

Planning and Replanning Near-optimal Robot Coverage Paths in Partially Unknown Environments

by

Megnath Ramesh

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2025

© Megnath Ramesh 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner: Stefano Carpin
Professor, Dept. of Computer Science and Engineering,
University of California, Merced

Supervisor(s): Stephen L. Smith
Professor, Dept. of Electrical and Computer Engineering,
University of Waterloo

Baris Fidan
Professor, Dept. of Mechanical and Mechatronics Engineering,
University of Waterloo

Internal Member(s): Yash Vardhan Pant
Assistant Professor, Dept. of Electrical and Computer Engineering,
University of Waterloo

Mark Crowley
Associate Professor, Dept. of Electrical and Computer Engineering,
University of Waterloo

Internal-External Member: Stephen Vavasis
Professor, Dept. of Combinatorics and Optimization,
University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We study the problem of planning near-optimal coverage paths for a robot operating in an environment such that a sensor or tool attached to the robot covers the environment. We consider environments that contain unknown obstacles that are detected by the robot during the execution of the coverage path. Such obstacles necessitate a replan of the coverage path that safely avoids the new obstacles while still minimizing a cost function (coverage time, length, and/or turns).

For an environment with known obstacles, existing coverage planning approaches discourage turns in the path by covering the environment along the least number of *coverage lines*, i.e., straight-line paths. This is because turns not only slow down the robot but also negatively affect the quality of coverage, e.g., tools like cameras and cleaning attachments commonly have poor performance around turns. However, existing approaches do not guarantee the minimum number of coverage lines. To this end, we propose a minimum-turn coverage planning approach, namely optimal axis-parallel rank partitioning (OARP), that guarantees coverage along the minimum number of axis-parallel (horizontal/vertical) coverage lines. Using simulations in real-world environments, we show that OARP improves upon state-of-the-art approaches in terms of turns and coverage time.

When there are unknown obstacles in the environment, recomputing such paths online is computationally expensive. In such cases, the robot may have to wait for a safe path to continue coverage, resulting in robot stoppages that increase coverage time. To enable replanning with minimal stoppage, we extend OARP to introduce an anytime coverage replanning approach. Upon detecting a new obstacle in the environment, we replan an initial coverage path obtained from OARP within a given time budget (e.g., time to reach the new obstacle). Given this budget, the replanning approach also aims to minimize the overall path turns and coverage time. We showcase the replanning framework in experiments using an industrial cleaning robot avoiding unknown obstacles.

We then study two problems towards improving upon the above framework. First, we analyse the length of the coverage paths returned by OARP and obtain an approximation guarantee that quantifies the sub-optimality of OARP path lengths. Following this, we propose an approach that improves upon this approximation factor and computes shorter coverage paths in practice than OARP. The second problem is to remove the axis-parallel constraint imposed by OARP on the coverage lines. We propose decomposing the environment into sectors, i.e., possibly overlapping rectangular sub-regions that can each be covered by straight-line paths parallel to the longest sector edge. Using results from sub-modular set cover (SSC) problems, we propose a greedy approach to compute sectors and provide an approximation guarantee on the number of sectors in the decomposition.

Acknowledgements

I would like to thank my supervisors, Prof. Stephen L. Smith and Prof. Baris Fidan, for their incredible support throughout my Ph.D. program. Their advice and insights were invaluable in shaping not only the research presented in this thesis, but also my overall approach to problems in general. I am also grateful for the many valuable opportunities they gave me to broaden my skillset.

I am thankful for my Ph.D. thesis committee, Profs. Stephen Vavasis, Yash Pant, Mark Crowley, and Stefano Carpin, for thoughtful feedback on this work. I am also grateful to have been taught by the incredible faculty at the University of Waterloo throughout my graduate program.

I would like to thank Avidbots and the Mitacs program for supporting this research. The Avidbots partnership was invaluable because it helped me connect my theoretical research with practical industrial applications, while working with some amazing minds in the industry. I am especially thankful to Frank, who is a fantastic collaborator, mentor, friend, and climbing buddy. Discussions with Frank were integral in shaping the work in this thesis. I would also like to thank Florentin for bringing in a product perspective to the research (and for showing us how to do every climb in the gym).

I am thankful to members of the Autonomous Systems Lab (ASL), most of whom have become my close friends over the last few years. I am especially grateful to the "hen pack": Rod, Shamak, and Jack, who were a large part of my graduate school experience (thus completing our cycle of acknowledgments). I will cherish the times we spent on long bike rides, grad house hangouts, go-karting, and helping me face my demons in a 44 km cross-country ski trip. A huge shout-out to Alex Botros, who was always up for discussing any problem or topic, including some truly obscure ones, and was an incredible office mate. I am also thankful to all of the wonderful friends I have made during these last four years. The list is too long to name, but you know who you are.

Few bonds last the test of time better than the ones I have with the Hype (Toronto) crew: Freddy, Rahul, Arjun, Tanay, and OJ. I am thankful for this brotherhood that has lasted for over a decade. It's amazing to see how much we've grown and all the impressive things we've accomplished during the time that we've known each other. While we may all be in different places over the coming years, I am sure we will still be spitting out hot sports takes and embracing our inner uncles when we're 80 and older. I am also grateful for my friendships with Vinoth and Karthik, who have always been a supportive presence since high school, albeit in our messed-up Saidapet way.

Speaking of long-lasting bonds, I am deeply grateful to Michelle. I could write an entire thesis-long document about how lucky we are to have met each other in that board game cafe, how the last three years have been the best years of my life so far (road trip to Ottawa, coffees in Italy, and countless other beautiful moments), and how excited I am about our lives moving forward.

Last, but surely not least, I am eternally thankful to my family. My parents emphasized the importance of a good education to me, but they also encouraged me to follow my dreams in my way. They have always supported me to explore and find something that interests me, even if it took me to different shores far from home. Without their support, I would not be the person I am today.

Dedication

To Amma and Appa. This thesis is a product of our collective hardships and does not exist without you.

Table of Contents

Examining Committee Membership	ii
Author's Declaration	iii
Abstract	iv
Acknowledgements	v
Dedication	vii
List of Figures	xii
List of Tables	xvii
1 Introduction	1
1.1 Thesis Contributions	4
1.2 Organization	4
2 Literature Review and Background	7
2.1 The Coverage Path Planning (CPP) Problem	7
2.2 Literature Review	8
2.2.1 Offline CPP	9
2.2.2 Online CPP	11

2.2.3	Multi-robot CPP	13
2.3	Preliminaries	14
2.3.1	Graphs	14
2.3.2	Matching Problems in Graphs	14
2.3.3	Traveling Salesman Problems	15
2.3.4	Submodularity	16
3	Minimum-Turn CPP for Known Environments	18
3.1	Introduction	18
3.2	Problem Definition	20
3.2.1	IOP Decomposition	21
3.2.2	Minimum Rank Partitioning Problem	21
3.3	MILP Formulation	22
3.4	Proof of Proposition 3.1	26
3.5	Tour Generation	28
3.6	Simulation Results	28
3.6.1	Decomposition Comparisons	30
3.6.2	Rank Partitioning Comparisons	30
3.6.3	Coverage Tour Comparisons	31
3.6.4	Comparison against BCD	32
3.6.5	ROS Simulation Case Study	33
3.7	Summary	34
4	Coverage Replanning for Partially Unknown Environments	35
4.1	Introduction	35
4.1.1	Organization	37
4.2	Problem Definition	37
4.2.1	Coverage Replanning Problem	37

4.2.2	Rank-based Coverage Replanning	38
4.3	Solution Framework	39
4.3.1	The Effect of Changing Ranks	39
4.3.2	Algorithm	40
4.4	The RANK-REPLAN Subroutine	42
4.4.1	The Base ILP	43
4.4.2	Constraining New Ranks	45
4.4.3	On Reducing ILP Variables	50
4.5	The TOUR-REPLAN Subroutine	54
4.6	Simulation Results	55
4.6.1	Simulation Setup	57
4.6.2	Comparisons with Online Approaches	59
4.6.3	ILP Runtime Comparisons	63
4.6.4	Comparisons with Offline Approaches	64
4.6.5	Simulation Case Study	66
4.7	Robot Experiments	66
4.8	Summary	69
5	CPP with Approximation Guarantees	70
5.1	Introduction	70
5.2	Problem Definition	72
5.2.1	Rank-touring Framework	72
5.2.2	Rank-based Coverage with Minimum Length	72
5.3	Approximation Factor for Rank-Touring	73
5.4	A $(1.5 + \epsilon)$ -Approximation Algorithm	76
5.4.1	General Framework	76
5.4.2	Sub-tour Merging	78
5.4.3	Approximation on Path Length	80

5.4.4	Turn Minimization	81
5.5	MLC-Approx with Lazy Evaluations	81
5.6	Simulation Results	83
5.6.1	Path Length Comparisons	84
5.6.2	Turns Comparisons	85
5.6.3	Runtime Improvements with Lazy Evaluations	86
5.7	Summary	87
6	Multi-Directional CPP with Approximate Decompositions	88
6.1	Introduction	88
6.2	Problem Definition	90
6.3	Link to Submodular Set Cover	91
6.3.1	Coverage Path Optimality	92
6.3.2	Submodularity of Sector Coverage	94
6.4	Sector Decomposition Approach	95
6.4.1	Sector Identification	96
6.4.2	Greedy Decomposition	96
6.4.3	Analysis	97
6.4.4	Local Sector Merges	98
6.5	Sector Touring	99
6.6	Simulation Results	100
6.7	Summary	103
7	Conclusions and Future Work	104
7.1	Future Work	106
	References	108

List of Figures

1.1	Applications of Coverage Planning.	2
1.2	Visualizing missed area coverage by a large industrial cleaning robot while turning. In addition, turns take time to execute, so CPP approaches aim to minimize robot turning.	3
2.1	Illustrations of typical CPP approaches applied to an example environment.	9
3.1	(a) An example environment and (b) its optimal rank partitioning obtained using the proposed OARP method: the orange partitions are horizontal ranks and the purple are vertical ranks.	19
3.2	An Integral Orthogonal Polygon with light squares representing the polygon and grey areas representing the boundaries and holes. Each grid cell is as wide as the coverage tool.	20
3.3	Illustration of merging oriented grid cells to form ranks and the corresponding coverage lines. The cell orientations are determined by the MILP from Section 3.3, and the merge-able cells are merged.	22
3.4	The directed graphs G_H and G_V for an example IOP.	25
3.5	Comparison of our rank partitioning method with that proposed in [88]: (a) Cell decomposition runtime for each map. (b) The number of ranks determined for each test environment.	29
3.6	Coverage path of an indoor environment generated using the proposed LP method.	30
3.7	Average % improvement exhibited by the OARP method in comparison with [88] for each map.	32

3.8	Comparison results of the coverage tour generated using OARP and the BCD method [9]. The x-axis specifies the number of vertices representing the holes in the maps.	33
3.9	Simulation results of covering an example environment using the Avidbots Neo robot. The coverage planners used were (a) OARP and (b) the heuristic method from [88].	34
4.1	Replanning coverage paths in partially unknown environments. (a) An initial coverage path for the known base environment, (b) the robot (blue triangle) covers part of the environment (gray region) and observes a new obstacle (red box) interrupting the initial path (red path), and (c) a near-optimal replanned path (green path) computed online using the proposed approach (OARP-Replan).	36
4.2	The effect of an obstacle (black box) on replanning a coverage path consisting of ranks (black lines) and transition paths (green lines). The ranks of the initial path (left) when interrupted can be replanned in many ways, two of which are shown here. Example 1 has fewer new ranks (red lines in dotted box) than example 2 and is preferable for shorter planning time. Example 2 has fewer total ranks and is preferable for shorter drive time.	39
4.3	A run of OARP-Replan with one obstacle blocking a set of grid cells in the IOP (dotted grey box). The visible part of the obstacle (pink dots) is used to identify blocked grid cells (grey) and obstacle encounters (blue dots) along the current path. For each encounter $i \leq k$, the planner attempts to replan the remaining path (red path) within the time τ_i to reach the encounter (black path). To compute τ_i , previous encounters are replanned using GD replan (green dotted lines).	41
4.4	Integral orthogonal polygon (IOP) of an environment where each grid cell is either oriented horizontally (orange) or vertically (purple). The left, right, top, and bottom rank endpoints are also shown (see legend). Each cell c_i (zoomed-in cell) has four endpoint indicator variables. If the cell is an endpoint (e.g. c_i is a right endpoint), the corresponding variable is active ($y_R^i = 1$).	43
4.5	The directed graphs G_L, G_R, G_T , and G_B for the example IOP.	44

4.6	The matching edges used to detect ranks for a given row of grid cells. Each cell c_i has edges that match it with another cell in the row if c_i is a right (<i>using magenta edges</i>) or left (<i>using cyan edges</i>) endpoint. Here, we illustrate an example where c_i is a left endpoint and is matched with a right endpoint in the row (bold cyan edge). No matching occurs if c_i is not an endpoint. .	46
4.7	Illustrations of the cases considered for Lemma 4.1's proof.	48
4.8	Illustration of rank endpoint changes as a result of rank replanning. Each row shows horizontal (orange lines) and vertical (purple lines) ranks blocked by obstacles (black boxes) and the corresponding replanned ranks and endpoint changes.	53
4.9	All possible changes to an example initial tour of only horizontal ranks (orange circles). The initial tour requires replanning if the ranks (i) switch orientations to form vertical ranks (purple circles), (ii) are split by obstacles, and (iii) become unreachable due to obstacles (black circle). The resulting GTSP sets are shown consisting of (i) path sets (yellow), (ii) rank sets for each new replanned rank, and (iii) artificial vertices representing the start and end of the initial path, with a directed zero-cost edge connecting them.	56
4.10	Data used to estimate TOUR-REPLAN runtime, where we plot the number of ranks (x-axis) and the time taken to compute all transitions and replan GTSP (y-axis). The red line shows the cubic polynomial fit to the data.	58
4.11	Comparison of the online replanning approaches using the fixed clutter dataset: (a) Difference between total coverage time and base coverage time. (b) Minimum stop time of the robot during coverage (GD and SB omitted as replanning time is negligible). The line shows the average and the lighter region shows the data interval.	60
4.12	Qualitative results of replanning an initial coverage path for the example environment with unknown obstacles.	61
4.13	Comparison of the online replanning approaches using the varying clutter dataset: (a) Difference between total coverage time and base coverage time. (b) Minimum stop time of the robot during coverage (GD and SB omitted as replanning time is negligible). The line shows the average and the lighter region shows the data interval.	62

4.14	Comparison of average ILP runtimes for the OARP-Replan approaches. (a) Fixed clutter dataset comparisons, with maps sorted by <i>base map area</i> . (b) Varying clutter dataset comparisons. The line shows the average and the lighter region shows the data interval.	64
4.15	Comparison results with offline planning approaches. (a) Coverage time comparisons between OARP-Replan and offline baselines. (b) Violin plot of percent coverage achieved by the OARP-based approaches (with data points plotted). BCD-Offline and LC-Offline are omitted as they are exact approaches that achieve 100% coverage.	65
4.16	Robot operating system (ROS) simulation of covering an example environment with unknown obstacles. (a) An initial coverage path for the base environment (b) replanning the path online using OARP-Replan.	67
4.17	Snapshots from experiments using the Avidbots Neo robot	68
4.18	Robot experiment results of replanning an initial coverage path using OARP-Replan. The final robot path (pink curves) and the covered area (blue) are shown for both scenarios.	68
5.1	An IOP where a rank-touring algorithm (left) computes a path of length $5/3$ times the optimal coverage path length (right).	74
5.2	MLC-Approx stages: (a) obtain coverage lines (orange) from the minimum rank partition and compute min-length perfect matching paths (green), and (b) identify the resulting sub-tours (colored paths) and evaluate possible merges (black lines) as determined by an ordering (dotted green lines). . .	76
5.3	Corner junction merging: (a) Corner junction formed by 4 grid cells. (b) Edge swap to merge two sub-tours at the junction. (c & d) Modifying sub-tour S to allow an edge swap by removing the red edges and adding the green edges.	78
5.4	Path length comparisons. (a) Average path lengths for each map. (b) % length difference from TSP across all maps.	82
5.5	Coverage paths for an example environment generated using (a) OARP, (b) TSP, and (c) the proposed MLC-Approx.	83
5.6	Turn comparisons. (a) Number of turns (bold - fewest turns). (b) Coverage path straightness as denoted by the number of turns per 100 meters of path length.	86

5.7	Runtime comparisons. (a) Number of edge evaluations (i.e., obstacle-free paths planned) by MLC-Approx with/without lazy. (b) Runtimes for each map (red arrow - exceeds 5000s).	87
6.1	Coverage path resulting from a sector decomposition containing three sectors. Each sector is covered by a lawnmower path and the robot transitions between sectors using (i) intra-sector paths, and (ii) inter-sector paths. . .	91
6.2	G-Sect results with varying sector overlap and merging.	101
6.3	Sector decomposition results for the test maps.	102

List of Tables

3.1	Robot parameters used for experiments	31
4.1	Robot parameters used for experiments	57
5.1	Statistics for dataset maps.	82
6.1	Robot parameters used for experiments	99
6.2	Comparison of number of sectors in each decomposition	101
6.3	Comparison of coverage planning performance for the test maps.	101

Chapter 1

Introduction

An increasingly compelling application of modern robotics is to automate repetitive tasks that are mundane and dangerous to human beings. This includes tasks such as cleaning [44] and lawn mowing [78], where the task simply involves a human or robot sweeping a tool (for cleaning or grass cutting) over all points in a given environment. Similar tasks include underwater inspections [71, 36] and GIS surveys [73], where the robot instead carries a sensor such that all points in the environment are fully sensed. To automate such tasks using robots, a path must be generated for the robot to follow such that its tool or sensor covers the area completely.

Coverage path planning (CPP) is the automation problem in which a robot must find a path such that its tool or sensor covers the entire environment [37]. Including the above applications, this problem is also prevalent in agricultural robots [39], visual inspection of 3D structures [82, 49, 10], and more recently, autonomous disinfection of hospitals during the COVID-19 pandemic [66]. Fig. 1.1 illustrates some applications of robots where CPP is used. While the requirement of these paths is to cover an environment (or structure) with the robot's tool, a desirable coverage path typically aims to minimize a cost associated with the path, such as coverage time, path length, number of turns, etc.

For a fully known environment with obstacles, CPP is proven to be an NP-Hard problem [8, 7]. Therefore, computing optimal solutions for the CPP problem does not scale well for large and complex environments with obstacles. Moreover, in real-world coverage scenarios, the robot may encounter new obstacles that it was not previously aware of, which increases the problem difficulty. In this case, the robot may need to either plan a new coverage path *on-line* (during coverage) that avoids newly detected obstacles or replan the coverage path that it is currently following. Additionally, if the goal is to minimize coverage time, the

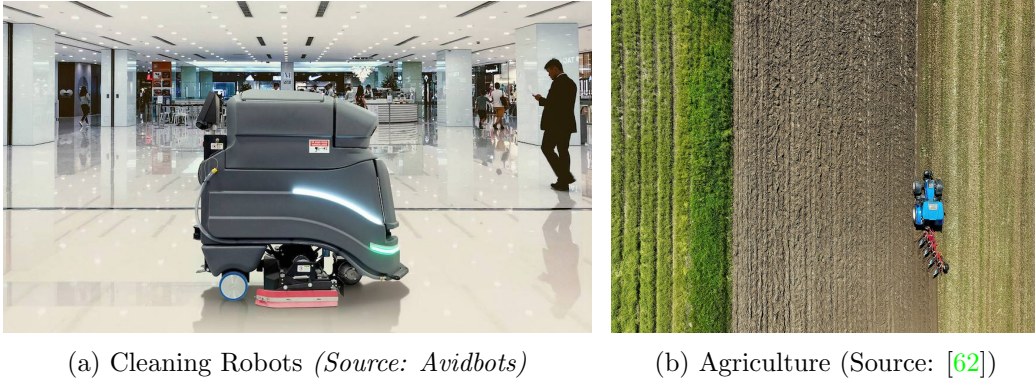


Figure 1.1: Applications of Coverage Planning.

CPP algorithm must be able to compute a new path for the robot before reaching the obstacle to avoid stoppages caused by the robot not having any safe paths to continue coverage.

In this thesis, we focus on coverage planning in environments that may contain unknown static obstacles. The high-level idea proposed in this thesis involves the following two steps: (i) generate a coverage path for a base environment with known static obstacles, and (ii) replan the initial path upon the detection of new static obstacles. Since one can usually obtain base maps for indoor environments using existing floor plans or modern simultaneous localization and mapping (SLAM) techniques, this framework aims to minimize total coverage cost by starting with a near-optimal initial path to cover the base environment and performing replans that minimize the added path cost to account for newly discovered obstacles.

