \neq t(\pi_j)$, (ii) path π_i does not cross path π_j, (iii) $G_{\{\pi_i,\pi_j\}}$ has no cycle, (iv) ms is the length of the longest path in $G_{\{\pi_i,\pi_j\}}$, and (v) $ms < ub$. For each pair of anchor points $\{a_i, a_j\} \subseteq \mathcal{A}$, we post the table constraint $(z_i, z_j, M_{i,j}) \in \mathcal{T}_{ij}$, and we post the constraint $Obj \geq M_{i,j}$.

The solution that minimizes Obj is a lower bound of the optimal makespan because some constraints have been relaxed. Indeed, table constraints ensure that there is no deadlock between two paths and Obj 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 two paths. Also, in the case of dependency chains of more than two paths, the makespan may be larger than all $M_{i,j}$ variables (*e.g.*, when π_1 and π_2 share a vertex and π_2 and π_3 share another vertex, the actual makespan may be larger than $M_{1,2}$ and $M_{2,3}$). In the next section, we introduce an approach that lazily generate constraints to compute the optimal solution.

5.5.2 Lazy constraint generation

Lazy constraint generation is often used when there are too many constraints. This is the case for our problem, as we should add a constraint for each possible subset of

[2] In Section 4.3.3, we use y to denote the largest path length. Here we use Obj for this new definition of makespan.

anchor points in order to ensure that the paths associated with these anchor points do not create deadlocks[3]. In Algorithm 6, we describe an approach based on lazy constraint generation. We enumerate all solutions of the model described in Section 5.5.1. Each time a solution s is found, we compute the precedence graph $G_{\{s(z_i):a_i \in \mathcal{A}\}}$ (where $s(X)$ denotes the value assigned to a variable X in solution s). If s implies a deadlock, *i.e.*, the precedence graph contains a cycle, we search for a cycle c in the precedence graph, we compute the set of anchor points involved in c, and we add a nogood constraint to prevent the search from enumerating again the paths associated by s to these anchor points (lines 3-4). If there is no deadlock, we search for the longest path π in the precedence graph (line 6): the makespan corresponds to the length of this path. If this makespan is smaller than ub, we have found a new improving solution and we constrain Obj to be smaller than this makespan (line 7). Otherwise, we search for all anchor points involved in π and we add a nogood constraint to prevent the search from enumerating again the paths associated by s to these anchor points (line 8).

5.5.3 Dichotomous Approach

A main issue is related to the initial bound l. This bound limits the length of the paths computed in the set Π_l used to generate the relaxed CP model. The larger l, the more time needed to compute paths in Π_l, the larger the domains of z_i variables and the more triples in $\mathcal{T}_{i,j}$ tables. The upper bound ub computed with COMBINED-VNS is often quite close to the optimal solution (the average gap is smaller than 10% for U and B instances, and smaller than 20% for A instances when $k_{max} = 7$). However, in most cases, ub is much larger than the longest selected path in the optimal solution because waiting times are added. As a consequence, when setting l to ub, a considerable number of paths of Π_l do not contribute to the optimal solution. To avoid this unnecessary computation, we use a dichotomous approach for setting l, as described in Algorithm 7.

lb and ub are the lower and upper bounds of Obj, respectively (lb is initialized to $makespan(s_{LBAP})$ and ub to an upper bound computed by LSAP or COMBINEDVNS). α is a parameter used to control when to switch from dichotomous search to a classical search. While the gap between ub and $(lb + ub)/2$ is larger than $\alpha * ub$, we set l to $(lb + ub)/2$, we compute the set Π_l and launch LAZYAPPROACH (lines 2-4). If the bound returned by LAZYAPPROACH is smaller than l, then we have found the optimal solution (line 5); otherwise, we increase the lower bound of Obj to l as we know that the optimal solution is greater than or equal to l (line 6). When the gap between ub and $(lb + ub)/2$ becomes smaller than $\alpha * ub$, we stop the dichotomous

[3]Another possibility is to design a global constraint for ensuring that the selected set of paths does not contain deadlocks. We have implemented a propagator for this global constraint (see Appendix A), that incrementally detects cycles in the precedence graph. However, experimental results showed us that this approach is less efficient than a lazy approach, on most instances.

Algorithm 7: DICHOTOMOUSAPPROACH(lb, ub, α)

Input: a lower bound lb, an upper bound ub, and a parameter $\alpha \in [0, 1]$
Output: the optimal makespan

```
while (ub − (lb + ub)/2) > α ∗ ub do
    l ← (lb + ub)/2
    Compute the set Π_l of all paths of length smaller than l
    ub ← LAZYAPPROACH(ub, l)
    if ub ≤ l then return ub;
    lb ← l
end
Compute the set Π_ub of all paths of length smaller than ub
return LAZYAPPROACH(ub, Π_ub)
```

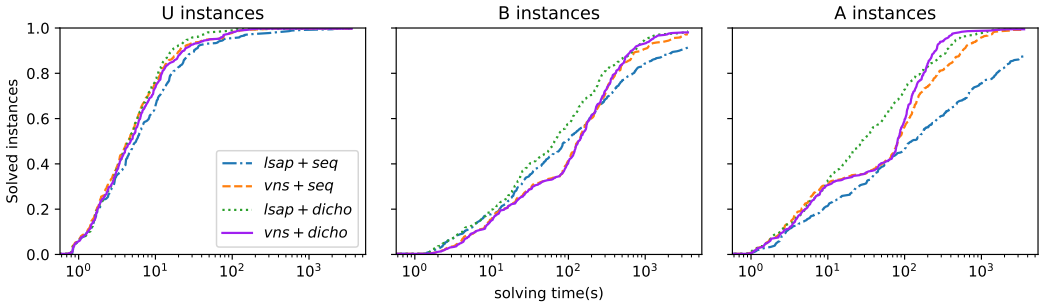


FIGURE 5.9: Percentage of solved instances with respect to time when $d = U$ (left), B (middle), and A (right).

approach and run LAZYAPPROACH with the set Π_{ub} of all paths of length smaller than ub.

5.5.4 Experimental results

Algorithm 6 has been implemented in Java, using the Choco CP library [PFL16]. Algorithm 7 has been implemented in Python.

The sequential lazy approach described in Section 5.5.2 is denoted SEQ and the dichotomous approach described in Section 5.5.3 is denoted DICHO. For DICHO, the rate α is set to 0.05 (*i.e.*, we switch to SEQ when the gap between ub and $(lb + ub)/2$ is smaller than or equal to 5%). For each approach X $\in$ {SEQ,DICHO}, the bound l is either initialized with $makespan(s_{LSAP})$ (denoted LSAP+X) or with the bound computed by COMBINEDVNS with $k_{max} = 7$ (denoted VNS+X).

In Fig. 5.9, we display the ratio of solved instances with respect to time for the four approaches X+Y with X $\in$ {SEQ,DICHO} and Y $\in$ {LSAP,VNS} (CPU times include the time for computing the initial bound l with Y). For each instance type $d \in \{U, B, A\}$, there are 360 instances (30 instances per value of $o \in [5, 10, 15, 20]$ and $n \in [10, 20, 30]$). DICHO is more successful than SEQ. This is more particularly sensible when the upper bound is computed with LSAP as, in this case, the initial upper bound is much greater. When the upper bound is computed with VNS,

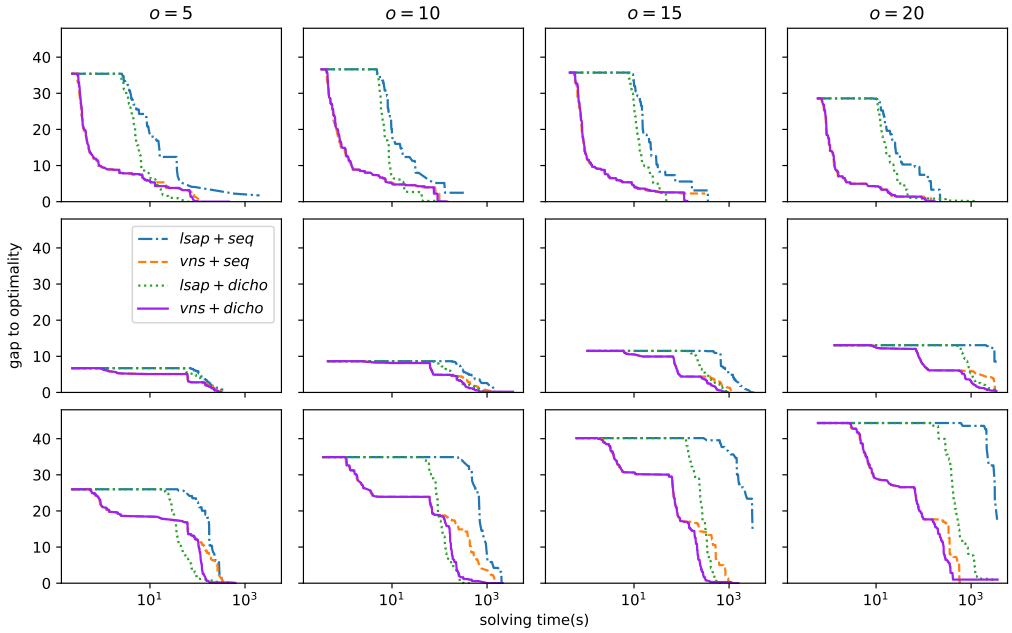


FIGURE 5.10: Evolution of the gap to optimality (in percentage) with respect to time for $n = 30$, $o \in \{5, 10, 15, 20\}$ and $d = U$ (Top), B (Middle), or A (Bottom).

VNS+SEQ and VNS+DICHO obtain very close results for U and B instances. However, VNS+DICHO is much more successful than VNS+SEQ for A instances for time limits greater than 100s (corresponding to the time spent by COMBINEDVNS to compute the upper bound when $k_{max} = 7$). This comes from the fact that the upper bound returned by COMBINEDVNS is quite close to the optimal solution for U and B instances (less than 5% for U instances, and less than 6% for B instances), whereas the gap is equal to 18% for A instances (see Fig. 5.8).

In Fig. 5.10, we display the evolution of the gap to optimality (in percentage) with respect to time. For LSAP+SEQ and LSAP+DICHO, the curve is horizontal at the beginning of the solution process. This corresponds to the time needed to compute paths of Π_l. This horizontal part is shorter for LSAP+DICHO (especially for B and A instances), as paths are computed up to $(lb + ub)/2$ instead of ub for LSAP+SEQ. VNS+SEQ and VNS+DICHO have identical gaps at the beginning of the search process. This corresponds to the 120 seconds spent by COMBINEDVNS to improve the upper bound. After 120s, VNS+DICHO has smaller gaps than VNS+SEQ, and it is the best performing approach for most instances.

An NC-AMAPF instance has a parameter dt which corresponds to the time needed by a robot to let another robot pass before it at some shared vertex. In all experiments, this parameter has been set to four. We made experiments with other values of dt and observed that it has an impact on instance hardness: the higher dt, the greater the upper bound (computed with VNS) and, therefore, the more expensive the computation of all paths of length smaller than the upper bound. However, the relative performance of the different considered approaches are not drastically

changed when dt is changed.

5.6 Conclusion

In this chapter, we extend the work of Chapter 4 on non-crossing AMAPF by considering the impact of robots's physical size. We show that motion constraints can be translated into precedence constraints, that imply waiting times when computing the makespan. We prove that the solution of LSAP cannot contain deadlocks and always provides a valid upper bound for the new problem. We propose as well an alternative way to calculate an upper bound from the LBAP solution. Experimental results show us that the two upper bounds have rather similar qualities but the LSAP upper bound is more quickly computed.

We introduce a novel VNS approach that also considers non-shortest paths as neighbors, and we show that it has complementary performance with the VNS approach in Section 4.2.2. We propose to combine them sequentially to improve robustness in practice. To solve the problem optimally, we firstly introduce a relaxed CP model that disregards interactions between more than two robots, thus it provides theoretically a lower bound. Then a lazy constraint generation approach is applied to compute the optimal solution.

In this problem, one of the main issues affecting efficiency is related to the initial upper bound used 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 unnecessary path computations, we adopt a dichotomous approach to choose an appropriate upper bound. Experimental results on randomly generated instances have shown us that a sequential combination of VNS and CP enables efficient computation of solutions for this novel variant of AMAPF problem. Additionally, we have introduced parameters in each method to effectively control efficiency and solving time, thereby enhancing its generalization capability for solving diverse instances.

Chapter 6

Single Tethered Robot Coverage Path Planning

Contents

6.1 Problem Statement 75
6.1.1 Notations and Definitions 75
6.1.2 Computing the Cable Configuration 75
6.1.3 Definition of the CPP Problem for Tethered Robots 76
6.2 Complexity Study 78
6.2.1 MCT Problem without Forbidden Areas 78
6.2.2 MCT Problem with Forbidden Areas 79
6.3 Configuration Enumeration Based Approach (Model $\mathcal{M}_0$) 83
6.4 Approximation Algorithms 86
6.4.1 Model $\mathcal{M}_1$ 86
6.4.2 A Fixed-Parameter Tractable (FPT) Algorithm (Model $\mathcal{M}_2$) . 87
6.4.3 Model $\mathcal{M}_3$ and $\mathcal{M}_3 h$ 90
6.5 Experimental results 92
6.5.1 Description of Benchmarks 93
6.5.2 Computational Performance 94
6.5.3 Scalability Analysis 95
6.6 Discussion 99
6.7 Conclusion 100

In Chapter 4 and Chapter 5, we focused on solving the anonymous multiple tethered robots path finding problems, which essentially involve finding a set of shortest paths. In this chapter, we shift our attention to another common task referred to as **C**overage **P**ath **P**lanning for a single **T**ethered robot, denoted as TCPP. Compared with the classic CPP problem, the main challenge is related to the constraints with the cable, e.g. the cable has a limited length and cannot cross itself. In addition, we consider scenarios where the workspace might contain forbidden zones, where the robot and its cable are forbidden to pass. This restriction could be due to potential damage to the robot and its cable or the presence of humans who might be impacted

by the robot or cable. A main difference between a forbidden area and an obstacle comes from the fact that the cable is blocked by obstacles (wrapping around them), whereas it is not blocked by forbidden areas. In such cases, the robot needs to follow a path that prevents the cable from crossing these zones, as depicted in Fig. 6.1.

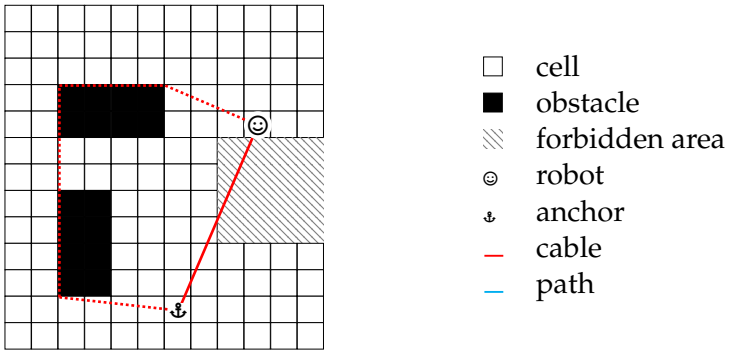


FIGURE 6.1: Example forbidden areas. The red solid line is an invalid configuration because the tether overlaps the forbidden area. The robot can choose to bend around the obstacles to reach the target with the cable always being inside the workspace.

In Section 6.1, we formally define the TCPP problem and propose the adaptation of the STC algorithm to find an solution for this new problem. By discretizing the workspace as a special graph structure called G_4, a new objective problem is formulated to find a maximum tree in G_4, ensuring that the corresponding cable configuration at each vertex satisfies the constraints.

Section 6.2 focuses on a theoretical analysis of the problem's complexity. In general case, we demonstrate that the problem can be solved in polynomial time and introduce an algorithm that combines the Dijkstra method with spanning tree generation. Based on that, we introduce a new constraint involving forbidden areas in the workspace. We prove that this new problem belongs to the class of $\mathcal{NP}$-complete problem through a reduction from a Planar 3-SAT instance.

In Section 6.3, we present an exact approach for solving the problem when forbidden areas are present. It involves enumerating all possible cable configurations and representing them as a graph structure. In this method, we implement an ILP model to solve it.

In Section 6.4, we introduce four heuristic models aimed at efficiently finding a lower bound for the problem. These models are tested on randomly generated instances, and the results are presented in Section 6.5. The findings indicate that the model $\mathcal{M}_3h$, based on a Fixed Parameter Tractable (FPT) heuristic, performs the best. Additionally, we examine how the problem's hardness evolves when varying the cable length and the number of obstacles in the workspace.

6.1 Problem Statement

6.1.1 Notations and Definitions

Consider a workspace $\mathcal{W} \subseteq \mathbb{R}^2$ which is defined by a finite region B consisting of a set of polygonal obstacles $\mathcal{O} = \bigcup_j \mathcal{O}_j$ and holes[1] $\mathcal{H} = \bigcup_j \mathcal{H}_j$, and $\mathcal{W} = B \setminus \{\mathcal{O} \cup \mathcal{H}\}$. We denote $\mathcal{V}_\mathcal{O}$ the set of obstacle vertices. In this work we discretize the workspace into a 4-connected grid graph g such that each cell of the grid has the same size as the robot's footprint, as described in Section 2.2.4.

Let us consider a connected graph $G_4 = (\mathcal{V}_4, \mathcal{E}_4)$ such that each vertex of G_4 corresponds to a non-overlapping group of 2×2 adjacent cells in g, and edges of G_4 correspond to adjacency relations between these 2×2 cell groups. An example is depicted in Fig. 2.9. In this work, we assume that g can be perfectly transformed into G_4, i.e., there are no remaining cells in g. For this purpose, we assume that the obstacles and holes are all in the form of blocks of 2×2 cell groups, which can be both convex and concave.

We assume that a tethered robot is initially located at its anchor point r. Let X_s, X_t be respectively the initial and target cable configuration, and $\pi_{s->t}$ a path allowing to move from s to t.

6.1.2 Computing the Cable Configuration

In Section 2.1.3, we have introduced different methods to compute X_t given X_s and $\pi_{s->t}$. In a workspace with polygonal obstacles, determining the shortest path between two configurations takes $\mathcal{O}(k_p|\mathcal{V}_\mathcal{O}|^2 \log|\mathcal{V}_\mathcal{O}|)$ time, where k_p denotes the number of vertices of X_s, and $|\mathcal{V}_\mathcal{O}|$ is the number of obstacle vertices. Algorithm 8 presents an approach to incrementally update the cable configuration on a grid graph when the robot moves between two adjacent vertices u and v, where uv is a segment rather than a polyline as in the general case. Specifically, let $X_s = q_0 q_1 ... q_k u$. When the robot moves from u to v, the cable either detaches some vertices on X_s (case in Fig. 6.2(b)) or bends around new obstacles (case in Fig. 6.2(a)), but never combines them as in Fig. 2.3(left). Therefore, to compute X_t, the algorithm backtracks on the vertices $X_s \setminus \{u\}$ until the last vertex q_{i+1} visible to v is found. If $q_{i+1} = q_k$, then the algorithm searches for other obstacle vertices within the pseudotriangle $\triangle q_k uv$ (line 5). This step takes $\mathcal{O}(|\mathcal{V}_\mathcal{O}| \log|\mathcal{V}_\mathcal{O}|)$ time with a preprocessing by computing the visibility graph of the workspace.

If X_s is valid and the movement $u \to v$ does not traverse any holes, Algorithm 8 ensures that X_t will never cross any holes neither (in our discretization scheme, we suppose that the width of a hole is at least of 2 grid cells). Since the length of a cable configuration is the sum of its segments, it can be also incrementally updated in time $\mathcal{O}(k_p)$. Consequently we can efficiently compute and check whether X_t is valid in $\mathcal{O}(|\mathcal{V}_\mathcal{O}| \log|\mathcal{V}_\mathcal{O}| + k_p)$ time.

[1] In computational geometry, the term "holes" generally refers to the interior boundaries within an enclosed shape. In the context of our problem, "holes" specifically denote forbidden areas.

Algorithm 8: NEXTCONFIG(X_s, v)

Input: The cable configuration $X_s = q_0q_1...q_ku$, and the target point v
Output: The cable configuration X_t when arriving at v

backtrack X_s from q_k, until a vertex q_i not visible to v, then stop
if $i < k$ **then**
  return $q_0q_1...q_{i+1}v$
else
  find all obstacle vertices $z_0, ..., z_m$ in the interior of $\triangle q_kuv$ and order them by angle relative to u
  return $q_0, ...q_kz_0, ...z_mv$
end

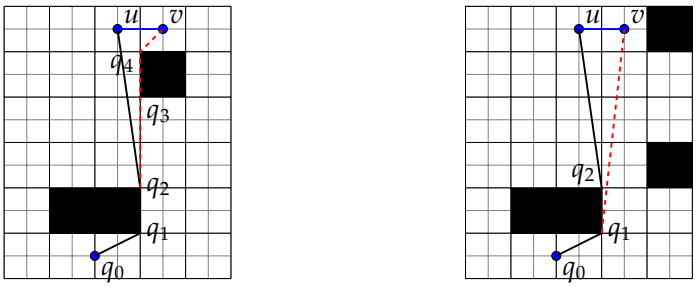


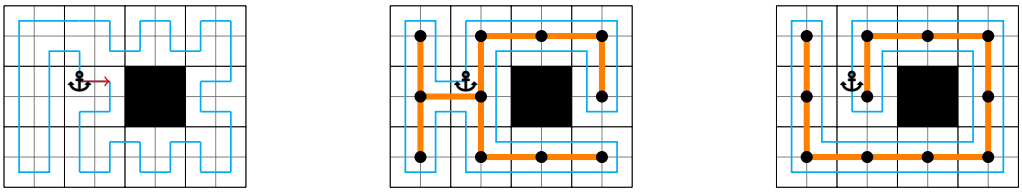
FIGURE 6.2: Examples of cable configuration in a grid graph. (a): $X_s = q_0q_1q_2u$, $X_t = q_0q_1q_2q_3q_4v$. (b): $X_s = q_0q_1q_2u$, $X_t = q_0q_1v$.

In general, k_p is smaller than $|\mathcal{V}_\mathcal{O}|$. The process of computing the cable configuration and verifying its validity is a fundamental operation in our problem. To simplify matters, we represent its complexity as $\alpha(|\mathcal{V}_\mathcal{O}|)$, where $\alpha(|\mathcal{V}_\mathcal{O}|)$ is a polynomial function in terms of $|\mathcal{V}_\mathcal{O}|$.

6.1.3 Definition of the CPP Problem for Tethered Robots

Definition 6.1.1 (TCPP problem)**.** The problem of Tethered Coverage Path Planning Problem (TCPP) involves a tethered robot operating within a workspace $\mathcal{W}$ described in Section 6.1.1, where an anchor point $r \in \mathcal{W}$ is fixed and a cable of maximum length ℓ is attached. In this context, we consider a discretized graph g associated with $\mathcal{W}$. The goal of TCPP is to find a *shortest cycle* that fully covers g and returns to its starting point, subject to the following constraints: (i) at each vertex $v \in \pi$, the corresponding cable configuration X_v remains valid, *i.e.*, it does not exceed the cable length, it does not cross itself and it does not cross any holes in $\mathcal{H}$; and (ii) when the robot returns to the anchor point, the cable is fully retracted.

According to the definition above, if there exists a Hamiltonian cycle in g, we actually have a shortest coverage path as each cell is traversed exactly once. The complexity of deciding of the existence of a Hamiltonian cycle depends on the properties of g: it is in $\mathcal{O}(1)$ for rectangular grids with no obstacles whereas it becomes $\mathcal{NP}$-complete in case of obstacles [IPS82].



(a) (b) (c)

FIGURE 6.3: (a): A Hamiltonian cycle in graph g that encircles an obstacle results in the cable getting crossed at some point if the robot starts off in the direction of the red arrow. (b)(c): By constructing a Hamiltonian cycle based on a spanning tree of G_4 (highlighted in orange), it is ensured that the cable can be completely retracted when the robot returns to its initial position. The Hamitonian cycle in (b) can avoid cable entanglements, while (c) not.

In this work, we consider to apply Spanning Tree Coverage (STC) to compute an optimal solution when the workspace can be perfectly discretized into G_4 [2]. The motivation comes from the fact that there exists a Hamiltonian cycle in g if and only if G_4 is connected and given a spanning tree T of G_4, a Hamiltonian cycle in g can be simply constructed by circumnavigating T, as illustrated in Fig. 2.9. Another reason is that the Hamiltonian cycle constructed in this way can always ensure that the cable is fully retracted and and the self-crossing can be avoided if the spanning tree used to construct the Hamiltonian cycle is approriately selected, while the other Hamiltonian cycles cannot retract the cable and may also lead to cable entanglements, as illustrated in Fig. 6.3. In Section 6.2.1, we will discuss how to generate such a spanning tree in order to meet cable constraints.

In the upcoming section, we will demonstrate that the existence of a spanning tree satisfying the cable-related constraints is not guaranteed in all cases, particularly when considering forbidden areas within the workspace. Consequently, our focus shifts to finding a general tree structure in G_4 that satisfies the constraints, which may not necessarily be a spanning tree capable of generating a coverage path in g.