The first problem we will tackle in this thesis is CPP for *known* environments with non-convex boundaries and obstacles. To minimize path overlap (or “double-coverage”), such environments are typically covered through a lawnmower-style path consisting of parallel *coverage lines*, which are non-overlapping straight-line paths. Thus, the main differentiation between coverage plans for an environment is the transitions between these lines, where the robot performs little to no additional coverage. This has resulted in a growing body of work focused on *minimizing turns*, i.e., minimizing the time spent transitioning between coverage lines [46, 11, 9]. Turns also have the following adverse effects: (i) the robot travels slower around turns which increases total coverage time, (ii) in cleaning applications, the tool does not properly pick up water and dust while turning (e.g., Fig. 1.2), and (iii) in sample retrieval applications, the robots may experience high pose estimation errors and poor sensor coverage quality during turns [28]. We add to this body of

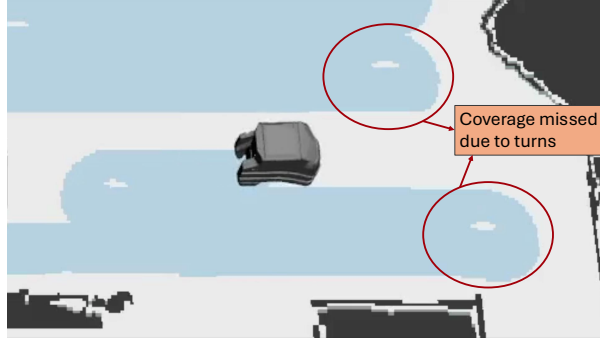


Figure 1.2: Visualizing missed area coverage by a large industrial cleaning robot while turning. In addition, turns take time to execute, so CPP approaches aim to minimize robot turning.

work by proposing a state-of-the-art CPP method that generates a minimum-turn coverage path for a known environment. The key to our approach is that we efficiently compute a partitioning of the environment that guarantees a coverage path with the minimum number of axis-parallel (horizontal/vertical) coverage lines. Minimizing the number of lines typically reduces the number of turns in the coverage path.

The subsequent topic we address in this thesis is replanning the coverage path generated by the above approach when new static obstacles are discovered in the environment by the robot’s sensors. We refer to this as the *coverage replanning* problem (CRP). To solve CRP, we present an anytime (i.e., given a time budget) replanning algorithm that uses the time available to reach the new obstacle along the current coverage path to replan as much of the path as possible. The replanning algorithm extends the axis-parallel partitioning step from above, where a time budget is incorporated to control any changes to the partitioning during replanning.

In the rest of the thesis, we address two possible improvements to the above replanning framework, where we (i) compute paths with approximation guarantees, and (ii) allow multiple coverage orientations (not just axis-parallel). We first present an approach that builds upon on the above axis-parallel partitioning-based framework to plan minimum-length coverage paths with approximation guarantees. In practice, the proposed algorithm generates paths with near-optimal lengths. Moreover, the algorithm allows paths to be computed *lazily*, i.e., with incomplete information about paths in the environment. This lazy variation offers computation speed-ups that will be useful for replanning. Finally, we address the removal of the axis-parallel constraints imposed by all of the previous methods, where we compute decompositions of the environment that allow for multi-directional coverage paths. With this approach, we also provide theoretical guarantees on the computed

decomposition and demonstrate the performance improvements in simulations, specifically for complex environments with non-orthogonal boundaries.

1.1 Thesis Contributions

The specific contributions of this thesis are the following.

- We propose a method to compute an optimal axis-parallel (horizontal/vertical) partitioning of a non-convex environment with obstacles to generate turn-minimizing coverage paths. We use this method to propose a minimum-turn CPP approach that outperforms the state-of-the-art approach of the same class.
- We then extend the axis-parallel partitioning approach to replan coverage paths for partially unknown static environments, where not all obstacles in the environment are known a-priori.
- We analyse the path lengths computed by the partitioning-based CPP approach and derive an approximation factor for the approach. We then propose a lazy approach that improves upon this approximation factor to generate minimum-length coverage paths efficiently.
- Finally, we propose an approximation algorithm to decompose the environment into the minimum number of rectangular sectors, i.e., sub-regions that can each be covered along parallel straight line paths. We use this decomposition to remove the axis-parallel constraint of the previous algorithms and compute multi-directional coverage paths in the environment.

1.2 Organization

We now review the contributions of each chapter in more detail.

Chapter 2: We start by introducing the coverage path planning (CPP) problem and provide a review of CPP literature. We also review the core concepts that we used to design the algorithms proposed in this thesis.

Chapter 3: In this chapter, we present the coverage path planning (CPP) problem. We introduce a turn-minimizing CPP approach named Optimal Axis-Parallel Rank Partitioning (OARP) for general non-convex environments. The proposed approach uses the key

idea that one can compute an optimal axis-parallel (horizontal/vertical) partitioning of the environment using a linear program (LP) that can be solved in polynomial time. The chapter details the LP-based partitioning approach and how to compute a coverage path from the partitioning, which we validate in simulations and comparisons against state-of-the-art CPP approaches.

The work presented in this chapter is based on the following publication:

- Megnath Ramesh, Frank Imeson, Baris Fidan, and Stephen L. Smith. Optimal Partitioning of Non-Convex Environments for Minimum Turn Coverage Planning. *IEEE Robotics and Automation Letters*, 7(4):9731–9738, October 2022 [74].

Chapter 4: In this chapter, we extend the OARP approach proposed in the previous chapter to be applied to static environments with unknown obstacles. We propose a framework that takes an initial coverage path that was planned using a base environment (i.e., with known obstacles and boundaries) and *replans* the path when new obstacles are detected in the environment. The proposed replanning approach aims to balance minimizing the number of turns and the computation time required to replan the coverage path.

The work presented in this chapter is based on the following publication:

- Megnath Ramesh, Frank Imeson, Baris Fidan, and Stephen L. Smith. Anytime Replanning of Robot Coverage Paths for Partially Unknown Environments. *IEEE Transactions on Robotics*, 40:4190–4206, 2024 [75].

The following chapters address two aspects of improvement for the OARP approach: (i) computing coverage paths with approximation guarantees, and (ii) removing the axis-parallel constraints on the path.

Chapter 5: In this chapter, we discuss planning coverage paths with approximation guarantees on the path length, where the number of turns is considered as a secondary objective. We start by analyzing the path lengths computed by OARP and obtain an approximation factor for the approach. We then propose an algorithm that improves upon this factor and computes coverage paths with near-optimal lengths. Moreover, we show that these coverage paths can be computed *lazily* without losing the approximation guarantee while reducing the empirical runtime in practice.

The work presented in this chapter is based on the following publication:

- Megnath Ramesh, Frank Imeson, Baris Fidan, and Stephen L. Smith. Minimum-Length Coverage Path Planning for Grid Environments with Approximation Guarantees. IEEE Robotics and Automation Letters, 2025 (To Appear).

Chapter 6: In previous chapters, we discussed CPP approaches where coverage of the environment is conducted along mostly axis-parallel (horizontal/vertical) paths. This chapter proposes an approach that removes the axis-parallel constraint and plans coverage paths along multiple orientations. The approach decomposes the environment into the minimum number of rectangular *sectors* (i.e., possibly overlapping sub-regions) that can each be covered optimally with straight-line paths parallel to the longer side of the rectangle. We show that this decomposition problem is analogous to a well-studied optimization problem in the literature, for which the greedy algorithm provides a good approximation guarantee on the solution. Using this link, we propose an algorithm to solve the decomposition problem with optimality guarantees. In simulation, we observe improved coverage paths for complex non-orthogonal environments compared to CPP approaches in the literature.

The work presented in this chapter is based on the following publication:

- Megnath Ramesh, Frank Imeson, Baris Fidan, and Stephen L. Smith. Approximate Environment Decompositions for Robot Coverage Planning using Submodular Set Cover. In IEEE Conference on Decision and Control, pages 7528–7533, December 2024 [76].

Chapter 7: This chapter concludes the thesis by providing a summary of the work presented and potential directions for future work.

Chapter 2

Literature Review and Background

2.1 The Coverage Path Planning (CPP) Problem

We start by introducing the CPP problem we focus on in this thesis. Consider a 2D (possibly non-convex) environment with known obstacles, where $\mathcal{W} \subseteq \mathbb{R}^2$ is a closed and bounded set that represents all free points within the environment. We now consider a robot with a state composed of its position $\mathbf{x} \in \mathcal{W}$, where $\mathbf{x}$ is a vector in $\mathbb{R}^2$, and a heading $\theta \in [0, 2\pi)$ radians. The robot is carrying a coverage tool with a footprint given by $\mathcal{A} \subseteq \mathbb{R}^2$. Conventionally, the coverage tool is represented by a simple geometric shape such as a square of a fixed width $l > 0$ [37, 7, 32, 88].

Assumption 2.1. *The coverage tool is a square with width $l > 0$.*

Let $\mathcal{A}(\mathbf{x}, \theta) \subseteq \mathcal{W}$ represent the placement of the tool with respect to the robot's state. We consider the square coverage tool to be centered at $\mathbf{x}$ and oriented along θ . In reality, the square tool may not be able to cover the entire environment, so we look to cover an approximation of the environment $\widetilde{\mathcal{W}} \subseteq \mathcal{W}$ that the tool can reach. This includes grid approximations of $\mathcal{W}$ containing square grid cells of width l . Our goal is to compute a cost-minimizing coverage path for $\widetilde{\mathcal{W}}$. We consider the coverage path to be *closed*, i.e., starts and ends at the same point. This enables the robot to return to its initial position after covering the environment. Let $\mathcal{P}$ be the class of all closed paths in $\widetilde{\mathcal{W}}$, where each $P \in \mathcal{P}$ is a trajectory passing through a sequence of robot states $(\mathbf{x}, \theta)$. We now formally define the coverage path planning (CPP) problem of interest as follows.

Problem 2.1 (Coverage Path Planning (CPP)). *Given an approximation $\widetilde{\mathcal{W}}$ of a 2D environment $\mathcal{W}$ and a robot carrying a coverage tool of footprint $\mathcal{A}$, plan a path P from the set $\mathcal{P}$ of all possible paths which solves*

$$\min_{P \in \mathcal{P}} J(P) \tag{2.1}$$

$$s.t. \quad \bigcup_{(\mathbf{x}, \theta) \in P} \mathcal{A}(\mathbf{x}, \theta) = \widetilde{\mathcal{W}}, \tag{2.2}$$

where $J(P)$ is the cost of the path P .

In this thesis, we typically consider the path cost $J(P)$ to be the time for the robot to cover the given environment along P , which is determined by both the path length and the turns present in the path. In Chapter 5, we study a slight variation of this problem where path length is our primary objective, but the number of turns is minimized as a secondary objective. We use this variation to derive an approach with guarantees on the resulting coverage path length.

2.2 Literature Review

The CPP problem has been studied extensively in the robotics literature, with varying CPP approaches proposed for a wide range of robotic setups. Many detailed surveys of CPP approaches can be found in the literature [21, 37, 12, 85]. For the general class of environments, CPP is proven to be an NP-Hard problem [8, 7]. As such, the CPP problem must be solved using a solution approach that exploits properties of the environment to plan the coverage path. Therefore, an important component of the CPP problem is the environment set $\mathcal{W}$. Based on prior knowledge of $\mathcal{W}$, CPP methods can be broadly classified into *offline* and *online approaches*.

- *Offline CPP*: If $\mathcal{W}$ is known, a CPP algorithm may solve for the coverage path in an *offline* manner, i.e., planning before the path execution starts. Offline planning approaches may sometimes require computation time, but provide some guarantees on the complete coverage of the environment. Some approaches even have guarantees on the quality of the solution.
- *Online CPP*: However, in many applications, the environment is either partially known or completely unknown. In such cases, CPP must be done in a *online* manner,

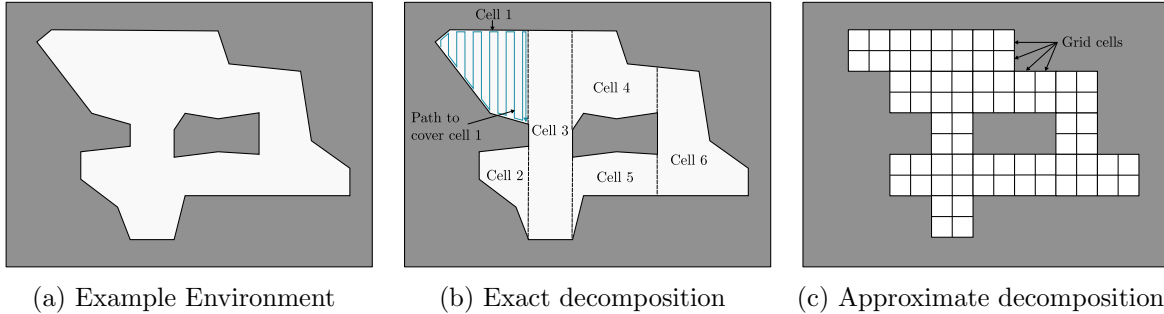


Figure 2.1: Illustrations of typical CPP approaches applied to an example environment.

i.e., path planning/replanning during coverage. In some cases, the environment is generally known, but may contain dynamic (moving) obstacles, in which case the coverage path must also be computed/repaired online.

2.2.1 Offline CPP

In the offline CPP problem, we assume that the environment is known and undergoes no changes. Given this, one looks to compute a coverage path that covers the environment. For the general class of environments, offline CPP is an NP-hard problem [8] and thus research is primarily on computing approximations to the optimal coverage path using a variety of solution frameworks. We first briefly discuss the typical solution frameworks for offline CPP.

Offline CPP is a well-studied robotics problem. Early works in offline CPP involve using simple heuristics and maneuvers to cover the environment, albeit with minimal guarantees of completely covering the environment (i.e., *complete coverage*) [15, 35, 44]. In [70], a simple algorithm using random cleaning sweeps is proposed for a robot to cover its environment. The algorithm is as follows: the robot simply moves straight until it reaches an obstacle, after which it moves back a small amount, turns at a random angle, and repeats the process. This approach guarantees complete coverage of simple environments given enough execution time, as the robot is likely to cover new areas with every sweep. However, when presented with environmental challenges, this random walk approach would require significant execution time to complete the coverage of the environment.

To ensure complete coverage of the environment, modern CPP approaches typically follow a *decomposition-based* algorithmic framework, where (i) the environment is decomposed into smaller sub-regions, (ii) an optimal coverage path for each sub-region is computed in-

dividually, and (iii) a tour of the sub-region paths is computed to form the coverage path. Based on the *type* of decomposition used by the approach, the decomposition-based approaches can be further categorized into 2 sub-classes: (i) *exact decomposition* approaches, and (ii) *approximate decomposition* approaches.

Exact Decomposition Approaches: In exact decomposition approaches, the environment is decomposed into sub-regions called *cells*, where the union of the cells gives back the environment. In other words, the decomposition does not omit any part of the environment. As such, covering each cell guarantees complete coverage of the environment. The cells are usually non-overlapping and convex, but this is not always the case. A popular approach of this class, namely the *boustrophedon approach*, computes an exact decomposition of the environment where the cells are not necessarily convex [22]. Each cell is covered by a path that either sweeps the cell (i) back-and-forth along coverage parallel lines [22] (illustrated in Fig. 2.1(b)), (ii) along spiral paths [17], or (iii) a combination of both [16]. There are also turn-minimizing approaches that cover each cell in a locally optimal direction, generally determined using the cell’s geometry [46, 9, 28]. Huang [46] studies an approach that computes a minimum-turn coverage path for each cell in a convex decomposition by using the cell’s *minimum altitude*, i.e., orienting the cell where its “height” is shortest.

As exact decompositions do not omit any region in the environment, they typically guarantee complete coverage of the environment. However, there are a few drawbacks that stem from this guarantee. For environments with complex boundaries, specifically cluttered environments, the decomposition is likely to contain many narrow cells that are not as wide as the coverage tool. We refer to this as an *over-decomposition*, which typically results in instances of redundant coverage. For cluttered environments, computing these cells may require time and cause unnecessarily overlapping coverage paths.

Approximate Decomposition Approaches: Approximate decompositions are environment representations where the union of the cells does not give the exact environment. A grid decomposition of an environment is an example, as illustrated in Fig. 2.1(c). Grid-based CPP approaches use an approximate representation of the environment as a set of (usually square) *grid cells* and compute a path that covers all of them. This path can be computed either by solving the traveling salesman problem (TSP) [12], using a wave-front algorithm to determine visitation order [93], or using a spanning tree of the grid cells (Spanning Tree Coverage or STC) [32]. STC is a widely studied approach for CPP in grid approximations, where a spanning tree generated in the grid is used to plan the coverage path. Many variations of STC have been proposed in later studies [33, 34, 87, 57]. Turn-minimizing grid-based CPP approaches additionally look to minimize the number of axis-parallel (horizontal or vertical) coverage lines in the coverage path [88, 57].

Grid-based approaches have also been introduced with approximation guarantees on the resulting coverage path length [8] and turns [7, 30]. In fact, given a square coverage tool of width l , the STC approach is shown to provide coverage paths with optimal lengths for coarse grids with cells of width $2l$ each [32]. However, a similar guarantee is difficult to achieve for general grid environments. Arkin et. al. proposed a $6/5$ approximation algorithm for minimum-length coverage planning, but it does not aim to minimize the number of turns in the path [8]. Approximation algorithms for minimum-turn coverage planning typically result in longer paths [7, 30]. Later in this thesis, we propose an approximation algorithm to plan minimum-length coverage paths, while also aiming to minimize the number of turns in the path that achieve comparable performance to turn-minimizing approximate approaches [88, 9].

2.2.2 Online CPP

We now shift our focus to online coverage planning, where the CPP approach either assumes partial or no knowledge of the environment and plans the coverage path *during* the coverage operation. To detect unknown obstacles in the environment, the coverage robots carry a sensor to observe the environment online. In sensor coverage applications, the same sensor may be used for obstacle detection and coverage. Complete coverage of such environments is difficult to guarantee, but online CPP approaches aim to maximize area coverage given the current observed state of the environment.

Online CPP approaches have gained recent interest, and as such, there is a wide range of different approaches in the literature. Similar to offline CPP, decomposition-based approaches are typically used for online CPP, which are further classified into (i) exact decomposition and (ii) approximate decomposition approaches. Apart from decomposition-based approaches, we also review decomposition-free online approaches that have gained recent popularity.

Exact Decomposition Approaches: Early approaches in online CPP aim to compute exact decompositions of the environment during coverage. Acar et. al. [2] propose an approach that incrementally decomposes the environment into *cells* by detecting connectivity changes in the environment (e.g., areas split by obstacles). Each cell in the decomposition can be covered using a boustrophedon (back-and-forth sweep) path. However, this approach assumes access to a short-range occupancy sensor to detect connectivity changes while executing a boustrophedon coverage path [1]. For long-range occupancy sensors, a modification of this method was proposed by [3]. Here, the coverage planner constructs a generalized Voronoi diagram (GVD) of the environment during coverage planning. The GVD is used to generate a hierarchical decomposition of an environment into

(i) vast regions, which are covered using boustrophedon paths, and (ii) narrow regions, which are covered by following the GVD edges. Online exact decomposition approaches guarantee complete coverage for unknown static environments, but also suffer from over-decomposition, similar to analogous offline approaches, resulting in long coverage paths with many turns.

Approximate Decomposition Approaches: Grid-based methods extend well for online CPP as one can easily update a grid with new occupancy information. A popular approach of this class is online spanning tree coverage, i.e., Online-STC [34, 33]. This approach is similar to the offline STC approach introduced in the previous section [32], except that the spanning tree of the grid cells is incrementally constructed online. With new obstacle-free grid cells detected, the spanning tree and, by extension, the coverage path are changed to cover the new region. However, given a square coverage tool of width l , online-STC methods are generally applied to coarse grids with cells of width $2l$ and do not generalize well to finer grids with cell width l .

Other grid-based online approaches use various local search and optimization methods to determine the coverage plan in an unknown environment. Song et. al. [81] propose an approach using artificial potential fields (APF) to determine a coverage path online. The APF is used to assign a value to each grid cell that indicates whether the cell is covered, uncovered, or occupied. Depending on the value, the robot decides which neighboring grid cell to cover next. A similar technique to APFs is using bio-inspired neural networks (BINNs) to model and search the environment. This was originally proposed by Luo et. al. [58] and has gained popularity in many recent works [65, 84, 38]. However, these local search-based approaches are susceptible to dead-locks, where the robot enters a failure state where it either does not know where to navigate to, or is in a fatal state where the robot cannot continue coverage autonomously. Dead-locks are expensive to resolve and affect overall coverage time.

To plan minimum-turn coverage paths online, the above approaches can be modified to penalize turns in a local search cost function. However, other grid-based approaches can be used to guarantee minimum turns in the path overall [75, 51]. These approaches usually involve solving an optimization problem defined on the grid graph. In [50, 51], a hexagonal grid is used to plan turn-minimizing coverage paths online for non-holonomic robots.

Decomposition-free Approaches: Online coverage planning shares similarities with exploration planning, which involves covering unknown environments using sensors [26]. Exploration planning algorithms use information theory [24, 45] and frontiers [92] to determine the next-best scan of the environment, rather than using decompositions.

A popular approach of this class, namely Boustrophedon A* (BA*) was proposed by [90]. Here, the robot starts by executing a single boustrophedon coverage path to cover a part of the environment, without constructing an exact decomposition. While executing this path, the robot’s sensor is used to detect points in the environment that the current boustrophedon path cannot reach. For example, an obstacle in the environment may force the robot to turn before reaching this point. These points are referred to as *backtracking* points, as they are starting points for a second boustrophedon path that can be executed after the first. From the set of all backtracking points detected during coverage, the A* algorithm is used to pick the best starting point for the next boustrophedon path. Further variations of this approach have been proposed in later works [89, 20]. Recently, a decomposition-free online coverage planner was proposed that uses frontiers and search-based planning to identify the next-best region to cover [56].

2.2.3 Multi-robot CPP

Each of the above-discussed approaches is predominantly used for *single-robot* CPP. For multi-robot swarms, variants of the above CPP approaches enable the coverage tasks to be split between the swarm, thereby improving the overall coverage time [79, 52, 49, 63, 50, 53]. Multi-robot CPP however also has some additional planning challenges associated with robot coordination and communication. An example is to prevent overlaps between the coverage paths of different robots in the swarms. Many recent works have been proposed to plan multi-robot CPP paths with these considerations [18, 64, 14, 13, 86]. However, these approaches typically do not consider robot turns or provide approximation guarantees on the resulting path.