Definition 6.1.2 (Coverage Tree)**.** Consider a graph g obtained by discretizing a workspace $\mathcal{W}$, and let $G_4 = (\mathcal{V}_4, \mathcal{E}_4)$ be a connected graph derived from g. A *coverage tree* is an arborescence rooted at the source vertex r, which covers a subset of vertices from $\mathcal{V}_4$, such that the corresponding cable configuration at each vertex is valid.

Definition 6.1.3 (Maximum Coverage Tree Problem(MCT))**.** Given ($\mathcal{W}$, G_4, r), compute a coverage tree $T = (\mathcal{V}_T, \mathcal{E}_T)$ of G_4 that maximize $|\mathcal{V}_T|$.

Therefore, the primary challenge lies in finding a coverage tree that maximizes the number of vertices covered. This is essential in order to discover the shortest coverage path for our original **TCPP** problem.

[2] In the case where the workspace cannot be perfectly structured into G_4—meaning that there are residual cells of g unaffected—STC can still provide a highly satisfactory approximation by introducing an additional cost for unaffected cells, as stated in Section 2.2.4.

Algorithm 9: DIJKSTRACOVERAGE(G_4, r, ℓ)

Input: A graph G_4, the source r and the maximal cable length ℓ
Output: An arborescence T rooted at r such that for each vertex $v \in T$, the path from r to v minimizes the cable configuration and whose length does not exceed ℓ

```
for each vertex v in G do
    initialise dist[v] to +∞, prev[v] to undefined, config[v] to empty
end
let Q be the set of all vertices in G_4
dist[r] ← 0
while Q is not empty do
    u ← vertex in Q with minimum dist[u], remove u from Q
    if the distance of config[u] is larger than ℓ then break;
    for each neighbor v of u still in Q do
        q ← NEXTCONFIG(config[u], v), let alt be the length of q
        if alt < dist[v] then
            dist[v] ← alt, prev[v] ← u, config[v] ← q
        end
    end
end
T ← prev \ {w | dist[w] > ℓ}
```

6.2 Complexity Study

In this section, our objective is to investigate the complexity of the **MCT** problem. We will specifically focus on how the complexity of this problem is influenced by the properties of the workspace, with a particular emphasis on the presence of forbidden areas. We will analyze and discuss the complexity for each case separately.

6.2.1 MCT Problem without Forbidden Areas

We consider firstly that the workspace contains only obstacles. In this case, the size of the coverage tree is restricted by the cable length. When the cable length is limited to ℓ, some cells may become out of reach. This occurs when the shortest path from the anchor point to these cells exceeds ℓ. In such case, these out-of-reach cells are discarded.

There are several ways to compute a coverage tree in G_4, like a breadth-first-search (BFS). However, the length of a path in g may be longer than the corresponding cable length as the cable is kept taut by a system that pulls on it. A shortest path in g does not necessarily leads to a shortest cable length, as illustrated in Fig. 6.4. As a consequence, using a BFS to compute a spanning tree in G_4 is not enough to ensure that the corresponding Hamiltonian path in g will never exceed the cable length. In Algorithm 9, we show how to adapt Dijkstra's algorithm to compute a coverage tree in G_4 that minimizes, for each reachable vertex v, the cable length between the anchor point and v. Like the classical Dijkstra algorithm [Cor+09], we

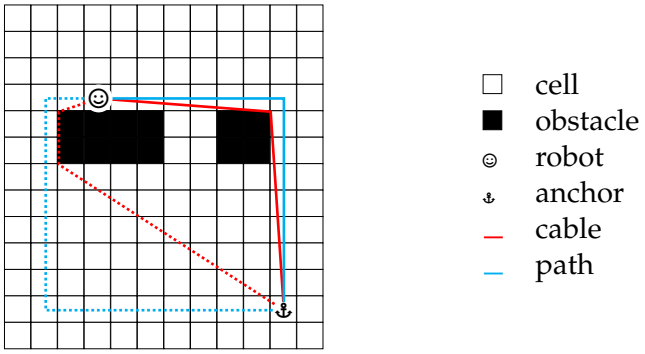


FIGURE 6.4: Example of paths (in blue) and cable positions (in red). The blue solid path is smaller than the blue dashed path. However, when cables are kept taut, the red solid cable length (29.44 units, where 1 unit = the discretizing size) is longer than the red dashed cable length (29.28 units).

maintain an arborescence that gives for each vertex v the vertex that is visited just before v, denoted $prev[v]$, and we maintain the best known distance from the root r to v, denoted $dist[v]$. However this distance is not the length of the path used to reach v (as defined in $prev$), but the length of the cable configuration when v is reached from $prev[v]$. This length is computed by calling $NextConfig$, as defined in Algorithm 8. This allows us to compute in polynomial time an optimal solution that satisfies the cable length constraint.

Another advantage to construct a coverage tree such that the Euclidean distance from the anchor point to each vertex is minimized ensures that the resulted Hamiltonian path will never lead to any cable entanglements (see Fig. 6.3 (b)).

Theorem 6.2.1. When there are no forbidden areas in the workspace, the **MCT** problem can be optimally resolved within $\mathcal{O}(|\mathcal{V}_4| \log |\mathcal{V}_4| \alpha(|\mathcal{V}_\mathcal{O}|))$.

Proof. Algorithm 9 is an adaption of the classical Dijkstra's algorithm, and its principles are similar. Consequently, it has a time complexity of $\mathcal{O}(|\mathcal{V}_4| \log |\mathcal{V}_4|)$, as the number of edges in G_4 is in $\mathcal{O}(|\mathcal{V}_4|)$. Additionally, each invocation of $NextConfig$ (defined in Algorithm 8) requires $\mathcal{O}(\alpha(|\mathcal{V}_\mathcal{O}|))$ time, resulting in an overall complexity of $\mathcal{O}(|\mathcal{V}_4| \log |\mathcal{V}_4| \alpha(|\mathcal{V}_\mathcal{O}|))$. It is important to note that the value of $|\mathcal{V}_4|$ is limited by the cable length ℓ in this expression. □

6.2.2 MCT Problem with Forbidden Areas

More generally, when the workspace contains forbidden areas, we show that the presence of these forbidden areas increases the complexity of the problem of deciding whether it is possible to find a coverage tree with at least k vertices. We prove that this problem becomes $\mathcal{NP}$-complete by a reduction from planar 3-SAT [Lic82].

The decision problem associated with the **MCT** problem is defined as follows.

Instance: A workspace $\mathcal{W}$ and the associated graph G_4, root node r, an integer k

Question: Does there exist a coverage tree T of G_4 rooted at r that contains at least k vertices?

Theorem 6.2.2. In case of forbidden areas, the decision problem associated with the **MCT** problem is $\mathcal{NP}$-complete.

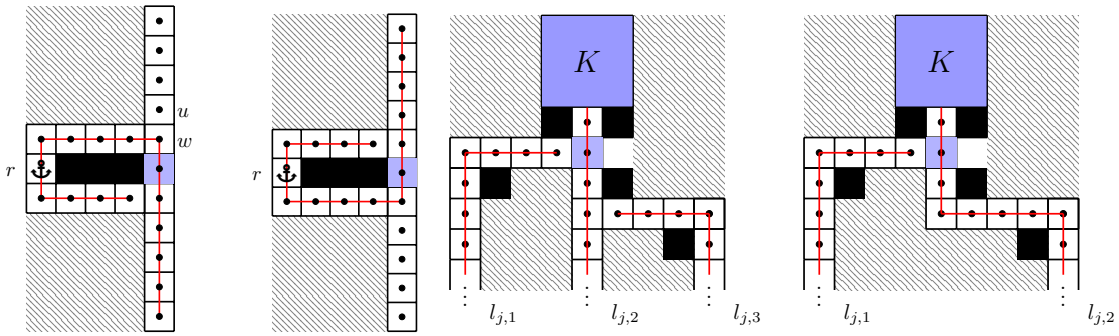
Proof. The problem belongs to $\mathcal{NP}$. Starting from the source and following T, the cable configuration at each vertex can be incrementally computed from its predecessor according to Algorithm 8. It can be checked whether a given tree T has a size greater than k, and subsequently, whether each vertex satisfies the conditions specified in Definition 2.1.2 (Cable configuration), all within polynomial time.

To prove the $\mathcal{NP}$-hardness, we reduce the Separable Planar 3-SAT problem to our problem, and the former has been proved to be $\mathcal{NP}$-Complete [Lic82]. An instance of Separable Planar 3-SAT is defined by a triple (X, F, μ) such that:

- $X = \{x_1, ..., x_n\}$ is a set of n boolean variables;
- $F = C_1 \wedge ... \wedge C_m$, a conjunction of m clauses, where $C_j = l_{j,1} ... \vee l_{j,k}$ a disjunction of 2 or 3 literals;
- every variable x_i occurs in 2 or 3 clauses and at least once positively and once negatively;
- $\mu : X \cup F \rightarrow \mathbb{R}^2$ is is a plane embedding that associates 2D coordinates to every variable and every clause such that (i) it is possible to draw lines between every clause and its variables without line crossings, and (ii) when drawing a line that goes through all clauses, thus separating the space in two parts, every positive occurrence of a given variable are located in the same part whereas every negative occurrence of this variable are located in the other part, as illustrated on the top of Fig. 6.6 (see [Sol+15] for more details).

From a Separable Planar 3-SAT instance (X, F, μ), a maximum coverage tree instance is constructed as follows.

- For each variable x_i, the workspace $\mathcal{W}$ contains a "T"-shaped gadget where the empty zone can be considered as the outside of $\mathcal{W}$ or forbidden zones (see the gadget for x_i in Fig. 6.5(a) and 6.5(b)). In this particular workspace, the robot sits initially at the anchor point r. We cannot find a coverage tree to visit all the free cells, as a choice must be made to either discover the vertices on the top or on the bottom. For example, in Fig. 6.5(a), the vertex u cannot be connected to w, otherwise, in order to reach u, the cable will be outside of the workspace. According the direction to expand the tree at the blue cell, we label the gadget as x_i or $\neg x_i$.
- For each clause $C_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$, we construct a gadget (see Fig. 6.5(c)) which corresponds to the junction of the gadgets of C_j's literals. The junction vertex (blue cell) is connected to a large extended area of size K that can be reached from it. That area is convex and does not contain any forbidden areas. When the junction vertex can be reached from more than one "arm" of these literal



(a) x_i (b) $\neg x_i$ (c) $C_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$ (d) $C_j = l_{j,1} \vee l_{j,2}$

FIGURE 6.5: Gadgets. Black cells are obstacles; forbidden areas are displayed in grey. (a): gadget for x_i when x_i is set to true; (b): gadget for x_i when x_i is set to false; (c) gadget for a clause with three literals; (d) gadget for a clause with two literals.

gadgets, we connect it to only one of them, otherwise, a cycle occurs in the coverage tree. In addition, the clause gadget ensures that when the blue cell is connected by an arm, it cannot continue to expand the other two arms in the opposite direction.

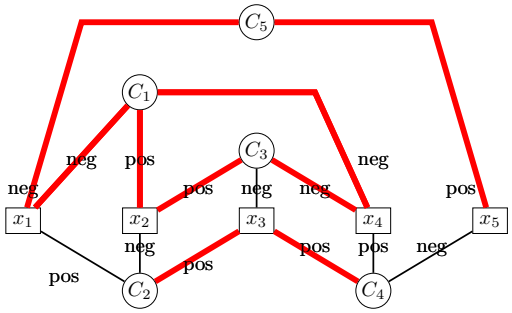
- For each clause $C_j = l_{j,1} \vee l_{j,2}$, a similar gadget can be constructed as shown in Fig. 6.5(d).

These gadgets of variable X are arranged in a straight line, with the two "arms" of each gadget being positioned respectively towards above and below them. The separable embedding μ of the planar graph associated with (X, F) ensures that the clause gadgets are properly nested so that the arms between variables and clauses do not cross each other. This condition is important because if there is any other crossing vertex except the junction vertex, there must be a cycle in the solution. This is why we use an instance of Separable Planar 3-SAT to make this reduction, rather than the classical 3-SAT. We make K greater than the total number of the vertices in order to reach these extended areas associated to each clause. This condition can be satisfied if we scale properly the length of each arm and the bridge parts. After such a workspace $\mathcal{W}$ has been constructed, then we set $k = mK$. Fig. 6.6 displays an example of the reduction from the Separable Planar 3-SAT instance to a workspace for coverage task. The construction can be done in polynomial time in the size of F. We now prove that F is satisfiable iff G has a coverage tree of size greater than k.

$\boxed{\Rightarrow}$ Let $\{\tilde{x}_1, ..., \tilde{x}_n\}$ be a truth assignment of X satisfying F. For each variable x_i, the robot chooses an arm oriented towards above or below to discover. A true clause C_j means that at least one of its literal is set to true, this is to say, the robot can follow at least one arm to reach to the extended area of size K associated to C_j. Therefore, the resulting coverage tree has a size greater than k.

$\boxed{\Leftarrow}$ On the other hand, if the workspace that we construct in this way has a solution T with $\#T >= k$, it forces the robot to reach all the extended areas associated to the clauses, since we assume that K is greater than the total size of free vertices of

$$
\begin{aligned}
X &= \{x_1, x_2, x_3, x_4, x_5\} \\
F &= (\neg x_1 \vee x_2 \vee \neg x_4) \\
&\wedge (x_1 \vee \neg x_2 \vee x_3) \\
&\wedge (x_2 \vee \neg x_3 \vee \neg x_4) \\
&\wedge (x_3 \vee x_4 \vee \neg x_5) \\
&\wedge (\neg x_1 \vee x_5)
\end{aligned}
$$



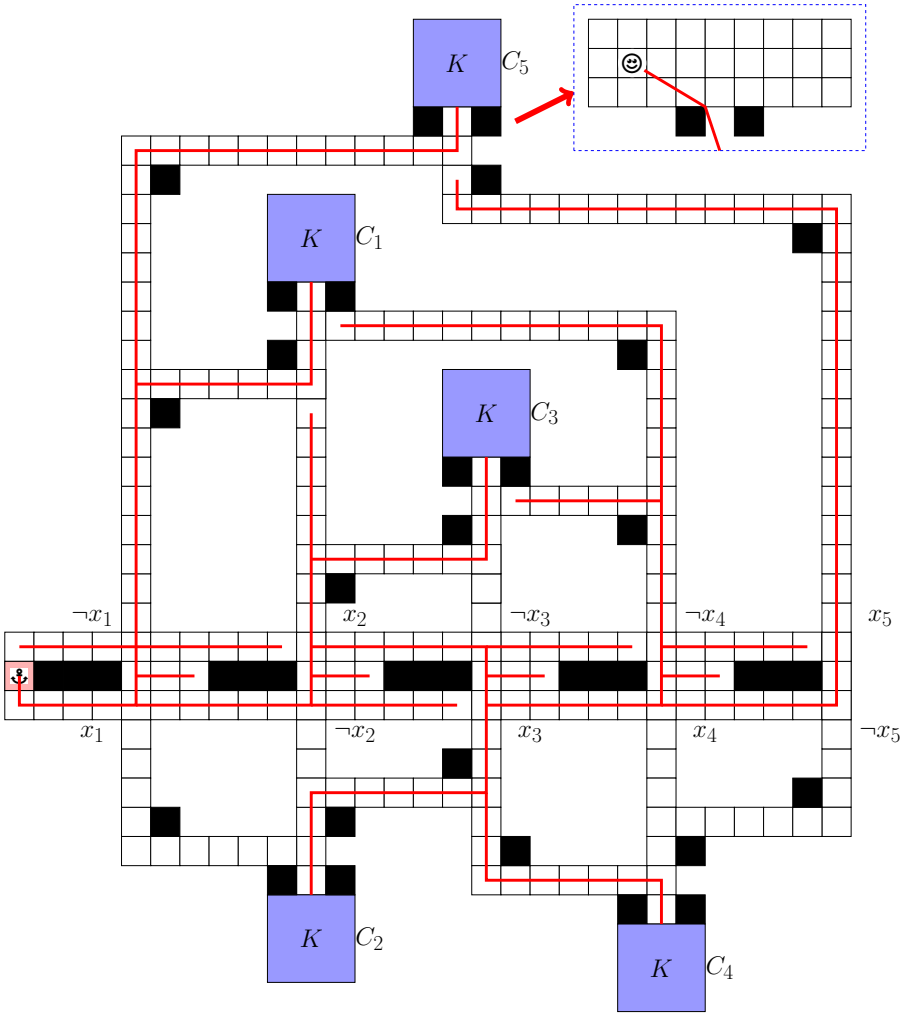


FIGURE 6.6: Proof by a reduction from a Separable Planar 3-SAT instance. Top right: the embedding graph of X where the edges marked in red is a satisfiable solution for X. Bottom: the workspace for coverage constructed from X and the corresponding optimal solution (in red). Each blue block represents an extended area of K cells which can be reached by the robot, as illustrated in the dashed box. Empty areas are considered as forbidden zones.

the variable gadgets and that of junction area in each clause. To make a clause true, the robot can choose to follow either the arm above or below. If it follows an arm corresponding to a positive (resp., negative) form of the variable $\tilde{x}_i$ belongs to this clause, let $\tilde{x}_i = true$ (resp., false). If the robot does not follow any arm, it means that the value of x_i is irrelevant, and we choose $\tilde{x}_i = true$ by convention. The resulting assignment satisfies F.

□

6.3 Configuration Enumeration Based Approach (Model $\mathcal{M}_0$)

We begin by introducing an exact approach to address the MCT problem with forbidden areas. In the proof of Theorem 6.2.2, it becomes evident that the complexity of the maximum coverage tree problem arises due to certain vertices being reachable only with specific cable configurations, in the presence of forbidden areas. The challenge lies in the fact that selecting a configuration to reach one cell may be incompatible with reaching another cell. To compute an optimal coverage tree, it is necessary to explore all possible cable configurations to reach each cell. The interconnections between these configurations can be effectively represented as a graph, similar to the homotopy-augmented graph.

Let us first introduce the notion of consistent configuration, that corresponds to a valid cable configuration from the anchor point to a vertex of G_4: a sequence $s = q_1q_2...q_k$ is a consistent configuration if (i) $q_1 = r$, (ii) $q_k \in \mathcal{V}_4$, (iii) $\forall i \in [2, k-1]$, $q_i \in \mathcal{V}_\mathcal{O}$, (iv) s is the shortest path in its homotopy class, (v) s is non self-crossing, and (vi) the length of s does not exceed ℓ.

Definition 6.3.1 (The configuration graph)**.** The configuration graph associated with G_4 is the graph $G_{cfg} = (\mathcal{V}_{cfg}, \mathcal{E}_{cfg})$ such that $\mathcal{V}_{cfg}$ is the set of all consistent configurations, $\mathcal{E}_{cfg} = \{(q_1...q_k, q'_1...q'_l) \in \mathcal{V}_{cfg} \times \mathcal{V}_{cfg} \mid (q_k, q'_l) \in \mathcal{E}_4$ and $q'_1...q'_l$ is homotopic to $q_1...q_kq'_l\}$.

Each vertex in the configuration graph corresponds to a consistent configuration to reach some cell, and each edge connects two adjacent cells through a safe path. As illustrated in Fig. 6.7, for instance, there could be three possible configurations to reach cell 13: $r13$, $rABCD13$ and $rDCBA13$, while two configurations exist for cell 14, namely $r14$ and $rDCB14$. When the cable configuration at cell 13 is $rABCD13$, the robot cannot advance to cell 14 without causing a self-crossing of the cable. There are $|\mathcal{V}_4|$ cells in the workspace and each of them can be reached from at most $2^{|\mathcal{O}|}$ kinds of homotopy class where $|\mathcal{O}|$ is the number of obstacles. Consequently, the size of $\mathcal{V}_{cfg}$ can be expressed as $\mathcal{O}(2^{|\mathcal{O}|}|\mathcal{V}_4|)$.

To efficiently enumerate all the consistent configurations, we introduce a tree-structured graph representing the set of valid subconfigurations that allows to construct G_{cfg}.

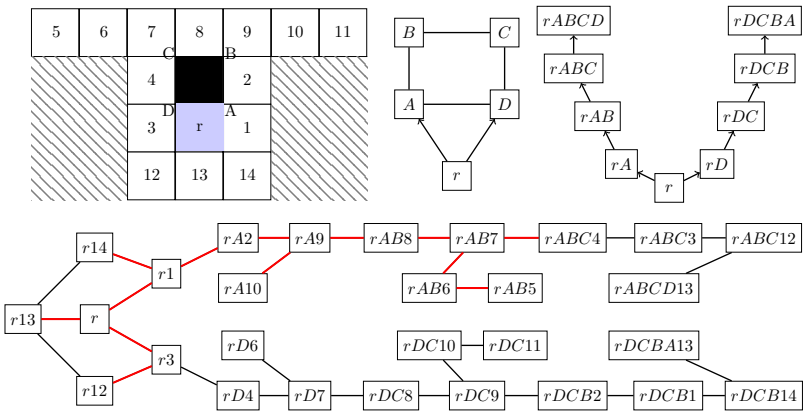


FIGURE 6.7: Example of G_{cfg}. Top left: a workspace that contains an obstacle $ABCD$. The anchor point is at r. The robot must take the path $\langle r, 1, 2, 9, 8, 7, 6, 5\rangle$ in order to reach the cell 5. Similarly, the cell 11 can be only reached by the path $\langle r, 3, 4, 7, 8, 9, 10, 11\rangle$. Top middle: the visibility graph G_{vis}. Top right: the associated prefix configuration graph. Bottom: The corresponding configuration graph. The tree composed of these red edges is an optimal solution for this example.

Definition 6.3.2 (The prefix configuration graph)**.** Let $G_{vis} = (\mathcal{V}_{vis}, \mathcal{E}_{vis})$ be the visibility graph associated with the workspace and the anchot point. The prefix configuration graph is the directed graph $G_p = (\mathcal{V}_p, \mathcal{E}_p)$ such that $\mathcal{V}_p$ is the set of all cable configuration prefixes, *i.e.*, $\mathcal{V}_p = \{q_1...q_k \mid \exists q_{k+1} \in \mathcal{V}_4 \text{ and } q_1...q_{k+1} \in \mathcal{V}_{cfg}\}$, $\mathcal{E}_p = \{(q_1...q_k, q_1...q_kq_{k+1}) \in \mathcal{V}_p \times \mathcal{V}_p \mid (q_k, q_{k+1}) \in \mathcal{E}_{vis}\}$.

An example of G_p is illustrated in Fig. 6.7 (top right). The prefix configuration graph can be constructed by performing a DFS on G_{vis}. Starting from r, each node of G_p represents a consistent cable configuration and can be incrementally discovered by adding an edge of $\mathcal{E}_{vis}$ to its parent node. To add an edge $(q_k, q_{k+1}) \in \mathcal{E}_{vis}$ to the configuration $q_1 \dots q_k$ associated with a parent node, it is necessary to check that (q_k, q_{k+1}) does not cross $q_1 \dots q_k$ and that $q_1 \dots q_{k+1}$ is a taut path, which requires linear time in terms of $|\mathcal{V}_{\mathcal{O}}|$ if the visibility graph is ready. The exploration process terminates when the length of the consistent configuration exceeds ℓ. Thus constructing G_p takes $\mathcal{O}(|\mathcal{V}_p||\mathcal{V}_{\mathcal{O}}|) = \mathcal{O}(2^{|\mathcal{O}|}|\mathcal{V}_{\mathcal{O}}|^2)$ time where $|\mathcal{V}_p| = \mathcal{O}(2^{|\mathcal{O}|})$, and constructing G_{cfg} from G_p takes $\mathcal{O}(2^{|\mathcal{O}|}|\mathcal{V}_4|)$ time.

We define $V_c = \{q_1...q_k \in \mathcal{V}_{cfg} \mid q_k = c\}$ as the set of configurations that can reach a cell c. As a result, $\mathcal{V}_{cfg}$ can be represented as a collection of clusters: $\mathcal{V}_{cfg} = \bigcup_{c \in \mathcal{V}_4} V_c$. Let $G_C = (\mathcal{V}_C, \mathcal{E}_C)$ denote the quotient graph that characterizes the overall connections between different clusters. Here, $\mathcal{V}_C$ is the set $\{V_1, \dots, V_{|\mathcal{V}_4|}\}$, and $(V_i, V_j) \in \mathcal{E}_C$ if there exists $v_i \in V_i$ and $v_j \in V_j$ such that $(v_i, v_j) \in \mathcal{E}_{cfg}$.

The interest of constructing G_{cfg} and G_C is that our initial problem of finding a maximal coverage tree in G_4 is transformed into identifying a maximal tree $T_{cfg} = (\mathcal{V}_{T_{cfg}}, \mathcal{E}_{T_{cfg}})$ in G_{cfg}, such that: (i) T_{cfg} is rooted at r; (ii) for each cell $c \in \mathcal{V}_4$, $|\mathcal{V}_{T_{cfg}} \cap V_c| \leq 1$. Constraint (ii) implies that for each cell, we can select at most one configuration to reach it. An optimal solution refers to any maximum tree that meets

conditions (i) and (ii).