In this thesis, we focus on the single-robot CPP case, as a large number of industrial and domestic applications still utilize single-robot systems. For example, our industry partner, Avidbots, utilizes a single cleaning robot to cover large environments such as departmental stores and warehouses. Moreover, the approaches proposed in this thesis offer foundations to be used in the development of multi-robot coverage planning and replanning approaches that consider turns and provide guarantees. We discuss this further as future work in Chapter 7.

2.3 Preliminaries

2.3.1 Graphs

A graph $G = (V, E)$ is defined as a tuple of two sets: a set V of vertices, and a set $E = \{(u, v) \mid u, v \in V\}$ of edges connecting pairs of vertices. A directed graph G is one where each edge (u, v) in the graph has a direction. More specifically, the edge (u, v) is an ordered tuple indicating flow from vertex u to v . In contrast, an undirected graph has edges (u, v) which are unordered tuples that each indicate flow between vertices u and v in any direction (u to v , or v to u).

In Chapters 3 and 4, we propose CPP approaches that use directed graphs derived from a square grid representation of an environment. Specifically, we use the *node-arc incidence* matrix derived from these graphs.

Definition 2.1 (Node-Arc Incidence (NAI) Matrix). *The node-arc incidence matrix B of a directed graph G is a $|V| \times |E|$ matrix, where v_i is the vertex corresponding to row i , e_j is the edge corresponding to column j , and*

$$B_{ij} = \begin{cases} -1 & \text{if } e_j \text{ leaves vertex } v_i \\ 1 & \text{if } e_j \text{ enters vertex } v_i \\ 0 & \text{otherwise} \end{cases}.$$

An important property of NAI matrices is total unimodularity.

Definition 2.2 (Total Unimodularity (TU) Matrix). *A matrix B is said to be a totally unimodular (TU) matrix if every square sub-matrix of B has determinant $d \in \{0, 1, -1\}$.*

We use the TU property of NAI matrices in chapters 3 and 4 to show that we can compute partitions of the environment efficiently.

2.3.2 Matching Problems in Graphs

Given an undirected graph $G = (V, E)$, a matching $M \subseteq E$ is a set of edges in the graph where no two edges in M share a common vertex. A vertex v is said to be matched if there exists an edge $e \in M$ that is incident upon v . We use concepts of vertex matching in Chapters 4 and 5.

Definition 2.3 (Perfect Matching). *A matching $M \subseteq E$ of a graph $G = (V, E)$ is a perfect matching where all vertices $v \in V$ are matched.*

A graph $G = (V, E)$ can only contain a perfect matching if it has an even number of vertices. The size of the resulting matching $|M| = |V|/2$. For other types of graphs, a *maximal matching* $|M| \leq |V|/2$ can be computed that maximizes $|M|$, but in this thesis, we will only deal with graphs that allow perfect matchings.

2.3.3 Traveling Salesman Problems

The coverage path planning problem is analogous to traveling salesman problems (TSP). We exploit this relationship in the algorithms proposed in this thesis.

Given an undirected complete (fully-connected) graph $G = (V, E)$, a cycle is a sequence of vertices $C = (v_1, v_2, \dots, v_k, v_1)$, where only v_1 is visited twice. A tour T is a cycle that visits all vertices $v \in V$ in the graph (except the start vertex). For each edge $(u, v) \in E$, where $u, v \in V$, let $J(u, v)$ be a cost associated with the edge. Given a tour T , let the cost of the tour $J(T) = \sum_{i=1}^{k-1} J(v_i, v_{i+1}) + J(v_k, v_1)$.

Definition 2.4 (Traveling Salesman Problem (TSP)). *Given an undirected complete graph $G = (V, E)$ and a cost function J as above, the traveling salesman problem is to compute a tour T that minimizes the cost $J(T)$.*

The generalized traveling salesman problem (GTSP) is a variant of TSP where the vertices V are partitioned into m sets $\{V_1, V_2, \dots, V_m\}$, i.e., $V_i \cap V_j = \emptyset$ and $\cup_{i=1}^m V_i = V$. The goal of GTSP is to compute a cycle C in G that visits exactly one vertex in each of the m sets and minimizes the cost $J(C)$. The GTSP problem shows up in almost all chapters of this thesis, with details provided in Chapters 3 and 4.

Metric-TSP: Both TSP and GTSP are NP-Hard problems, but effective heuristics have been proposed to efficiently compute good solutions in practice [40, 68]. However, we review a special class of graphs for which there exists an efficient algorithm to solve TSP with approximation guarantees. We use similar concepts to this algorithm in Chapter 5 to design an approximation algorithm for minimum-length CPP with turn cost considerations.

Consider a graph $G = (V, E)$, where each edge $(u, v) \in E$ has a cost $J(u, v)$. Additionally, we consider that the cost function has the following properties:

- (i) *Non-negativity:* $J(u, v) \geq 0$ for all $u, v \in V$,

(ii) *Symmetry*: $J(u, v) = J(v, u)$ for all $u, v \in V$

(iii) *Triangle inequality*: $J(u, v) + J(v, w) \geq J(u, w)$ for all $u, v, w \in V$.

A function $J(u, v)$ that satisfies the above properties is called a *metric function*, and we refer to G as a *metric graph*. The traveling salesman problem defined on G is called the *metric traveling salesman problem* or *metric-TSP*.

Given a complete metric graph $G = (V, E)$ as determined by the edge cost function J , Christofides [23] proposed a 1.5-approximation algorithm, which is briefly described here. The algorithm starts by computing a minimum spanning tree (MST) in G . Given the MST, let O be the set of vertices in the MST that have an odd degree. Since G always contains a tour (complete graph), O must have an even number of vertices. One can therefore compute a minimum-weight perfect matching M of the vertices in the sub-graph of G induced by the vertices in O . Combining the edges of the MST and M , we get a sub-graph G' of G that contains only even-degree vertices. Such a graph contains an Eulerian circuit.

Definition 2.5 (Eulerian circuit). *An Eulerian circuit in a graph G is a walk along a sequence of edges that visits each edge exactly once.*

The Eulerian circuit may revisit vertices in the graph, in which case a shortcut using other edges in G is found to compute a tour T . The length of the Eulerian circuit is 1.5 times the TSP tour cost in the worst case, thus giving rise to the approximation factor.

2.3.4 Submodularity

In this section, we provide a brief background on submodular functions and the submodular set cover (SSC) problem on which our decomposition approach is based. We use these concepts in chapter 6 to compute approximate environment decompositions for coverage paths along multiple orientations.

Submodular Functions: Let us consider a set X (not necessarily finite) and let the set 2^X be the power set of X , i.e., the set of all subsets of X . A function $f : 2^X \rightarrow \mathbb{R}_{\geq 0}$ is *submodular* if for all $A \subseteq B \subseteq X$ and $x \in X \setminus B$, we have

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B).$$

This is known as the property of diminishing returns. Equivalently, $f : 2^X \rightarrow \mathbb{R}_{\geq 0}$ is submodular if for every $A, B \subseteq X$,

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B).$$

In addition to submodularity, we consider functions f that possess the following additional properties:

1. *Monotonicity*: For all $A \subseteq B \subseteq X$, $f(A) \leq f(B)$,
2. *Normalization*: For the empty set $\emptyset$, $f(\emptyset) = 0$.

Submodular Set Cover (SSC): Consider a set X and a monotone submodular function $f : 2^X \rightarrow \mathbb{R}$ where $f(X)$ is finite. The submodular set cover problem aims to compute a minimum cardinality subset $S \subseteq X$ subject to a submodular constraint $f(S) = f(X)$, i.e.,

$$\min_{S \subseteq X} |S| \tag{2.3}$$

$$\text{s.t. } f(S) = f(X). \tag{2.4}$$

This problem is NP-Hard, however, a simple greedy algorithm provides the best-known performance guarantee for it [91]. For a given $S \subseteq X$ and $x \in X \setminus S$, let

$$\delta_x(S) = f(S \cup \{x\}) - f(S).$$

denote the marginal return obtained by adding x to S . The greedy algorithm for this problem iteratively constructs a set S as follows. We start with a set $S^0 = \emptyset$ (empty set). At each iteration r , the greedy algorithm adds the element $x \in X \setminus S$ that maximizes the marginal return $\delta_x(S)$. Let S^r be the set constructed at iteration r , i.e., $S^r = S^{r-1} \cup \{x\}$. The greedy algorithm terminates after a feasible solution S where $f(S) = f(X)$ is obtained. Following [91], this algorithm provides a solution S after T iterations that has the following approximation factor:

$$\frac{|S|}{|S^*|} \leq 1 + \ln \left(\frac{f(X) - f(\emptyset)}{f(X) - f(S^{T-1})} \right).$$

We apply this result in chapter 6 to provide a guarantee for the approximation environment decompositions we compute.

Chapter 3

Minimum-Turn CPP for Known Environments

In this chapter, we present a turn-minimizing coverage planning approach for a known non-convex environment with holes. We also present experimental results using real-world maps and compare against the state-of-the-art approaches.

3.1 Introduction

Coverage planning with minimum turns is an NP-Hard problem [7]. As such, a common solution framework is to simplify the problem into three steps:

- (i) *decompose* the environment into sub-regions or *cells*,
- (ii) use the cells to help *place coverage lines* (straight-line paths) in the environment, and
- (iii) construct the coverage path by computing a *tour* of the coverage lines.

For example, exact decomposition methods like [22] decompose the environments into convex cells, each of which is covered by a lawnmower-style path. Following this framework, the robot only turns when transitioning from one coverage line to another. As a result, the number of turns in the path is often reduced by minimizing the number of coverage lines.

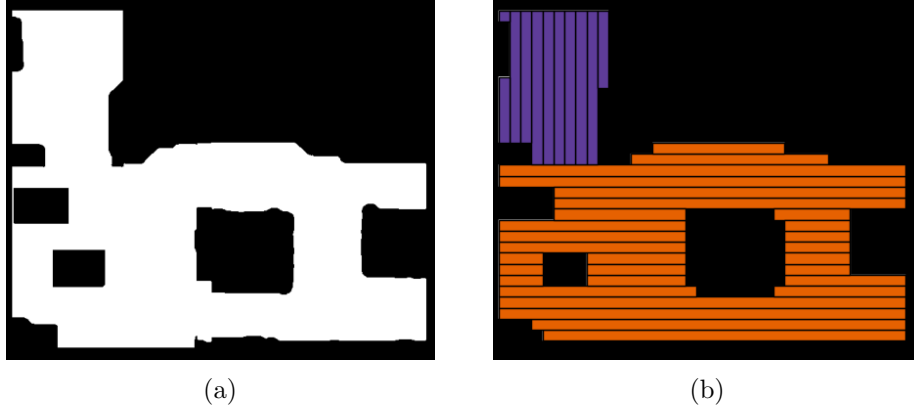


Figure 3.1: (a) An example environment and (b) its optimal rank partitioning obtained using the proposed OARP method: the orange partitions are horizontal ranks and the purple are vertical ranks.

A state-of-the-art method which follows this general framework is Vandermeulen et. al. [88], where they aim to cover the environment along a minimum number of axis-parallel (horizontal/vertical) coverage lines. Such a method is particularly well suited for indoor environments, in which many walls are either parallel or orthogonal. Their method decomposes the environment into cells, which are then used to partition the environment into *ranks*, i.e., thin disjoint rectangles of width equal to the robot’s tool width, where the center line of each rank along its length defines a coverage line. However, to solve the partitioning step, the authors propose an iterative heuristic with no optimality guarantees. In this chapter, we provide a coverage planner that follows the above framework and plans axis-parallel coverage paths with theoretical guarantees and with better performance.

Chapter contributions: The specific contributions of this chapter are as follows:

1. We prove that the axis-parallel rank partitioning problem is tractable and propose a polynomial-time solver that is guaranteed to find the optimal partition. We do this by posing the problem as a mixed integer linear program (MILP) and then proving that the linear relaxation provides optimal solutions.
2. We develop the **Optimal Axis-Parallel Rank Partitioning** (OARP) method that leverages the above solver and plans a coverage path for non-convex environments.
3. We present experimental results using maps of real-world environments and show that OARP outperforms the state-of-the-art method from [88] in minimizing the

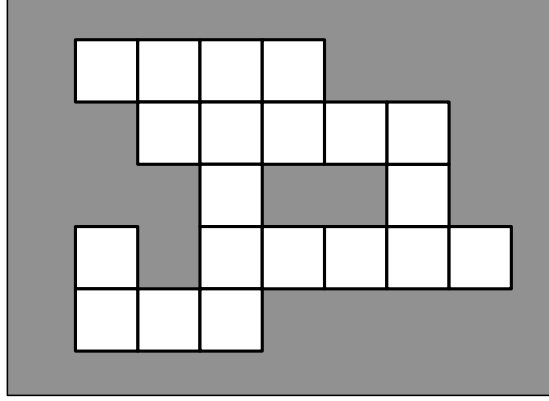


Figure 3.2: An Integral Orthogonal Polygon with light squares representing the polygon and grey areas representing the boundaries and holes. Each grid cell is as wide as the coverage tool.

decomposition time, number of ranks, number of turns, and consequently the cost of the coverage plan.

In Section 3.2, we define our decomposition approach and introduce the rank partitioning problem. In Section 3.3, we formulate this problem as a Mixed Integer Linear Program (MILP) and then prove in Section 3.4 that this problem can be relaxed to an LP and optimally solved in polynomial time. In Section 3.5, we briefly discuss how we compute a tour of the ranks using a Generalized Travelling Salesman Problem (GTSP) solver to obtain the full coverage plan. In Section 3.6, we present our experimental results on a set of real-world environments and compare the performance of our algorithm with the method proposed in [88].

3.2 Problem Definition

We aim to tackle the CPP problem in Chapter 2.1 with the objective of minimizing robot turns. In this section, we introduce a simple decomposition of the environment that constitutes $\widetilde{\mathcal{W}}$ in the CPP problem definition. We then define the rank partitioning and touring problems which we will be solving in the rest of the chapter.

3.2.1 IOP Decomposition

We now look at an extension of Problem 2.1 where we minimize the number of turns in the coverage path p . Following from Assumption 2.1, we decompose the environment $\widehat{\mathcal{W}}$ using a square grid and focus on paths where the tool moves only in axis-parallel directions during coverage (horizontal and vertical). We refer to this decomposition as an Integral Orthogonal Polygon (IOP), which is a set of square *grid cells* of size $l \times l$ [7]. Fig. 3.2 shows an example IOP generated for a simple non-convex environment with one hole. Computing the IOP representation of the environment constitutes the decomposition step of the OARP method. We also use the IOP decomposition in many other parts of this thesis.

3.2.2 Minimum Rank Partitioning Problem

We now consider a partition of the IOP that will help quantify the number of potential turns in the final coverage plan. We look at partitioning the environment into regions that will each be covered with a straight-line path (no turns). A *rank* is the cumulative footprint of the coverage tool when traversing a straight-line path. For a square coverage tool of width l , a rank is a rectangular region of width l .

In an IOP, the robot can perform coverage by moving its square tool from one grid cell to another. A rank in an IOP is therefore a series of horizontally or vertically adjacent grid cells (e.g. Fig. 3.3). Since each grid cell is of the same size as the coverage tool, a rank can be covered along a single coverage line (straight-line path) with endpoints at the centers of grid cells. We can thus define the main problem of this chapter as follows.

Problem 3.1 (Minimum Rank Partitioning Problem). *Given an IOP representation of $\widehat{\mathcal{W}}$, compute a partition of the IOP into the minimum number of axis-parallel (horizontal and vertical) ranks.*

From the rank partition determined in Problem 3.1, we generate a coverage path for the environment by solving the following problem.

Problem 3.2 (Tour Generation Problem). *Given a set of axis-parallel ranks (partitioning of an IOP), compute a tour of the ranks that minimizes the total cost of transitions between ranks.*

In a tour of the ranks, the coverage tool moves from one rank’s endpoint to another along transition paths determined by the robot’s dynamics.

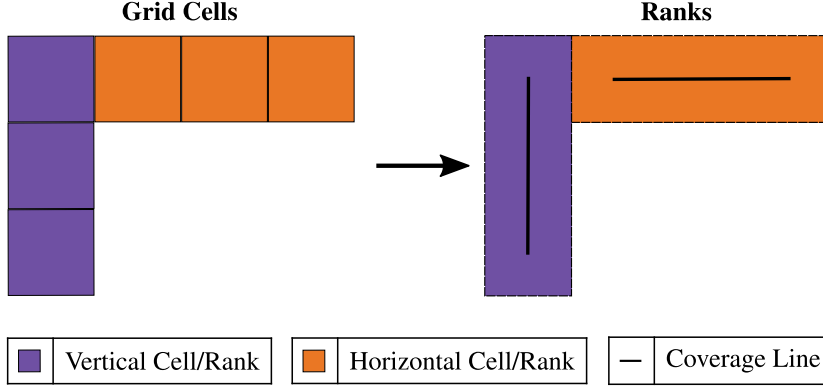


Figure 3.3: Illustration of merging oriented grid cells to form ranks and the corresponding coverage lines. The cell orientations are determined by the MILP from Section 3.3, and the merge-able cells are merged.

3.3 MILP Formulation

In this section, we formulate Problem 3.1 as a mixed integer linear program (MILP), which involves a set of variables and linear constraints. Consider an IOP with n grid cells, each of which is represented by a variable c_i in $C = \{c_1, c_2, \dots, c_n\}$. We start by introducing an operator $\text{orient}(c_i) \in \{H, V\}$ for each grid cell c_i to represent its coverage orientation (H for horizontal or V for vertical). We also introduce a series of variables that bound the number of ranks in the final partition. The goal of this MILP is to compute the orientations of grid cells that minimize the number of ranks.

Next, we use the concept of neighbouring grid cells to help represent each rank and construct the objective function to minimize the number of ranks. We do this by formulating constraints that can be interpreted as a series of merges for neighbouring cells.

Definition 3.1 (Merge-able grid cell). *A grid cell c_i with orientation $\text{orient}(c_i) \in \{H, V\}$ is merge-able with its neighbour if: (i) the cell and its horizontal neighbour are horizontally oriented, or (ii) the cell and its vertical neighbour are vertically oriented.*

A rank is obtained by merging a set of merge-able cells, and so the rank partition of the IOP depends on the assigned cell orientations. Fig. 3.3 illustrates the intuition of determining ranks from orientations.

We now work towards constructing a set of operators that count the ranks for a given set of oriented cells. The following lemma highlights a useful relationship for this task.

Lemma 3.1. *In any minimal rank partition, each rank terminates at cells with less than two merge-able neighbours.*

Proof. By definition, each grid cell in the IOP can have at most two merge-able neighbours across an axis. Let $R = (r_1, r_2, \dots, r_k)$ be an ordered set of k similarly-oriented cells $r_i \in C$ representing a rank along an axis. By our definition of R , these cells are connected in a sequential order $(r_1, r_2, \dots, r_k)$.

Consider one of the endpoints of R : r_1 . By definition, r_1 and r_2 are merge-able neighbours. Let r_1 have another neighbour r_t that is covered by a rank $\tilde{R}$. If r_t is a merge-able neighbour of r_1 , then R and $\tilde{R}$ can be merged to a single rank, thereby reducing the number of ranks and contradicting the optimality of the partition. This would mean one of two things: (i) r_t does not exist, i.e., r_t is a border or an obstacle, or (ii) r_t is not a merge-able neighbour. Similarly, we can show that the other endpoint r_k has only one merge-able neighbour r_{k-1} , which completes the proof of Lemma 3.1. $\square$

Given the above lemma, we can calculate the number of ranks in an optimal partition by counting one of the two unique endpoints of each rank. For the rest of this chapter, we only consider detecting and counting the *left* and *top* endpoints of horizontal and vertical ranks respectively. The choice of left and top endpoints is arbitrary and does not effect the solution approach. To identify and count the rank endpoints at the IOP border or holes, we add some artificial cells next to the border cells. These cells are not assigned an orientation, but simply indicate whether there is a rank that has an endpoint at the border. We will refer to these cells as *border identifiers*.

The following logical statements summarize how we will constrain our endpoint counting variables. For each grid cell c_i , denoting the left and top neighbors of c_i by $\text{left}(c_i)$ and $\text{top}(c_i)$ respectively, we define our endpoint operators as follows:

$$\begin{aligned} \text{end}_H(c_i) &= \begin{cases} 1 & \text{if } \text{orient}(c_i) = H \text{ and } \text{orient}(\text{left}(c_i)) = V \\ 1 & \text{if } \text{orient}(c_i) = H \text{ and } \text{left}(c_i) \text{ is a border identifier} \\ 0 & \text{otherwise} \end{cases}, \\ \text{end}_V(c_i) &= \begin{cases} 1 & \text{if } \text{orient}(c_i) = V \text{ and } \text{orient}(\text{top}(c_i)) = H \\ 1 & \text{if } \text{orient}(c_i) = V \text{ and } \text{top}(c_i) \text{ is a border identifier} \\ 0 & \text{otherwise} \end{cases}, \end{aligned}$$

where $\text{end}_H(c_i)$ and $\text{end}_V(c_i)$ are binary operators determining if c_i is a horizontal or a vertical endpoint respectively. We now use these functions to formulate the following

binary programming problem:

$$\min \quad \sum_{i=0}^n \text{end}_H(c_i) + \sum_{i=0}^n \text{end}_V(c_i) \quad (3.1)$$

$$\text{s.t.} \quad \text{orient}(c_i) \in \{H, V\}, \forall i \in \{1, 2, \dots, n\}. \quad (3.2)$$

Next, we will convert the above binary programming problem to a MILP by replacing the operators with a set of new variables. Let us start with $\text{end}_H(c_i)$ and create the auxiliary variable y_h^i that bounds $\text{end}_H(c_i)$. The upper index (i) of y_h^i matches the lower index of c_i . We also introduce a set of binary variables to denote whether a cell is horizontally or vertically oriented. For the horizontal case, we introduce x_h^i which equals 1 if $\text{orient}(c_i) = H$ and 0 otherwise. For the left neighbour, $c_l = \text{left}(c_i)$, we observe the following:

$$x_h^i - x_h^l = \begin{cases} 1 & \text{if } \text{orient}(c_i) = H \text{ and } \text{orient}(\text{left}(c_i)) = V \\ 1 & \text{if } \text{orient}(c_i) = H \text{ and } \text{left}(c_i) \text{ is a border identifier} \\ -1 & \text{if } \text{orient}(c_i) = V \text{ and } \text{orient}(\text{left}(c_i)) = H \\ 0 & \text{otherwise} \end{cases}.$$

The above, along with a non-negativity constraint, gives the following encoding of y_h^i :

$$\begin{aligned} y_h^i &\geq x_h^i - x_h^l \\ y_h^i &\geq 0, \end{aligned}$$

where $y_h^i \geq \text{end}_H(c_i)$. With these constraints, we observe that minimizing y_h^i gives us an optimal solution where $y_h^i = \text{end}_H(c_i)$. We now determine the values of y_h^i for all grid cells using the following system of vector inequalities:

$$\mathbf{y}_h \geq A_H \mathbf{x}_h \quad (3.3)$$

$$\mathbf{y}_h \geq 0, \quad (3.4)$$

where $\mathbf{y}_h$ and $\mathbf{x}_h$ are n -dimensional vectors composed of the variables y_h^i and x_h^i respectively. On further inspection, A_H in (3.3) is the *node-arc incidence (NAI) matrix* [67] of the directed graph G_H (see Fig. 3.4) composed of all horizontal path flows for an IOP grid cell from its left neighbour/border identifier. Similarly, we encode $\text{end}_V(c)$ using the auxiliary variable vector $\mathbf{y}_v$ and the NAI matrix A_V of the directed graph G_V (Fig. 3.4).

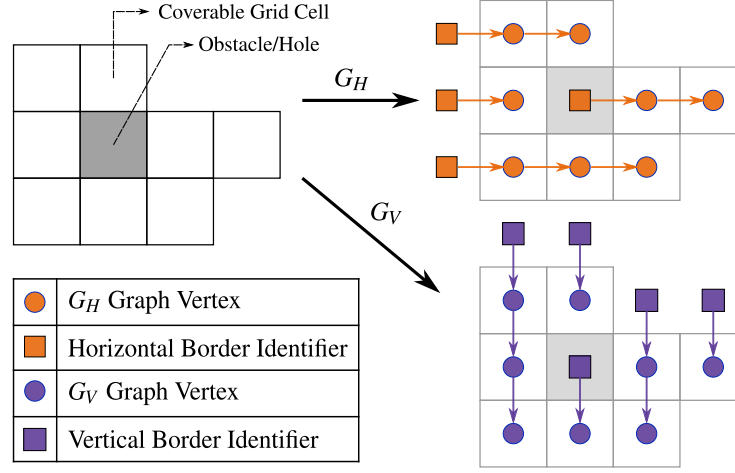


Figure 3.4: The directed graphs G_H and G_V for an example IOP.

Using the defined matrices and vectors, we write the following MILP to solve Problem 3.1:

$$\min \sum_{i=0}^n y_h^i + \sum_{i=0}^n y_v^i \quad (3.5)$$

$$\text{s.t. } A_H \mathbf{x}_h - \mathbf{y}_h \leq \mathbf{0} \quad (3.6)$$

$$A_V \mathbf{x}_v - \mathbf{y}_v \leq \mathbf{0} \quad (3.7)$$

$$\mathbf{x}_h + \mathbf{x}_v = \mathbf{1} \quad (3.8)$$

$$\mathbf{x}_h, \mathbf{x}_v \in \{0, 1\}^n \quad \mathbf{y}_h, \mathbf{y}_v \geq \mathbf{0}, \quad (3.9)$$

where $\mathbf{1}$ is the column vector of ones and $\mathbf{0}$ is the vector of zeros. Eq. (3.8) ensures that each grid cell is assigned one of two orientations (either x_h^i or x_v^i equals 1). The objective in Eq. (3.5) gives us the number of ranks while $\mathbf{x}_h$ and $\mathbf{x}_v$ gives us the corresponding grid cell orientations. The solution of the MILP gives the minimum rank partition as determined by the grid cell orientations, where we obtain the ranks by merging all merge-able cells.

We also obtain an LP relaxation from this MILP using Eqs. (3.5)-(3.8) and replacing the binary variables $\mathbf{x}_h$ and $\mathbf{x}_v$ in (3.9) with continuous variables as follows:

$$\mathbf{x}_h, \mathbf{x}_v, \mathbf{y}_h, \mathbf{y}_v \geq \mathbf{0}. \quad (3.10)$$

Proposition 3.1. *The relaxed LP denoted by Eqs. (3.5)-(3.8), (3.10) computes an integral optimal solution and hence solves Problem 3.1 in polynomial time.*

3.4 Proof of Proposition 3.1

In general, solving MILPs is NP-hard while LPs can be solved in polynomial time [54]. An LP relaxation of a MILP in general will not yield optimal integral solutions. However, our LP relaxation belongs to a special class of problems that yield integral optimal solutions. This is mainly because of our use of NAI matrices, which are totally unimodular (TU) [72]. A matrix is TU if the determinants of all its square submatrices are in $\{-1, 0, 1\}$ [67]. In this section, we prove that the matrix of all constraints in Eqs. (3.5)-(3.8), (3.10) is TU. It will then follow from the Hoffman-Kruskal Principle [43] that the relaxed LP can directly solve the MILP in polynomial time, since our matrix is TU.

Theorem 3.1 (Hoffman-Kruskal Principle [43]). *If an integral matrix A is TU, then for an integral vector $\mathbf{b}$, the extreme points of the polyhedron $\{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b}\}$ has integral coordinates.*

Since the extreme points of the polyhedron has integral coordinates, the optimal solution obtained from solving the LP defined in the same polyhedron is also integral.

Lemma 3.2. *The constraints of the LP formulation in Eqs. (3.5)-(3.8), (3.10) are TU.*

Proof. First, we write our problem in Standard Equality Form (SEF) using slack variables to eliminate the inequality constraints. We then rewrite the summations in Eq. 3.5 using a vector multiplication with $\mathbf{1}$ and obtain

$$\min \quad \mathbf{1}^T \mathbf{y}_h + \mathbf{1}^T \mathbf{y}_v \quad (3.11)$$

$$\text{s.t.} \quad A_H \mathbf{x}_h - \mathbf{y}_h + \mathbf{z}_h = \mathbf{0} \quad (3.12)$$

$$A_V \mathbf{x}_v - \mathbf{y}_v + \mathbf{z}_v = \mathbf{0} \quad (3.13)$$

$$\mathbf{x}_h + \mathbf{x}_v = \mathbf{1} \quad (3.14)$$

$$\mathbf{x}_h, \mathbf{x}_v, \mathbf{y}_h, \mathbf{y}_v, \mathbf{z}_h, \mathbf{z}_v \geq \mathbf{0}. \quad (3.15)$$

The constraints are now of the form $A\mathbf{x} = \mathbf{b}$, where

$$A = \begin{bmatrix} A_H & \bar{\mathbf{0}} & -I & \bar{\mathbf{0}} & I & \bar{\mathbf{0}} \\ \bar{\mathbf{0}} & A_V & \bar{\mathbf{0}} & -I & \bar{\mathbf{0}} & I \\ I & I & \bar{\mathbf{0}} & \bar{\mathbf{0}} & \bar{\mathbf{0}} & \bar{\mathbf{0}} \end{bmatrix}, \quad (3.16)$$

$$\mathbf{b} = [\mathbf{0}^T \quad \mathbf{0}^T \quad \mathbf{1}^T]^T, \quad (3.17)$$

$$\mathbf{x} = [\mathbf{x}_h^T \quad \mathbf{x}_v^T \quad \mathbf{y}_h^T \quad \mathbf{y}_v^T \quad \mathbf{z}_h^T \quad \mathbf{z}_v^T]^T, \quad (3.18)$$

$\bar{\mathbf{0}}$ is the matrix of zeros and I is the identity matrix. Clearly $\mathbf{b}$ is an integral vector.

To prove A is TU, let us start with the matrix

$$\tilde{A} = \begin{bmatrix} A_H^T & \bar{\mathbf{0}} & I \\ \bar{\mathbf{0}} & A_V^T & I \end{bmatrix}.$$

Each row of the NAI matrices A_H and A_V signifies a directed edge in the graph, with a -1 for the source grid cell (outgoing), a +1 for the sink grid cell (incoming), and 0s otherwise [67].

Due to this construction, $\tilde{A}$ satisfies the following sufficient conditions for TU matrices as outlined in [43].

1. All elements are in $\{-1, 0, +1\}$ (all entries in A_V and A_H are either -1, 0, or +1).
2. Each column of $\tilde{A}$ has at most two non-zero elements (each column of A_H^T and A_V^T will have two non-zero entries, one for the source and one for the sink).
3. There exists a partition of the rows of $\tilde{A}$ into two disjoint sets T_1 and T_2 such that:
 - (i) If any column of $\tilde{A}$ contains two nonzero entries of the same sign, then one is in a row of T_1 and the other is in a row of T_2 .
 - (ii) If any column of $\tilde{A}$ contains two nonzero entries of the opposite sign, then they are both in a row of T_1 or in a row of T_2 .

For $\tilde{A}$, T_1 consists of all rows in

$$[A_H^T \quad \bar{\mathbf{0}} \quad I]$$

and T_2 consists of all rows in

$$[\bar{\mathbf{0}} \quad A_V^T \quad I].$$

It follows from [67] that the transpose of $\tilde{A}$ is also TU. We can construct A from $\tilde{A}^T$ by appending columns of the positive and negative signed unit matrices $\bar{I}$ and $-\bar{I}$ where

$$\bar{I} = \begin{bmatrix} I & \bar{\mathbf{0}} & \bar{\mathbf{0}} \\ \bar{\mathbf{0}} & I & \bar{\mathbf{0}} \\ \bar{\mathbf{0}} & \bar{\mathbf{0}} & I \end{bmatrix}.$$

Appending these columns preserves the total unimodularity of the resulting matrix, see [67]. Because of this, the matrix A is TU, which completes the proof of Lemma 3.2. $\square$

Since the constraints are TU, it follows from Theorem 3.1 that the relaxed LP computes optimal integral solutions for the formulated MILP, thereby completing the proof of Proposition 3.1. We can therefore use efficient polynomial-time LP solvers [54] to solve Problem 3.1 optimally.

3.5 Tour Generation

We now address Problem 3.2 and plan our full coverage path by computing a tour of the ranks obtained from the LP in Section 3.3. We do this by formulating a GTSP to determine a visitation order for the ranks. Using a GTSP formulation offers us the flexibility to use existing approaches to compute the tour [80, 68]. We employ a similar formulation that was proposed in [11] and create an auxiliary graph representing all possible connections between the ranks. The vertices of this graph are grouped into sets, where each set consists of two vertices representing the two directions to cover the rank (e.g., for a horizontal line we can cover it from left to right or right to left). The GTSP aims to minimize the cost of visiting one vertex in every set (traversing each rank in a specific direction).

The edge costs between the vertices in different sets of the auxiliary graph are given by the time to travel between rank endpoints along an obstacle-free *transition path*. These transition paths are planned using the dynamics of the robot, which depends on the robot’s design. For this work, we assume semi-holonomic dynamics where (i) the robot stops to turn in-place with a constant angular velocity, and (ii) the robot has piecewise constant acceleration (when accelerating or decelerating) while travelling in a straight line with a maximum linear velocity. The same dynamics model is used in [88] and [9], which we compare our method against in Section 3.6. We compute each transition path by constructing a visibility graph within the IOP boundaries and planning the shortest path using an A* search with the straight-line Euclidean distance used as the heuristic [69]. The resulting path is a sequence of straight lines and intermediary turns, for which the travel time is computed.

3.6 Simulation Results

In this section, we present our experiments to test the performance of OARP and compare it to two different coverage planning approaches from literature. Finally, we present ROS simulations on an example environment with the Avidbots Neo robot model.

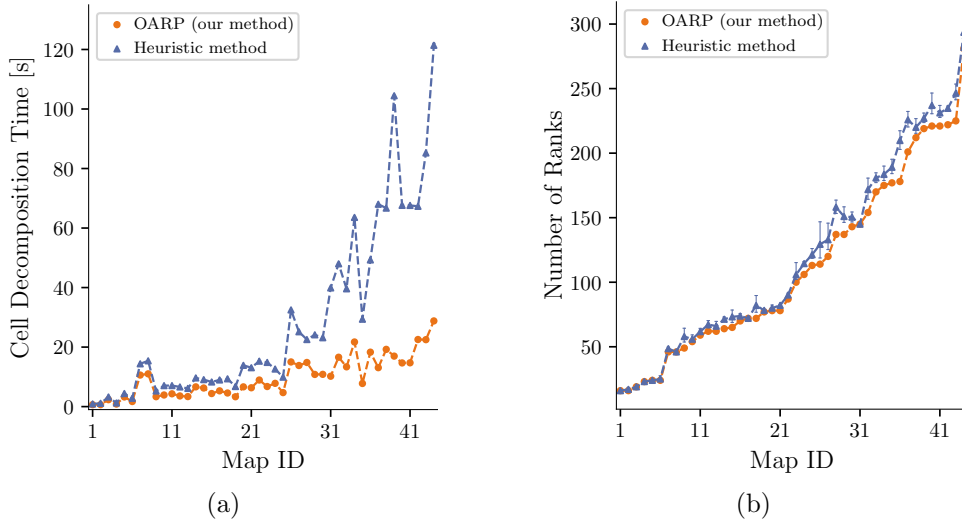


Figure 3.5: Comparison of our rank partitioning method with that proposed in [88]: (a) Cell decomposition runtime for each map. (b) The number of ranks determined for each test environment.

Firstly, in Sections 3.6.1 to 3.6.3, we compare OARP to the heuristic approach from [88]. We chose the heuristic approach for our main comparisons because it produced the least turns and the lowest tour costs in our analysis of the available approaches. While the work in [88] has applied the heuristic approach for both single and multi-robot cases, the rank partitioning step is itself independent of the number of robots, which is similar to OARP. We focus on the single-robot coverage tours in this comparison to analyze how the different rank partitioning methods affect the tour. For this comparison, we use a dataset containing 2D maps of 44 real-world environments obtained from Avidbots and used for their Neo cleaning robot. These environments have a minimum coverage path length ranging from 100 m to around 3700 m , where the minimum coverage path length of an environment is given by $(\text{total area})/(\text{tool width } l)$. The maps in this dataset are arranged/labeled by increasing complexity, i.e., the minimum number of ranks required to cover the environment. Due to the confidentiality of these scans, we cannot share this dataset but we have additionally generated a set of anonymized environments for visualization, which are shown in Figs. 3.1 and 3.6. Each method is run for 20 trials on each map in the dataset and we compare the results for every stage of the coverage planning process.

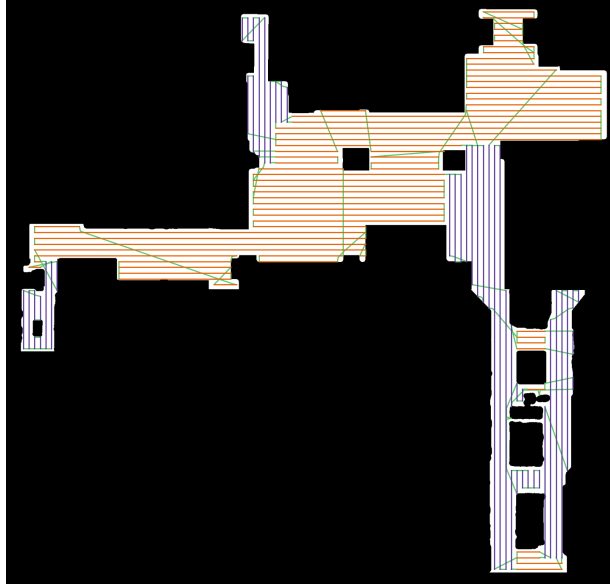


Figure 3.6: Coverage path of an indoor environment generated using the proposed LP method.

3.6.1 Decomposition Comparisons

For each map in our dataset, we compared the time it takes for both methods to decompose the environment into cells. The heuristic method decomposes the environment into a preliminary set of coarse rectangular cells [88], while the OARP method simply needs to express the IOP of the environment using a grid overlay. To construct the IOP, we include all grid cells where over 50% of its area is contained within the original environment. The results of the comparison are shown in Fig. 3.5a, where we observe that as the complexity of the map increases, the cell decomposition time for the heuristic method increases rapidly. In contrast, OARP only requires a grid approximation which can be computed much faster; about 50% faster on average and 5 times faster in some cases. This makes a large impact on the overall planner time, especially in cases where the coverage plan would need to be replanned due to perceived changes or uncertainty in the environment. Robustification of our approach to dynamic environment uncertainties is a future research direction.

3.6.2 Rank Partitioning Comparisons

We implement our rank partitioning step by expressing the LP using Eqs. (3.5)-(3.8), Eq. (3.10), and the IOP built in the decomposition step. The algorithm was prototyped in

Table 3.1: Robot parameters used for experiments

Coverage Tool Width	0.8 m
Maximum Linear Velocity	1 m/s
Linear Acceleration	$\pm 0.5 \text{ } m/s^2$
Angular Velocity	$30^\circ/s$

Python and we used the free LP solver, GLPK [59] to solve the LP. The coverage tool width was set to 0.8 m to model Avidbots’ robot. Fig. 3.1 shows the rank partitioning of an example map with the horizontal (orange) and vertical (purple) ranks as determined by the LP solver. Solving the LP takes an average of 0.4 seconds across all trials for all maps.

For the heuristic method, we solve the same IOP coverage problem by only including the interior ranks from their method (we ignore the perimeter ranks). The heuristic method is an iterative algorithm, where running for more iterations improves the performance. So, one would need to allow enough iterations to find high quality solutions, but avoid extra iterations after the optimal has been found. In contrast, OARP avoids unnecessary computation by deterministically finding an optimal solution before terminating.

To standardize the comparison, we limit the runtime of the heuristic method’s partitioning step to that of OARP’s partitioning step (the LP’s runtime) for each map. Fig. 3.5b shows the results of the experiments. As expected, we observe that the number of ranks obtained by OARP is always as good or better than the heuristic method, with the performance gap widening for more complex maps. In general, the heuristic method does quite well, likely due to the tractability of the problem. However, since this problem is solvable in polynomial time, one should choose an optimal solver.

3.6.3 Coverage Tour Comparisons

To generate the coverage tour from the ranks, we use the GTSP formulation in Section 3.5 to compute a tour that minimizes the transition time between the ranks. Table 3.1 documents our choice of parameters used to compute the transition costs by simulating the robot’s coverage tool and dynamics. We used GLNS [80] to solve the formulated GTSP problem for both OARP and the heuristic methods. Fig. 3.6 shows an example coverage tour for an anonymized environment from our dataset, with the transitions (green lines) between the ranks as determined by the GTSP solver.

We also compared our tours to those obtained using the heuristic method from [88]. Since the ranks are non-overlapping for both methods, the cost of the tour is related to the

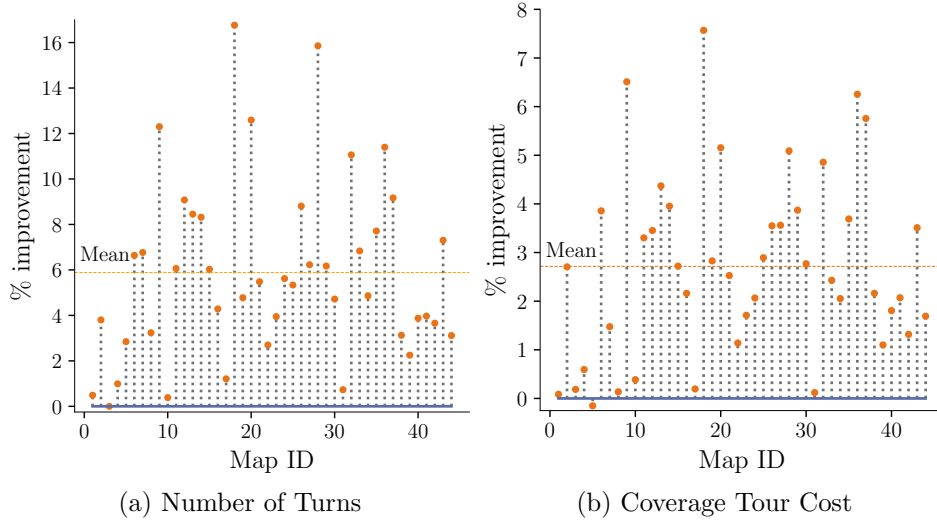


Figure 3.7: Average % improvement exhibited by the OARP method in comparison with [88] for each map.

number of turns. Fig. 3.7 shows the comparison results, where we plot the average percent improvement of OARP over the heuristic method for each map on two metrics: a) number of turns, and b) total coverage tour cost. We observe that OARP improves the coverage tours on both metrics for almost all maps. For a small number of maps, we see little to no improvement due to the heuristic method providing the same number of ranks as OARP for those maps. On average for all maps, OARP improved the number of turns by 5.9% and the tour cost by around 2.7%. For some maps, we get a 16.8% improvement in the number of turns, and a 7.6% improvement in total coverage time. From this comparison, we observe the intended trend that less ranks leads to less turns, and less turns leads to shorter coverage paths.