With this approach, considering that the size of each cluster V_c is limited to $2^{|\mathcal{O}|}$, the complexity of searching for a maximum tree in G_{cfg} can be bounded by $\mathcal{O}(2^{|\mathcal{V}_4||\mathcal{O}|})$. Consequently, the overall complexity of this approach amounts to $\mathcal{O}(2^{|\mathcal{V}_4||\mathcal{O}|} + 2^{|\mathcal{O}|}(|\mathcal{V}_4| + |\mathcal{V}_\mathcal{O}|^2))$.

We propose using integer linear programming to solve this problem. We adopt the *local-global* formulation proposed in the Section 4.4 of [Pop20]. The 0-1 variable $x_e = x_{ij}$ equals 1 iff the edge $e = (i, j) \in \mathcal{E}_{cfg}$ is selected, i.e. appears in the solution tree T_{cfg}, and the 0-1 variable z_i for the vertex $i \in \mathcal{V}_{cfg}$ equals 1 when the solution tree T_{cfg} contains vertex i. The 0-1 vector y describes a spanning tree on G_C: $y_{ij} = 1$ means that $(V_i, V_j) \in \mathcal{E}_C$. For every triple of nodes (k, i, j), λ_{kij} equals to 1 if j is the parent of i when the tree is rooted at k, otherwise 0. The integer linear program is:

$$\max \quad \sum_{i \in \mathcal{V}_{cfg}} z_e$$

$$\text{s.t.} \quad \sum_{i \in V_k} z_i \leq 1 \qquad \forall k \in K = \{1, ..., |\mathcal{V}_C|\} \tag{6.1a}$$

$$z_{root} = 1 \tag{6.1b}$$

$$\sum_{j \in V_r} x_{ij} \leq z_i \qquad \forall r \in K, \forall i \in \mathcal{V}_{cfg} \setminus V_r \tag{6.1c}$$

$$\sum_{e \in \mathcal{E}_{cfg}} x_e = \sum_{i \in \mathcal{V}_{cfg}} z_i - 1 \tag{6.1d}$$

$$\sum_{i \in V_l, j \in V_r} x_{ij} \leq y_{lr} \qquad \forall l, r \in K, l \neq r \tag{6.1e}$$

$$\sum_{i,j} y_{ij} = |\mathcal{V}_C| - 1 \tag{6.1f}$$

$$y_{ij} = \lambda_{kij} + \lambda_{kji} \qquad \forall 1 \leq k, i, j \leq |\mathcal{V}_C|, i \neq j \tag{6.1g}$$

$$\sum_{j} \lambda_{kij} = 1 \qquad \forall 1 \leq k, i, j \leq |\mathcal{V}_C|, i \neq k \tag{6.1h}$$

$$y_{kkj} = 0 \qquad \forall 1 \leq k, j \leq |\mathcal{V}_C| \tag{6.1i}$$

$$y_{ij}, \lambda_{kij} \geq 0 \qquad \forall 1 \leq k, i, j \leq |\mathcal{V}_C| \tag{6.1j}$$

$$x_e, z_i \in \{0, 1\} \qquad \forall i \in \mathcal{V}_{cfg}, \forall e \in \mathcal{E}_{cfg} \tag{6.1k}$$

Compared to the GMSTP problem described in Example 3.2.1 (see Chapter 3), this problem does not require a vertex to be selected in each cluster, as indicated in constraint (6.1a). In addition, our problem is to maximize the objective function, as opposed to minimizing it in the GMSTP. The cluster constraint is addressed through constraints (6.1a)-(6.1d). Constraints (6.1f)-(6.1k) ensure that y constitutes a spanning tree. Constraint (6.1e) requires the solution being part of y, thereby containing no cycles. In ensuring that y is a tree structure, the introduction of the auxiliary variables λ avoids the elimination of all sub-tours from the solution, instead of explicitly enumerating them. Besides, it is worth noting that constraint (6.1f) becomes

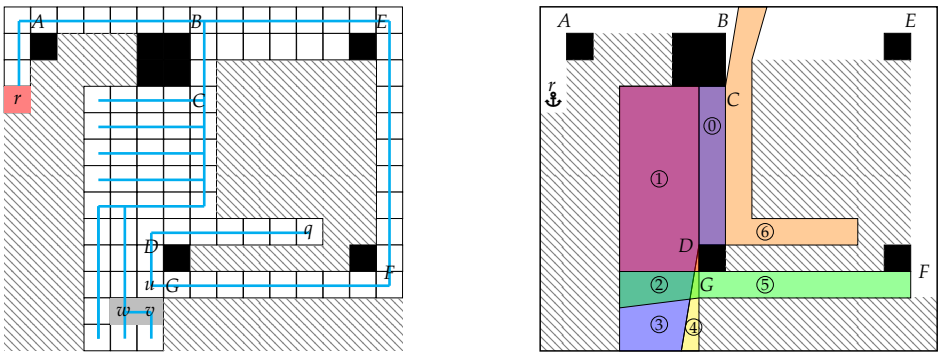


FIGURE 6.8: An example showing that the search of a maximum tree in $\hat{G}_{cfg}$ is a suboptimal solution. Left: the cyan path is an optimal solution found in G_{cfg} that can cover all the cells in the workspace. Right: a partition (incomplete) of the workspace associated with G_q. The small regions between I_2, I_5 and the obstacle with apex D can be ignored compared to the discretization precision.

unnecessary in our case, given our objective of maximizing the objective function, favoring the inclusion of more edges in y while retaining its tree structure.

6.4 Approximation Algorithms

The exact methods described above can be difficult to solve for some hard instances (see Section 6.5). We introduce some approximation methods in this section.

6.4.1 Model $\mathcal{M}_1$

The graph G_{cfg} in which we search for a maximum tree is not directed. A first relaxation may be obtained by directing its edges, using the length of cable configurations to define edge orientations. More precisely, given a non directed configuration graph G_{cfg}, we define the directed graph $\hat{G}_{cfg} = (\hat{\mathcal{V}}_{cfg}, \hat{\mathcal{E}}_{cfg})$ such that $\hat{\mathcal{E}}_{cfg} = \{(q_1 \dots q_k, q'_1 \dots q'_l) \in \mathcal{E}_{cfg} \mid$ the length of $q_1 \dots q_k$ is smaller than the length of $q'_1 \dots q'_l\}$. Note that $\hat{G}_{cfg}$ is a Directed Acyclic Graph (DAG) as it is not possible to have a cycle of decreasing configuration lengths.

It can be proven that a lower bound may be computed by finding the maximum arborescence in $\hat{G}_{cfg}$ due to the fact that $\hat{G}_{cfg}$ is a subgraph of G_{cfg}. An example illustrating the suboptimality of this solution is shown in Fig. 6.8. In the optimal solution, cell q can only be reached via the configuration $rABEFGDq$ which passes through u, and consequently v cannot be connected to u and must be connected to w. In such way, it can be observed that $rABCDv$ is smaller than $rABCw$, but the edge $(rABCw, rABCDv) \notin \hat{\mathcal{E}}_{cfg}$. This implies that working with $\hat{G}_{cfg}$ may yield a suboptimal solution. However, investigating this lower bound is still valuable, as it can be of good quality for certain instances, and can be solved more efficiently than the exact solution, as we will explore further in the subsequent sections.

The ILP model can be also used to solve the relaxed problem. In comparison to the model described in Section 6.3, we can simplify the tree constraint and eliminate the variables y since $\hat{G}_{cfg}$ is a DAG.

$$\max \quad \sum_{i \in \hat{\mathcal{V}}_{cfg}} z_i$$

$$\text{s.t.} \quad \sum_{i \in V_k} z_i \leq 1 \qquad \forall k \in K = \{1, ..., |\mathcal{V}_4|\} \tag{6.2a}$$

$$z_{root} = 1 \tag{6.2b}$$

% Each node (except the root) has one parent node

$$\sum_{j \in V_r} x_{ij} = z_i \qquad \forall r \in K, \forall i \in \hat{\mathcal{V}}_{cfg} \setminus V_r \tag{6.2c}$$

$$\sum_{e \in \hat{\mathcal{E}}_{cfg}} x_e = \sum_{i \in \hat{\mathcal{V}}_{cfg}} z_i - 1 \tag{6.2d}$$

%The two nodes of a selected edge are selected

$$x_{ij} \leq z_i z_j \qquad \forall e = (i, j) \in \hat{\mathcal{E}}_{cfg} \tag{6.2e}$$

$$x_e, z_i \in \{0, 1\} \qquad \forall i \in \hat{\mathcal{V}}_{cfg}, \forall e \in \hat{\mathcal{E}}_{cfg} \tag{6.2f}$$

In the case of a DAG, the tree constraint can be simply ensured by constraints (6.2c)-(6.2e). Constraint (6.2e) can be formulated with linear constraints by introducing the auxiliary variables w_e associated to each edge e. For each edge $e = (i, j) \in \hat{\mathcal{E}}_{cfg}$, it necessitates satisfying:

$$\begin{aligned} x_e &\leq w_e \\ w_e &\leq z_i \\ w_e &\leq z_j \\ z_i + z_j - 1 &\leq w_e \end{aligned}$$

6.4.2 A Fixed-Parameter Tractable (FPT) Algorithm (Model $\mathcal{M}_2$)

As shown above, the size of $\mathcal{V}_{cfg}$ is proportional to $|\mathcal{V}_4|$, and the complexity of computing an optimal solution is at least exponential to $|\mathcal{V}_4|$. Consider the case where a large workspace contains only few obstacles. Since proximately distributed cells generally have the same type of accessibility (or homotopy class), we need only make choice at some critical cells and their neighbors can be reached with the same homotopy class. In this section we will introduce a more compact representation of G_{cfg} by constructing its quotient graph with respect to the equivalence relation defined below.

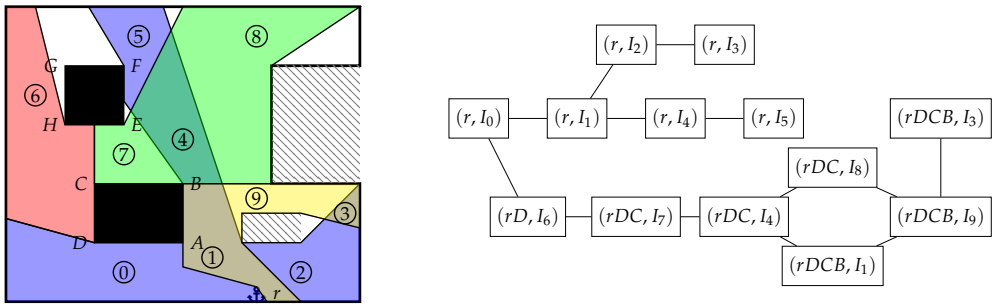


FIGURE 6.9: Example of a partition (incomplete) of the workspace. Left: the workspace is divided into a set of disjoint subregions, numbered from 0 to 9. We note the k^{th} subregion as I_k, and I_0 is where the root is located. Right: the corresponding G_q.

Definition 6.4.1 (The equivalence relation on $\mathcal{V}_{cfg}$)**.** For each cell $c \in \mathcal{V}_4$, let $\Pi(c) = \{\pi \mid \pi \cdot c \in \mathcal{V}_{cfg}\}$ be the set of prefix cable configuration to reach c. Two configurations $\pi_1 \cdot c_1, \pi_2 \cdot c_2 \in \mathcal{V}_{cfg}$ are equivalent if $\pi_1 = \pi_2$ and $\Pi(c_1) = \Pi(c_2)$. We denote $\pi_1 \cdot c_1 \sim \pi_2 \cdot c_2$.

Definition 6.4.2 (Quotient graph of G_{cfg})**.** Let $\mathcal{V}_p$ be the set of consistent configuration prefixes, and $Part(\mathcal{V}_4)$ be the set containing all subsets of $\mathcal{V}_4$. The quotient graph of G_{cfg} is the graph $G_q = (\mathcal{V}_q, \mathcal{E}_q)$ such that

- $\mathcal{V}_q = \{(\pi, I) \in \mathcal{V}_p \times Part(\mathcal{V}_4) \mid \forall c_1, c_2 \in I, \pi \cdot c_1 \sim \pi \cdot c_2\}$
- $\mathcal{E}_q = \{((\pi_1, I_1), (\pi_2, I_2)) \in \mathcal{V}_q \times \mathcal{V}_q \mid \exists c_1 \in I_1, c_2 \in I_2$ such that $(\pi_1 \cdot c_1, \pi_2 \cdot c_2) \in \mathcal{E}_{cfg}\}$

A geometrical illustration[3] of G_q is showed in Fig. 6.9. Let π be a prefix configuration, we denote $\mathcal{P}(\pi) = \{v \in \mathcal{V}_4 \mid \pi \cdot v \in \mathcal{V}_{cfg}\}$. For example, $\mathcal{P}(rD) = I_6$. When the workspace and obstacles are all polygonal, $\mathcal{P}(\pi)$ should also be polygonal. These polygons intersect and are split into a set of disjoint and smaller polygons, like $\mathcal{P}(r) = I_0 \cup I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5$ and $\mathcal{P}(rDC) = I_7 \cup I_4 \cup I_8$. Each vertex $(\pi, I) \in G_q$ corresponds to the subregion of all cells in I when they are reached from π. The connectivity between two adjacent subregions (π_1, I_i) and (π_2, I_j) forms an edge in $\mathcal{E}_q$.

Let $\mathbb{I} = \{I_i \mid \exists \pi_i \in \mathcal{V}_p$ such that $(\pi_i, I_i) \in \mathcal{V}_q\}$ be the partitioning of G_4 associated with $\mathcal{V}_q$. We define $V_I = \{(\pi, I) \in \mathcal{V}_q\}$ as a cluster that contains all the vertices originated at I. Then $\mathcal{V}_q$ can be written as the disjoint union of a set of clusters $\bigcup_{I \in \mathbb{I}} V_I$.

Observation 6.4.1. From the definition of G_q, we can learn that for any vertex $(\pi, I) \in \mathcal{V}_q$, if there exists $c \in I$ that appears in the objective coverage tree with $\pi \cdot c$ being the associated cable configuration, then the other cells in I can be also covered in the same way since I is connected.

[3]For visualization matters, this partition is incomplete and some areas are not taken into account, as the same case for Fig. 6.8.

This property implies that instead of searching for a maximum tree in G_{cfg}, we could first find a maximum tree T_q in G_q and then connect the cells contained in each vertex of T_q in polynomial time. With this in mind, we can explore how to build T_q from G_q and whether T_q allows to construct an optimal solution for our original problem. To ascertain this, it is crucial to investigate the arrangement of these partitions in $\mathbb{I}$.

Consider five adjacent regions $I_{u_1}, I_{u_2}, I_{v_1}, I_{v_2}, I_w$ in $\mathbb{I}$ as illustrated in Fig. 6.10. Any two regions, for instance, I_{u_1} and I_w, can be connected if there exist π_1 and π_2 such that $((\pi_1, I_{u_1}), (\pi_2, I_w)) \in \mathcal{E}_q$. Let us assume we have four edges in $\mathcal{E}_q$: $e_1 = ((\pi_1, I_{u_1}), (\pi_2, I_w))$, $e_2 = ((\pi_2, I_w), (\pi_3, I_{v_1}))$, $e_3 = ((\pi_4, I_{u_2}), (\pi_2, I_w))$, and $e_4 = ((\pi_2, I_w), (\pi_5, I_{v_2}))$. We observe that the presence of e_1 and e_2 in T_q may not always be compatible with the presence of e_3 and e_4. Their relationship can be classified into two modes, as illustrated in Fig. 6.10.

- *The exclusive mode*. In this case, to construct an optimal solution, the presence of e_1 and e_2 will prohibit e_3 and e_4. Consider the example in Fig. 6.9. If the objective tree covers the cells in (r, I_5), then it must pass through (r, I_4). In this case, the tree can cover partial cells in (rDC, I_4), but cannot reach (rDC, I_8), or vice versa. If it covers (rDC, I_8), then it cannot cover (r, I_5), without creating a cycle. In other words, if the coverage tree covers (r, I_5), whether it covers (rDC, I_4) or not will not contribute to the size of the tree. To address this, an exclusive constraint is applied to V_{I_4}, assuming the tree either completely covers the nodes of (r, I_4) or (rDC, I_4).

- *The sharing mode*. In this case, the cells of I_w are partitioned into two parts: one part is connected to I_{u_1} and I_{v_1}, while the other is connected to I_{u_2} and I_{v_2}. As illustrated in Fig. 6.8 (right), in the optimal solution, we observe that $(rABC, I_0) \rightarrow (rABC, I_1) \rightarrow (rABC, I_2) \rightarrow (rABC, I_3) \rightarrow (rABCD, I_4)$ and $(rABEF, I_5) \rightarrow (rABEF, I_2) \rightarrow (rABEFG, I_1) \rightarrow (rABEFGD, I_0) \rightarrow (rABEFGD, I_6)$. This implies that I_0, I_1, and I_2 are shared by two different branches in T_q, and cannot be exclusively covered by a single configuration, as in the exclusive mode, in order to achieve an optimal solution.

Based on Observation 6.4.1, we can derive a lower bound for our original problem by reformulating it as a tree search problem in G_q. In this reformulation, we assign a cost to every $(\pi, I) \in \mathcal{V}_q$ as $|I|$, and the objective is to find a tree $T_q = (\mathcal{V}_{T_q}, \mathcal{E}_{T_q}) \subset G_q$ that maximizes $\sum_{(\pi,I)\in\mathcal{V}_{T_q}} |I|$. However, there are two constraints to consider: (i) T_q is rooted at (r, I_r) where $r \in I_r$; (ii) for each $I \in \mathbb{I}, \#\mathcal{V}_{T_q} \cap V_I \leq 1$. The constraint (ii) implies that only the exclusive mode is allowed and T_q can completely cover at most one node in each cluster V_I. We adopt this heuristic strategy to avoid refining each node, as required in the sharing mode.

The number of clusters in $\mathcal{V}_q$ refers to computing the number of partitions illustrated in Fig. 6.9, thus should be polynomial to the number of obstacle vertices,

FIGURE 6.10: Illustration of two types of connectivity between vertices in G_q. (a) The exclusive mode: if I_{u_1} and I_{v_1} are connected through I_w, then I_{u_2} and I_{v_2} cannot be connected, and vice versa. (b) The sharing mode: I_{u_1} and I_{u_2} can be respectively connected to I_{v_1} and I_{v_2} through I_w simultaneously.

i.e., $|\mathcal{V}_\mathcal{O}|^\beta$ [4]. Within this approach, since the size of each cluster V_I is exponential to the number of obstacles, the complexity to find the optimal solution should be $\mathcal{O}(2^{|\mathcal{O}||\mathcal{V}_\mathcal{O}|^\beta})$. Compare to $\mathcal{M}_0$, the complexity is reduced from $\mathcal{O}(2^{|\mathcal{O}||\mathcal{V}_4|})$ to $\mathcal{O}(2^{|\mathcal{O}||\mathcal{V}_\mathcal{O}|^\beta})$, where the exponential terms depends on $|\mathcal{V}_\mathcal{O}|$, rather than $|\mathcal{V}_4|$.

We propose to apply the same ILP model as that of $\mathcal{M}_0$ to solve this problem. The only difference is that in this case, the objective function becomes: $\max \sum_{i \in \mathcal{V}_q} c_i z_i$ where c_i is the number of cells in the i^{th} vertex of $\mathcal{V}_q$.

6.4.3 Model $\mathcal{M}_3$ and $\mathcal{M}_3 h$

We can further relax this problem by converting G_q into a DAG, similar to what has been done with G_{cfg}. To achieve this, we can define a precedence order between any two neighboring vertices in V_q as follows.

Definition 6.4.3 (Region order $\prec_r$)**.** For any edge $((\pi_1, I_1), (\pi_2, I_2)) \in \mathcal{E}_q$, we define $\prec_r$ as a strict order on $\mathcal{V}_q$ where $(\pi_1, I_1) \prec_r (\pi_2, I_2)$ if the minimum length of cables configurations in (π_1, I_1) is shorter than that of (π_2, I_2).

Note that $\prec_r$ signifies a strict ordering, indicating that no vertex can precede its own predecessor. In case where (π_1, I_1) and (π_2, I_2) have the same shortest cable configuration length, a predefined order is used to break the equality. Consequently, G_q can be reduced to a DAG, denoted as $\hat{G}_q$, as depicted in Fig. 6.11. A lower bound can be computed by searching for a maximum arborescence $\hat{T}_q$ in $\hat{G}_q$. We can apply the same model as $\mathcal{M}_1$ introduced in Section 6.4.1 to address this problem, and we refer to this model as $\mathcal{M}_3$.

We can also strengthen this model by considering some heuristics. Let (π_1, I_1) and (π_2, I_2) be two vertices in $\hat{T}_q$, and $\pi_1 \neq \pi_2$, then in some cases, it follows that π_1 and π_2 cannot intersect each other. For example, in Fig. 6.8, the vertices $(rABCD, I_4)$ and $(rABEFGD, I_0)$ are not compatible due to the intersection of $rABCD$ and $rABEFGD$ at point D, which would results in cycles in the coverage tree.

[4] The exact value of β is unknown. These partitions are formed by the edges of the visibility graph, whose number is bounded by $|\mathcal{V}_\mathcal{O}|^2$. Depending on the shape of each partition, e.g. a triangular partition is formed by 3 edges, the number of triangular partitions must be less than $\binom{|\mathcal{V}_\mathcal{O}|^2}{3}$. In fact, the number of partitions should be well below this number for geometric constraints.

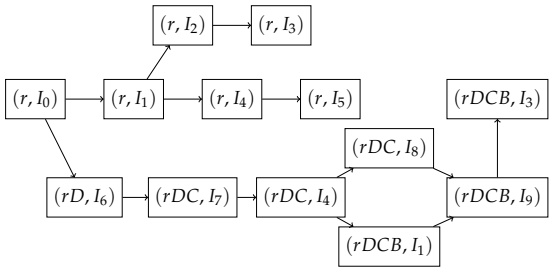


FIGURE 6.11: Illustration of $\hat{G}_q$, corresponding to the example in Fig. 6.9.

For each vertex $v \in \{r\} \cup V_{\mathcal{O}}$, we enumerate all the consistent prefix configurations from r to v as $\Pi(v) = \{\pi \cdot v \in \mathcal{V}_p\}$. Let $\Pi_{\hat{T}_q} = \{\pi \mid (\pi, I) \in \hat{T}_q\}$ be the set of prefix configurations issued with $\hat{T}_q$, then the heuristic that any two prefix configurations $\pi_1, \pi_2 \in \Pi_{\hat{T}_q}$ cannot intersect can be interpreted as the following constraints in G_p:

- (i) For each vertex $v \in \{r\} \cup V_{\mathcal{O}}$, at most one prefix configuration is selected from $\Pi(v)$.
- (ii) if a prefix configuration $rq_1q_2..q_k \in \mathcal{V}_p$ is selected, then all its sub-configurations $rq_1q_2...q_l$ with $l < k$ are also selected.
- (iii) If two intersecting edges $\overline{u_1v_1}, \overline{u_2v_2} \in \mathcal{E}_{vis}$ intersect at a point other than their ends (e.g., $\overline{BH}$ and $\overline{CE}$ in Fig. 6.9), then it is not possible to simultaneously select any two prefix configurations π_1 and π_2 such that $\overline{u_1v_1}$ or $\overline{v_1u_1}$ is a segment of π_1 and $\overline{u_2v_2}$ or $\overline{v_2u_2}$ is a segment of π_2.

Based on (i)(ii), we can simplify (iii) by enumerating all the forbidden node pairs $F = \{(\pi_1 \cdot u_1v_1, \pi_2 \cdot u_2v_2) \mid \overline{u_1v_1}$ intersects $\overline{u_2v_2}\}$ and adding a constraint to prevent these forbidden pairs to be simultaneously selected as shown in (6.3c). In fact, the selected nodes together form a subtree of G_p rooted at r, as highlighted in red in Fig. 6.12.

We introduce the 0-1 variable w_i which equals 1 iif $\pi_i \in \mathcal{V}_p$ is selected. Let $R_i = \{(\pi_i, I_k)\} \subset \mathcal{V}_q$ the set of subregions that are reached with π_i. Then we can formulate the above constraints as follows. In the constraint (6.3d), z_j is a 0-1 variable for the presence of a vertex $q_j \in \mathcal{V}_q$ in the solution $\hat{T}_q$. We channel z_j and w_i by the inequality since $w_i = 1$ is a necessary condition for any element from R_i to be selected.

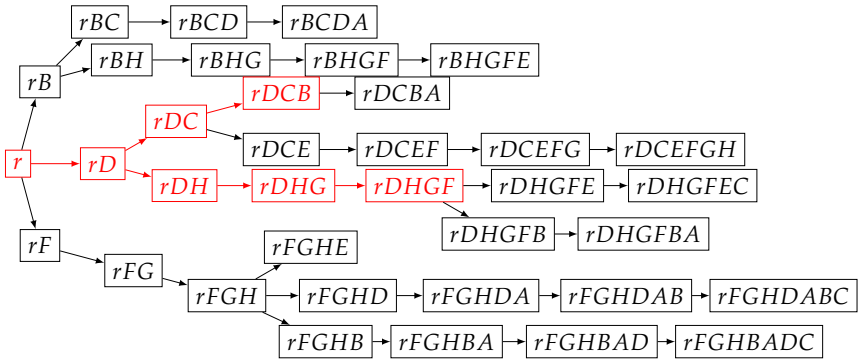


FIGURE 6.12: An example of the prefix configuration graph G_p associated with the workspace in Fig. 6.9.