3.6.4 Comparison against BCD

We additionally compared the proposed OARP method against the Boustrophedon Cell Decomposition (BCD) method from [9]. For this comparison, we used the dataset from [9] containing over 300 aerial scans of environments with varying number of holes (obstacles). The complexity measure of the maps in this dataset is given by *hole vertices*, i.e., the total number of vertices required to represent the holes in the map (e.g., one triangular hole has 3 vertices). The maximum number of hole vertices in this dataset is 86, whereas in Avidbots’ dataset, this number goes up to 900. We do not test BCD on Avidbots’

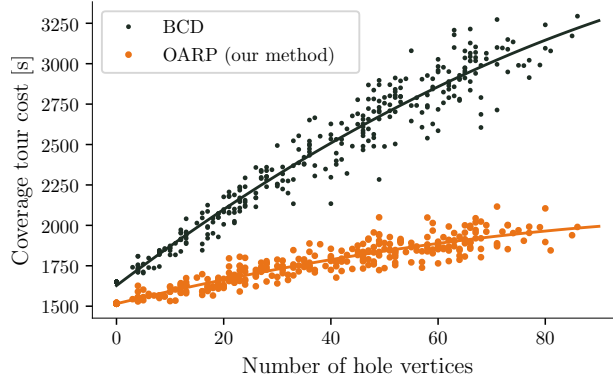


Figure 3.8: Comparison results of the coverage tour generated using OARP and the BCD method [9]. The x-axis specifies the number of vertices representing the holes in the maps.

dataset as it would require significant modifications to the BCD implementation, but we expect to see a similar but exaggerated trend due to the large number of hole vertices. To perform this comparison, we use the same parameters from [9]; a tool width of 3 m , linear acceleration of $\pm 1 \text{ m/s}^2$, and maximum velocity of 3 m/s . We also penalize turns by adding turning time to the cost function, where the robot turns in-place with an angular velocity of $30^\circ/s$. Fig. 3.8 shows the results of comparing the cost of the coverage tours generated by OARP and BCD using identical parameters. We observed that OARP outperforms BCD for all maps, and improves the coverage tour by 25.5% on average. We believe that this performance gap is due to OARP’s better coverage line placement. We also tested the heuristic method from [88] on this dataset and observed that, while OARP still outperforms [88], the average performance improvement is small ($<1\%$) due to the relative simplicity of the maps in this dataset.

3.6.5 ROS Simulation Case Study

In this subsection, we provide a case study of simulating the coverage of an example environment using the Avidbots Neo robot. The simulation was conducted using ROS on an anonymized hybrid environment as shown in Fig. 3.9. The Neo robot has tricycle dynamics and so coverage was planned with sufficient space at the boundaries to allow comfortable turns. The robot was also equipped with a local planner to replan difficult parts of the coverage plan (e.g. areas with small coverage lines) if necessary. Fig. 3.9 shows the resulting environment coverage using plans generated by OARP and the heuristic method [88]. We observed that OARP’s plan was nearly 13% faster than the heuristic method’s plan and

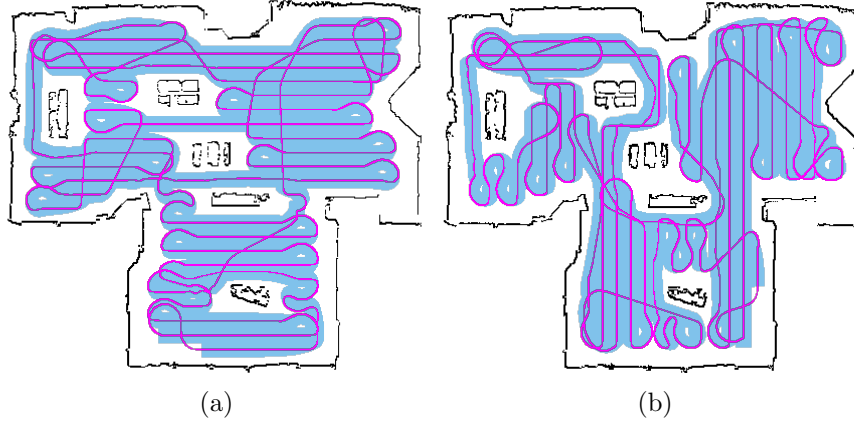


Figure 3.9: Simulation results of covering an example environment using the Avidbots Neo robot. The coverage planners used were (a) OARP and (b) the heuristic method from [88].

3.3% shorter in path length. We also observed that OARP’s plan covered more area of this map than the heuristic method’s plan (1.1% more). The reasoning for this is two-fold: OARP’s path consists of less turns (poor coverage at turns), and OARP’s plan required less replans by the local planner. We also attempted to test the plan generated using the BCD planner from [9], but the plan contained many sharp turns that are infeasible for the Neo robot.

3.7 Summary

In this chapter, we proposed the OARP coverage planning method that aims to minimize the number of turns needed to cover the robot’s environment. We do this by partitioning the environment into thin axis-parallel ranks using a linear program (LP). We proved that this formulation computes the optimal rank partitioning in polynomial time. We then conducted experiments on maps of real-world environments and showed that OARP always achieves the best rank partition in comparison with the state-of-the-art method. Furthermore, OARP is also shown to have around 6% lesser turns and 3% shorter coverage tours on average than the other method.

Chapter 4

Coverage Replanning for Partially Unknown Environments

4.1 Introduction

In this chapter, we tackle the problem of replanning optimal coverage paths that have been interrupted by previously unknown static obstacles. We refer to this as the *coverage replanning* problem. This problem arises when businesses such as retail stores and warehouses change their environments frequently to accommodate new products and remapping beforehand each time would be too costly.

One approach to coverage replanning is to stop executing the current coverage path and plan a new path for the uncovered environment with the new obstacle(s). However, coverage planning for arbitrary environments is generally NP-Hard [8] and this approach does not scale well for large environments with many unknown obstacles. Moreover, an obstacle may only affect a small section of the path, i.e., the unbroken parts of the path may still be drivable and near-optimal. An alternate approach is to only replan the affected section(s) of the path (*local replanning*) while keeping the rest of the path intact. A simple example of local replanning would be where the robot simply drives around the obstacle whenever a collision is imminent. However, a more practical approach would be to use the time while driving towards the blockage to find more optimal solutions. In this chapter, we propose an anytime coverage replanner that utilizes the time spent driving to conduct near-optimal replans on the provided coverage paths (see Fig. 4.1).

In the previous chapter [74], we developed the Optimal Axis-Parallel Rank Partitioning (OARP) approach to plan coverage paths that minimize the number of turns in the

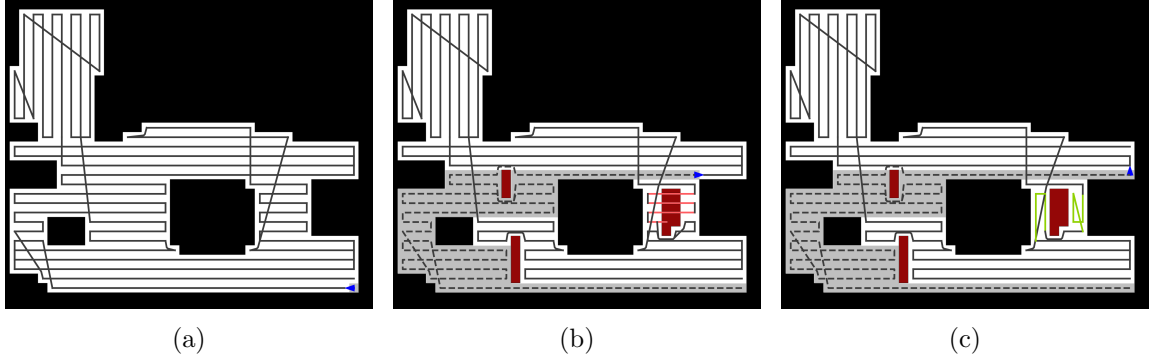


Figure 4.1: Replanning coverage paths in partially unknown environments. (a) An initial coverage path for the known base environment, (b) the robot (blue triangle) covers part of the environment (gray region) and observes a new obstacle (red box) interrupting the initial path (red path), and (c) a near-optimal replanned path (green path) computed online using the proposed approach (OARP-Replan).

solution. Minimizing turns is motivated by path quality and optimality as robots can have decreased performance while turning (speed and coverage). In this chapter, we extend OARP to provide a coverage replanning approach, namely *OARP-Replan*, that replans coverage paths.

Chapter contributions: Our specific contributions are as follows:

1. We present an Integer Linear Program (ILP) that replans the ranks of a coverage path such that a tour of the ranks (i.e. a coverage path) can be replanned within a time budget.
 - a) We leverage LP relaxations of this ILP to replan ranks in polynomial time (efficient to compute).
 - b) We present a second ILP that reduces the number of decision variables, and therefore replans the coverage ranks faster than the first ILP.
2. We provide an anytime coverage replanning method named *OARP-Replan* that leverages the above ILPs to conduct near-optimal replans to the coverage path within a given time budget.
3. Finally, we conduct simulations and robot experiments to validate our proposed approach. Our simulation results include comparisons against other state-of-the-art approaches, including offline approaches (i.e. all obstacles are known) [9, 5].

4.1.1 Organization

The rest of this chapter is organized as follows. In Section 4.2, we define the problem of replanning a coverage path within a given time budget. In Section 4.3, we present the high-level framework of our proposed approach, OARP-Replan, that constrains the runtime to replan the coverage path, which solves the problem in two stages: (i) *rank replanning*, and (ii) *touring*. In Section 4.4, we describe the rank replanning stage where we use ILPs to replan the ranks of the coverage path such that the overall runtime is constrained. In Section 4.5, we describe the touring stage where we formulate a Generalized Traveling Salesman Problem (GTSP) to compute a tour of the replanned ranks to obtain the replanned path. In Section 4.6, we provide simulation results on a range of real-world environments and compare our proposed approach against state-of-the-art coverage planning methods in online and offline settings. In Section 4.7, we present experimental results of testing our replanning approach using an industrial-level autonomous cleaning robot.

4.2 Problem Definition

In this section, we define the problem of replanning the coverage path. This is an extension of the coverage path planning (CPP) problem introduced in Chapter 2.1. Recall that $\mathcal{W}$ is the robot’s environment and $\widetilde{\mathcal{W}}$ is the environment approximation (i.e., IOP) we look to cover. The CPP problem plans an initial coverage path P that covers $\widetilde{\mathcal{W}}$ using a square tool of width l while minimizing the path cost $J(P)$.

4.2.1 Coverage Replanning Problem

Let us consider the case where our robot is trying to follow its initial coverage path P but it observes a set $\mathcal{O}$ of previously unknown obstacles. In other words, the actual environment is $\overline{\mathcal{W}} \subseteq \mathcal{W} \setminus \mathcal{O}$. We assume the robot is equipped with a sensor that can observe its environment within a sensor footprint relative to the robot’s position, e.g., an optical range sensor. Suppose, at position $\mathbf{x}_i$, the robot detects a new obstacle interrupting the path. Let τ be the time in which the robot would *encounter* the obstacle, i.e., reach it along its current path from $\mathbf{x}_i$. While the robot might be able to continue along P for time τ , the subsequent path must be replanned. One solution is to solve the full CPP for the new environment $\overline{\mathcal{W}}$. While this method would give us the optimal coverage path, it has a major drawback. If solving CPP takes longer than τ , the robot may have to stop and

wait for a new path, which is undesirable. To address this, we instead solve the coverage replanning problem that takes into account a time budget τ .

Let P_r be the remaining path after the first interruption by the obstacle. Let $\mathcal{W}_r \subseteq \widetilde{\mathcal{W}}$ be the area covered by P_r , where $\mathcal{O} \cap \mathcal{W}_r \neq \emptyset$. The replanning problem is to plan a new coverage path P'_r before τ such that it covers the same areas as P_r without colliding into $\mathcal{O}$. We are now ready to pose the main problem we look to solve in this chapter.

Problem 4.1 (Coverage Replanning Problem (CRP)). *Given a path P covering $\widetilde{\mathcal{W}}$, a set of newly observed obstacles $\mathcal{O}$, a time τ until the first such obstacle is encountered along P , the remaining path P_r after the first encounter, and the area $\mathcal{W}_r \subseteq \widetilde{\mathcal{W}}$ covered by P_r , find a new coverage path P'_r within time τ which solves*

$$\min_{P'_r \in \mathcal{P}} J(P'_r) \quad (4.1)$$

$$s.t. \quad \bigcup_{(\mathbf{x}, \theta) \in P'_r} \mathcal{A}(\mathbf{x}, \theta) = \mathcal{W}_r \setminus \mathcal{O}. \quad (4.2)$$

CRP is similar to CPP when τ is infinite and $P = P_r$. However, for a finite τ , CRP must find a trade-off between minimizing coverage cost and ensuring a path is computed within τ . The larger the τ value, the closer the coverage cost is to the optimal path (CPP solution).

4.2.2 Rank-based Coverage Replanning

CPP is NP-Hard [8], which indicates that Problem 4.1 is intractable. Therefore, we look for approximations of Problem 4.1 that are more solvable. A common approach is to solve this problem in stages. While this may remove some optimal solutions in the search space, it reduces the complexity of the problem and makes it tractable.

In this chapter, we focus on replanning initial coverage paths that are represented as a series of axis-parallel (horizontal/vertical) ranks covering an IOP decomposition of our initial environment, e.g., paths generated by OARP. With this setup, we approach Problem 4.1 in two stages: (i) rank replanning, and (ii) touring. In the rank replanning stage, we compute the minimum number of axis-parallel ranks to cover the IOP of the remaining region $\mathcal{W}_r \setminus \mathcal{O}$. In the touring stage, we obtain the replanned path P'_r by computing a tour of the ranks that minimizes the cost of transitioning between the ranks. We also look to ensure that both stages are completed within the time budget τ . The following problem captures this requirement.

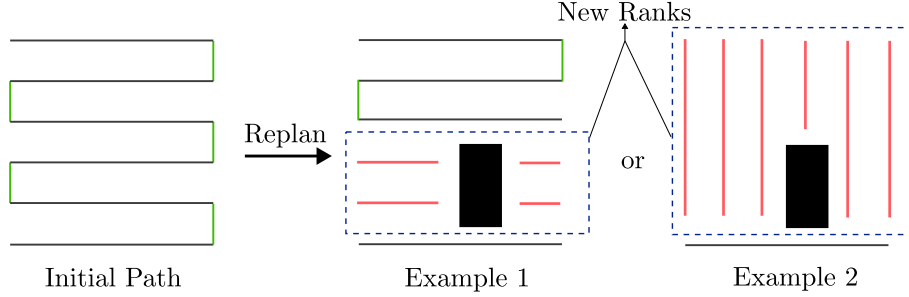


Figure 4.2: The effect of an obstacle (black box) on replanning a coverage path consisting of ranks (black lines) and transition paths (green lines). The ranks of the initial path (left) when interrupted can be replanned in many ways, two of which are shown here. Example 1 has fewer new ranks (red lines in dotted box) than example 2 and is preferable for shorter planning time. Example 2 has fewer total ranks and is preferable for shorter drive time.

Problem 4.2 (Rank-based Coverage Replanning). *Given a set of newly observed obstacles $\mathcal{O}$ that interrupts the remainder P_r of the coverage path P , and a time budget τ before the robot’s first encounter with an obstacle in $\mathcal{O}$, compute the following within time τ : (i) a minimum set of ranks that covers the IOP representation of the uncovered free region $\mathcal{W}_r \setminus \mathcal{O}$ and (ii) a tour of the ranks that minimize the total cost of transition between ranks.*

4.3 Solution Framework

In this section, we describe our solution framework for coverage replanning. Unlike OARP, we look to re-use sections of the initial coverage plan which adds an additional coupling between the rank replanning and touring stages. In this section, we first explore this coupling with an example, and then present the high-level algorithm for the proposed approach.

4.3.1 The Effect of Changing Ranks

We aim to solve rank replanning efficiently so that the majority of the time budget is available for the touring stage, which is formulated as a Generalized Travelling Salesman Problem (GTSP), an NP-hard problem. However, the rank replanning output affects the GTSP input size, and in turn, the runtime. Therefore, we design the rank replanning stage such that the tour of the replanned ranks can be computed within the time budget τ .

To better understand this coupling, consider the example coverage path shown in Fig. 4.2 with an initial set of horizontal ranks. Let us now introduce an obstacle, for which rank replanning returns two different sets of ranks (example 1 and example 2). Each set contains ranks that are unchanged from the initial path and *new ranks*, i.e., ranks that were not in the initial solution. To reduce the computational complexity of the touring step, we reuse sections of the initial path to cover unchanged ranks. This allows us to spend the time budget on connecting new ranks to the unchanged path sections. As a result, the GTSP problem size is driven by the number of *new ranks*. In Fig. 4.2, Example 1’s second stage has a GTSP size of 6 sets (4 new ranks + 2 retained paths), whereas Example 2 has a GTSP size of 7 sets (6 new ranks + 1 retained path of one rank) despite having fewer total ranks. Note that the number of unchanged disconnected path sections is also bounded by the number of new ranks.

Let m be the number of total ranks in the replanned solution and $m_{\text{new}} \leq m$ be the number of new ranks. The GTSP input size is dependent on m_{new} and the estimated GTSP runtime would be a monotonically increasing function $\hat{T}(m_{\text{new}})$. Therefore, given a time budget τ , we bound m_{new} such that $\hat{T}(m_{\text{new}}) \leq \tau$. In other words, a larger τ allows us to change more ranks (m_{new}) so that we can minimize the total number of ranks (m). To compute this bound, we first determine $\hat{T}$ using data collected from previous GTSP runs (e.g. data from the offline planning of the initial coverage path). We then set $\bar{m} = \hat{T}^{-1}(\tau)$ as the maximum GTSP instance size, where $\hat{T}^{-1}$ is the inverse of $\hat{T}$. Thus, we add the following constraint in rank replanning to ensure that the touring stage finishes within time τ :

$$m_{\text{new}} \leq \hat{T}^{-1}(\tau) = \bar{m}. \quad (4.3)$$

In Section 4.4, we will propose a rank replanning approach that constrains the number of new ranks m_{new} in its output within a budget $\bar{m}$ derived from τ . In Section 4.6.1, we provide more details about how $\hat{T}$ can be determined.

4.3.2 Algorithm

We now describe the high-level algorithm of our proposed approach, namely OARP-Replan, for which the pseudocode is given in Algorithm 1. We consider a robot that is equipped with a sensor similar to that of a laser scanner (LIDAR) that is capable of detecting obstacles (or parts of obstacles) in the environment. OARP-Replan is run when potential collisions or *encounters* are detected along the current path P using the robot’s sensor model. More specifically, we determine the grid cells in the IOP decomposition that cannot be covered

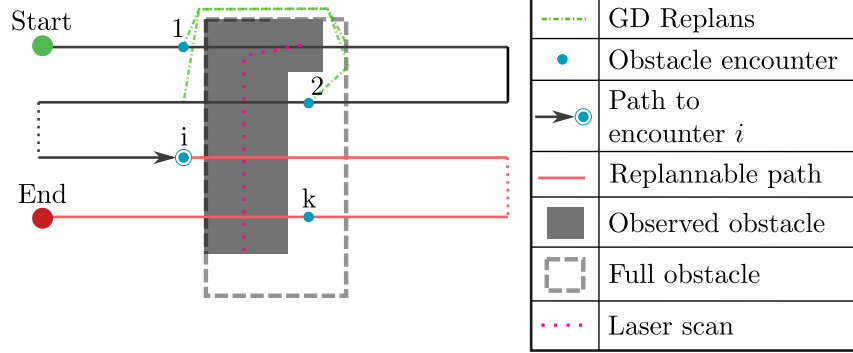


Figure 4.3: A run of OARP-Replan with one obstacle blocking a set of grid cells in the IOP (dotted grey box). The visible part of the obstacle (pink dots) is used to identify blocked grid cells (grey) and obstacle encounters (blue dots) along the current path. For each encounter $i \leq k$, the planner attempts to replan the remaining path (red path) within the time τ_i to reach the encounter (black path). To compute τ_i , previous encounters are replanned using GD replan (green dotted lines).

anymore due to newly detected obstacles (if the obstacle was partially observed, this process would be repeated for each scan that reveals more of the obstacle). Let k be the number of *all* projected *encounters* along P (each obstacle may cause multiple encounters). For each encounter, the robot has the option of replanning the remaining path (the preferred solution) or simply going around the obstacle while covering the interrupted rank (if there is no time to replan). We refer to the latter method as the *greedy detour* (GD) replan, which can be performed using a local planner with little to no computational time and thus no robot stoppage. We propose only using GD replan as a fallback method in the case where the main planner does not have enough time to replan the path.

Fig. 4.3 illustrates an example run of the algorithm. Note that we replan using the obstacle encounters that have been observed so far by the robot. The encounters that were not observed (even from an obstacle that was partially observed) is replanned once the robot has observed them later in the coverage run. We iterate through each obstacle encounter $i \leq k$ and evaluate whether the main planner can replan the path after the i^{th} encounter. The path for all previous obstacle encounters ($\{1, 2, \dots, i-1\}$) is replanned using GD replan. In other words, we look for an obstacle encounter that gives the planner enough time to replan the path after the encounter. Let τ_i be the time for the robot to reach the i^{th} encounter along the current path, which will be the time budget for the replan.