.

$$w_i \geq w_j \qquad \forall (i,j) \in \mathcal{E}_p \tag{6.3a}$$

$$\sum_{\pi_j \in \Pi(u) \cap \mathcal{V}_p} w_j \leq 1 \qquad \forall u \in \mathcal{V}_{vis} \tag{6.3b}$$

$$w_{i_1} + w_{i_2} \leq 1 \qquad \forall (i_1, i_2) \in F \tag{6.3c}$$

$$z_j \leq w_i \qquad \forall q_j \in \mathcal{V}_q \cap R_i, \forall i \in \{1, ..., |\mathcal{V}_p|\} \tag{6.3d}$$

$$w_i \in \{0,1\} \qquad \forall i \in \mathcal{V}_p \tag{6.3e}$$

We denote the new model based on $\mathcal{M}_3$ with these additional constraints from (6.3a) to (6.3e) as $\mathcal{M}_3h$. It is important to note that the solution computed by $\mathcal{M}_3h$ is a lower bound of the result of $\mathcal{M}_3$. In the next section, we will show that $\mathcal{M}_3h$ is more efficient for some hard instances.

6.5 Experimental results

So far, we have introduced 5 models to compute exact or approximate solutions.

- $\mathcal{M}_0$: search for an arborescence in G_{cfg}.
- $\mathcal{M}_1$: transform G_{cfg} to a DAG $\hat{G}_{cfg}$ by defining an order between any two configurations, then search for an arborescence in $\hat{G}_{cfg}$.
- $\mathcal{M}_2$: construct a quotient graph G_q of G_{cfg}.
- $\mathcal{M}_3$: transform G_q to a DAG $\hat{G}_q$.
- $\mathcal{M}_3h$: add some heuristics to restrict $\mathcal{M}_3$.

$\mathcal{M}_0$ is an exact model and all the other four models can provide lower bounds. We use $opt_{\mathcal{M}_0}$, $lb_{\mathcal{M}_1}$, $lb_{\mathcal{M}_2}$, $lb_{\mathcal{M}_3}$ and $lb_{\mathcal{M}_3h}$ to denote the solutions computed by these methods.

Theorem 6.5.1. $opt_{\mathcal{M}_0} \geq lb_{\mathcal{M}_1}$ and $opt_{\mathcal{M}_0} \geq lb_{\mathcal{M}_2} \geq lb_{\mathcal{M}_3} \geq lb_{\mathcal{M}_3h}$

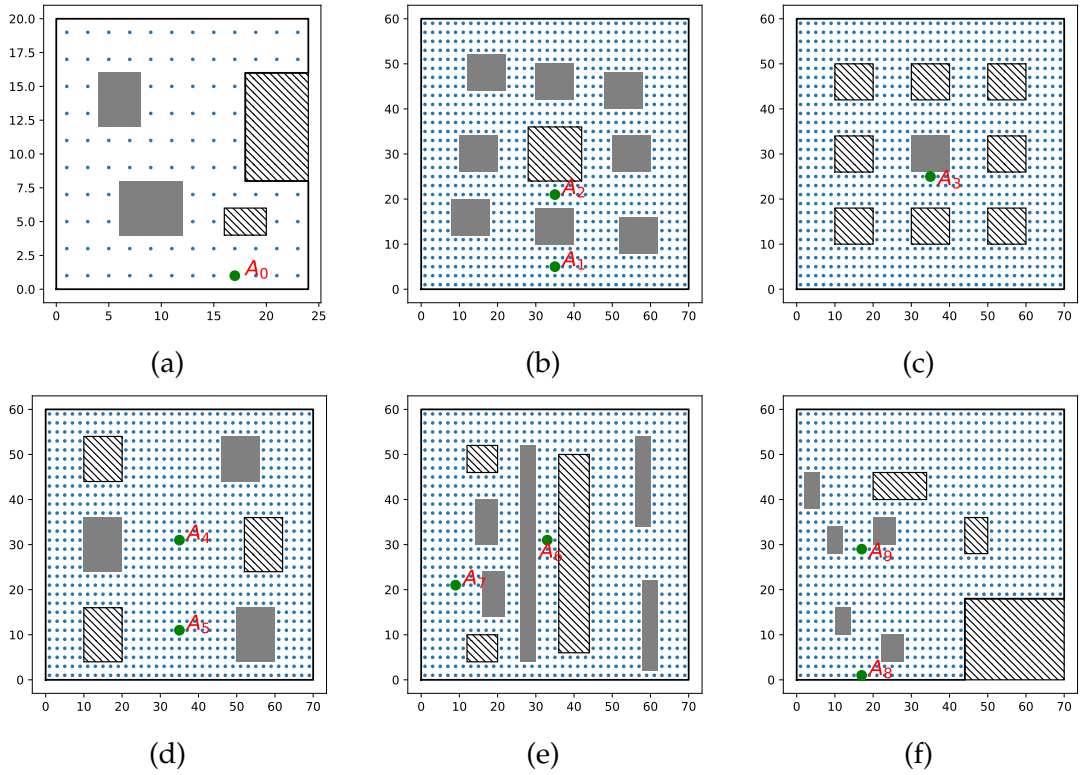


FIGURE 6.13: The 6 simulated workspaces tested in our experimental analysis. In each workspace, the gray rectangles are the obstacles, the forbidden areas are filled with slash lines, and the green points are the anchor points. Workspaces are discretized, and the blue points depict the vertices in G_4.

Proof. This relationship is straightforward from the principle of these models. □

6.5.1 Description of Benchmarks

To analyze the factors affecting the complexity of this problem, we generated 6 workspaces by incorporating randomly positioned rectangular obstacles and forbidden areas, as depicted in Figure 6.13. The bounding polygon for workspace (a) is defined as the rectangle $[0, 25] \times [0, 20]$, and the other 5 workspaces are bounded by $[0, 70] \times [0, 60]$. Another variable in our problem is the location of the anchor point, and by choosing different anchor points in each workspace, we generate a total of 10 scenarios, as described in Table 6.1.

In sections 6.3 and 6.4, we have presented five methods for addressing this problem, comprising one exact approach and four approximate approaches. For each method and each scenario, we fix a limit ℓ_{max} on the maximum cable length value, as defined in Table. 6.1. When increasing the cable length, the problem becomes harder, and we have chosen limits that make it possible to solve the problem, for each model.

For each scenario described in Table. 6.1, we have generated a set of instances of increasing difficulty by increasing the value of the cable length ℓ from 10 to ℓ_{max}, by

S	W	R	N_{free}	$\mathcal{M}_0$			$\mathcal{M}_2$			$\mathcal{M}_1, \mathcal{M}_3, \mathcal{M}_3h$		
				ℓ_{max}	N_{max}	$\frac{N_{max}}{N_{free}}$	ℓ_{max}	N_{max}	$\frac{N_{max}}{N_{free}}$	ℓ_{max}	N_{max}	$\frac{N_{max}}{N_{free}}$
I_0	(a)	A_0	96	50	96	1.0	50	96	1.0	50	96	1.0
I_1	(b)	A_1	848	20	169	0.199	60	728	0.858	100	848	1.0
I_2	(b)	A_2	848	20	157	0.185	60	786	0.927	100	848	1.0
I_3	(c)	A_3	870	20	194	0.223	50	492	0.566	50	492	0.566
I_4	(d)	A_4	880	10	81	0.092	100	880	1.0	1.0	880	1.0
I_5	(d)	A_5	880	10	81	0.092	120	880	1.0	140	880	1.0
I_6	(e)	A_6	820	50	247	0.301	140	705	0.86	140	705	0.86
I_7	(e)	A_7	820	30	257	0.313	110	816	0.995	150	820	1.0
I_8	(f)	A_8	862	20	141	0.164	50	495	0.574	140	856	0.993
I_9	(f)	A_9	862	10	63	0.073	40	593	0.688	100	856	0.993

TABLE 6.1: The scenarios tested in the experimental evaluation. Each line successively gives: the scenario name S, the workspace W, the anchor position R (as defined in Fig. 6.13), the number N_{free} of cells (excluding obstacles and forbidden areas) in W, and the maximum cable length ℓ_{max} of the cable depending on the considered model. For each length, we give the number N_{max} of cells that can be reached with this length, and the coverage ratio $\frac{N_{max}}{N_{free}}$.

steps of 10, resulting in 24 instances for $\mathcal{M}_0$, 78 for $\mathcal{M}_2$, and 107 for $\mathcal{M}_1$, $\mathcal{M}_3$ and $\mathcal{M}_3h$. Specifically, for workspace (c), we assigned a shorter cable length due to the presence of forbidden areas that restrict the reachable region. In this case, a longer cable length would not improve the coverage ratio in any way.

For each method, the overall solving time is composed of ILP model building time and ILP solving time, and we will report them separately. Specifically, the building time includes the time to enumerate all consistent cable configurations, and to construct graphs, like $\hat{G}_{cfg}$ for $\mathcal{M}_1$ and $\hat{G}_q$ for $\mathcal{M}_3h$. The ILP models are solved using the Gurobi Solver [Gur23], and the computation time is limited to 3600 seconds. All experiments reported here are implemented in Python and run on a computer with an Intel Core Intel Xeon E5-2623v3 processor operating at 3.0 GHz with 16 cores and 32GB of RAM.

6.5.2 Computational Performance

We denote ub an upper bound of this problem, which counts all the cells that can be reached by a certain cable configuration under the given cable length, hence it can be larger than the optimal solution. Each model computes a lower bound lb_x (x represents the model) and these lower bounds are not necessarily equal. To evaluate the quality of these bounds, we calculate the bound gap $g_x = \frac{ub - lb_x}{ub}$ for each method, where ub represents the upper bound. The results reveal that

- For $\mathcal{M}_3h$, 39.3% of the instances have a bound gap of 0, indicating optimality, and 74.8% of the instances have a bound gap below 0.05. The maximum observed gap is 0.197.

- For $\mathcal{M}_1$ and $\mathcal{M}_3$, similar conclusions are drawn, with 40.2% and 39.3% of the instances, respectively, having a bound gap of 0, and 73.8% and 72.9% of instances below 0.05.
- For $\mathcal{M}_0$ and $\mathcal{M}_2$, 91.7% and 85.9% of the instances, respectively, yield a bound gap of 0, with all instances falling below 0.05. These results are not surprising given that these methods solve fewer instances.

In addition, when comparing the values of these lower bounds, the conclusion in Theorem 6.5.1 is also supported by the experimental results.

In Figure 6.14, we present a comparison of the solving times for the five models described earlier. Since each model is tested on a different number of instances, and they compute different bounds, the comparison is conducted by pairing the models as follows: $\mathcal{M}_0$ vs $\mathcal{M}_1$, $\mathcal{M}_1$ vs $\mathcal{M}_3h$, $\mathcal{M}_2$ vs $\mathcal{M}_3$, and $\mathcal{M}_3$ vs $\mathcal{M}_3h$. We evaluate them based on the bound gap g_x and the instance hardness, as measured by the cable length. For each pair, these two models are compared on the same subset of instances. However, from a pair to another, the subset of instances may be different.

In each row of Fig. 6.14, we set a different threshold value for g_x. For instance, the first row represents the results when $g_x \leq 0.2$, while the last row is the case where the optimal solution is obtained. Additionally, we segment the cable length into five ranges, each represented by different colors and shapes. Generally, a longer cable length indicates a more challenging instance.

The comparison reveals that $\mathcal{M}_3$ performs comparably to $\mathcal{M}_3h$, with the latter outperforming it on the most challenging instances. This illustrates the effectiveness of the heuristics introduced in $\mathcal{M}_3h$, which accelerates the resolution process. Following $\mathcal{M}_3$ are $\mathcal{M}_1$ and $\mathcal{M}_2$, while $\mathcal{M}_0$ can solve the fewest instances and at a higher cost. These results align with the complexity analysis conducted in Section 6.4.2, where the FPT method proves to be more efficient when $|\mathcal{V}_4| \gg |\mathcal{V}_\mathcal{O}|^\beta$, which often corresponds to instances characterized by large scale but a small number of obstacles. Additionally, the effectiveness of the greedy approach applied to a DAG as opposed to an undirected graph is evident when comparing $\mathcal{M}_0$&$\mathcal{M}_1$ and $\mathcal{M}_2$&$\mathcal{M}_3$.

6.5.3 Scalability Analysis

In Fig. 6.16, we present the evolution of the solving times (the second column) of $\mathcal{M}_1$ and $\mathcal{M}_3h$ with respect to the cable length for each scenario. To better understand the factors impacting solving time, we display the graph size (the number of vertices, as shown in the first column of Fig. 6.16) in each method, in more detail, $|\hat{\mathcal{V}}_{cfg}|$ in $\mathcal{M}_1$ and $|\hat{\mathcal{V}}_q|$ in $\mathcal{M}_3h$. The difficulty of an instance is influenced by factors such as the workspace scale, anchor point position, and the number of obstacles. Generally, larger workspaces require more time to solve, especially when the cable length limit is sufficiently large. We observe that the graph size of both $\hat{\mathcal{V}}_{cfg}$ and $\hat{\mathcal{V}}_q$ increases

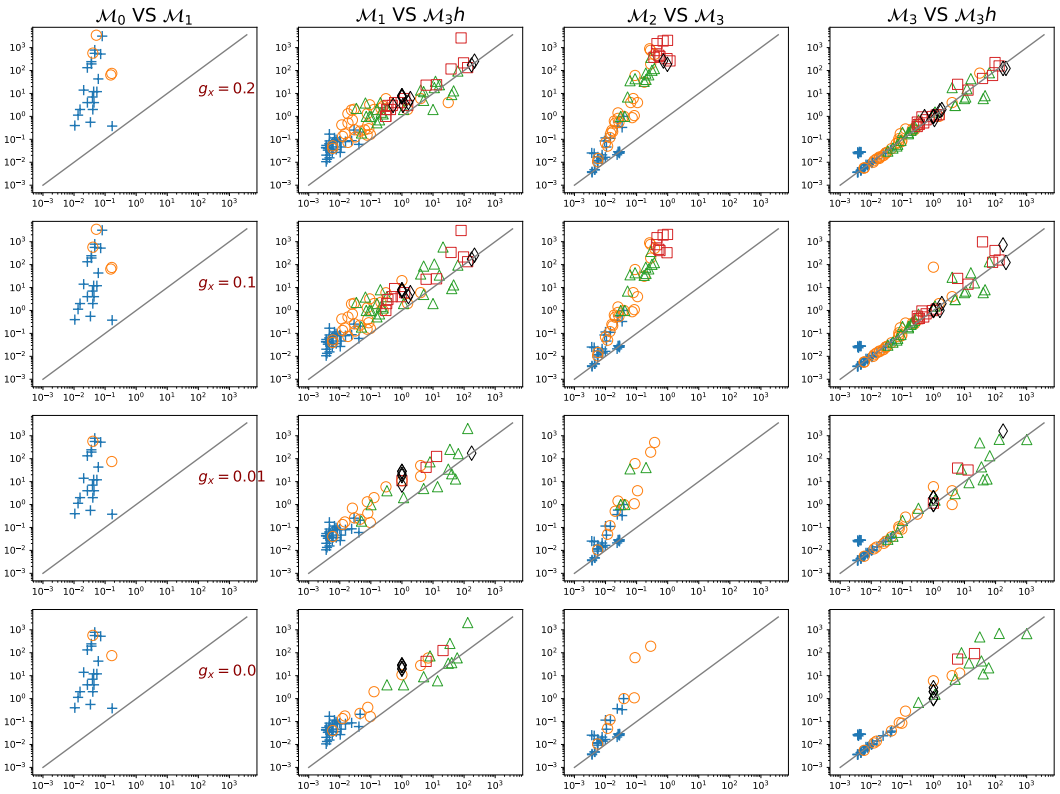
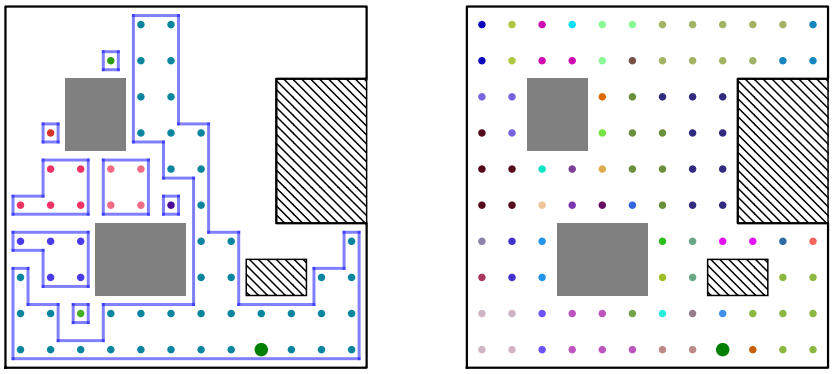


FIGURE 6.14: Comparison of the 5 ILP models' solving times with regard to solution quality and instance hardness. For each scatter plot, we display a point (x, y) for each instance such that x is the time of $\mathcal{M}_0$ (resp. $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$) and y is the time of $\mathcal{M}_1$ (resp. $\mathcal{M}_3h$, $\mathcal{M}_3$, and $\mathcal{M}_3h$) in the first (resp. second, third, and fourth) column. We consider the time needed to find a gap smaller than or equal to a threshold value g_x such that $g_x = 0.2$ (resp. 0.1, 0.01, and 0) on the first (resp. second, third and fourth) row. Colours and shapes are different depending on the cable length: blue $+$ when $10 \leq l \leq 30$, orange $\bigcirc$ when $40 \leq l \leq 60$, green $\triangle$ when $70 \leq l \leq 90$, red $\square$ when $100 \leq l \leq 120$ and black $\Diamond$ when $130 \leq l \leq 150$.



(a) (b)

FIGURE 6.15: Examples of workspace partitions resulting from the FPT methods ($\mathcal{M}_3$ and $\mathcal{M}_3h$). (a): the cable length is set to 20. (b): the cable length is set to 50.

exponentially with cable length and number of obstacles. This effect is evident when comparing instances I_2 and I_3.

It is observed that the performance of $\mathcal{M}_3h$ is degraded as the cable length increases. Consider the example of instance I_9, when the cable length has reached 140, the difference between the graph size of $\hat{G}_{cfg}$ and $\hat{G}_q$ is narrowed, and so is the solving time. In fact, as the number of homotopy classes expands with the cable length, the weight of each vertex of $\hat{G}_q$, which is equivalent to the size of the corresponding partition of $\hat{G}_{cfg}$ decreases. In other words, the partitions of $\hat{G}_{cfg}$ become finer and $\hat{G}_q$ is closer to $\hat{G}_{cfg}$. An example is illustrated in Fig. 6.15. When the cable length is set to 20, the accessible area can be divided into 8 connected components (where points of the same color form an component, and delimited by a blue circle), the largest of which has a size of 43. When the cable length is increased to 50, the workspace can be reached anywhere, resulting in 42 components, and the maximum size of these components is 9. This demonstrates how the partitions become more refined as the cable length increases.

Even though it is the presence of forbidden zones that renders the problem $\mathcal{NP}$-complete, the problem's complexity depends on the number of obstacles rather than the number of forbidden zones. With more obstacles, a longer cable length allows to explore more homotopy classes, and thus the graph size increases and so does the solving time. Another factor that could have an impact on the hardness of problem is the anchor position, as we can compare the result of I_6 with I_7, and I_8 with I_9. If the obstacles are intensely distributed and the anchor point is just near the center of these obstacles, then this creates many more homotopy classes.

In addition to the ILP solving times, in the third column of Fig. 6.16, we also display the time it takes to build these ILP models. Generally, this model building time is relatively smaller compared to the ILP solving time. It increases with the cable length ℓ when ℓ is small, and then it eventually reaches a saturation phase as

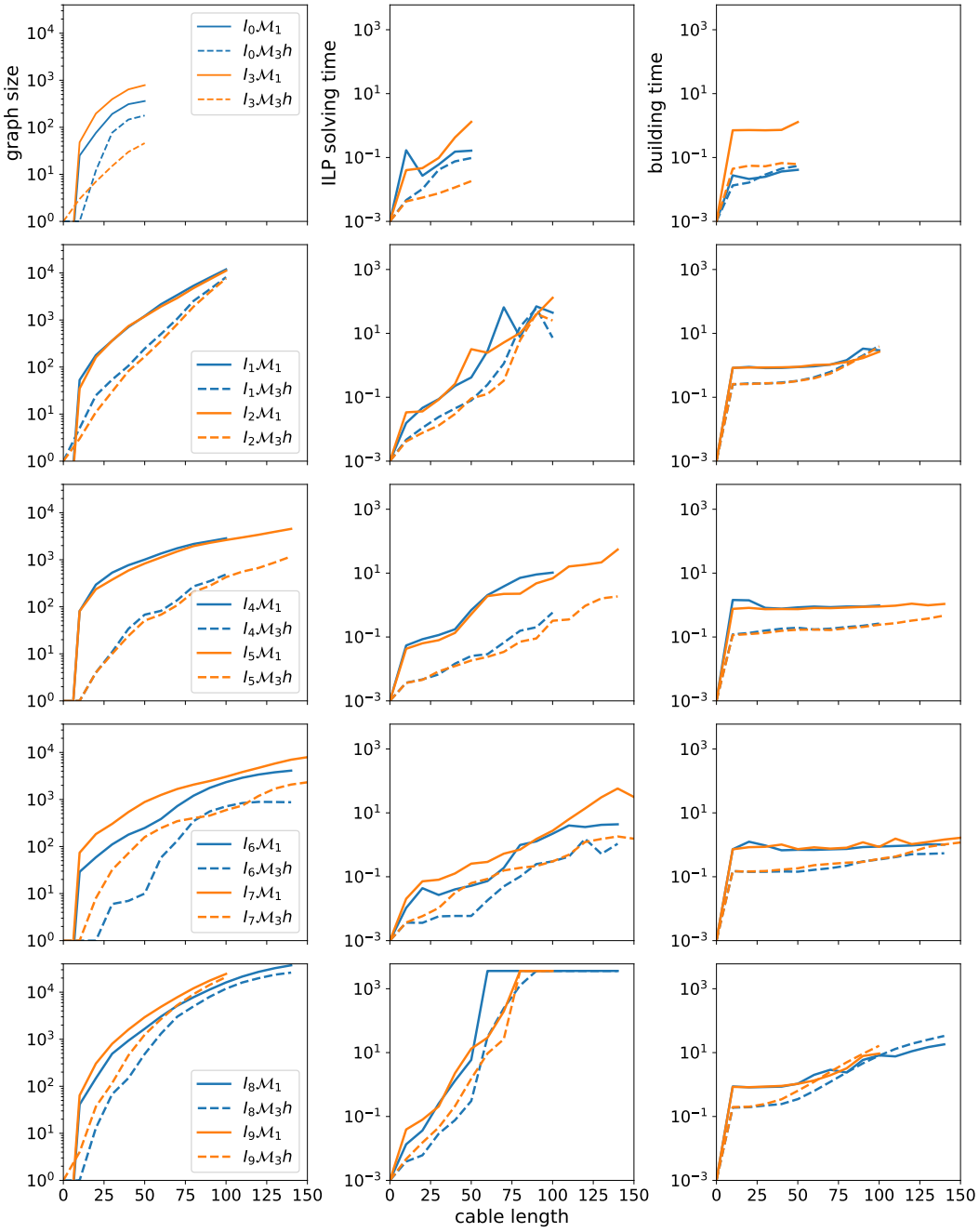


FIGURE 6.16: Evolution of graph sizes and solving times (y-axis with a log scale) with respect to the cable length (x-axis). In each figure, $I_x\mathcal{M}_y$ means the instance I_x solved by model $\mathcal{M}_y$. First column: the number of vertices in G_{cfg} (model $\mathcal{M}_1$) and G_q (model $\mathcal{M}_3h$). Second column: resolution times of the ILP models. Third column: building times of ILP models.

a longer cable cannot generate more consistent cable configurations due to the non-crossing constraint. However, in the presence of more obstacles in the workspace, this model building time increases exponentially, as can be observed in the results for instances I_8 and I_9 (see the last row in Fig. 6.16. When ℓ is small, it takes less time to construct $\hat{G}_q$ than $\hat{G}_{cfg}$ because $|\hat{\mathcal{V}}_{cfg}| > |\hat{\mathcal{V}}_q|$. As ℓ becomes larger, $|\hat{\mathcal{V}}_q|$ gets closer to $|\hat{\mathcal{V}}_{cfg}|$, and connecting two vertices in $\hat{G}_q$ is more computationally expensive than connecting two simple vertices in $\hat{G}_{cfg}$. This is because we have to find the boundary cells and check their connectivity in $\hat{G}_q$. It explains why the time for $I_8\mathcal{M}_3h$ exceeds that for $I_8\mathcal{M}_1$ when ℓ increases, and the same reason applies to instance I_9.

In this study, we only provide a qualitative analysis of these results, as we believe that the hardness of an instance is the result of several topological factors combined, and that it is therefore difficult to discuss each factor separately. Our aim is to show in which cases a **TCPP** instance can be hard, and how to calculate a good approximate solution for it.

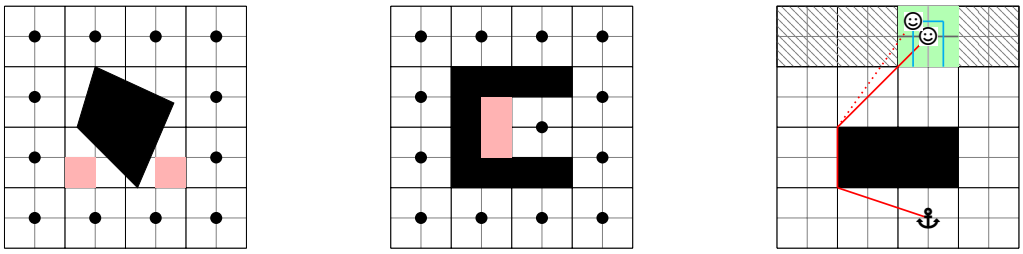
6.6 Discussion

In this work, we primarily focus on the rectangular obstacles, for the sake of simplifying the computation of cable configurations. However, our methodology can be also extended to obstacles of arbitrary shape. There are several cases that should be taken into account.

- **Non-Rectangular Obstacles:** In cases where obstacles have non-rectangular shapes, as depicted in Fig. 6.17(a), cells which are (partially) occupied are ignored, and those near the obstacles (shown in red) cannot be included in G_4 directly, but can still be connected to G_4, and can be covered with an additional cost. This approach, while not optimal, allows coverage of these challenging regions, as discussed in Section 2.2.4.

- **Non-Convex Obstacles:** Fig. 6.17(b) illustrates a scenario with a non-convex obstacle. It is always possible to explore the workspace strategically in order to construct a G_4 as large as possible. The left cells (shown in red) can either be ignored or covered with an additional cost.

- **Discretization Issues:** In addition to the shape of obstacles, there is another issue related to the discretization, as depicted in Fig. 6.17(c). The center of the green cells group, corresponding to a vertex of G_4, can be reached from the anchor point. However, if the robot moves along the coverage path, it may happen that the cable length exceeds its limit or that the cable goes out of the workspace. This typically occurs with cells located at the boundary of forbidden zones or the outer boundary of the workspace. To simplify our planning, we overlook these situations and assume that these cells are reachable in our planning process.



(a) (b) (c)

FIGURE 6.17: Discretization issues related to the shape of obstacles and forbidden zones. In each scenario, the vertices of G_4 are represented by bold black dots.

6.7 Conclusion

In this work, we revised the CPP problem for tethered robots, considering the specific constraints imposed by the cable, such as its limited length and the requirement to prevent entanglement. We investigated the feasibility of applying the widely used STC method, which has proven successful in solving the classical CPP problem. Our findings demonstrate that the STC method can be easily adapted to meet these two constraints.

When searching for a spanning tree within the STC method, the BFS algorithm does not consistently guarantee compliance with the cable length limit. To address this, we propose to apply the Dijkstra's algorithm instead, which can effectively resolve the problem within polynomial time. Based on this result, we introduce a new constraint that considers forbidden areas within the workspace where the cable is prohibited from passing through. The presence of forbidden areas significantly alters the complexity of the problem since the workspace is no longer convex. Notably, we demonstrate that TCPP problem incorporating forbidden areas is $\mathcal{NP}$-complete.

The challenge posed by these forbidden areas is that certain cells can be only reached via specific homotopy classes. To overcome this challenge, we employ the concept of the homotopy augmented graph, which represents all possible cable configurations within a graph structure G_{cfg}. We select the most favorable homotopy classes to form the solution. Based on this approach, we explore the different heuristics to reduce the search space, such as transforming G_{cfg} into a directed graph. An original contribution in this work is that we propose a heuristic by defining a quotient graph of G_{cfg} that groups the neighboring cells which can be reached by the same set of homotopy classes as a new vertex. A theoretical analysis shows this new heuristic can reduce the search space from $2^{|\mathcal{O}||\mathcal{V}_4|}$ to $2^{|\mathcal{O}||\mathcal{V}_\mathcal{O}|^\beta}$, namely, it depends on the number of the obstacle vertices, not the number of cells in the workspace. Thus this method is referred as a FPT algorithm where the parameter is the number of obstacle vertices. Based on the different heuristics, we present five ILP models to address the problem, comprising one exact approach and four approximate approaches. Our results indicate that the approximate approaches are generally more efficient than the exact approach, and the gap between the obtained lower bound

and upper bound is limited to 20%.

The hardness of a TCPP instance in case of forbidden areas is impacted by the workspace scale, the cable length and the number of obstacles ($|\mathcal{O}|$). Notably, for the factor $|\mathcal{O}|$, the more obstacles there are in the workspace, the more homotopy classes there are to explore, so the larger the search space. Our tests on simulated instances show that when $|\mathcal{O}|$ is not very large, the $\mathcal{M}_3h$ model, based on the FPT method, with an additional heuristic on the choice of cable configuration, is the most efficient. When $|\mathcal{O}|$ gets larger, as $|\mathcal{V}_\mathcal{O}|$ increases with $|\mathcal{O}|$, its performance degenerates and approaches that of a non-FPT method. In future research, we intend to investigate alternative parameterized algorithms to address this challenge. Furthermore, while the surface of forbidden areas does not significantly affect the problem's hardness, it does impose limitations on the maximum coverage areas. In Chapter 7, we aim to explore the potential utilization of multiple tethered robots for the coverage task.

Chapter 7

Multiple Tethered Robots Coverage Path Planning

Contents

7.1 Problem Statement 104
7.2 Methodologies 105
7.2.1 Ham-Sandwich Based Polygon Division 106
7.2.2 DARP 107
7.2.3 Voronoi-Based Partition 108
7.3 Simulation Results 111
7.4 Equitable division in nonconvex polygon 116
7.5 Conclusion 116
7.6 Open Problems 117
7.6.1 Diameter Constraint 117
7.6.2 Reachability Constraint 118
7.6.3 Anchor points placement 119

In Chapter 6, we have presented the **C**overage **P**ath **P**lanning (CPP) problem for a single tethered robot. This chapter aims to extend the scope to a system of **M**ultiple **T**ethered robots and propose solutions for the MTCPP problem, as the primary objective of this project. Extensive literature exists on the multi-robot coverage path planning problem (MCPP). Typically, the general approach involves solving the problem in two steps [MO07]: firstly, dividing the workspace into partitions and allocating them optimally to the robots considering relevant constraints such as capacity and initial location; secondly, allowing each robot to operate independently within its assigned area. The advantage of this method is the avoidance of collision risks and no need for communications among robots, which is considered as a key challenge in the application of multiple robots systems. We aim to adapt this strategy for coverage tasks with tethered robots, specifically employing the STC method and leveraging the findings from Chapter 6 for the second step. This chapter focuses primarily on workspace partitioning and allocation methods for tethered robots.

To the best of our knowledge, this work is the first to consider the multiple tethered robot coverage problem. Compared with robots without cables, the constraints

associated with cables pose new challenges for the planning. Even if each robot operates independently within its assigned area, cable entanglement is possible, especially in nonconvex subregions. Additionally, the anchor point of each robot is fixed and should belong to its designated subregion, which is not always the case in some optimal partitioning approaches [Pav+11; Pal+19].

In this chapter, we compare existing polygon partitioning methods and conduct a qualitative comparison among them, primarily in terms of solution quality and computational efficiency. Our strategy involves applying these methods in our context to meet the geometric constraints for tethered robots. In Section 7.1, we formally define the new partitioning problem for tethered robots, introducing shape constraints for each subregion. Section 7.2 provides an overview of existing polygon partitioning approaches for robots without cables, demonstrating that the *additively weighted Voronoi diagram* generally exhibits superior performance, though it is not yet widely adopted in the robotics community. These partitions also satisfy shape constraints, effectively solving the problem for tethered robots. Section 7.3 includes simulated instances for qualitative comparison. This work is still ongoing and further quantitative studies are expected in future work.

7.1 Problem Statement

We consider a workspace $\mathcal{W}$, defined by a finite region $\mathcal{B}$ consisting of a set of obstacles $\mathcal{O} = \cup_j \mathcal{O}_j$, where $\mathcal{W} = \mathcal{B} \setminus \mathcal{O}$. Let $X = \{x_1, ..., x_n\}$ be a set of anchor points in $\mathcal{W}$. The objective is to find a subdivision of $\mathcal{W}$ into a set of equal-area subregions $\{P_1, ..., P_n\}$, where $\mathcal{W} = \bigcup_{1 \leq i \leq n} P_i$, $P_i \cap P_j = \emptyset$ for any two parts P_i and P_j, and such that $x_i \in P_i$ for each point $x_i \in X$. This problem is also referred to as *locus-based equal-area division problem*[AP10].

Most workspace partitioning approaches are designed for robots without cables. In that case, the collision avoidance is already ensured, and thus the geometric properties of the partitioning are not an issue. While for tethered robots, an independently assigned area is not sufficient to avoid the cable entanglements. The shape of each subregion can also be crucial, specifically, a convex partition (where each subregion P_i is convex) can ensure that the cables never cross, as the result of Voronoi diagram shown in Fig. 7.1(a)(we will discuss it later in Section 7.2.3). However, the convexity is a strong property which can be hard to achieve and is not necessary in some cases, as shown in Fig. 7.1(b): a star-shaped nonconvex polygon can also ensure that the cable never gets out of its area. Furthermore, other constraints associated with the characteristics of the workspace, like the obstacles, can also affect the properties of a subregion (see in Fig. 7.1(c)). We use the notions of *visibility* and *reachability* to generalize these properties.

Definition 7.1.1 (visibility)**.** Any two points $u, v \in \mathcal{W}$ are said to be visible if there is a line segment that joins u and v and is entirely contained in $\mathcal{W}$. For a point $x \in \mathcal{W}$,

the visible polygon $\mathcal{V}(x)$ is a collection of all the points in $\mathcal{W}$ that can be visible from x. We also say that $\mathcal{V}(x)$ is *relatively star-convex* to x.

In a more general case, with the presence of obstacles in the workspace, the notion of *reachability* is defined, and it can be used as a metric to evaluate the quality of partition. The *reachability* is also called *relatively star-convexity* [CCD13].

Definition 7.1.2 (Reachability)**.** A point $q \in \mathcal{W}$ is said to be reachable from p if there exists a taut path from p to q that is contained in $\mathcal{W}$. The set of all points $q \in P$ reachable from p is the reachability set $R(p)$ with respect to p.

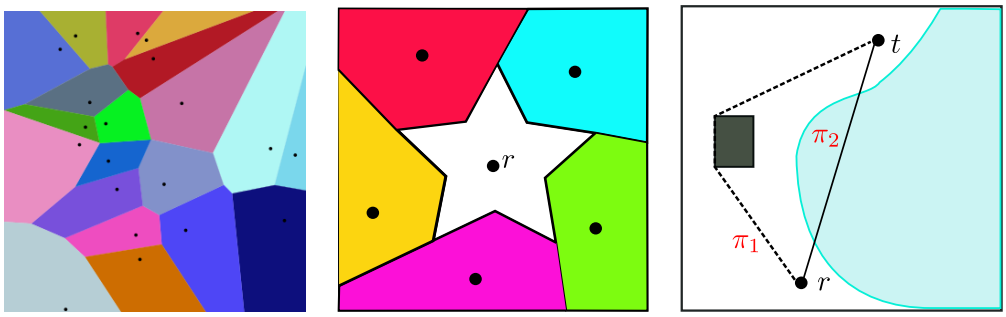


FIGURE 7.1: Different cases of reachability. Each figure represents an example of partitioning with the different subregions distinguished by color. (a): the Voronoi diagram[1]. (b): the central subregion is of star-convex shape with respect to r. (c): the subregion on the left is nonconvex. t can be reached from r by the path π_1 which bends around the obstacle and does not pass through the blue area.

In the context of addressing the partitioning problem for tethered robots, our objective is to find an equitable subdivision such that each subregion is relatively star-convex to its anchor point.

7.2 Methodologies

Existing works on the area partitioning problem commonly incorporate three constraints:

- *Locus-based:* each subregion contains one predefined point.
- *Equal-area:* each subregion has an equal surface area.
- *Convexity:* each subregion is convex.

Finding an optimal solution that simultaneously satisfies all three constraints is not trivial. This section provides an overview of current research on the locus-based equal-area division problem and explores the feasibility of satisfying the convexity

[1]This picture is cited from https://en.wikipedia.org/wiki/Voronoi_diagram

constraint. Additionally, in the presence of the locus-based constraint, we relax the convexity requirement to be relatively star-convex in relation to a point.

Existing works on the locus-based equal-area division problem fall into two primary categories: geometry-based division and optimization-based division. The main methods investigated are *ham-sandwich-based polygon division*, *DARP*, and *Voronoi-based partition*. We assume that the bounding region of the workspace, i.e., $\mathcal{B}$, is convex, and the nonconvex case will be discussed in Section 7.4.

We provide a concise explanation of each method and present a general comparison between them.

7.2.1 Ham-Sandwich Based Polygon Division

First, we introduce a geometric method. In this method, we assume that $\mathcal{B}$ constains no obstacles and $\mathcal{W}$ is a convex polygon. Literally, if we consider a sandwich consisting of one slice of ham and two slices of bread, arbitrarily placed, there always exists an angle under which the three slices can be cut into two equal halves, all at once. Mathematically, the generalized ham sandwich theorem deals with the existence of a $(n-1)$-hyperplane that simultaneously cuts into two n-subsets in $\mathbb{R}^n$ with finite Lebesgue measure [ST42]. This concept is widely applied to polygon division problems [AP10; CAY10]. Since the ham-sandwich theorem is intended for a 2-partition scenario, extending it to the case of n-partition is straightforward when n is a power of 2, by recursively applying this theorem. For a general integer value of n, the conclusion is completed by considering the case of 3-partition, and the existence of a 3-cutting is proven in [BKS00].

Theorem 7.2.1 (Equitable 3-cutting [BKS00])**.** Let n_1,n_2, and n_3 be three positive integers with $n_1 + n_2 + n_3 = n$. Let g and h be two positive integers. For any $n * g$ red points and $n * h$ blue points in the plane in general position, there exists a partition of the plane into 3 convex subregions such that the i-th subregion contains $n_i * g$ red points and $n_i * h$ blue points.

This implies that if n is odd, there exists an equitable 3-partition $(1, \lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{2} \rfloor)$. Then the following conclusions are forthright for the existence of an n-partition.

Theorem 7.2.2 (Equitable n-subdivision [BKS00])**.** Given ng red points and nh blue points in the plane in general position, there exists a subdivision of the plane into n convex polygonal regions each of which contains g red and h blue points, and it can be computed in $\mathcal{O}(N^{4/3} \log^3 N \log n)$ time where $N = n(g+h)$.

Theorem 7.2.3 (Equitable n-subdivision, continuous version [BKS00])**.** Let μ_1 and μ_2 be measurable functions $\mathbb{R}^2 \to [0, \infty)$ with $\iint_{\mathbb{R}^2} \mu_i dA = 1$. For any integer $n > 0$, there exists a subdivision of the plane into n convex regions $P_1, ..., P_n$ such that

$$\iint_{P_j} \mu_i dA = \frac{1}{n} \quad \text{for } i = 1, 2 \quad \text{and} \quad j \in [1, n]$$

In our problem, we focus on the case on a 2-dimension plane, where $\mathcal{W}$ is a convex polygon with m vertices, and X is set of n anchor points in general position. We define μ_1 is an atomic measure and μ_2 a density function. An equitable convex partition of $\mathcal{W}$ and X can be found in $\mathcal{O}(nN \log N)$ time where $N = m + n$, by recursively searching the 2-partition and 3-partition cuts [CAY10].

7.2.2 DARP

In [KCK17], a discrete algorithm, named DARP, is proposed to optimally divide a surface into n robot-exclusive regions. The surface has no specific requirements on its shape and might contain obstacles. In this algorithm, the surface is discretized into a finite set of equal grid cells, and each grid cell is assigned to the robots according to metrics like the distance function and the robots' reachability level. The principle of DARP is to optimize the grid cell assignment problem such that the target subregions have quasi-equal size.

Some notations in DARP are defined as follows. Let L be the grid map to be divided (for sake of simplicity, we assume $\mathcal{B}$ is a rectangle in this case), which is assumed to be a matrix of size $rows \times cols$, where $L_{x,y} = 0$ if the cell (x, y) is occupied by an obstacle, otherwise $L_{x,y} = 1$ for the free cells. For each robot r_i, an evaluation matrix E_i of the same size as L is maintained, which characterizes the reachability from the initial position of r_i to the other cells and it is initialized by their distance. An assignment matrix A is defined as $A_{x,y} = \underset{i \in \{1,\dots,n\}}{\arg\min} E_i(x, y), \forall (x, y) \in L$, that assigns a grid cell to the robot that reaches it with the minimum cost. The area assigned to the i^{th} robot is $L_i = \{(x, y) \in L : A(x, y) = i\}$ and its size is noted as $k_i = |L_i|$. There are two constraints to be considered in this problem.

Equitable division

In order to compute an equitable division, they evaluate the fairness by

$$J = \frac{1}{2} \sum_{i=1}^{n} (k_i - k_f)^2 \tag{7.1}$$

where $k_f := \frac{|L|}{n}$ denotes the fair assignment. The core idea of this algorithm is that we introduce a vector scalar m of length n to "correct" each E_i as $E_i = m_i E_i$. m_i can be seen as a correction factor in the objective function. A cyclic coordinate descent method is applied to optimize J. Each robot's contribution $J_i = \frac{1}{2}(k_i - k_f)^2$ is proven to be a convex function to m_i. At each iteration, the update rules proceeds as

$$\begin{aligned} m_i &\leftarrow m_i - \eta \frac{\partial J_i}{\partial m_i} \\ &\leftarrow m_i - \eta (k_i - k_f) \frac{\partial k_i}{\partial m_i} \end{aligned} \tag{7.2}$$

The values $\frac{\partial k_i}{\partial m_i}$ are negative and nearly identical, hence the update policy can be approximated as follows:

$$m_i \leftarrow m_i + c(k_i - k_f) \tag{7.3}$$

with c a positive tunable parameter. In [KCK17], the author demonstrate that this process ultimately converge to an optimal value $J(m^*)$ and where all k_i are nearly equal.

Spatial connected areas

The other constraint to be satisfied is that each area L_i must be connected. For those robots who occupy more than one distinct connected areas, a corrective multiplier is defined as

$$C_{i|x,y} = \min_{r \in \mathcal{R}_i}(||(x,y) - r||) - \min_{q \in \mathcal{Q}_i}(||(x,y) - q||) \tag{7.4}$$

where $\mathcal{R}_i$ denotes the connected area assigned to r_i and where r_i's initial position lies, and $\mathcal{Q}_i$ denotes the union of the other grid cells assigned to r_i. It works as a penalization term to reward the cells around the R_i and penalize the cells around the other connected components. In this way, if all grid cells assigned to r_i belong to the same closed shape region, then C_i will be the all-one-matrix. Finally, the update of E_i is computed as

$$E_i \leftarrow C_i \odot (m_i E_i) \tag{7.5}$$

where $\odot$ denotes the element-wise product.

7.2.3 Voronoi-Based Partition

The partitioning problem can be also approached by means of computing the Voronoi diagram. Given a planar region $\mathcal{W}$ and a set of points $X = \{x_1, ..., x_n\} \subset \mathcal{W}$, the Voronoi diagram is a partition of $\mathcal{W}$ defined as

$$P_i = \{x \in \mathcal{W} : ||x - x_i|| \leq ||x - x_j||, \forall j\} \tag{7.6}$$

An additively weighted Voronoi diagram is a generalization of (7.6) by introducing a vector of positive weights w, and the properties of each partition P_i might be changed due to w, as a larger weight w_i indicates a larger area assigned to x_i.

$$P_i = \{x \in \mathcal{W} : ||x - x_i|| - w_i \leq ||x - x_j|| - w_j, \forall j\} \tag{7.7}$$

Besides, according to the distance function chosen (most cases we use euclidean distance), different variants of partitions can be defined. For example, the diagram defined in (7.8) is called **power diagram**.

$$P_i = \{x \in \mathcal{W} : ||x - x_i||^2 - w_i \leq ||x - x_j||^2 - w_j, \forall j\} \tag{7.8}$$

The locus-based equitable partition problem can be viewed as a kind of assignment problem. Let $d(x, x_i)$ be the distance between a point x and x_i, $f(x)$ be a density function, and $c = (c_1, ..., c_n)^T$ be a predefined quotas vector parameter, interpreted as the target "capacity" or the surface of each assigned area, then it can be formulated as a special case of the *Monge-kantorovich transportation problem* [Amb+03].

$$\begin{aligned} \underset{I_1(\cdot),...,I_n(\cdot)}{\text{minimize}} \quad & \sum_{i=1}^{n} \iint_{\mathcal{W}} f(x) d(x, x_i) I_i(x) dA \\ \text{subject to} \quad & \iint_{\mathcal{W}} f(x) I_i(x) dA = c_i, \ \forall i \\ & \sum_{i=1}^{k} c_i = 1, \ c_i > 0 \\ & \sum_{i=1}^{k} I_i(x) = 1, \ \forall x \in \mathcal{W} \\ & I_i(x) \in \{0, 1\}, \ \forall i, x \end{aligned} \tag{7.9}$$

where $I_i(.)$'s are indicator functions, and its support can be relaxed to $[0, +\infty)$. The objective function to minimize is regarded as "workload" in the transportation problem. This optimization problem is convex and can be approached by Lagrangian multiplier method, where the solution is in the following form according to the Theorem 2 in [JER12].

$$I_i^*(x) = \begin{cases} 0 & \text{if } d(x, x_i) - \lambda_i^* > d(x, x_j) - \lambda_j^* \text{ for some } j \\ 1 & \text{if } d(x, x_i) - \lambda_i^* < d(x, x_j) - \lambda_j^* \text{ for all } j \neq i \end{cases} \tag{7.10}$$

where λ^* is the dual vector. The result can be interpreted as assigning a point x to x_i with index i where $d(x, x_i) - \lambda_i^*$ has the minimum value. It can be observed that the diagram generating models defined in (7.6)-(7.8) are special cases of (7.10). To compute an equitable division, there are several points to be considered.

- $f(x)$ and c: $f(x)$ is a continuous density function that is set to a constant value 1, and $c_i = \frac{1}{n}, \forall i \in [1, n]$ in our case.
- $d(x, x_i)$: The distance function takes the shortest path between x and x_i in $\mathcal{W}$, based on the euclidean distance, and we adopt the L1 norm. When there are no obstacles, d is computed as $d(x, x_i) = ||x - x_i||$, consequently, in such scenarios, the boundaries separating different areas are characterized by hyperbolic arcs. The difference between the norms of order 2 and order 1 is depicted in Figure. 7.2(a) and (b). In case of L2 norm, the boundary is a line orthogonal to $\overline{x_i x_j}$ and intersects it at point p, charaterized by $||x - x_i||^2 - ||x - x_j||^2 = w_i - w_j = ||x_i - p||^2 - ||x_j - p||^2$. The position of the boundary line depends on the value of $w_i - w_j$, and it might be at the location of the dotted line (which

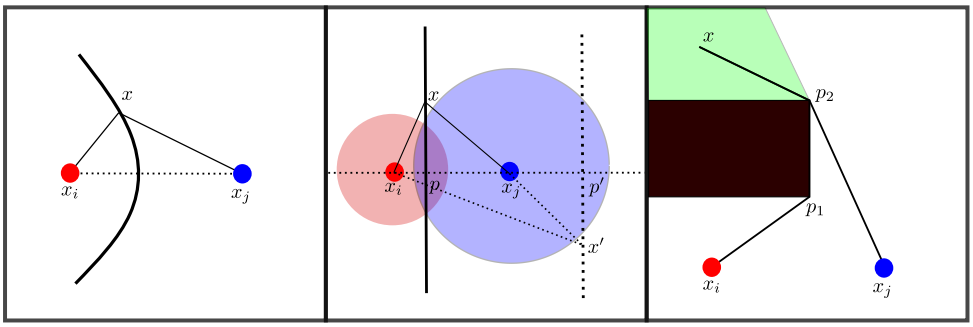


FIGURE 7.2: Examples of different $d(x, x_i)$. (a): d is defined as the distance of L1 norm. The boundary is generated by the equality case in (7.7). (b): the case corresponding to L2 norm(defined in (7.8)). (c): the case of L1 with the presence of obstacles.

intersects $\overline{x_i x_j}$ at p'). In this way, x_i and x_j are all on the same side of the boundary, and the constraint that each initial point is contained in its subregion cannot be satisfied.