To evaluate if the path after encounter i can be replanned, the planner runs a sub-routine named RANK-REPLAN to get a set of replanned ranks for which a tour can be

Algorithm 1 OARP-Replan

Input: Current path P , Environment IOP $\widetilde{\mathcal{W}}$ **Output:** Replanned path P'

```
1: for each encounter  $i \leq k$  on  $P$  do
2:    $\mathcal{W}_i \leftarrow$  Part of  $\widetilde{\mathcal{W}}$  to cover after  $i^{\text{th}}$  encounter
3:    $R \leftarrow$  Ranks from coverage path  $P$ 
4:    $\tau_i \leftarrow$  Time to reach  $i^{\text{th}}$  encounter
5:    $\bar{m}_i \leftarrow \hat{T}^{-1}(\tau_i)$   $\triangleright$  Maximum instance size given  $\tau_i$ 
6:    $R' \leftarrow \text{RANK-REPLAN}(\mathcal{W}_i, R, \bar{m}_i)$ 
7:   if  $R' \neq \emptyset$  then
8:      $P' \leftarrow \text{TOUR-REPLAN}(R', R)$ 
9:     return  $P'$ 
10:  else
11:    Apply GD replanning to  $i^{\text{th}}$  encounter
12:  end if
13: end for
```

computed in time τ_i (Line 6). If RANK-REPLAN returns a set of ranks, we compute a tour of the ranks using another sub-routine called TOUR-REPLAN (Line 8) that solves the GTSP to obtain our replanned path. This sub-routine runs as a background process while the robot continues to follow the current coverage path. However, if RANK-REPLAN returns an empty set, then this indicates that there is no solution that will finish within the expected time budget. In that case, GD replan is used to replan encounter i (Line 11), and the procedure is repeated for the next encounter. In the end, we obtain a replanned path where some (or all) encounters are replanned using GD replan, while the remaining path undergoes a near-optimal replan before the robot reaches the corresponding encounter.

In our implementation, we run OARP-Replan using the path that the robot is *currently executing*, i.e. if a new obstacle is detected before P' is returned, we start a new replan of P . In the rest of the chapter, we will expand on how the RANK-REPLAN and TOUR-REPLAN sub-routines work.

4.4 The RANK-REPLAN Subroutine

In this section, we extend the integer linear program (ILP) from the OARP approach in our previous work [74] to formulate the RANK-REPLAN subroutine that minimizes the

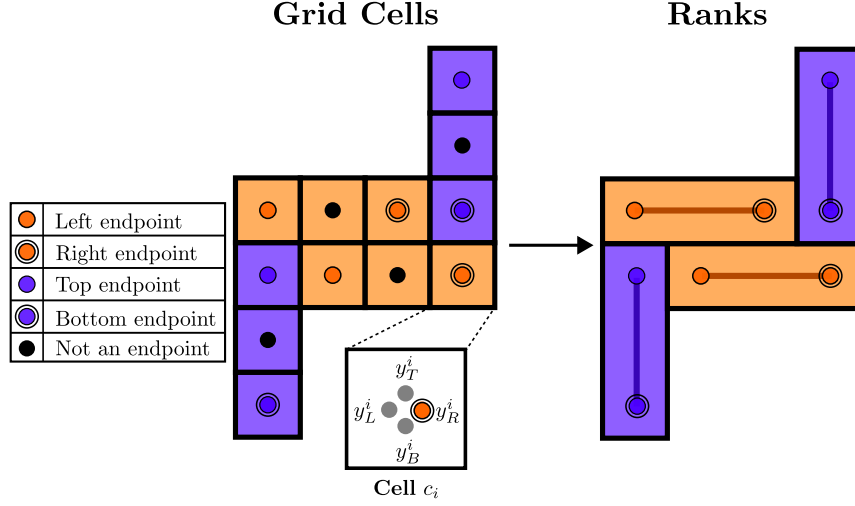


Figure 4.4: Integral orthogonal polygon (IOP) of an environment where each grid cell is either oriented horizontally (orange) or vertically (purple). The left, right, top, and bottom rank endpoints are also shown (see legend). Each cell c_i (zoomed-in cell) has four endpoint indicator variables. If the cell is an endpoint (e.g. c_i is a right endpoint), the corresponding variable is active ($y_R^i = 1$).

number of replanned ranks while constraining the number of new ranks within a budget $\bar{m}$. We first provide a slightly modified of the original ILP with some additional variables to allow for extension. Then, we propose two extensions to this ILP: the first extension solves the rank replanning step optimally, and the second extension solves it efficiently with fewer ILP variables (no optimality guarantees). The user may choose either ILP based on their replanning preferences (optimality vs efficiency).

4.4.1 The Base ILP

Consider an integral orthogonal polygon (IOP) containing n grid cells representing the input environment, where each cell c_i for $1 \leq i \leq n$ is of size $l \times l$ (l is the tool width). The goal of the ILP is to assign a *coverage orientation* (horizontal or vertical) for each cell c_i such that the number of ranks covering the cells is minimized. We obtain ranks by merging *similarly-oriented neighboring* grid cells. To this end, we introduce variables to represent cell orientations and count the resulting ranks after all merges are complete.

Let x_H^i and x_V^i be binary variables that denote whether cell c_i is oriented *horizontally* or *vertically* (cannot be both). Fig. 4.4 shows an example IOP with oriented grid cells and

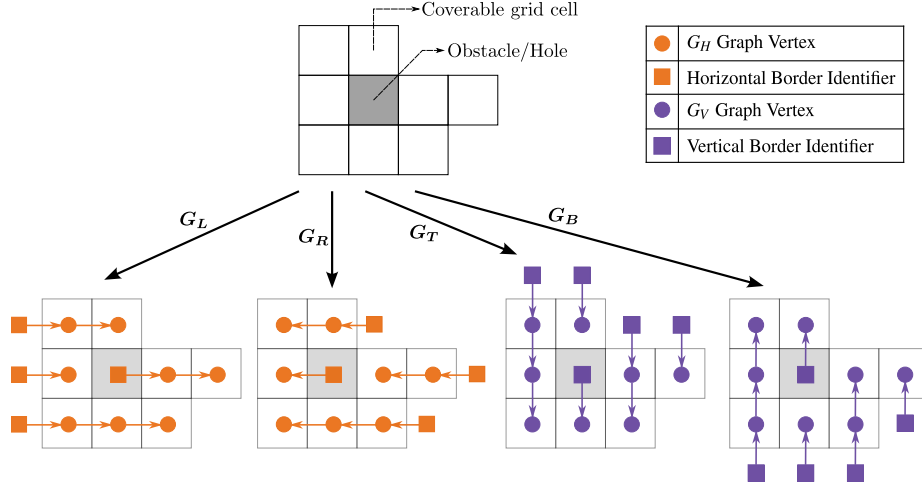


Figure 4.5: The directed graphs G_L, G_R, G_T , and G_B for the example IOP.

their corresponding ranks. We also add variables to identify the rank *endpoints*. Let y_L^i and y_R^i be binary variables that indicate whether the cell, c_i , is the *left* endpoint and/or the *right* endpoint of a horizontal rank. Similarly, let y_T^i and y_B^i be binary variables that indicate whether c_i is a *top* and/or *bottom* endpoint of a vertical rank. This notation is different from the original OARP approach, which only has indicator variables for the left and top endpoints. Fig. 4.4 also illustrates the endpoint variables and rank endpoints for the oriented grid cells.

Let $\mathbf{x}_H, \mathbf{x}_V, \mathbf{y}_L, \mathbf{y}_R, \mathbf{y}_T, \mathbf{y}_B$ be n dimensional binary vectors of the corresponding variables for all grid cells. Let $\mathbf{y}_H = [\mathbf{y}_L^T \ \mathbf{y}_R^T]^T$ and $\mathbf{y}_V = [\mathbf{y}_T^T \ \mathbf{y}_B^T]^T$ be the vectors of all horizontal rank and vertical rank endpoints respectively, where $\mathbf{y}_H, \mathbf{y}_V \in \{0, 1\}^{2n}$. Since each rank has exactly two endpoints, the number of coverage ranks m is given by half the sum of all rank endpoints:

$$m = \frac{1}{2}(\mathbf{1}^T \mathbf{y}_H + \mathbf{1}^T \mathbf{y}_V), \quad (4.4)$$

where $\mathbf{1}$ is the column vector of ones. This constitutes the objective of the optimization problem.

We formulate the ILP constraints by encoding the relationship between the orientations (x variables) and endpoints (y variables). Following Chapter 3, we use directed graphs in the IOP to detect whether a neighboring cell can be merged. Let G_L and G_R be directed graphs composed of all horizontal path flows for each grid cell from its left and right neighbors respectively. For cells at the IOP border, we add artificial nodes called *border*

identifiers to simulate path flow into those cells. Similarly, let G_T and G_B be the graphs composed of all vertical path flows for each grid cell from its top and bottom neighbors respectively. Fig. 4.5 illustrates these graphs for an example IOP. From these graphs, let A_L, A_R, A_T , and A_B be matrices representing the corresponding graphs, where each *row* of a matrix signifies a directed edge in the graph, with a -1 for a source grid cell (outgoing edge), a $+1$ for a sink grid cell (incoming edge), and 0s otherwise. These matrices resemble the node-arc incidence (NAI) matrices for the graphs G_L, G_R, G_T , and G_B .

We now pose the rank minimization ILP, namely ILP-0, that we will extend in the rest of this section:

ILP-0:

$$\min_{\mathbf{x}_H, \mathbf{x}_V, \mathbf{y}_H, \mathbf{y}_V} m \quad (4.5)$$

$$\text{s.t.} \quad [A_L^T \ A_R^T]^T \mathbf{x}_H - \mathbf{y}_H \leq \mathbf{0} \quad (4.6)$$

$$[A_T^T \ A_B^T]^T \mathbf{x}_V - \mathbf{y}_V \leq \mathbf{0} \quad (4.7)$$

$$\mathbf{x}_H + \mathbf{x}_V = \mathbf{1} \quad (4.8)$$

$$\mathbf{x}_H, \mathbf{x}_V \in \{0, 1\}^n \quad (4.9)$$

$$\mathbf{y}_H, \mathbf{y}_V \in \{0, 1\}^{2n}. \quad (4.10)$$

In this ILP, equations (4.6) and (4.7) ensure that, for an *optimal* solution of ILP-0, vectors $\mathbf{y}_H$ and $\mathbf{y}_V$ correspond to the endpoints of the optimal number of ranks covering the IOP. This follows from the minimization of m , which in-turn minimizes $\mathbf{y}_H$ and $\mathbf{y}_V$ (Eq. (4.4)), thereby tightening equations (4.6) and (4.7). Equation (4.8) ensures that each cell is only assigned one orientation (can be either horizontal or vertical).

Note that the ILP constraints formulated here follows that of OARP in Chapter 3 except for the additional endpoint variables, which we will use in the extension for our replanning approach.

4.4.2 Constraining New Ranks

We now look to constrain the number of new ranks m_{new} to be within the rank budget $\bar{m}$. To compute m_{new} , we identify the new ranks in the replanned solution by comparing them with the ranks from the initial path. This comparison requires an encoding of ranks using the ILP variables. To this end, we propose an encoding based on perfect matching [55] that matches the endpoints for each rank in the ILP solution. The new ranks are therefore the difference between the old and new endpoint matchings.

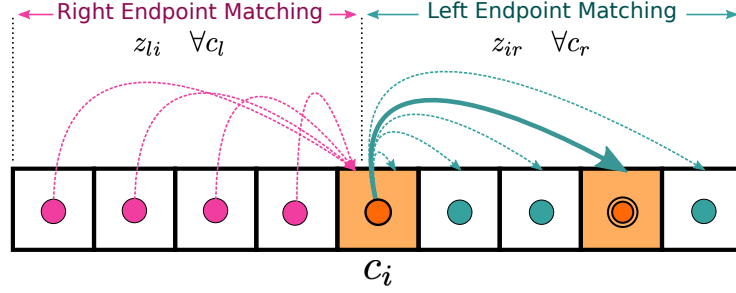


Figure 4.6: The matching edges used to detect ranks for a given row of grid cells. Each cell c_i has edges that match it with another cell in the row if c_i is a right (using magenta edges) or left (using cyan edges) endpoint. Here, we illustrate an example where c_i is a left endpoint and is matched with a right endpoint in the row (bold cyan edge). No matching occurs if c_i is not an endpoint.

To simplify the explanation of the encoding, we first look at the horizontal case and simply extend the same ideas to the vertical case. Consider a horizontal row of *connected* (unobstructed) grid cells in the IOP as depicted in Fig. 4.6. Now, consider the case where the row contains a horizontal rank with a left endpoint at a cell c_i . This indicates that the rank either (i) covers cells to the right of c_i or, (ii) covers only c_i . In both cases, we observe that there must exist a cell to the right of c_i (including c_i) that is the right endpoint of this rank. We look to identify this rank by matching its left and right endpoints.

We formulate this with variables that represent directed edges added between c_i and every cell c_r to its right, as any of them could be a right endpoint. We also add an edge from c_i to itself to account for a single-cell rank. For each edge between c_i and a cell c_r , we add a binary variable $z_{ir} \in \{0, 1\}$ that takes a value of 1 if the edge is included in the matching (i.e. c_r is a right endpoint). To ensure that c_i and c_r are matched only if they are left and right endpoints respectively, we enforce the following constraints:

$$\sum_{\forall r} z_{ir} = y_L^i, \quad (4.11)$$

$$\sum_{\forall i} z_{ir} = y_R^r, \quad (4.12)$$

$$z_{ir} \in \{0, 1\} \quad \forall 1 \leq i, j \leq n. \quad (4.13)$$

Lemma 4.1. *Consider a horizontal row of connected IOP grid cells and two cells c_i and c_r in this row where c_r is to the right of c_i . Then there exists a horizontal rank with its left endpoint at c_i and right endpoint at c_r iff $z_{ir} = 1$.*

Proof. ($\Rightarrow$ direction) If there exists a rank between c_i and c_r , then $y_L^i = 1$ and $y_R^r = 1$. As a result, we have the following for c_i :

$$\sum_{\forall r} z_{ir} = 1, \quad \text{and} \quad \sum_{\forall i} z_{ir} = 1.$$

Assume that $z_{ir} = 0$ (see Fig. 4.7(a)). This indicates that c_i must be matched with another right endpoint c_k to its right, i.e. $z_{ik} = 1$. Since there exists a rank between c_i and c_r , c_k cannot be between the two cells. Therefore, c_k must be to the right of c_r . WLOG, let us assume that the rank between c_i and c_r is the left-most horizontal rank in the row, meaning that there are no other left endpoints for c_r to match with. This is an infeasible solution for the set of constraints unless $z_{ir} = 1$, thereby proving the statement.

($\Leftarrow$ direction) If $z_{ir} = 1$, then y_L^i and y_R^r have to both be 1, indicating that they are left and right endpoints respectively. However, consider the case where there is a vertically-oriented cell between them, indicating that there is no horizontal rank between c_i and c_r (see Fig. 4.7(b)). In this case, there must exist another cell c_k that is a right endpoint (to the left of the vertical cell), where $y_R^k = 1$. WLOG, let us assume that c_i is the only endpoint to the left of c_k . This indicates that c_k has no other left endpoint to match with, which in turn gives an infeasible solution unless $z_{ik} = 1$ and $z_{ir} = 0$. This contradicts the fact that $z_{ir} = 1$, and therefore shows that there must exist a rank between c_i and c_r if $z_{ir} = 1$. This proves the lemma. $\square$

Following Lemma 4.1, we now have a way of representing a horizontal rank as a matching of its endpoints. We extract each horizontal row of connected grid cells in the given IOP and add the following constraints for each cell c_i in the row:

$$\sum_{\forall r} z_{ir} = y_L^i, \quad \text{where } c_r \text{ is to the right of } c_i, \quad (4.14)$$

$$\sum_{\forall l} z_{li} = y_R^i, \quad \text{where } c_l \text{ is to the left of } c_i, \quad (4.15)$$

$$z_{ij} \in \{0, 1\} \quad \forall 1 \leq i, j \leq n. \quad (4.16)$$

We simply extend the setup to represent vertical ranks as a matching of its endpoints in a vertical row of connected grid cells. For each cell c_i in the vertical row, we add the

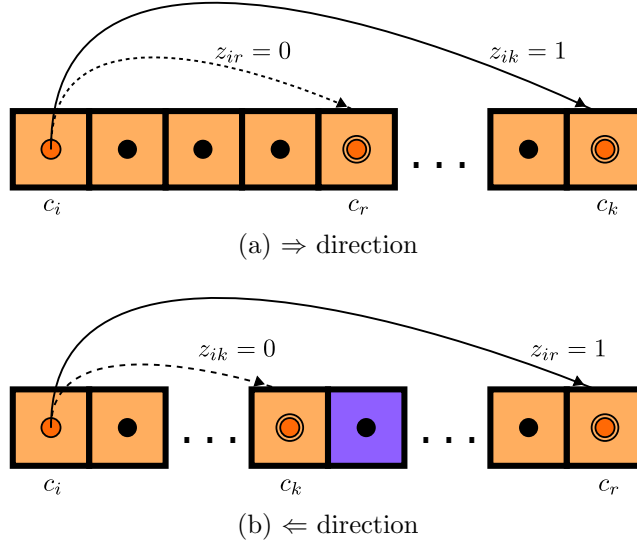


Figure 4.7: Illustrations of the cases considered for Lemma 4.1's proof.

following constraints:

$$\sum_{\forall b} z_{ib} = y_T^i, \quad \text{where } c_b \text{ is to the bottom of } c_i, \quad (4.17)$$

$$\sum_{\forall t} z_{ti} = y_B^i, \quad \text{where } c_t \text{ is to the top of } c_i \quad (4.18)$$

$$z_{ij} \in \{0, 1\} \quad \forall 1 \leq i, j \leq n. \quad (4.19)$$

Now, let $\bar{z}_{ij}$ be an input to the ILP such that $\bar{z}_{ij}$ is 1 if the initial path contains a rank with cells c_i and c_j as endpoints and 0 otherwise. From Lemma 4.1, if the ILP solution has a rank with c_i and c_j as endpoints ($z_{ij} = 1$), then it is a new rank if $\bar{z}_{ij} = 0$. Using this relation for all pairs of cells c_i and c_j with matching edges, we compute m_{new} as follows:

$$m_{\text{new}} = \sum_{\forall i} \sum_{\forall j} (1 - \bar{z}_{ij}) z_{ij}. \quad (4.20)$$

We now propose our extended ILP, namely ILP-1, that minimizes the number of ranks in the replanned solution while constraining the number of new ranks to be within a budget $\bar{m}$. We also add the term ϵm_{new} to the ILP's objective function, where $\epsilon > 0$ is used to compute the rank-minimizing solution with the smallest m_{new} . Specifically, in the case of

a large budget $\bar{m}$, the additional term helps prevent rank changes that do not improve the number of ranks in the solution. The resulting ILP is as follows:

ILP-1:

$$\min_{\mathbf{x}_H, \mathbf{x}_V, \mathbf{y}_H, \mathbf{y}_V} \quad m + \epsilon m_{\text{new}} \quad (4.21)$$

$$\text{s.t.} \quad \text{Eqs. (4.6), (4.7), (4.8),}$$

$$\text{Eqs. (4.14), (4.15), (4.17) (4.18),} \quad \forall 1 \leq i \leq n, \quad (4.22)$$

$$m_{\text{new}} \leq \bar{m},$$

$$z_{ij} \in \{0, 1\} \quad \forall 1 \leq i, j \leq n, \quad (4.23)$$

$$\mathbf{x}_H, \mathbf{x}_V \in \{0, 1\}^n, \quad (4.24)$$

$$\mathbf{y}_H, \mathbf{y}_V \in \{0, 1\}^{2n}. \quad (4.25)$$

We now show that we can configure ILP-1 to solve the rank replanning stage (Problem 4.2(i)) *optimally*, i.e. the solution (i) minimizes the number of ranks m , (ii) subject to the constraint that $m_{\text{new}} \leq \bar{m}$.

Proposition 4.1. *If $\epsilon < 1/\bar{m}$, then ILP-1 computes optimal solutions to Problem 4.2(i).*

Proof. Based on our criteria of optimality, Problem 4.2(i) can be simplified as follows:

$$\begin{aligned} \mathbf{P1:} \quad & \min \quad m \\ & \text{s.t.} \quad m_{\text{new}} \leq \bar{m}. \end{aligned}$$

Any solution to ILP-1 must satisfy Eq. 4.22 and is therefore a feasible solution to P1. What remains to be shown is whether ILP-1 computes the solution with minimum m , i.e., there exists an $\epsilon > 0$ such ILP-1 returns an optimizer to P1. To show this, consider the simplified version of ILP-1:

$$\begin{aligned} \mathbf{P2:} \quad & \min \quad m + \epsilon m_{\text{new}} \\ & \text{s.t.} \quad m_{\text{new}} \leq \bar{m}. \end{aligned}$$

Note that since the constraints of P1 and P2 are the same, they share the same feasible solution space. Let m^* and m^{**} be the number of ranks corresponding to the optimizers to P1 and P2 respectively, where m_{new}^* and m_{new}^{**} are the corresponding values of m_{new} . Clearly, $m^* \leq m^{**}$.

Let us assume, by way of contradiction, that $m^* \neq m^{**}$. Since m is an integer, we have $m^{**} - m^* \geq 1$. Since the optimizer of P1 is feasible for P2, we have the following:

$$\begin{aligned} m^* + \epsilon m_{\text{new}}^* &\geq m^{**} + \epsilon m_{\text{new}}^{**} \\ \epsilon (m_{\text{new}}^* - m_{\text{new}}^{**}) &\geq m^{**} - m^* \geq 1. \end{aligned}$$

From the constraints of P1 and P2, we have the following:

$$(m_{\text{new}}^* - m_{\text{new}}^{**}) \leq m_{\text{new}}^* \leq \bar{m}.$$

Combining the inequalities, we get:

$$\begin{aligned} \epsilon \bar{m} &\geq 1 \\ \epsilon &\geq 1/\bar{m}. \end{aligned}$$

Following this, for values of $\epsilon < 1/\bar{m}$, our assumption that $m^* \neq m^{**}$ is contradicted, which proves our proposition. We choose $\epsilon = 1/(\bar{m} + 1)$ for the rest of this work. $\square$

Remark 4.1 (Linear Relaxation). *The linear relaxation of ILP-1 is obtained by replacing the integrality constraints for the decision variables with real-valued constraints as follows:*

$$\begin{aligned} 0 &\leq z_{ij} \leq 1 \quad \forall 1 \leq i, j \leq n, \\ \mathbf{0} &\leq \mathbf{y}_H, \mathbf{y}_V \leq \mathbf{1}, \\ \mathbf{x}_H, \mathbf{x}_V &\geq \mathbf{0}. \end{aligned}$$

We observed that solving this relaxation consistently gives integral solutions despite the constraints not being totally unimodular (TUM). Our reasoning for this is that (i) the constraints of ILP-0 is TUM (following the proof from [74]), and (ii) perfect matching on a bipartite graph (e.g. graph partitioned as left endpoints (y_L^i variables) and right endpoints (y_R^i variables)) can be solved using a linear program (LP) [55]. For our experiments, we simply solve the linear relaxation as LPs can be solved in polynomial time [54].

4.4.3 On Reducing ILP Variables

In the previous section, we formulated ILP-1 which solves the rank replanning step optimally. However, one drawback of ILP-1 is that it adds a large number of decision variables ($\mathcal{O}(n^2)$) to the base ILP, which affects the runtime of the ILP. In this section, we propose another ILP extension, namely ILP-2, that uses fewer decision variables and can be more

successfully used for large environments. Instead of matching rank endpoints to count the number of new ranks m_{new} , we formulate a bound for m_{new} using the *number of rank endpoints* that have changed. Specifically, we look to identify and count two types of changes: (i) *added* endpoints and (ii) *extended* endpoints (i.e. endpoints removed when ranks from the initial solution are merged). Fig. 4.8 illustrates cases where the ranks are changed with the introduction of obstacles and the corresponding added and extended rank endpoints.

We first analyze the horizontal ranks to present our approach and extend the same approach to vertical ranks. Consider a binary vector $\bar{\mathbf{y}}_{\mathbf{H}} \in \{0, 1\}^{2n}$ that indicates if a cell was a horizontal (left or right) endpoint for a rank in the initial path. We take this vector as an input to the ILP. We look to compare vectors $\mathbf{y}_{\mathbf{H}}$ and $\bar{\mathbf{y}}_{\mathbf{H}}$ to identify added and extended endpoints.

An *added* endpoint represents an endpoint in $\mathbf{y}_{\mathbf{H}}$ but not in $\bar{\mathbf{y}}_{\mathbf{H}}$. This helps count the first four of the five possible rank changes illustrated in Fig. 4.8, as each rank after the change has at least one added endpoint. Let $\alpha_H \in \mathbb{R}$ be the number of added horizontal endpoints, which we compute as follows:

$$\alpha_H = (\mathbf{1} - \bar{\mathbf{y}}_{\mathbf{H}})^T \mathbf{y}_{\mathbf{H}}.$$

However, in the case of a merged rank (Fig. 4.8), the new rank may not contain any added endpoints. To account for this, we also count *extended* endpoints that count cells that were endpoints in $\bar{\mathbf{y}}_{\mathbf{H}}$ and are not endpoints in $\mathbf{y}_{\mathbf{H}}$ while *still maintaining the same coverage orientation* (horizontal in this case). Suppose a cell c_i was a left endpoint in $\bar{\mathbf{y}}_{\mathbf{H}}$, i.e. $\bar{y}_L^i = 1$. The cell is considered an extended endpoint if $y_L^i = 0$ and the cell is still covered horizontally ($x_H^i = 1$). We use the following piece-wise equation for cell c_i to capture this relationship.

$$\bar{y}_L^i(1 - y_L^i) + (x_H^i - 1) = \begin{cases} 1 & \text{if } c_i \text{ is an extended left endpoint} \\ \leq 0 & \text{otherwise} \end{cases}$$

Now, let $\beta_{\mathbf{H}}$ be a vector of binary variables that represents whether each cell is an extended horizontal endpoint. We use this to compute a value β_H that counts the number of extended horizontal endpoints as follows:

$$\begin{aligned} \beta_H &= \mathbf{1}^T \beta_{\mathbf{H}}, \\ \text{where } \beta_{\mathbf{H}} &\geq \bar{\mathbf{y}}_{\mathbf{H}} \circ (\mathbf{1} - \mathbf{y}_{\mathbf{H}}) + ([\mathbf{x}_{\mathbf{H}}^T \quad \mathbf{x}_{\mathbf{H}}^T]^T - \mathbf{1}), \\ \beta_{\mathbf{H}} &\in \{0, 1\}^{2n}, \end{aligned}$$

where the $\circ$ operator is the element-wise product of two vectors. The endpoint changes that remain to be counted are the cells that were endpoints in $\bar{\mathbf{y}}_H$ and are not endpoints in $\mathbf{y}_H$ *while changing coverage orientations* (to vertical in this case). However, such a cell will now be a part of a vertical rank that would either have an added or extended *vertical* rank endpoint. Therefore, we do not count these endpoints as it tracks rank changes that are already accounted for.

We extend these concepts to the vertical case to count all endpoint changes. Let $\bar{\mathbf{y}}_V$ be a binary vector indicating whether a cell was a vertical endpoint for a rank in the initial path. The total number of added endpoints α and extended endpoints β is then obtained as follows:

$$\alpha = (\mathbf{1} - \bar{\mathbf{y}}_H)^T \mathbf{y}_H + (\mathbf{1} - \bar{\mathbf{y}}_V)^T \mathbf{y}_V \quad (4.26)$$

$$\beta = \mathbf{1}^T \boldsymbol{\beta}_H + \mathbf{1}^T \boldsymbol{\beta}_V, \quad (4.27)$$

$$\text{where } \boldsymbol{\beta}_H \geq \bar{\mathbf{y}}_H \circ (\mathbf{1} - \mathbf{y}_H) + ([\mathbf{x}_H^T \ \mathbf{x}_H^T]^T - \mathbf{1}), \quad (4.28)$$

$$\boldsymbol{\beta}_V \geq \bar{\mathbf{y}}_V \circ (\mathbf{1} - \mathbf{y}_V) + ([\mathbf{x}_V^T \ \mathbf{x}_V^T]^T - \mathbf{1}), \quad (4.29)$$

$$\boldsymbol{\beta}_H, \boldsymbol{\beta}_V \in \{0, 1\}^{2n}. \quad (4.30)$$

We now look to bound the number of new ranks m_{new} using α and β , for which we exploit the following property.

Lemma 4.2. *Consider α and β to be the number of added and extended endpoints respectively for a set of replanned ranks. Then the number of new ranks m_{new} in the set is bounded as follows:*

$$m_{\text{new}} \leq \alpha + \beta/2.$$

Proof. We start by considering all possible changes to the ranks as illustrated in Fig. 4.8. For the first four possible changes, the minimum number of added endpoints for each new rank is 1. However, for a merged rank, the number of added endpoints for the new rank is 0, so we instead count the minimum number of extended endpoints. Merging two ranks can be thought of as extending two endpoints so that the ranks join to form a new rank. Therefore, a merged rank results in at least two extended endpoints. Note that merging more than 2 ranks to form one new rank results in more than 2 extended endpoints. As a result, the maximum number of merged ranks is $\beta/2$. With this, we have the following bound on m_{new} :

$$m_{\text{new}} \leq \alpha + \beta/2,$$

Rank Change	Before Interruption	After Interruption
Broken Rank		
Shortened Rank		
New Rank		
Extended Rank		
Merged Rank		

Added Endpoint

Extended Endpoint

Figure 4.8: Illustration of rank endpoint changes as a result of rank replanning. Each row shows horizontal (orange lines) and vertical (purple lines) ranks blocked by obstacles (black boxes) and the corresponding replanned ranks and endpoint changes.

which proves the lemma. $\square$

We now propose our second ILP extension. We use the bound we derived in Lemma 4.2 to constrain m_{new} in the ILP. Following Proposition 4.1, we add an additional term $\epsilon (\alpha + \beta/2)$ to the objective function, where $\epsilon = 1/(\bar{m} + 1)$, to compute a solution with the smallest m_{new} without dominating the minimization of m . The resulting ILP is as follows:

ILP-2:

$$\min_{\mathbf{x}_H, \mathbf{x}_V, \mathbf{y}_H, \mathbf{y}_V} \quad m + \epsilon (\alpha + \beta/2) \quad (4.31)$$

$$\text{s.t.} \quad \begin{aligned} & \text{Eqs. (4.6), (4.7), (4.8), (4.26) - (4.30)} \\ & \alpha + \beta/2 \leq \bar{m} \end{aligned} \quad (4.32)$$

$$\mathbf{x}_H, \mathbf{x}_V \in \{0, 1\}^n \quad (4.33)$$

$$\mathbf{y}_H, \mathbf{y}_V \in \{0, 1\}^{2n}. \quad (4.34)$$

Solving ILP-2 guarantees a solution where $m_{\text{new}} \leq \bar{m}$. Moreover, ILP-2 only adds $\mathcal{O}(n)$ decision variables to the base ILP (β_H and β_V). So, we expect ILP-2 to run faster than ILP-1. We observed that Remark 4.1 also applies for ILP-2, so we simply solve the linear relaxation to obtain integral solutions. Since there are negative terms involving $\mathbf{y}_H$ and $\mathbf{y}_V$ (Eqs. (4.28), (4.29)), the following constraint in the linear relaxation ensures that the optimization problem is bounded:

$$\mathbf{y}_H, \mathbf{y}_V \leq \mathbf{1}.$$

Unlike ILP-1, ILP-2 does not guarantee an optimal solution to the rank replanning problem, as we use a pessimistic bound on m_{new} in our ILP constraint. However, we observe in our simulation results that the paths replanned with ranks from ILP-2 are close in cost to those from ILP-1.

4.5 The TOUR-REPLAN Subroutine

In this section, we describe our approach to compute a tour of the replanned ranks to obtain our new coverage path. Similar to the approaches from [11] and [74], we formulate this as a Generalized Travelling Salesman Problem (GTSP) on a fully-connected auxiliary graph representing all connections between ranks. Each set in the graph represents a rank and consists of two vertices to represent the directions in which the rank can be traversed. We call these sets as *rank sets*. The edge costs are the *transition times* between two ranks,

i.e. the time for the robot to travel from one rank endpoint to another. Since the transition times depend on the robot’s design and dynamics, we compute the edge costs by assuming dynamics similar to [74] and [88], where (i) the robot traverses a straight line with piecewise constant acceleration with a maximum velocity and (ii) the robot stops and turns in place with a constant angular velocity. Solving GTSP on this graph obtains a tour that covers each rank while minimizing the transition times between the ranks. In Section 4.6.1, we provide more details on the robot dynamics parameters used to compute transition times in our simulations.

However, we look to address two additional challenges in the tour replan. Firstly, we look to limit the size of the GTSP instance by considering only the interrupted sections of the initial path. Fig 4.9 gives an illustration of possible path interruptions caused by (i) ranks changing orientations (ii) ranks split by an obstacle, and (iii) unreachable ranks. The ranks that are unchanged correspond with sub-sections of the initial path that are unchanged. We retain each such section by representing it as its own GTSP set, consisting of two vertices representing possible traversal directions. We refer to this set as a *path set*. The GTSP is computed on a graph containing these path sets and rank sets that each represent a new rank obtained from RANK-REPLAN to be added to the tour.

The second challenge is that we must also take into account the position of the robot and the desired end of the tour. To address this, we add artificial start and end vertices to the GTSP graph, as in Fig. 4.9. To ensure that the solution is a path from the start vertex to the end vertex, we add a directed zero-cost edge from the end vertex to the start vertex. Solving the GTSP on this graph gives us the replanned coverage path.

In this chapter, we created a fully-connected auxiliary graph. There are alternative approaches where the auxiliary graph need not be fully-connected, such as those using lazy edge evaluations to compute a tour [77]. However, the above challenges must still be addressed for such a formulation using the approach described for the fully-connected case.

4.6 Simulation Results

In this section, we evaluate the OARP-Replan approach in simulation using real-world maps. We compare the performance of OARP-Replan in online coverage replanning scenarios against greedy detour (GD) replan and other planning approaches [74, 88, 56]. We also compare OARP-Replan with offline coverage planning approaches provided with complete obstacle information [9, 5], where OARP-Replan is given no prior obstacle in-

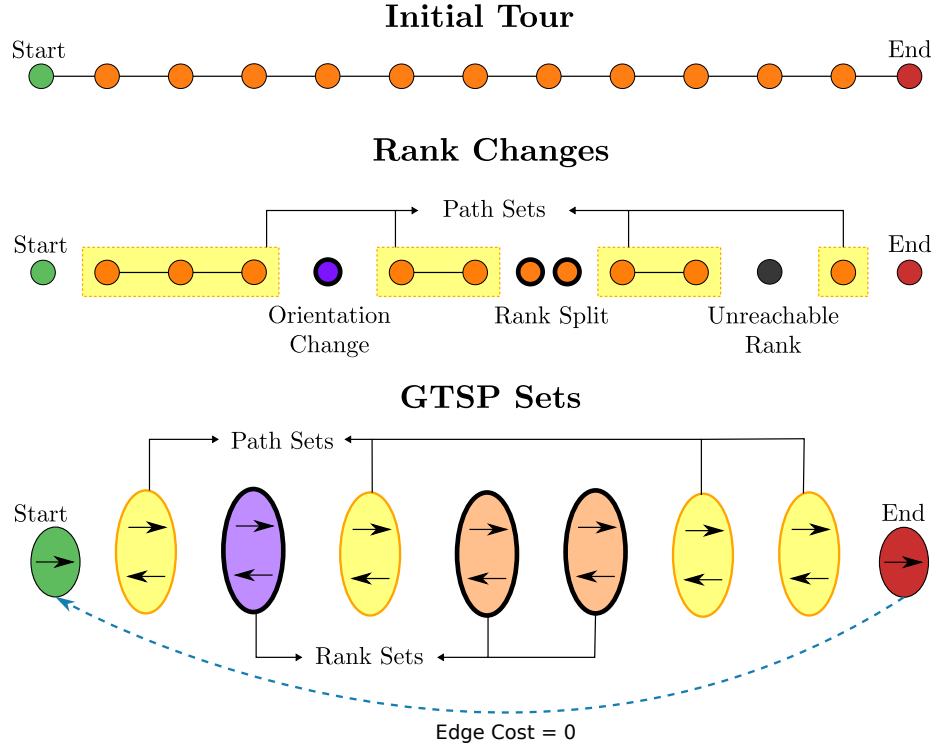


Figure 4.9: All possible changes to an example initial tour of only horizontal ranks (orange circles). The initial tour requires replanning if the ranks (i) switch orientations to form vertical ranks (purple circles), (ii) are split by obstacles, and (iii) become unreachable due to obstacles (black circle). The resulting GTSP sets are shown consisting of (i) path sets (yellow), (ii) rank sets for each new replanned rank, and (iii) artificial vertices representing the start and end of the initial path, with a directed zero-cost edge connecting them.

Table 4.1: Robot parameters used for experiments

Parameter	Value
Coverage tool width	0.8 m
Maximum linear velocity	1 m/s
Linear acceleration	$\pm 0.5 m/s^2$
Angular velocity	$30^\circ/s$
Radial sensor range	5.6 m

formation. Finally, we demonstrate using OARP-Replan to replan coverage paths in a high-fidelity ROS simulation.

4.6.1 Simulation Setup

We simulate the robot covering partially known environments: the base environment is known but there are unknown obstacles that the robot discovers during coverage using its sensor. We plan initial coverage paths for the base environments using the OARP method [74] and replan the path as new obstacles are detected.

Dataset: We use a dataset of 8 real-world 2D maps obtained from Avidbots as our base environments, with their minimum coverage path lengths ranging from 145 m to 1900 m . We then add randomly generated obstacles with varying sizes and orientations to create a set of unknown environments. Specifically, we create two datasets of unknown environments: (i) *fixed clutter* dataset where the new obstacles occupy 10% of the base map’s free area for all maps, and (ii) *varying clutter* dataset where starting with a candidate base environment, we generate unknown environments by varying the % area occupied by new obstacles. For the varying clutter dataset, we chose the base environment to be the one used in Fig. 4.1(a) and we test with 2 to 20 % area occupied. Due to the confidentiality of the environments, we are unable to share the complete dataset, but we have created anonymized environments for visualization, as seen in Figs. 4.1, 4.12, and 4.16. Fig. 4.12 shows an example base environment containing three unknown obstacles (red boxes) for which the initial coverage path (dark path) needs replanning.

Implementation: We test the two variants of OARP-Replan (Algorithm 1) based on which RANK-REPLAN method is used: (i) *OARP-Replan-1* which uses ILP-1, and (ii) *OARP-Replan-2* which uses ILP-2. Both approaches were implemented in Python. For each ILP, we solve the corresponding linear relaxation using the free SCIP solver [4].

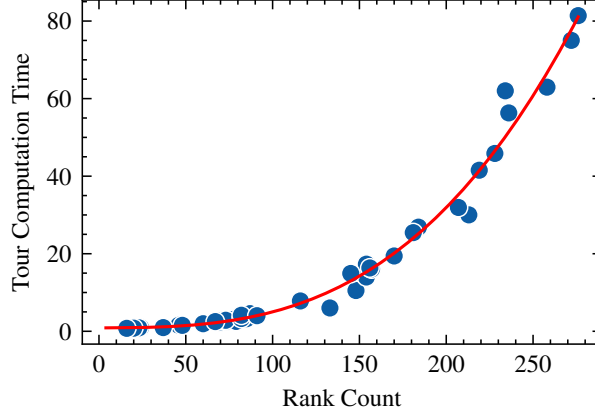


Figure 4.10: Data used to estimate TOUR-REPLAN runtime, where we plot the number of ranks (x-axis) and the time taken to compute all transitions and replan GTSP (y-axis). The red line shows the cubic polynomial fit to the data.

Then, we compute a tour of the replanned ranks by solving the GTSP formulation from Section 4.5, for which we use the open-source GLNS solver [80]. Table 4.1 shows the robot parameters used to compute the transition time between ranks. The robot is also equipped with a radial sensor in simulation (radius in Table 4.1) that detects new obstacles and runs OARP-Replan.

Metrics: Our main metric of interest is the time taken by the robot to complete coverage, i.e. *total coverage time*. This is computed as a sum of drive time and *stoppage time*, where stoppages occur when the robot is waiting for a new path to be computed. Table 4.1 shows the robot parameters used to compute the path drive time using the dynamics model from Section 4.5. To measure stoppage, we compute the minimum excess time taken by the algorithm to compute the replanned path as follows:

$$\begin{aligned} \text{Minimum stoppage time} = & \max(\{0, [\text{Time to plan transitions}] \\ & + [\text{Time to find GTSP tour}] \\ & - [\text{Time to obstacle encounter}]\}). \end{aligned}$$

Replan runtime estimator: In each simulation run, the robot conducts many replans of the coverage path depending on the obstacles it observes. For each replan, we compute a time τ to reach the next obstacle encounter along the current path using the robot model from Section 4.5. Using τ , we then determine a budget $\bar{m}$ for rank replanning using an estimator function $\hat{T}(m)$ of the runtime needed to compute a tour for m ranks. The inverse of $\hat{T}(m)$ is then used to compute the replanning budget, i.e. $\bar{m} = \hat{T}^{-1}(\tau)$.

The runtime estimate $\hat{T}(m)$ includes (i) the estimated time to compute transition paths $\hat{T}_{\text{trans}}(m)$, and (ii) the estimated time to solve the GTSP $\hat{T}_{\text{GTSP}}(m)$. Transition path computation is included as it is a time-intensive process, especially for large maps. Formally,

$$\hat{T}(m) = \hat{T}_{\text{trans}}(m) + \hat{T}_{\text{GTSP}}(m).$$

We use data collected from previous coverage planning runs to estimate each function. Our solver of choice, GLNS, typically computes a tour in $\mathcal{O}(m^2)$ time for m ranks [80]. Therefore, we estimate $\hat{T}_{\text{GTSP}}$ by fitting a cubic polynomial through the data as shown in Fig. 4.10, where we plot the time to compute the tour for different rank counts (m values). For $\hat{T}_{\text{trans}}(m)$, we use the following map-specific linear estimator. Let t_{avg} be the average time to compute the transition paths for one rank in the given map. We compute $\hat{T}_{\text{trans}}(m)$ as follows:

$$\hat{T}_{\text{trans}}(m) = t_{\text{avg}}m.$$

Under-estimating the replan runtime will result in robot stoppage, as the robot must wait for the new path to be computed. Using the above estimator with OARP-Replan in our simulations, we observed little to no such robot stoppage. Therefore, we use this simple estimator for the rest of the chapter as it works well for OARP-Replan. Regardless, this estimator may be inaccurate in cases where the time to compute transition path differs significantly from t_{avg} (e.g. time to compute transitions increase when a large number of obstacles are added to an environment). While other sophisticated estimators can be used with OARP-Replan to handle this, we do not focus on this for this chapter.