- the obstacles in the workspace: we assume that all obstacles are polygonal. This simplification facilitates the computation of the shortest path. Let $\langle x_i, p_1, p_2, ..., p_k, x\rangle$ be the shortest path between x and x_i, then $d(x, x_i) = ||x_i - p_1|| + \sum_{j=2}^{k} ||p_j - p_{j-1}|| + ||x - p_k||$ where $p_1, p_2.., p_k$ are all the obstacle vertices. In this case, equation (7.10) may not be sufficient to reach a complete partitioning of the workspace, because there are certain regions, denoted as $R' \subset \mathcal{W}$ where equality holds in equation (7.10). In other words, there exist indices i and j and an area $R' \subset \mathcal{W}$ such that for every point $x \in R'$, we have $d(x, x_i) - \lambda_i^* = d(x, x_j) - \lambda_j^* = \min_{k\in[1,n]} d(x, x_k) - \lambda_k^*$, as illustrated in Fig. 7.2(c) where R' is marked in green. These R' regions must start at a *reflex vertex* on the obstacles (such as the point p_2), and they can be further subdivided into a set of interconnected and *relatively star-convex* subregions. The topological connections among these subregions can be effectively represented using a tree structure. By applying a DFS, these subregions can be reassigned to each x_i in order to achieve an equitable final partitioning, as explained in Fig.4 of [JER12].

Enforcing shape constraints

Since the objective defined in (7.7) aims to minimize the overall "workload", the subregions computed through this method must be connected. The definition of $d(x_i, x)$ specifically considers the shortest path connecting x and x_i, the *relatively star-convexity* is guaranteed by the same reason that an optimal (bipartite) Euclidean matching has no crossing-edges. A more detailed explanation refers to section 2.2 and 2.3 in [CD13].

Resolution of the problem (7.9)

As mentioned earlier, this constrained optimization problem can be approached by

Lagrange multiplier method. In general, the value of those λ_i^* can be obtained by Newton's method [KMT19]. However it should be pointed out Newton's method does not work if d is defined as unsquared Euclidean distance [HS20]. In case of L1 norm, it can be efficiently solved by using an *analytic-center cutting plane method* (ACCPM) [JER12], or an alternative method is to use a decentralized supergradient agent-based control scheme to iteratively update λ_i by

$$\lambda_i^{j+1} \leftarrow \lambda_i^j + \epsilon(\iint_{R_i^j} f(x)dA - \frac{1}{n}) \tag{7.11}$$

where λ_i^j is the value of component λ_i at iteration j, R_i^j denotes the region assigned to x_i and ϵ is the step coefficient. As can be observed that (7.11) is quite similar to (7.3), we will compare this method with DARP in the next section.

Another possibility is to solve problem in a discrete scheme: $\mathcal{W}$ is divided into m equal-sized grid cells, and we reformulate (7.9) as a classic linear assignment problem:

$$\begin{aligned} \underset{I \in \mathbb{R}_+^{m \times n}}{\text{minimize}} \quad & \sum_{i,j} I_{i,j} D_{i,j} \\ \text{subject to} \quad & I\mathbf{1} = \mathbf{a} \\ & I^T\mathbf{1} = \mathbf{c} \\ & I \geq 0 \end{aligned} \tag{7.12}$$

where $D \in \mathbb{R}_+^{m \times n}$ is the metric cost matrix defining the shortest Euclidean distance between each grid cell and each x_i, $\mathbf{a}$ is an histogram that represents the weight of each grid cell (analogue to the density function f) and $\mathbf{c} = \frac{\mathbf{1}}{\mathbf{n}}$. This linear programming has a polynomial complexity in terms of m.

7.3 Simulation Results

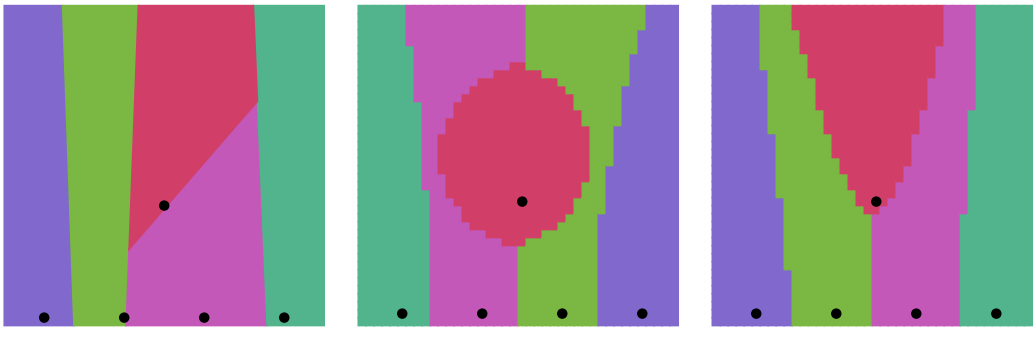
We report the simulation results and compare these three methods described above. We carried out tests on 5 scenarios, taking into account factors, such as the presence of obstacles, the location of anchor points, the density function imposed on the workspace and also compare the results with the work [Pal+19]. In our simulation, the workspace is a bounding rectangle and represented by a matrix of pixels.

- Scenario 1 (Fig. 7.3): a convex workspace of 40×40 without obstacles, and 5 anchor points (marked with black points);
- Scenario 2 (Fig. 7.4): a convex workspace of 100×100 without obstacles, and 10 anchor points;
- Scenario 3 (Fig. 7.5): a nonconvex workspace of 40×40 with obstacles (in gray), and 7 anchor points;

- Scenario 4 (Fig. 7.6): three nonconvex workspaces of 160×200 with obstacles, introduced in [Pal+19] [2];
- Scenario 5 (Fig. 7.7): a non-uniform density function is applied on the workspaces, as $f(x,y) = 0.9995(80 + 0.1y)(0.5 + 0.0025y)$. A visualization of the normalized $f(x,y)$ is shown in Figure 3 of [Pal+19].

For simplicity's sake, we refer to the three partition methods as follows: the Ham-Sandwich Theorem based method is denoted as HST, the Voronoi diagram based method as Voronoi, and the DARP method. HST is a continuous method, while DARP and Voronoi compute in a discrete way by assigning each grid cell to an anchor point. In particular, the solution of Voronoi is computed by solving (7.12) using the Python package POT [Fla+21]. DARP and Voronoi are implemented in Python and executed on an Intel Core Intel Xeon E5-2623v3 of 3.0GHz×16 with 32GB of RAM, while HST is run on Matlab Online. This work is still ongoing and we provide just some qualitative comparisons for these three methods.

Firstly we present two examples illustrating different partitioning algorithms in scenarios where there are no obstacles within the workspace. As depicted in Figure 7.3, the equitable partitions produced by the HST method result in a collection of convex polygons. In the Voronoi method, the boundaries exhibit a hyperbolic-like shape, and the resulting subregions are relatively star-convex to their corresponding anchor points. However, the property of relative-convexity is not guaranteed in the case of DARP. As observed, certain areas within the pink and green subregions are not visible from their anchor points due to the presence of the red subregion.



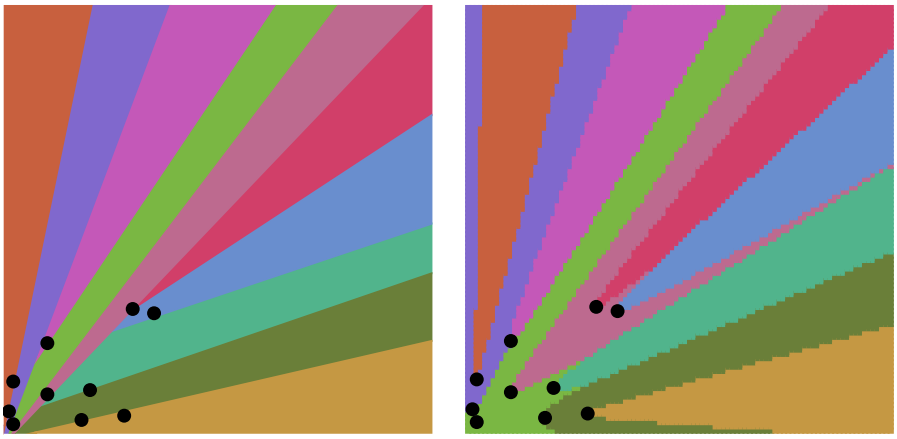
(a) (b) (c)

FIGURE 7.3: Scenario 1 - Comparison between the three partitioning methods in case without obstacles in the workspace. (a): HST; (b): DARP; (c): Voronoi.

For Scenario 2, as shown in Fig. 7.4, we display only the result of HST and Voronoi, which can be computed promptly, while the DARP method fails to find a solution within 20000 iterations and 640 seconds. The convergence speed of DARP heavily depends on the distribution of anchor points, especially when they are densely

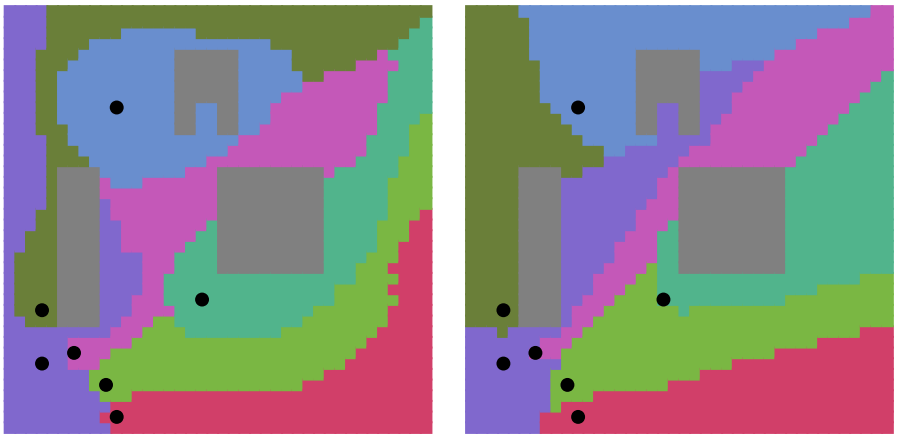
[2] There are four environments presented in their article, and the data for the environment "Maze" is not available, hence is not reported here.

distributed. It becomes challenging for DARP to simultaneously satisfy the connectivity constraint and equitable division under these circumstances. DARP operates on the principle of iteratively assigning cells based on gradient values and control variables. Notably, connectivity is not explicitly penalized in its objective function, which solely considers the discrepancy in the size of each subregion. Consequently, DARP can be sensitive to specific cells and may become trapped in local minima, particularly when anchor points are densely concentrated, as observed in this case with anchor points at the lower-left corner.



(a) (b)

FIGURE 7.4: Scenario 2 - The case where the anchor points are densely located. (a): HST; (b): Voronoi.



(a) (b)

FIGURE 7.5: Scenario 3 - Example of partitioning in the presence of obstacles. (a):DARP; (b):Voronoi.

In Scenario 3, we consider the case where obstacles are present. Given that HST cannot deal with the cases with obstacles, we display only the results of DARP and Voronoi in Fig. 7.5. In the result of Voronoi, each subregion can be fully reached from its anchor point. However the solution of DARP cannot ensure that, as in some subregions, the cable should go through the partitions of other robots, in order to reach some points in its partition. One advantage of DARP is that if a solution can be

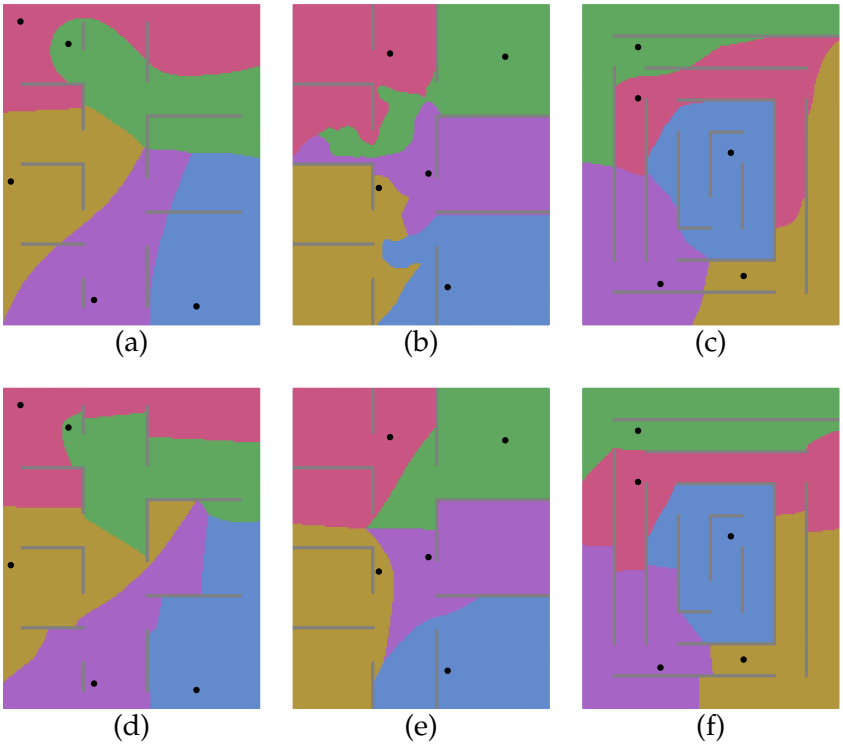


FIGURE 7.6: Scenario 4 - Example of partitioning in the three different environments introduced in [Pal+19]: (a)&(d): Open rooms; (b)&(e): Rooms; (c)&(f): Spiral. The results in the top row are computed by DARP, and the row below by Voronoi.

found, the connectivity of a subregion is always guaranteed, while some polynomial time post-processing might need for the solution of Voronoi (as explained in section 7.2.3).

In Scenario 4 and Scenario 5, we evaluate DARP and Voronoi using the environments introduced in [Pal+19]. In their work, partitions are computed based on power diagram. However, one drawback of power diagram is that the resulting subregions might be disconnected. To handle this issue, they proposed a control policy to adjust the locations of generators (equivalent to anchor points in our context) in order to achieve an equitable and connected partition. It's worth noting that they do not provide theoretical support to guarantee the convergence of the control law to a connected partition. Consequently, their algorithm might need to be executed multiple times with different initial conditions until a satisfactory connected partition is achieved. Additionally, their method does not ensure the relative-convexity of the subregions. In our application, we assume that the anchor points are fixed, and either DARP or Voronoi can compute a feasible solution. Certainly, when we compare the results in Fig. 7.6(b) and (e), it becomes evident that the Voronoi method provides a solution with a better quality in terms of the reachable area from the anchor point.

In Scenario 5, we extend the applicability of Voronoi on a workspace with a non-uniform density function. It turns out that the desired partition properties can still

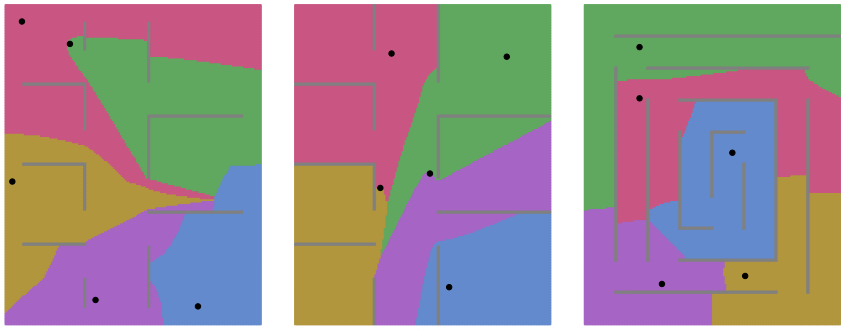


FIGURE 7.7: Scenario 5 - The results computed by the Voronoi-base method when the non-uniform density function is applied. (a): Open rooms; (b): Rooms; (c): Spiral.

be perserved as guaranteed by the theoretical basis. Specifically, in this scenario, where cells closer to the bottom of the map have a larger value of f, we can observe in Fig. 7.7 that the surface areas of the yellow, purple, and blue subregions are smaller than those of the red and green subregions. The weighted DARP has been explored in a study [IR22], and the *weighted Ham-Sandwich cuts* has also been studied in [BL05]. Further comparisons between these methods are expected to be conducted in future work.

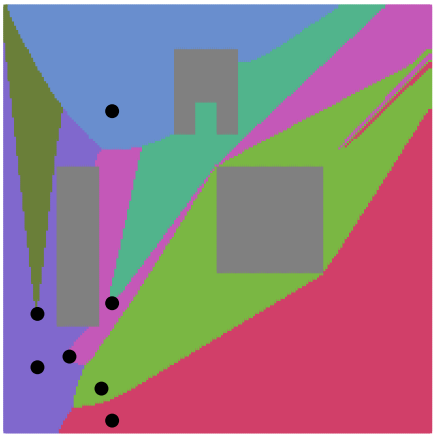


FIGURE 7.8: Illustration of Voronoi solution where the subregions are not compact.

Issues with Voronoi

One drawback associated with Voronoi is that the solution computed by solving (7.12) may include unconnected areas. An example is illustrated in Fig. 7.8. In this instance, a non-fair target capacity is applied, where we set $\mathbf{c} = [0.1, 0.2, 0.1, 0.1, 0.3, 0.05, 0.15]$, so that the resulting partition has a different area for each subregion. It can be observed that the red subregion and the pink one are not compacted, as there are some red and pink cells appear in the green subregion. This problem has been explained in Section 7.2.3 and illustrated in Fig. 7.2(c), where there is some ambiguity

that a cell can be assigned to either of the anchor points, the overall cost is still optimal. One straightforward solution is to arrange these outlier cells: swap the red outlier cells with these green cells at the boundary between the red and green subregions, and proceed in the same way to handle the pink outlier cells. It worth noting that this arranging method is not ad hoc, where a general assigning strategy has been mentioned earlier in Section 7.2.3. Although a post-processing step may be required in certain situations, it does not compromise the theoretical guarantee that resulting subregions within Voronoi are star-convex.

7.4 Equitable division in nonconvex polygon

The earlier work is built on the assumption that $\mathcal{B}$ is convex. However, in the nonconvex case where the boundary of $\partial\mathcal{B}$ includes reflex vertices, certain constraints need to be reevaluated. Consider a concave polygonal workspace denoted as P, with these reflex vertices. In such scenarios, it is possible that a partition fully reachable from some anchor points may not exist. However, if we view the envelope of the workspace as "walls" or obstacles, meaning the cable can be blocked by these reflex vertices, it still remains valuable to find a partition for a nonconvex polygon.

In the case of DARP, this new setting does not affect its functionality, and it can work as long as the unoccupied cells are connected. For the Voronoi-based method, we can generalize the concept of a shortest path between two points, allowing it to include the reflex vertices on ∂P. A study presented in [BBK06] demonstrates that there always exists a geodesic that can divide a general polygon into two equal halves. This conclusion can be extended to the existence of an n-equal relatively star-convex subdivision within a general polygon.

7.5 Conclusion

In this chapter, we have summarized three existing methods for computing locus-based equitable partitions of polygons. We conducted a qualitative comparison among these methods, considering various aspects such as workspace properties, solution quality, complexity, and extensibility to cope with other potential constraints, as displayed in Tab. 7.1.

In general, DARP and Voronoi exhibit broader applicability as they can operate on workspaces of any shape and handle scenarios with obstacles. However, HST requires that the workspace is polygonal and that the anchor points are in general positions. Regarding partitioning quality, HST generates subregions that are all convex polygons and are relatively star-convex when the bounding hull of the workspace is nonconvex. In terms of computational complexity, HST is the fastest, capable of being computed in polynomial time. On the other hand, both DARP and Voronoi rely on optimization methods, with Voronoi being significantly more efficient than DARP. Voronoi achieves quicker convergence to the optimal solution by

	DARP	HST	Voronoi
workspace	any shape	polygonal	any shape
obstacles	✔	✘	✔
anchors	✔	in general position	✔
partition	any shape	convex polygonal	relatively star-convex
complexity	+	+++	++
extensibility	++	+	++++

TABLE 7.1: Comparision between three partitioning methods

minimizing the "workload" based on the Euclidean shortest path in the objective function. However for DARP, the shortest path distance information is primarily used to update control variables, serving as a heuristic to guide the objective function optimization. DARP is less efficient and easier to get trapped in local minima, especially in the presence of several specific challenging grid cells. All three methods can be extended to compute weighted partitions when the workspace is equipped with a non-uniform density function. Furthermore, DARP and Voronoi can compute non-equitable divisions with respect to target "capacities". Voronoi, formulated as a continuous optimization problem, is more practical when it comes to adding new constraints to enhance desired geometric properties within the resulting subregions.

In multi-robot system applications, the *power diagram* is frequently applied to compute a partition, where the initial robot positions do not need to be fixed, and disconnections in the resulting solution can be resolved by adjusting the initial position of the robots. We demonstrate that computing a Voronoi diagram based on the L1 norm distance can effectively address this problem. We believe that these findings have broad applications in the field of multi-robot systems. Our work is the first to consider the task of multiple tethered robot coverage, and our approach of initially computing a locus-based equitable relatively-convex partition offers an efficient solution to avoid cable tangling. This greatly simplifies the coverage path planning task, as explored in Chapter 6.

7.6 Open Problems

In our application scenarios, besides the equal-area constraint, the constraints associated with the cable must also be taken into account, such as the cable length constraint, which can be interpreted as the **diameter constraint** that requires each sub-region to have a limited radius, and the **reachability constraint** in case of the presence of forbidden areas in the workspace.

7.6.1 Diameter Constraint

Let ℓ be the maximum cable length, for each anchor point x_i, a point x can be assigned to it must satisfy the condition $d(x, x_i) \leq \ell$, and it can be addressed by adding

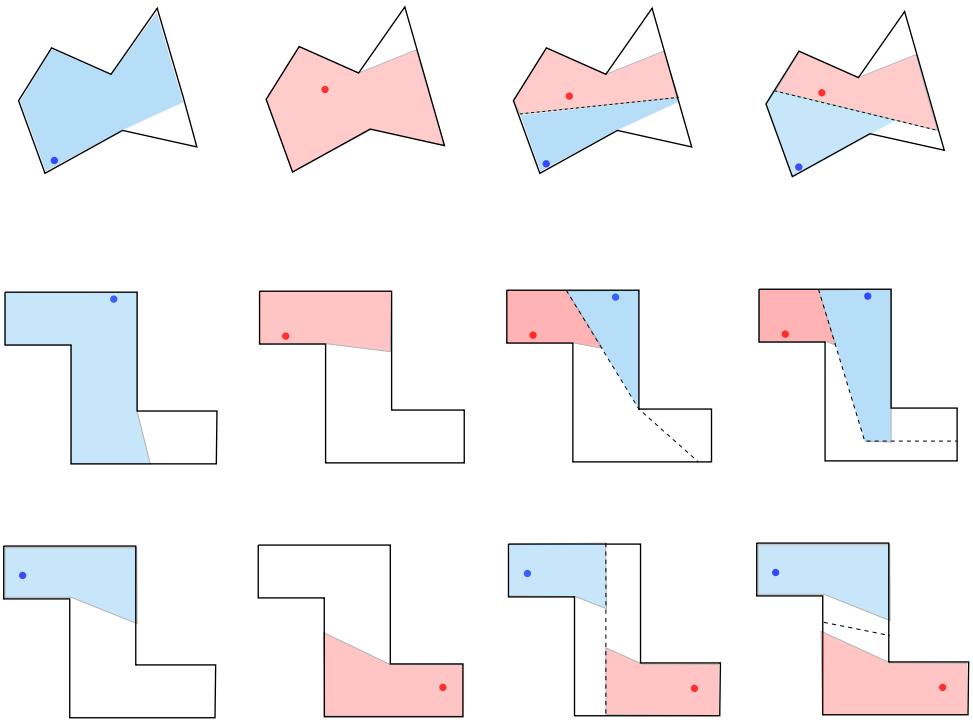


FIGURE 7.9: 2-partition for a concave polygon. In each row: the first two figures display respectively the visible polygons for the blue and red point within P; the third figure illustrates an equitable partition along with the visible polygon in each subregion; and the forth figure shows an alternative equitable partition which that results in a larger sum of visible polygon surfaces.

a constraint in (7.9):

$$I_i(x) = 0, \ \forall i : d(x, x_i) > \ell \tag{7.13}$$

As for its solution, it should depend on the value of ℓ: if ℓ has a large or small value, then (7.13) can be easily satisfied or no solution exists; otherwise, the solution will be like a collection of hyperbolic and circular curves. It will then be interesting to study whether the geometric properties of each sub-region can still be preserved, such as connectivity and relative star convexity. To our knowledge, a similar problem has been discussed in [CD13] while their basic problem is to find a partition that minimizes the maximum workload among these subregions.

7.6.2 Reachability Constraint

In case that the workspace is a concave polygon, and the bounding hull can be seen as "walls", namely, all the exterior area out of P are forbidden areas. Given a partition $\{P_1, ..., P_n\}$, we denote the visible polygon in P_i as $\mathcal{V}(P_i, x_i)$ and its surface as $\mathcal{A}(\mathcal{V}(P_i, x_i))$. Since such equitable partition is not unique, we can consider a new problem that seeks an equitable partition $\{P_1, ..., P_n\}$ that maximizes the total visible areas $\sum_{i=1}^{n} \mathcal{A}(\mathcal{V}(P_i, x_i))$. An example of 2-partition for a concave polygon is demonstrated in Fig. 7.9.

7.6.3 Anchor points placement

From Chapter 6, we can learn that in the case of forbidden zones, the area covered by a tethered robot is highly dependent on the location of its anchor point. In certain scenarios, if the anchor points are mobile and can be relocated, it becomes interesting to optimize their placement to achieve maximum coverage. Similar work is discussed in [Gan07], where the problem targets multi-agent deployment for visibility coverage in a non-convex polygonal environment with holes. Specifically, for a single agent, it proposes a gradient-based control policy to guide the agent to an optimal position along a trajectory where the visible area is nondecreasing. A possible extension of this concept involves considering the distinct impacts of forbidden zones (holes) and obstacles. In such cases, the objective shifts from maximizing the visible area to maximizing the reachable area.

Chapter 8

Conclusion

8.1 Summary

The goal of this thesis is to design efficient planning algorithms for a multiple tethered robots system to accomplish a coverage task over a large surface. As stated in Chapter 2, the challenges in this planning problem are issued with the cable constraints. Specially, the following constraints have been considered:

- The cable has a limited length;
- The cable cannot be self-crossing;
- The cable is kept taut when robots navigate in the workspace;
- The cable is vulnerable to some forbidden areas.

Our ultimate objective settles in coverage path planning, a complex task in robotics, where robots are tasked with covering the entire workspace rather than a single target point. However we believe that it is essential to initially address the fundamental non-crossing AMAPF problem, detailed in Chapters 4 and 5, where initial positions and target points are given. The challenge in this problem lies in collision avoidance as the cable is regarded as obstacles. With the assumption that the cable is regarded as line and the size of robots is ignored, Chapter 4 abstracts this problem into a geometrical challenge. The methodology used in Euclidean bipartite matching, such as LSAP, is proven to provide an upper bound for this problem. A VNS approach is introduced to improve the upper bound computed by LSAP, followed by a CP model to solve it optimally. The basis of non-crossing AMAPF involves coupling a bipartite matching problem that assigns each anchor point to an optimal target, and a path planning problem for each assigned anchor-target pair. Our strategy is to enumerate all the paths with respect to the upper bound improved by VNS, and then CP shows its strength to filter the inconsistent assignments and path choices.

The complexity of this problem depends on the number of robots, obstacles and their distributions in the workspace. In Chapter 5, when extending this work to non point-sized robots, a new definition of makespan is formulated that incorporates the precedence constraints on robots' motion and a safety distance to avoid cable entanglements. The precedence graph is introduced to represent all the precedence

constraints in a solution, and a deadlock occurs if it contains a cycle. The challenge in this new problem is that the resulting paths in a feasible solution are not only non-crossing, but also cannot cause any deadlocks. We adopt the same procedure as before to solve this problem. The solution of LSAP is proven to contain no deadlocks, providing an upper bound. Expressing this new makespan explicitly in a CP model can be challenging, and the lazy constraint generation method is proposed to handle the deadlock constraint. Additionally, a major factor impacting resolution efficiency is the upper bound used to generate the CP model. We propose a dichotomous search to choose an appropriate upper bound, avoiding unnecessary path computations.

Given that avoiding cable entanglements is a crucial constraint in multiple tethered robots path planning, our strategy for solving the coverage task involves assigning each robot a separated area of operation to reduce the risk of entanglement. Chapter 6 firstly investigates the coverage path planning for a single tethered robot (TCPP). We show that the Spanning Tree Coverage (STC) algorithm can efficiently solve this problem in polynomial time. Specifically, the solution produced by this method ensures no cable self-crossing occurs, and the cable can be fully retracted when the robots returns to its anchor point. To satisfy the cable length constraint, we propose the application of Dijkstra's algorithm during the spanning tree search. Additionally, considering that the cable may be vulnerable to the forbidden areas, we demonstrate that the TCPP problem, when incorporating forbidden areas, becomes $\mathcal{NP}$-complete. The problem's hardness is influenced by the workspace scale, cable length, and the number of obstacles. We provide various ILP models to compute approximate solutions for the problem.

Finally, Chapter 7 explores different polygon partitioning approaches. In our application context, two constraints are identified: each subregion must contain an anchor point and its shape should be relatively star-convex to the anchor point. Three methods are evaluated theoretically and experimentally on simulated instances. The results shows that partitioning based on the *additively weighted Voronoi diagram* performs the best to solve our problem.

8.2 Future Work

The work presented in this thesis is accomplished based on some hypothesis, which simplifies real industrial constraints to a certain extent. We also discuss some open questions that remain to be solved. The following paragraphs explores the major perspectives for future research.

8.2.1 Avoiding Paths Enumeration

When solving the non-crossing AMAPF problem, we adopted a strategy of enumerating all candidate paths that could contribute to the solution. The results show that

this step can be very expensive for some difficult instances, especially when the gap between the upper bound solved by VNS and the lower bound by LBAP is large. Instead of precomputing all paths and then selecting compatible ones, we could directly search for a solution in the visibility graph. Geometric constraints associated with paths, such as non-intersection, as well as their precedence relationships could be partially relaxed in the model (a path is a sequence of edges, and the constraints can be locally satisfied at each vertex by imposing constraints on edge selection). It is worth evaluating the interest of this approach for computing tighter bounds.

8.2.2 Initial Locations Being not at Anchor Points

In the non-crossing AMAPF problem, we assume that the robots are always located at their anchor point. This simplification ensures that the path we are looking for is also the cable position, however, this assumption does not always hold in practical applications. A potential avenue for future research involves considering scenarios where the initial locations do not coincide with the anchor points. In such cases, during collision checks, the area swept by the cables forms a triangle if the anchor point, initial position, and target point are not collinear; otherwise, it becomes a segment. Fig. 8.1 illustrates examples of this, where two robots fixed to anchor points a_1 and a_2 respectively move from s_1 (or s_2) to t_1 (or t_2). Examples (a)-(f) depict scenarios without obstacles, while (g) includes obstacles. In each case, if the two (pseudo)triangles $\triangle a_1 s_1 t_1$ and $\triangle a_2 s_2 t_2$ (in (g), $\triangle a_1 s_1 t_1$ is a pseudotriangle) intersect, cable entanglement may occur. To prevent this, coordinated motion strategies are necessary, depending on different triangle intersection situations and the relative positions of a, s, and t (as exemplified in (b) and (e)), as shown in Fig. 8.1.

In general, the intersection between a pair of triangles can be abstracted as cross intersection or parallel coplanar triangle intersections [SLM13]. Addressing this geometric problem through constraints could be interesting. By proposing specific coordinated motion strategies for distinct cases, a combined makespan can be computed. Exploring this new variant of the non-crossing AMAPF problem holds potential for further investigation.

8.2.3 Coverage Path with Minimum Turns

In the coverage path planning problem, our primary goal is to search for a shortest coverage path that enables robots to thoroughly explore the entire workspace. Another commonly considered factor in this problem is the number of turns, as robots need to decelerate during turns and turning motions can be time consuming. Generally, the challenge of coverage path planning with the minimum number of turns is known to be an $\mathcal{NP}$-hard problem [Ark+05].

In the scheme of using the STC algorithm to generate a coverage path, assuming the workspace can be perfectly structured to a G_4 (refer to Chapter 2), the resulting coverage path is already optimal in terms of path length. However, there is room for

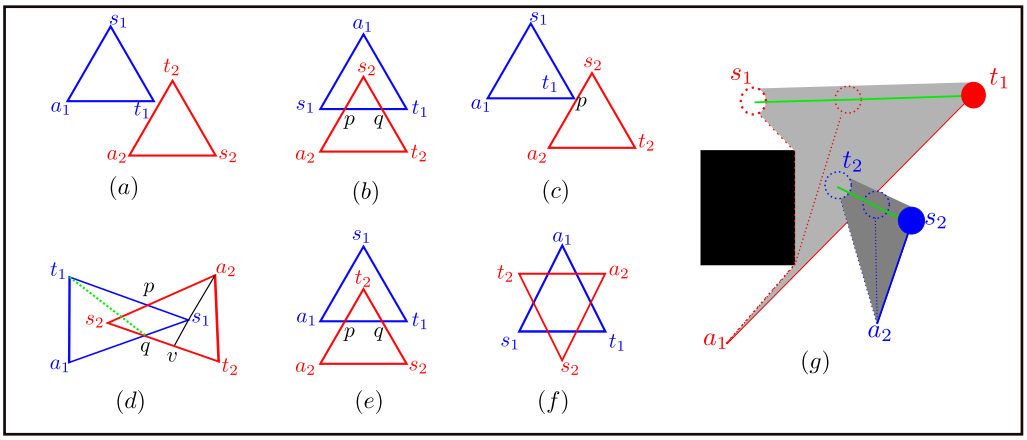


FIGURE 8.1: Illustrations of triangle intersections and corresponding coordinated robot motions. (a): Unrestricted movement of R_1 and R_2; (b)(c): Sequential order required, where R_1 must not pass point p until R_2 has left p, and a similar constraint applies at intersection point q; (d): R_1 moves toward a_1 to clear collision point q before R_2 reaches q; (e) R_2 retreats toward a_2 after reaching t_2 to allow collision-free arrival of R_1 at t_1; (f): A scenario combining various conflict types; (g): Intersection of two pseudotriangles.

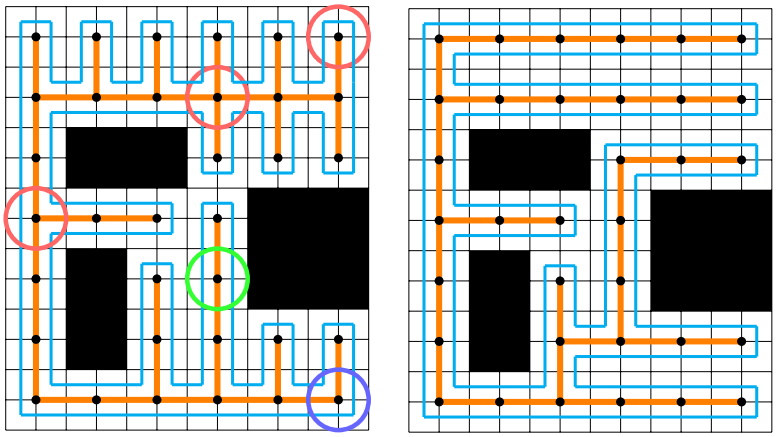
improvement by further minimizing the number of turns. The number of turns can be expressed as a variable dependent on the degree of each vertex in the spanning tree that is used to construct the coverage path, as illustrated in Fig. 8.2. This problem has been studied in [Lu+22], and a heuristic method based on ACO (ant colony optimization) is introduced to search for a spanning tree to optimize the number of turns, though without optimality guarantee.

However, three fundamental questions arise. First, despite the general problem of finding a coverage path with minimum turns being $\mathcal{NP}$-hard, is the problem of minimizing the number of turns in an STC-based coverage path is still $\mathcal{NP}$-hard? Their work lacks a rigorous proof in this regard. Second, it is worthwhile to explore constraint programming techniques, such as *tree constraints* [BFL05] or SAT solvers, to optimally solve this problem. Third, in a workspace that can be perfectly structured into a G_4, there are other types of Hamiltonian cycles not based on STC (an example is illustrated in Fig. 6.3, Chapter 6). Is it possible that a different kind of Hamiltonian cycle, not based on STC, results in a better solution?

Eventually, within the context of tethered robots, integrating the objective of minimizing turns with cable constraints could open up a valuable avenue for further research.

8.2.4 Partitioning with Forbidden Areas

As discussed in Section 7.6, the partitioning problem for multiple tethered robots can incorporate the diameter constraint to meet the cable length constraint. In specific workspaces with forbidden areas, finding a solution that maximizes the reachable area becomes an interesting challenge.



(a) (b)

FIGURE 8.2: Minimum turns problem with a coverage path generated by STC. The number of turns at each vertex is contingent on its degree. Consider these encircled vertices in (a), where a vertex of degree 4 can result in 4 turns, while vertices of degree 3 and degree 1 can produce 2 turns. For a vertex of degree 2, there are two cases to be considered, either 2 (blue) or 0 (green). (a): a random solution with 56 turns; (b): a solution with the minimum turns, 30.

8.2.5 Modeling Workspace with 3D Meshes

To address real industrial constraints associated with platform and hardware specifications, incorporating heterogeneous velocity in the robots' motion model during trajectory planning will be essential. Additionally, the models and methods presented in this paper are built on the assumption of a planar workspace, making them less adaptable to curved surfaces. Extending this work to 3D meshes would be interesting.

Appendix A

Global Constraint for Detecting Deadlocks

As explained in Chapter 5, when the physical size of robots are considered, an optimal solution for the non-crossing AMAPF problem should not create any deadlocks for robots' motion. Detecting deadlocks and computing the makespan relies on topological sorting of the precedence graph, a graph operation challenging to address in a CP model. This is why, in Section 5.5.1 and Section 5.5.2, we introduced a relaxed CP model for the original problem and proposed an approach based on lazy constraints generation to solve it. As we have discussed in Section 3.1, the use of global constraint can handle certain specific problems that cannot be addressed by ordinary constraints. In this section, we present some supplementary results obtained by solving the same CP model using a global constraint.

A.1 Incremental Topological Sorting

In this problem, the main interest of implementing a global constraint is to detect deadlock as early as possible rather than conducting an a posteriori check after the entire solution is generated. To achieve this, we implement an incremental topological sorting algorithm in the propagators.

Given a directed acyclic graph $G = (V, E)$, a topological sorting of G aims to find an ordering ord of the vertices in V, such that for every edge $(u \rightarrow v) \in E$, u precedes v in the ordering, as $ord(u) < ord(v)$. Typically, this can be accomplished by a depth-first-search. In our problem, we consider a dynamical implementation of this process, this is to say, maintaining a topological ordering of G in the presence of edge deletions and insertions. Consider the examaple illustrated in Figure. A.1: $\{y, x, y, a, c\}$ represents an ordered sequence of vertices in G, and their topological ordering is preserved by the edges $\{y \rightarrow a, a \rightarrow c, b \rightarrow x\}$. The objective is to determine the new ordering after adding a new edge $x \rightarrow y$ in G.

In [PK07] Pearce & Kelly proposed an algorithm for dynamic topological sorting that shows practical efficiency. Other incremental algorithms discussed in [Hae+12], exhibit varying performance depending on the density of graph, and require a rather

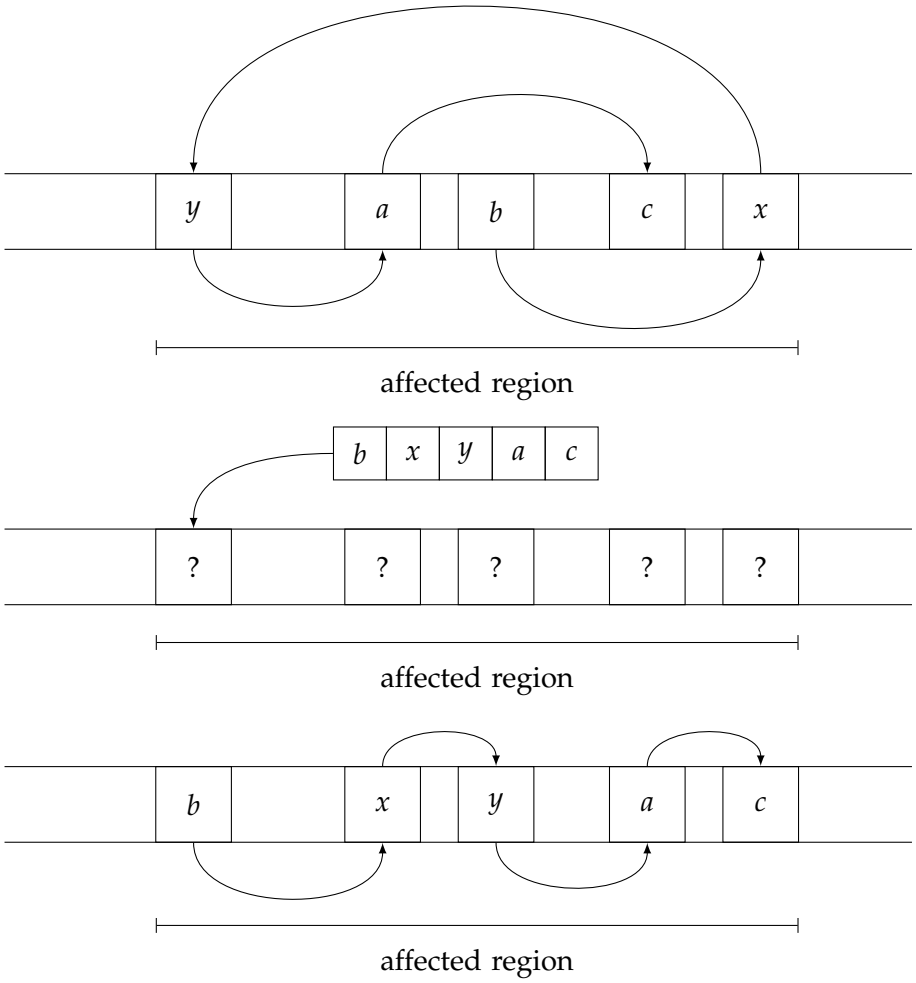


FIGURE A.1: An example from [PK07]: The vertices are places from left to right in topological order. A new edge $x \to y$ is added to the graph. We can find $\delta^F_{xy} = \{y, a, c\}$ and $\delta^B_{xy} = \{b, x\}$ with depth first search. For a correct ordering, vertices in δ^B_{xy} must be repositioned left of δ^F_{xy}, and we obtain $\{b, x, y, a, c\}$. Then we allocate the ordered vertices in the previous ordering list from the lowest available index to the highest, and the other unaffected vertices remain unchanged.

sophisticated data structure to handle Query, Insert and Delete operations in constant amortized time. To balance simplicity of data structures and practical performance, we implement the Pearce and Kelly algorithm in our approach. The principle behind Pearce & Kelly algorithm involves correcting the ordering by locally reorganizing the orders of affected vertices when an edge $x \rightarrow y$ is added to graph. The method operates as follows.

- **Comparison:** We compare x and y in the current ordering list, if $ord(x) < ord(y)$, then the relationship between them is already preserved and there is no need to correct the ordering list.
- **Two-Way Search:** When $ord(x) > ord(y)$, the algorithm use a two-way search to identify affected vertices: a forward search to find all the vertices in affected region δ_{xy} that can be reached from y (denoted as δ^F_{xy}), and a backward search to find all the vertices in δ_{xy} that can reach x (denoted as δ^B_{xy}). If x is reached during the forward search as $x \in \delta^F_{xy}$, it indicates that a circuit must exist.
- **Sorting:** The elements of $x \in \delta^F_{xy}$ and δ^B_{xy} are sorted to preserve their original order.
- **Repositioning:** Subsequently, the algorithm repositions vertices in δ^F_{xy} and δ^B_{xy} to ensure that all vertices in δ^B_{xy} have higher priority than those in δ^F_{xy}, and the other vertices remain unaffected.

A.2 Incremental Deadlock Check

In our problem context, we have a set of paths $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$ to check if a cycle exists in the precedence graph $G_\Pi = (V_\Pi, E_\Pi)$ based on Π (see Section 5.1.2). Here, a deadlock in our problem is the same as a cycle in G_Π. Rather than adding every node of a path in G_Π, we consider only the critical nodes, specifically, the first node of a path and the nodes shared by multiple paths. This avoids redundant vertices, reducing computational cost when maintaining a topological ordering in the graph. Let's introduce an incremental approach to detect deadlocks, as described in Algorithm 10.

The principle of Algorithm 10 is to maintain a set of paths already checked, denoted as Π_k, along with its associated precedence graph G_{Π_k}, and incrementally check the effects of a newly added path π_{k+1}. When π_{k+1} is processed, we first add its first node into G_{Π_k}, and then check if it intersects with every path in Π_k. If an intersection is detected, the process terminates with *Inconsistency*; otherwise, if there are nodes in common, we add these nodes (labeled by the path they belong to) and the resulting edges according to their precedence relationship in G_{Π_k}. There are three basic operations to be explained here.

- **AddVertex:** When a vertex is added, it is assigned the lowest priority order to avoid affecting the previous ordering.

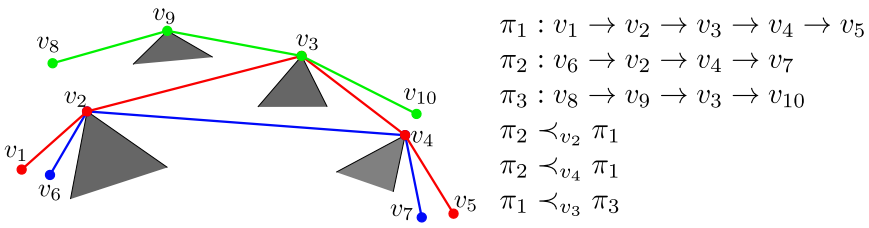


FIGURE A.2: The illustration for Example A.2.1.

- **AddEdge:** The algorithm Pearce & Kelly is applied to check this new edge. If cycles are found, the algorithm stops; otherwise a new ordering is constructed by local correction.
- **UpdatePath:** Since we only add some critical nodes of each path in the graph, each time a common node c between π_i and π_{k+1} is found, it is necessary to update the precedence relationship between c with other nodes already discovered on π_i and π_{k+1} respectively. In Example A.2.1, when path π_2 is processed, nodes (v_2, π_1) and (v_4, π_1) are added to the graph, along with two edges $1 \to 4$ and $4 \to 6$. In particular, when path π_3 is processed, on path π_1, we have edges $1 \to 4$, $4 \to 6$, $4 \to 9$ and $9 \to 6$. The edge $4 \to 6$ becomes redundant but is retained as it does not affect the resulting topological ordering.

Algorithm 10: INCREMDEADLOCKCHECK($\Pi_k, G_{\Pi_k}, \pi_{k+1}$)

Input: The set of paths already checked Π_k; the precedence graph based on Π_k: $G_{\Pi_k} = (V_{\Pi_k}, E_{\Pi_k})$; the path to be checked: π_{k+1}.
Output: The updated precedence graph $G_{\Pi_{k+1}}$ or *Inconsistency* in case path intersections or a deadlock are found.

1 **for** *each path* $\pi_i \in \Pi_k$ **do**
2 **if** π_{k+1} *does not intersect with* π_i **then**
3 Find all common nodes $C_{i,k+1} = \{c | \pi_i \prec_c \pi_{k+1}$ or $\pi_{k+1} \prec_c \pi_i\}$
4 **for** *each node* $c \in C_{i,k+1}$ **do**
5 AddVertex((c, π_i))
6 AddVertex((c, π_{k+1}))
 `/* According to the precedence relationship found in line 3. If a cycle occurs, the process returns Inconsistency. */`
7 AddEdge($(c, \pi_i), (c, \pi_{k+1})$)
8 **end**
9 UpdatePath(π_i)
10 UpdatePath(π_{k+1})
11 **else**
12 Return *Inconsistency*
13 **end**
14 **end**
15 $\Pi_{k+1} \leftarrow \Pi_k$
16 Return Π_{K+1}

Example A.2.1. Consider the example illustrated in Fig A.2. There are three paths π_1, π_2 and π_3, following the precedence relationships: $\pi_2 \prec_{v_2}$, $\pi_2 \prec_{v_4} \pi_1$ and $\pi_1 \prec_{v_3} \pi_3$. We process them one at a time. The evolution of the precedence graph and the ordering list is as follows.

1. $V = \emptyset, E = \emptyset, order = []$
2. $V = \{1 : (v_2, \pi_2), 2 : (v_2, \pi_1), 3 : (v_4, \pi_2), 4 : (v_4, \pi_1)\}$
   $E = \{1 \rightarrow 3, 3 \rightarrow 4, 2 \rightarrow 4\}$
   $order = [1, 2, 3, 4]$
3. $V = \{1 : (v_1, \pi_1), 2 : (v_6, \pi_2), 3 : (v_2, \pi_2), 4 : (v_2, \pi_1), 5 : (v_4, \pi_2), 6 : (v_4, \pi_1), 7 : (v_8, \pi_3), 8 : (v_3, \pi_3), 9 : (v_3, \pi_1)\}$
   $E = \{2 \rightarrow 3, 1 \rightarrow 4, 3 \rightarrow 5, 5 \rightarrow 6, 4 \rightarrow 6, 7 \rightarrow 8, 9 \rightarrow 8, 4 \rightarrow 9, 9 \rightarrow 6\}$
   $order = [1, 2, 3, 4, 5, 9, 6, 7, 8]$

Once G_Π has been constructed, the overall makespan can be computed as shown in Section 5.1.2. Note that when checking for deadlocks, there is no need to add the first and last nodes of each path into G_Π. However, when computing the makespan, this information must be taken into account. If a new node or edge is added, it is necessary to visit all the vertices starting from the first node that is affected in the ordering list. In the worst case, the time complexity is $\mathcal{O}(E_\Pi)$.

A.3 Global Constraint of Makespan

Following the notations defined in Section 5.5.1, where for each anchor point $a_i \in \mathcal{A}$, the integer variable x_i represents the target assigned to a_i, the integer variable z_i represents the path selected to reach x_i from a_i, and the integer variable Obj represents the makespan; We introduce a global constraint $GlobalMakespan(\{z_1, ..., z_n\}, Obj)$, which is satisfied if Obj equals to the longest path in $G_{\Pi(z_1,...,z_n)}$ where $G_{\Pi(z_1,...,z_n)}$ represents the precedence graph constructed based on the candidate paths set corresponding to $\{z_1, .., z_n\}$.

To propagate this variable, we maintain a set variable $pathpool$ to track all the paths z_i that have been checked. Each time a variable z_i is instantiated, we firstly verify whether it intersects the paths already in $pathpool$. A failure is triggered if an intersection is found. Otherwise, $G_{\Pi(pathpool)}$ is incrementally updated by Algorithm 10. We trigger a failure if a deadlock is found or the resulted makespan exceeds the upper bound of Obj; otherwise the lower bound of Obj is updated.

A.4 Experimental Results

In Fig. A.3, we compare the global constraint approach and the lazy constraint generation approach based on their solving times for the same CP model, denoting the

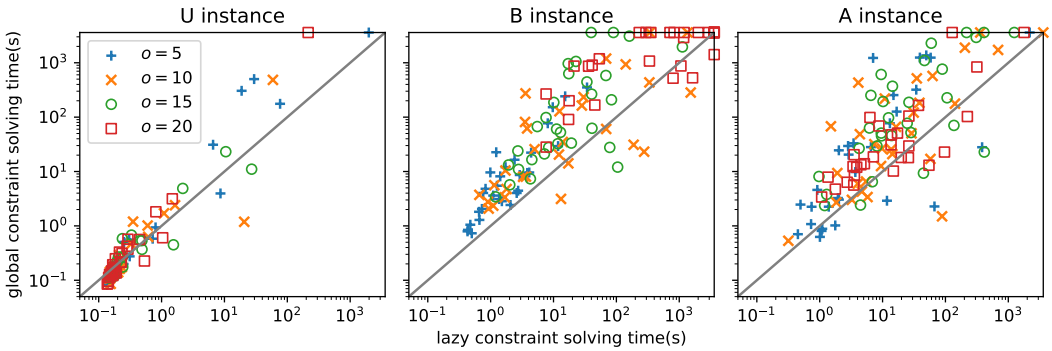


FIGURE A.3: Comparison of solving time by global constraint and lazy constraint generation approach: Each point (x, y) corresponds to an instance with $n = 30$, where x-axis represents the solving time by lazy constraint, and y-axis represents that of global constraint. The color of point depends on o. U(resp. B and A) instances are displayed on the left (resp. middle, right).

solving times as t_{gc} and t_{lazy}, respectively. The evaluation of their performance involves starting the resolution process with the same solution from the VNS step, and the dichotomous search is not applied. The results are presented on a per-instance basis when the number of robots $n = 30$, and the number of obstacles o varies in the range of $[5, 10, 15, 20]$. The solving time is limited to 3600 seconds.

For the majority of instances (U instances), the difference between the two methods is not evident, as most instances can be resolved within 10 seconds. The lazy constraint approach successfully solves all instances within 1 hour, while there are two instances that cannot be solved by the global constraint. When considering B instances and A instances, the lazy constraint approach shows better performance. Specifically, 90.0% of B instances show $t_{lazy} < t_{gc}$, and this percentage for A instances is 80.0%. Notably, for B instances, there are 8 instances unsolved by the global constraint compared to only 1 instance for the lazy constraint approach. Similarly, for A instances, there are 19 instances unsolved by the global constraint, whereas only 8 remain unsolved by the lazy constraint approach.

Bibliography

[AC02] Ercan U. Acar and Howie Choset. "Sensor-based Coverage of Unknown Environments: Incremental Construction of Morse Decompositions". In: *The International Journal of Robotics Research* 21 (2002), pp. 345–366.

[Ain+16] Sandip Aine et al. "Multi-Heuristic A*". In: *The International Journal of Robotics Research* 35 (2016). Publisher: SAGE Publications Ltd STM, pp. 224–243.

[Amb+03] Luigi Ambrosio et al. *Lecture notes on optimal transport problems*. Springer, 2003.

[ANB11] Pablo Abad-Manterola, Issa A. D. Nesnas, and Joel.W. Burdick. "Motion planning on steep terrain for the tethered axel rover". In: *2011 IEEE International Conference on Robotics and Automation*. ISSN: 1050-4729. 2011, pp. 4188–4195.

[AP10] D. Adjiashvili and D. Peleg. "Equal-area locus-based convex polygon decomposition". In: *Theoretical Computer Science*. Structural Information and Communication Complexity (SIROCCO 2008) 411 (2010), pp. 1648–1667.

[Ark+05] Esther M Arkin et al. "Optimal covering tours with turn costs". In: *SIAM Journal on Computing* 35 (2005), pp. 531–566.

[Bal+13] Daniel Balouek et al. "Adding virtualization capabilities to the Grid'5000 testbed". In: *Cloud computing and services science*. Vol. 367. Communications in computer and information science. Springer International Publishing, 2013, pp. 3–20.

[BBK06] Sergey Bereg, Prosenjit Bose, and David Kirkpatrick. "Equitable subdivisions within polygonal regions". In: *Computational Geometry*. Special Issue on the Japan Conference on Discrete and Computational Geometry 2004 34 (2006), pp. 20–27.

[BÇ99] Rainer E. Burkard and Eranda Çela. "Linear Assignment Problems and Extensions". In: *Handbook of Combinatorial Optimization: Supplement Volume A*. Springer US, 1999, pp. 75–149.

[Ber+08] Mark de Berg et al. *Computational Geometry: Algorithms and Applications*. 3rd ed. Springer-Verlag, 2008.

[Bes+07] Christian Bessiere et al. "The complexity of reasoning with global constraints". In: *Constraints* 12 (2007), pp. 239–259.

[BFL05] Nicolas Beldiceanu, Pierre Flener, and Xavier Lorca. "The tree constraint". In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2005, pp. 64–78.

[Bha+15] Subhrajit Bhattacharya et al. "A topological approach to using cables to separate and manipulate sets of objects". In: *The International Journal of Robotics Research* 34 (2015), pp. 799–815.

[Bha10] Subhrajit Bhattacharya. "Search-Based Path Planning with Homotopy Class Constraints". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 24 (2010). Number: 1, pp. 1230–1237.

[BKS00] S. Bespamyatnikh, D. Kirkpatrick, and J. Snoeyink. "Generalizing Ham Sandwich Cuts to Equitable Subdivisions". In: *Discrete & Computational Geometry* 24 (2000), pp. 605–622.

[BL05] Prosenjit Bose and Stefan Langerman. "Weighted Ham-Sandwich Cuts". In: *Discrete and Computational Geometry*. Lecture Notes in Computer Science. Springer, 2005, pp. 48–53.

[Bou+04] Frédéric Boussemart et al. "Boosting systematic search by weighting constraints". In: *ECAI*. Vol. 16. 2004, p. 146.

[BVX15] Peter Brass, Ivo Vigan, and Ning Xu. "Shortest path planning for a tethered robot". In: *Computational Geometry* 48 (2015), pp. 732–742.

[Cao+23a] Muqing Cao et al. "NEPTUNE: Nonentangling Trajectory Planning for Multiple Tethered Unmanned Vehicles". In: *IEEE Transactions on Robotics* 39 (2023). Conference Name: IEEE Transactions on Robotics, pp. 2786–2804.

[Cao+23b] Muqing Cao et al. *Path Planning for Multiple Tethered Robots Using Topological Braids*. arXiv:2305.00271 [cs]. 2023.

[Car+15] John Gunnar Carlsson et al. "A Bottleneck Matching Problem with Edge-Crossing Constraints". In: *International Journal of Computational Geometry & Applications* 25 (2015). Publisher: World Scientific Publishing Co., pp. 245–261.

[CAY10] John Gunnar Carlsson, Benjamin Armbruster, and Yinyu Ye. "Finding equitable convex partitions of points in a polygon efficiently". In: *ACM Transactions on Algorithms* 6 (2010), 72:1–72:19.

[CCD13] John Gunnar Carlsson, Erik Carlsson, and Raghuveer Devulapalli. "Balancing workloads of service vehicles over a geographic territory". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 209–216.

[CD13] John Gunnar Carlsson and Raghuveer Devulapalli. "Dividing a Territory Among Several Facilities". In: *INFORMS Journal on Computing* 25 (2013). Publisher: INFORMS, pp. 730–742.

[Cor+09] Cormen, Thomas H. et al. *Introduction to Algorithms*. 2009.

[CS14] Geoffrey Chu and Peter J. Stuckey. *Chuffed solver description*. 2014.

[Dan90] George B Dantzig. "Origins of the simplex method". In: *A history of scientific computing*. 1990, pp. 141–151.

[Fla+21] Rémi Flamary et al. "POT: Python Optimal Transport". In: *Journal of Machine Learning Research* 22 (2021), pp. 1–8.

[For56] Ford-Fulkerson, Lester R. *NETWORK FLOW THEORY*. Tech. rep. Section: Technical Reports. Rand Corp Santa Monica Ca, 1956.

[Gan07] Ganguli, Anurag. "Motion coordination for mobile robotic networks with visibility sensors". PhD thesis. University of Illinois at Urbana-Champaign, 2007.

[Gom58] Ralph Gomory. "Essentials of an algorithm for integer solutions to linear programs". In: *Recent Advances in Mathematical Programming* (1958), pp. 269–302.

[GR01] Yoav Gabriely and Elon Rimon. "Spanning-tree based coverage of continuous areas by a mobile robot". In: *Annals of Mathematics and Artificial Intelligence* 31 (2001), pp. 77–98.

[Gur23] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023.

[Hae+12] Bernhard Haeupler et al. "Incremental Cycle Detection, Topological Ordering, and Strong Component Maintenance". In: *ACM Transactions on Algorithms* 8 (2012), 3:1–3:33.

[HL96] Susan Hert and Vladimir Lumelsky. "The ties that bind: Motion planning for multiple tethered robots". In: *Robotics and Autonomous Systems* 17 (1996), pp. 187–215.

[HL97] Susan Hert and Vladimir Lumelsky. "Planar Curve Routing for Tethered-Robot Motion Planning". In: *International Journal of Computational Geometry & Applications* 07 (1997). Publisher: World Scientific Publishing Co., pp. 225–252.

[HL99] S. Hert and V. Lumelsky. "Motion planning in R/sup 3/ for multiple tethered robots". In: *IEEE Transactions on Robotics and Automation* 15 (1999). Conference Name: IEEE Transactions on Robotics and Automation, pp. 623–639.

[HM85] Stefan Hertel and Kurt Mehlhorn. "Fast triangulation of the plane with respect to simple polygons". In: *Information and Control*. International Conference on Foundations of Computation Theory 64 (1985), pp. 52–76.

[HS20] Valentin Hartmann and Dominic Schuhmacher. “Semi-discrete optimal transport: a solution procedure for the unsquared Euclidean distance case”. In: *Mathematical Methods of Operations Research* 92 (2020), pp. 133–163.

[HS94] John Hershberger and Jack Snoeyink. “Computing minimum length paths of a given homotopy class”. In: *Computational Geometry* 4 (1994), pp. 63–97.

[IPS82] Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. “Hamilton Paths in Grid Graphs”. In: *SIAM Journal on Computing* 11 (1982). Publisher: Society for Industrial and Applied Mathematics, pp. 676–686.

[IR22] Olivier Idir and Alessandro Renzaglia. “Multi-Robot Weighted Coverage Path Planning: a Solution based on the DARP Algorithm”. In: *2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2022, pp. 98–104.

[IS10] Takeo Igarashi and Mike Stilman. “Homotopic path planning on manifolds for cabled mobile robots”. In: *Algorithmic Foundations of Robotics IX: Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics*. Springer. 2010, pp. 1–18.

[JER12] John Gunnar Carlsson, Erik Carlsson, and Raghuveer Devulapalli. “Equitable partitioning with obstacles”. In: *Technical report*. 2012.

[KBK14] Soonkyum Kim, Subhrajit Bhattacharya, and Vijay Kumar. “Path planning for a tethered mobile robot”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 1050-4729. 2014, pp. 1132–1139.

[KCK17] Athanasios Ch. Kapoutsis, Savvas A. Chatzichristofis, and Elias B. Kosmatopoulos. “DARP: Divide Areas Algorithm for Optimal Multi-Robot Coverage Path Planning”. In: *Journal of Intelligent & Robotic Systems* 86 (2017), pp. 663–680.

[KF11] Sertac Karaman and Emilio Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The International Journal of Robotics Research* 30 (2011). Publisher: SAGE Publications Ltd STM, pp. 846–894.

[KL15] Soonkyum Kim and Maxim Likhachev. “Path planning for a tethered robot using Multi-Heuristic A* with topology-based heuristics”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 4656–4663.

[KMT19] Jun Kitagawa, Quentin Mérigot, and Boris Thibert. “Convergence of a Newton algorithm for semi-discrete optimal transport”. In: *Journal of the European Mathematical Society* 21 (2019), pp. 2603–2651.

[Kuh55] H. W. Kuhn. “The Hungarian method for the assignment problem”. In: *Naval Research Logistics Quarterly* 2 (1955), pp. 83–97.

[Lat12] Jean-Claude Latombe. *Robot Motion Planning*. Springer Science & Business Media, 2012.

[Lec11] Christophe Lecoutre. "STR2: optimized simple tabular reduction for table constraints". In: *Constraints* 16 (2011), pp. 341–371.

[Lec96] Michel Leconte. "A bounds-based reduction scheme for constraints of difference". In: *Proceedings of the Constraint-96 International Workshop on Constraint-Based Reasoning*. 1996, pp. 19–28.

[Lic82] David Lichtenstein. "Planar Formulae and Their Uses". In: *SIAM Journal on Computing* 11 (1982). Publisher: Society for Industrial and Applied Mathematics, pp. 329–343.

[Lu+22] Junjie Lu et al. "TMSTC*: A Turn-minimizing Algorithm For Multi-robot Coverage Path Planning". In: *arXiv preprint arXiv:2212.02231* (2022).

[LW66] Eugene L Lawler and David E Wood. "Branch-and-bound methods: A survey". In: *Operations research* 14 (1966), pp. 699–719.

[LW79] Tomás Lozano-Pérez and Michael A. Wesley. "An algorithm for planning collision-free paths among polyhedral obstacles". In: *Communications of the ACM* 22 (1979), pp. 560–570.

[Ma+17] Hang Ma et al. "Overview: A hierarchical framework for plan generation and execution in multirobot systems". In: *IEEE Intelligent Systems* 32 (2017), pp. 6–12.

[Ma+18] Hang Ma et al. *Multi-Agent Path Finding with Deadlines*. arXiv:1806.04216 [cs]. 2018.

[Mac77] Alan K Mackworth. "Consistency in networks of relations". In: *Artificial intelligence* 8 (1977), pp. 99–118.

[Mec+17] L.S.R. Mechsy et al. "A novel offline coverage path planning algorithm for a tethered robot". In: *2017 17th International Conference on Control, Automation and Systems (ICCAS)*. 2017, pp. 218–223.

[MH17] Seth McCammon and Geoffrey A. Hollinger. "Planning and executing optimal non-entangling paths for tethered underwater vehicles". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 3040–3046.

[MH97] N. Mladenović and P. Hansen. "Variable neighborhood search". In: *Computers & Operations Research* 24 (1997), pp. 1097–1100.

[MLT95] Young-Soo Myung, Chang-Ho Lee, and Dong-Wan Tcha. "On the generalized minimum spanning tree problem". In: *Networks* 26 (1995), pp. 231–241.

[MO07] Ivan Maza and Anibal Ollero. "Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms". In: *Distributed Autonomous Robotic Systems 6*. Springer, 2007, pp. 221–230.

[Net+07] Nicholas Nethercote et al. "MiniZinc: Towards a Standard CP Modelling Language". In: *Principles and Practice of Constraint Programming – CP 2007*. Lecture Notes in Computer Science. Springer, 2007, pp. 529–543.

[Pal+19] José Manuel Palacios-Gasós et al. "Equitable persistent coverage of non-convex environments with graph-based planning". In: *The International Journal of Robotics Research* 38 (2019). Publisher: SAGE Publications Ltd STM, pp. 1674–1694.

[Pav+11] Marco Pavone et al. "Distributed Algorithms for Environment Partitioning in Mobile Robotic Networks". In: *IEEE Transactions on Automatic Control* 56 (2011), pp. 1834–1848.

[PD22] Louis Petit and Alexis Lussier Desbiens. "TAPE: Tether-Aware Path Planning for Autonomous Exploration of Unknown 3D Cavities Using a Tangle-Compatible Tethered Aerial Robot". In: *IEEE Robotics and Automation Letters* 7 (2022). Conference Name: IEEE Robotics and Automation Letters, pp. 10550–10557.

[Pen07] David W. Pentico. "Assignment problems: A golden anniversary survey". In: *European Journal of Operational Research* 176 (2007), pp. 774–793.

[PFL16] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. "Choco solver documentation". In: *TASC, INRIA Rennes, LINA CNRS UMR* 6241 (2016).

[PK07] David J. Pearce and Paul H. J. Kelly. "A dynamic topological sort algorithm for directed acyclic graphs". In: *ACM Journal of Experimental Algorithmics* 11 (2007), 1.7–es.

[Pop20] Petrică C. Pop. "The generalized minimum spanning tree problem: An overview of formulations, solution procedures and latest advances". In: *European Journal of Operational Research* 283 (2020), pp. 1–15.

[PSS21] 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*. 2021, pp. 1–17.

[PSS23] Xiao Peng, Olivier Simonin, and Christine Solnon. "Non-Crossing Anonymous MAPF for Tethered Robots". In: *Journal of Artificial Intelligence Research (JAIR)* 78 (2023).

[Put79] Putnam. *Problem a4*. 1979.

[Ran+20] Vinitha Ranganeni et al. "Effective footstep planning using homotopy-class guidance". In: *Artificial Intelligence* 286 (2020), p. 103346.

[Rég94] Jean-Charles Régin. "A filtering algorithm for constraints of difference in CSPs". In: *AAAI*. Vol. 94. 1994, pp. 362–367.

[RVW06] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[SH15] 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)*. IEEE, 2015, pp. 4161–4166.

[Sha+15] Guni Sharon et al. "Conflict-based search for optimal multi-agent pathfinding". In: *Artificial Intelligence* 219 (2015), pp. 40–66.

[Sha+19] Gokarna Sharma et al. "A 2-Approximation Algorithm for the Online Tethered Coverage Problem". In: *Robotics: Science and Systems XV* (2019). Conference Name: Robotics: Science and Systems 2019 ISBN: 9780992374754 Publisher: Robotics: Science and Systems Foundation.

[Sin90] F.W. Sinden. "The Tethered Robot Problem". In: *The International Journal of Robotics Research* 9 (1990). Publisher: SAGE Publications Ltd STM, pp. 122–133.

[SLM13] Chaman L Sabharwal, Jennifer L Leopold, and Douglas McGeehan. "Triangle-triangle intersection determination and classification to support qualitative spatial reasoning". In: *Polibits* (2013), pp. 13–22.

[Sol+15] Christine Solnon et al. "On the complexity of submap isomorphism and maximum common submap problems". In: *Pattern Recognition* 48 (2015), pp. 302–316.

[SP20] Danylo Shapovalov and Guilherme A. S. Pereira. "Exploration of unknown environments with a tethered mobile robot". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 6826–6831.

[SR14] Iddo Shnaps and Elon Rimon. "Online Coverage by a Tethered Autonomous Mobile Robot in Planar Unknown Environments". In: *IEEE Transactions on Robotics* 30 (2014), pp. 966–974.

[ST42] A. H. Stone and J. W. Tukey. "Generalized "sandwich" theorems". In: *Duke Mathematical Journal* 9 (1942). Publisher: Duke University Press, pp. 356–359.

[Ste+19] Roni Stern et al. "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks". In: *Proceedings of the International Symposium on Combinatorial Search* 10 (2019). Number: 1, pp. 151–158.

[Su+22] Yao Su et al. "Object Gathering With a Tethered Robot Duo". In: *IEEE Robotics and Automation Letters* 7 (2022), pp. 2132–2139.

[TBN13] Melissa M. Tanner, Joel W. Burdick, and Issa A. D. Nesnas. "Online motion planning for tethered robots in extreme terrain". In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 5557–5564.

[TS14] Reza H Teshnizi and Dylan A Shell. "Computing cell-based decompositions dynamically for planning motions of tethered robots". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 6130–6135.

[TS21] Reza H. Teshnizi and Dylan A. Shell. "Motion planning for a pair of tethered robots". In: *Autonomous Robots* 45 (2021), pp. 693–707.

[WB18] Xiaolong Wang and Subhrajit Bhattacharya. "A Topological Approach to Workspace and Motion Planning for a Cable-Controlled Robot in Cluttered Environments". In: *IEEE Robotics and Automation Letters* 3 (2018), pp. 2600–2607.

[WBB21] Xiaolong Wang, Matthew Bilsky, and Subhrajit Bhattacharya. "Search-based configuration planning and motion control algorithms for a snake-like robot performing load-intensive operations". In: *Autonomous Robots* 45 (2021), pp. 1047–1076.

[WSB23] Xiaolong Wang, Alp Sahin, and Subhrajit Bhattacharya. "Coordination-free Multi-robot Path Planning for Congestion Reduction Using Topological Reasoning". In: *Journal of Intelligent & Robotic Systems* 108 (2023), p. 50.

[Xav99] P.G. Xavier. "Shortest path planning for a tethered robot or an anchored cable". In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. Vol. 2. 1999, pp. 1011–1017.

[Xia+18] Xuesu Xiao et al. "Motion Planning for a UAV with a Straight or Kinked Tether". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 8486–8492.

[Yan+22] Tong Yang et al. *Efficient Search of the k Shortest Non-Homotopic Paths by Eliminating Non-k-Optimal Topologies*. arXiv:2207.13604 [cs]. 2022.

[YL13a] Jingjin Yu and Steven LaValle. "Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 27 (2013). Number: 1, pp. 1443–1449.

[YL13b] Jingjin Yu and Steven M. LaValle. "Multi-agent Path Planning and Network Flow". In: *Algorithmic Foundations of Robotics X*. Springer Tracts in Advanced Robotics. Springer, 2013, pp. 157–173.

[YXW22] Tong Yang, Rong Xiong, and Yue Wang. "Efficient distance-optimal tethered path planning in planar environments: The workspace convexity". In: *arXiv preprint arXiv:2208.03969* (2022).

[ZP19] Xu Zhang and Quang-Cuong Pham. "Planning coordinated motions for tethered planar mobile robots". In: *Robotics and Autonomous Systems* 118 (2019), pp. 189–203.

FOLIO ADMINISTRATIF

THESE DE L'INSA LYON, MEMBRE DE L'UNIVERSITE DE LYON

NOM : PENG
(avec précision du nom de jeune fille, le cas échéant)

DATE de SOUTENANCE : 19/02/2024

Prénoms : Xiao

TITRE : Planning Problems in a Multi-Tethered-Robot System.

NATURE : Doctorat

Numéro d'ordre : 2024ISAL0018

Ecole doctorale : Informatique et Mathématiques

Spécialité : Informatique

RESUME :

Les systèmes de multiples robots ont été largement appliqué dans notre vie. En tant que type particulier de système mobile, les robots à câble jouent un rôle crucial dans des contextes spécifiques et des conditions difficiles, où le câble offre un accès stable à l'énergie et à la connectivité réseau. Cependant, les contraintes imposées par le câble introduisent également de nouveaux défis pour la planification des mouvements dans ces applications. Cette thèse se concentre sur les problèmes de planification pour une équipe de robots à câble, abordant deux problèmes majeurs: le Anonymous Multi-Agent Path Finding (AMAPF) sans croisement et le Multiple Tethered Coverage Path Planning (MTCPP). L'objectif du AMAPF sans croisement est de trouver un ensemble de trajectoires non croisées de manière à minimiser la longueur du plus long chemin (makespan). L'étude est divisée en deux cas, selon que l'on néglige la taille du robot ou non. Cette hypothèse influence significativement le calcul du makespan. Le problème est abstrait sous la forme d'un couplage bipartite euclidien, et nous montrons que des bornes peuvent être efficacement calculées en résolvant des problèmes d'affectation linéaires. Nous introduisons une méthode de recherche à voisinage variable pour améliorer les bornes supérieures, et un modèle de programmation par contraintes pour calculer des solutions optimales. L'approche est évaluée expérimentalement sur trois types différents d'instances. Le problème MTCPP est abordé en partitionnant initialement l'espace de travail en sous-régions équitables connectées, permettant à chaque robot de fonctionner indépendamment dans sa zone assignée. Nous proposons une approche basée sur le diagramme de Voronoi pondéré de manière additive, assurant une partition équitable qui impose la étoile-convexité relative de chaque sous-région par rapport au point d'ancrage associé, évitant ainsi l'emmêlement des cables. Pour la planification de la trajectoire de couverture de chaque robot, la méthode Spanning Tree Coverage permet de résoudre efficacement le problème tout en respectant les contraintes du câble.

MOTS-CLÉS : Multi Agent Path finding, Constraint Programming, Coverage Path Planning, Graph Theory

Laboratoire (s) de recherche : CITI, Inria

Directeur de thèse: Olivier SIMONIN, Christine SOLNON

Président de jury :

Composition du jury :
Aurélie BEYNIER, Yves DEVILLE, David COEURJOLLY, Cédric PRADALIER, Céderic Pralet, Olivier Simonin, Christine Solnon.