4.6.2 Comparisons with Online Approaches

In this section, we compare the total coverage time (drive time + stop time) of our two proposed OARP-Replan approaches against online coverage planning/replanning approaches. Our baseline approaches are as follows: (i) the OARP approach as proposed in [74], (ii) the heuristic (HR) approach from [88], (iii) the online search-based (SB) coverage planning approach from [56], and (iv) GD replan, i.e. the practical case of following a turn-minimizing coverage path with a local planner. The first two approaches minimize turns in the path but they are offline planners. So we modify them to be usable online for replanning as follows:

- (i) while the robot is following the path and detects new obstacle encounters, the planning approach is used to replan the ranks for the remaining coverage path, and

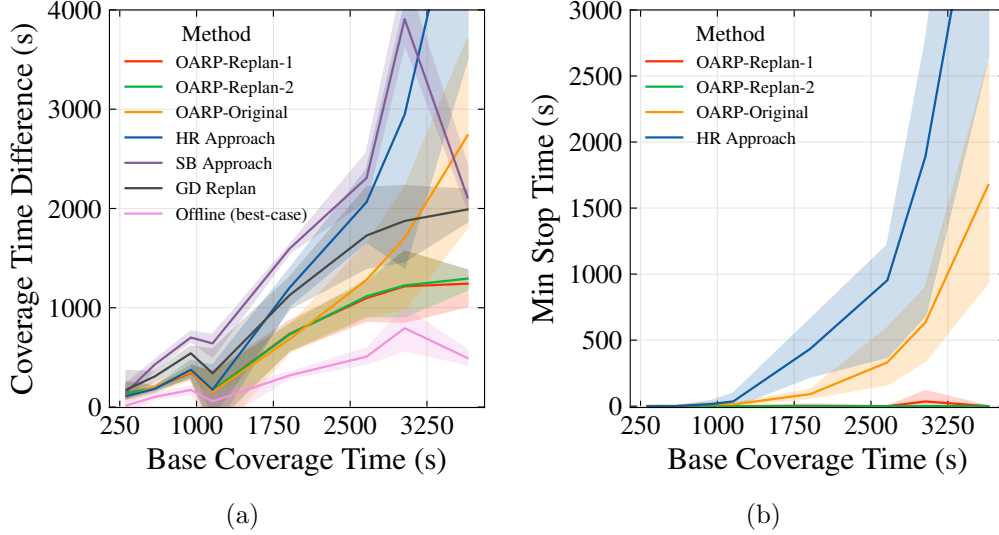


Figure 4.11: Comparison of the online replanning approaches using the fixed clutter dataset: (a) Difference between total coverage time and base coverage time. (b) Minimum stop time of the robot during coverage (GD and SB omitted as replanning time is negligible). The line shows the average and the lighter region shows the data interval.

- (ii) the tour of the resulting ranks is computed by solving the GTSP as formulated in Section 4.5, where unchanged parts of the coverage path are preserved.

For OARP, we replan coverage ranks by solving the linear relaxation of ILP-0 from Section 4.4 (identical to the LP in [74]). We refer to this online variant of OARP as *OARP-Original*. For the HR approach, we solve Algorithm 1 from [88] to replan ranks. The latter two approaches (SB and GD) are designed for online coverage planning with fast computation of coverage paths. As a result, these approaches have negligible stoppage time. The online SB approach plans the entire coverage path online and does not require an initial path. However, the SB approach minimizes path length rather than total coverage time with turns.

Fixed Clutter Dataset: We first review the results for the fixed clutter dataset. For each base environment, we ran 5 trials, where each trial was generated with a new random set of obstacles occupying 10% of the free space in the environment. The maps are sorted in the order of *base coverage time*, which is the time to cover the base environment (without the unknown obstacles) using the initial coverage path. Fig. 4.11(a) and (b) show the results of the comparison where we plot the following for each method: (a) the difference between the total coverage time (including stops) and the base coverage time (i.e. *total* –

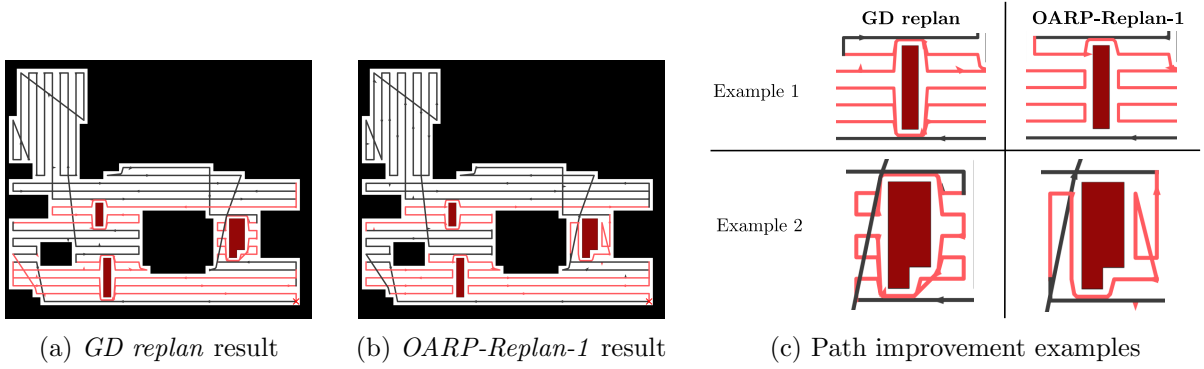


Figure 4.12: Qualitative results of replanning an initial coverage path for the example environment with unknown obstacles.

base), and (b) the minimum stop time. To visualize the *best-case* coverage time for each environment (a lower bound), we also include the results for an offline planner (OARP) when provided with all obstacle information a-priori. We observe that both OARP-Replan approaches result in lower total coverage times relative to all baseline approaches, and comes closest to the performance of the offline planner. In comparison with GD replan, OARP-Replan exhibits an average improvement of about 12% across all maps. To visualize this improvement, we show an example map with three unknown obstacles (red boxes) in Figs. 4.12(a) and (b) where the initial path was replanned using GD replan and OARP-Replan-1 respectively. Fig. 4.12(c) shows two examples where we observe qualitative differences between the approaches. In example 1, paths replanned using GD replan (red paths) have multiple long detours around the obstacle, whereas OARP-Replan-1 has no such detours. In example 2, GD replan generates messy overlapping paths when avoiding the large obstacle, while OARP-Replan-1 reorients the ranks in this region to obtain a path with fewer overlaps. Using OARP-Replan-2, we observe similar improvements in the replanned path for this example. Since the tour of the replanned ranks is computed with little to no stoppage, the replanning time does not affect the total coverage time.

When comparing with OARP-Original and the HR approach [88], both OARP-Replan approaches achieve faster coverage times, with the gap increasing as initial paths get longer (larger maps). This is because both ILP-0 and the HR approach do not limit the change to the coverage path, leading to longer replanning times and resulting robot stoppages (Fig. 4.11 (b)). In contrast, OARP-Replan approaches cause little to no stoppage. We expect this difference in performance to be exaggerated in weaker computation setups, such as onboard computers on robot platforms. The OARP-Replan approaches also outperform the SB approach as a result of minimizing the number of ranks (turns) in the coverage

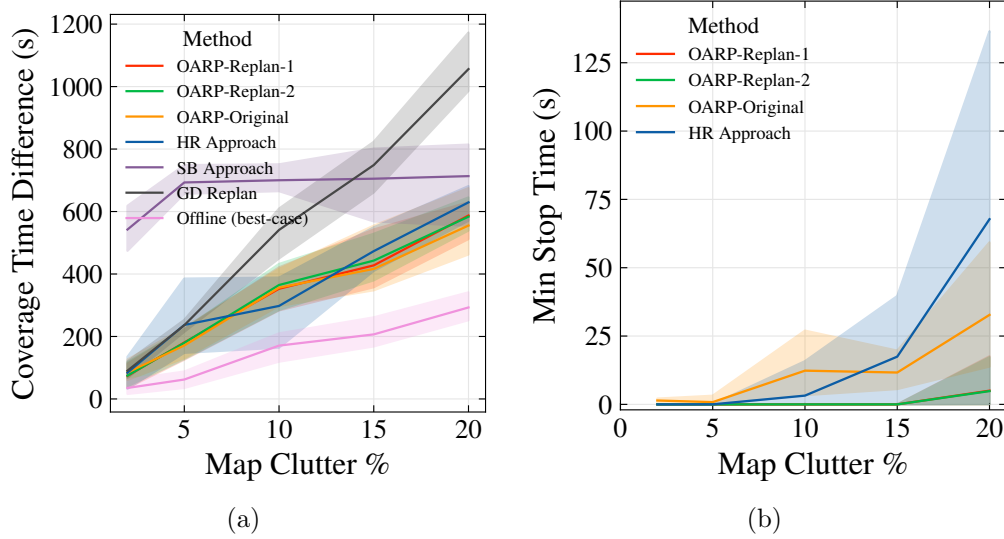


Figure 4.13: Comparison of the online replanning approaches using the varying clutter dataset: (a) Difference between total coverage time and base coverage time. (b) Minimum stop time of the robot during coverage (GD and SB omitted as replanning time is negligible). The line shows the average and the lighter region shows the data interval.

path. However, we observe that the SB approach performs well on the map with the longest base coverage time when compared to the other baselines. Note that this map is not the largest in terms of area, so we reason that the performance of SB is more closely related to map area than to base coverage time (which may be influenced by complex layouts/geometry). Between the OARP-Replan approaches, OARP-Replan-1 consistently plans paths with slightly shorter coverage times than OARP-Replan-2.

Varying Clutter Dataset: We also conducted similar experiments using the varying clutter dataset, where we ran 5 trials for each clutter level. The map chosen for this dataset has a base coverage time of 944s. Fig. 4.13 shows the results of the coverage time comparisons. In Fig. 4.13(a), we plot the difference between the total coverage time and base coverage time. We observe that the OARP-Replan approaches outperform GD replan by 18% on average, with the performance gap increasing with clutter level. This is because, with more obstacles, GD replan causes many detours, while the OARP-Replan approaches tend to shorten such detours.

In comparison with OARP-Original and the HR approach [88], the OARP-Replan approaches plan coverage paths with comparable coverage times. However, both OARP-Original and HR experience longer stops with increasing clutter (Fig. 4.13(b)). We also

observe some stoppage with using the OARP-Replan approaches in environments of 20% clutter. We believe this is because we do not update our runtime estimator during an experiment, making it inaccurate for highly cluttered environments (e.g. computing transitions take more time with more obstacles). A more sophisticated estimator that updates itself during coverage may eliminate this stoppage. We also observe that the coverage time using the SB approach plateaus with increasing clutter due to SB’s rapid online computation of coverage paths. Although OARP-Replan outperforms SB at all the clutter levels tested, this performance gap may decrease with higher clutter levels ($> 20\%$).

4.6.3 ILP Runtime Comparisons

In this section, we compare the average runtimes of the rank replanning ILPs for both OARP-Replan approaches. Our approach relies on the ILP being solved efficiently so that the GTSP tour can be reliably computed within the time budget (preventing robot stoppages). The ILP from OARP-Original is excluded from this comparison as it does not constrain the GTSP input size.

Fig. 4.14(a) and (b) show the comparison results in the fixed clutter and varying clutter datasets respectively, where we plotted the average ILP runtimes for each map. For the fixed clutter dataset, the maps are sorted in the order of increasing size (area) of the environment, and in-turn, the number of grid cells in the IOP. We observe that ILP-2’s runtime scales better than ILP-1 as the environment area increases. This is mainly due to the reduction in the ILP variables that speed up runtime. Moreover, we also observe in Fig 4.11(b) that using ILP-2 resulted in slightly shorter stop times for larger maps (the plot for OARP-Replan-2 is close to zero while OARP-Replan-1 goes up to 100s for one map). We believe that this difference will be exaggerated when computation power is constrained (e.g. low-cost robot hardware) to the point that it will affect coverage time. Therefore, for such applications, we recommend ILP-2 over ILP-1. A future improvement of this work could involve combining both ILPs, with ILP-2 as a fallback for ILP-1.

For the varying clutter dataset, we observe that the average runtimes of both ILPs are comparable, with ILP-1 performing better overall. Increasing clutter seems to slightly change both ILPs’ runtimes, and as such, we observed no drastic runtime difference as with the fixed clutter dataset. While this comparison was performed with a fixed tool width (grid cell size), changing the tool width could either increase or decrease the number of grid cells for all maps. However, since the change in map area also has the same effect, we expect the observed trend to be maintained with a change in tool width.

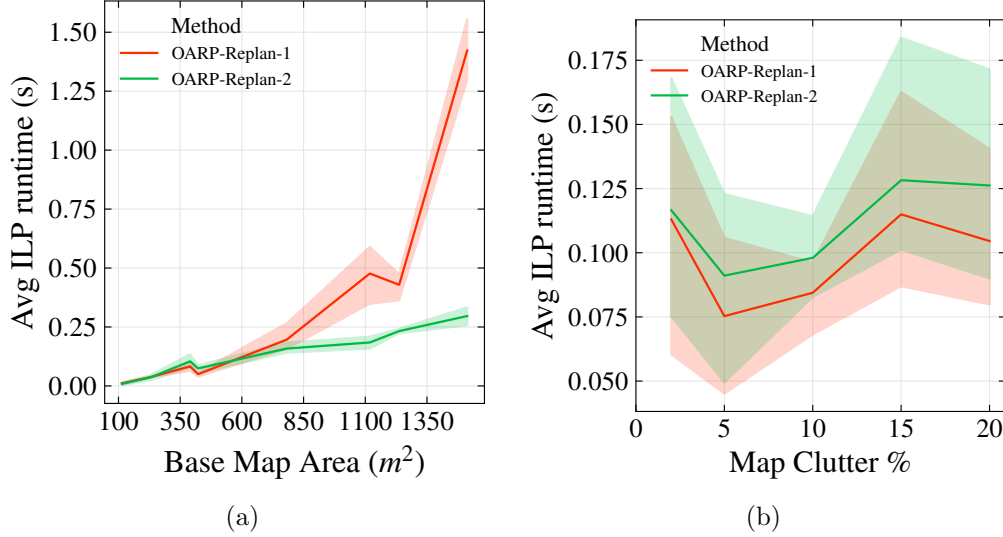


Figure 4.14: Comparison of average ILP runtimes for the OARP-Replan approaches. (a) Fixed clutter dataset comparisons, with maps sorted by *base map area*. (b) Varying clutter dataset comparisons. The line shows the average and the lighter region shows the data interval.

4.6.4 Comparisons with Offline Approaches

In this section, we compare the proposed OARP-Replan approach (i.e. OARP-Replan-1) against *offline* coverage planning approaches, which are provided with all obstacle information prior to coverage. In contrast, we provide OARP-Replan with no prior obstacle information, so it must replan the coverage path whenever the robot detects obstacles. We compare OARP-Replan against 3 offline approaches: (i) boustrophedon cell decomposition (BCD) method [9], (ii) line coverage (LC) approach [5], and (iii) the offline OARP planner [74]. For clarity, we will refer to the offline approaches as *BCD-Offline*, *LC-Offline*, and *OARP-Offline* for the rest of this section. The BCD-Offline algorithm also resembles the BCD approach from the ‘ipa_room_exploration’ ROS package, which is the most effective algorithm in the package according to the adjoining survey in [12].

While OARP-Replan was developed with mobile robots as a focus, it can also be used for coverage planning with aerial robots (fixed coverage tool width). As such, we apply the OARP-Replan approach to plan coverage paths for an aerial robot where the obstacles in the environment are not previously known. We use the dataset from the BCD paper [9] that consists of around 300 scans of environments with varying numbers of obstacles or *holes*. The outer boundary of each map is a square, which is then filled with the obstacles

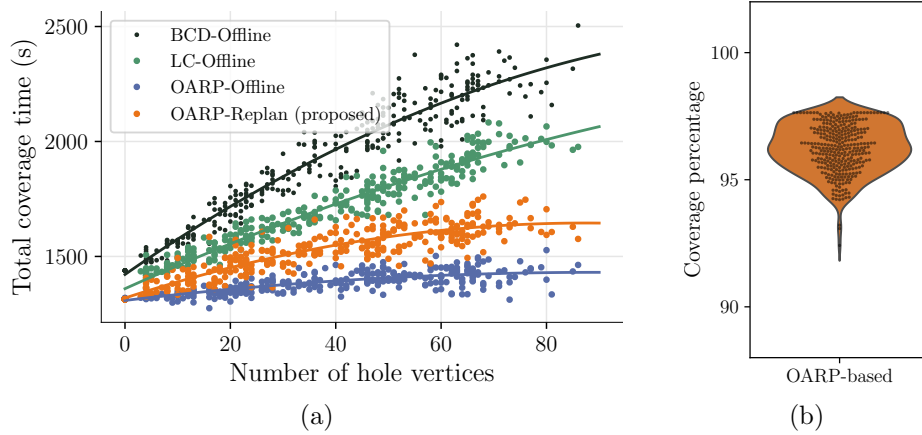


Figure 4.15: Comparison results with offline planning approaches. (a) Coverage time comparisons between OARP-Replan and offline baselines. (b) Violin plot of percent coverage achieved by the OARP-based approaches (with data points plotted). BCD-Offline and LC-Offline are omitted as they are exact approaches that achieve 100% coverage.

observed in the scans. The maps are arranged in the order of increasing complexity, which is given by the number of *hole vertices* (e.g., one square hole consists of 4 vertices). We also use robot parameters similar to that of the UAV from [9]: a tool width of 3 *m*, maximum velocity of 3 *m/s*, and linear acceleration of $\pm 1 \text{ m/s}^2$ (turning time omitted to match [9]). Additionally, OARP-Replan is provided with an initial path for the unobstructed base environment (a square), and a radial range sensor similar to previous sections to identify obstacles.

Fig. 4.15(a) shows the results of comparing the total coverage times of using all approaches. We observe that OARP-Replan outperforms BCD-Offline and LC-Offline across all maps, with an improvement of about 25% on average from BCD-Offline. We believe this improvement to be due to the optimal coverage lines produced by both OARP-based methods, as BCD-Offline paths tend to overlap closer to cell boundaries and obstacles. We also observe that OARP-Offline exhibits the best performance with an average improvement of 31% in coverage time from BCD-Offline. However, note that OARP-Offline is equivalent to OARP-Replan when provided with all obstacle information a-priori and unlimited solver time. We conclude that using OARP-Replan with a simple initial path is a competitive alternative to using BCD-Offline with complete obstacle information. However, if obstacle information is available beforehand, OARP-Offline is a better alternative as it computes paths with the fastest coverage times.

One thing to note is that both BCD-Offline and LC-Offline are exact approaches that

aim to cover the entire environment (100% coverage). In contrast, the OARP-based approaches use grid approximations that may not cover the environment fully. To address this, we plot the percent coverage achieved by both OARP-based methods (they cover the same grid) in Fig. 4.15(b), where we observe that the OARP-based approaches achieve about 96% coverage on average across all environments, with missed coverage usually occurring close to obstacles and boundaries. Therefore, we recommend OARP-Replan for covering environments with unknown obstacles where small amounts of missed coverage is acceptable.

4.6.5 Simulation Case Study

This section provides a case study of a Turtlebot 3 replanning coverage paths in a ROS simulation using GD replan and OARP-Replan (i.e. OARP-Replan-1). This is to demonstrate the real-world applicability of our replanning approach in contrast to GD replanning. We generated an anonymized environment as shown in Fig. 4.16, for which an initial coverage path was planned (Fig. 4.16(a)). We added new obstacles to the environment that occupied 10% of the initial area. As the robot observes new obstacles blocking this path, the robot replans the path either using OARP-Replan or GD replan. The replanning is done while driving and the robot switches to the new path when available. The robot is also equipped with a local planner to traverse the path and make small changes if the path goes close to the walls. Fig. 4.16(b) shows a snapshot of the robot replanning the coverage path using OARP-Replan. We use the robot parameters from Table 4.1 for this simulation with some changes to use the default Turtlebot controller in ROS: we set maximum velocity to 0.3 m/s and linear acceleration to 1 m/s^2 . With these parameters, we observed that OARP-Replan covered the environment 16.5% faster than GD replan along a path that was 12.2% shorter. This difference is mainly due to the reduction in long detours where the robot travels around obstacles many times. We included recordings of this case study in our video attachment.

4.7 Robot Experiments

In this section, we present the results of testing OARP-Replan (i.e. OARP-Replan-1) on an Avidbots Neo robot (Fig. 4.17(a)) covering a partially unknown environment. The robot's task is to cover a test environment for which an initial path is provided (Fig. 4.18(a)). The robot has tricycle dynamics and is equipped with a local planner to make small changes to difficult parts of the initial path. However, the robot may encounter large obstacles during



Figure 4.16: Robot operating system (ROS) simulation of covering an example environment with unknown obstacles. (a) An initial coverage path for the base environment (b) replanning the path online using OARP-Replan.

coverage (Fig. 4.17(b)) which may result in narrow regions that the robot cannot reach along its current path. Therefore, a replan of the coverage path is necessary to reach such regions and maximize the coverage of the environment.

We conducted our experiments using two test scenarios. Fig. 4.18 shows the results of applying OARP-Replan in both scenarios. In scenario 1 (Fig. 4.17(a) / Fig. 4.18(b)), the robot observes an obstacle that interrupts a large number of ranks in the initial path. The path is replanned using OARP-Replan, which computes horizontal ranks to comfortably cover the narrow region (bottom right of the environment in Fig. 4.18(b)). In scenario 2 (Fig. 4.17(b) / Fig. 4.18(c)), the robot encounters a large obstacle that it must observe over time. Later in the task, the robot observes a part of the obstacle that causes a narrow coverage region (top right of the environment in Fig. 4.18(c)). OARP-Replan allows the robot to cover this region by re-orienting the coverage ranks. We provide recordings of these experiments in our multimedia attachment.

To implement OARP-Replan, we require an occupancy map that uses the LIDAR scan collected by the robot to identify grid cells that correspond to static obstacles. However, this occupancy map must differentiate between a static obstacle and a noisy LIDAR scan. We do this by checking consecutive LIDAR readings for obstacles that block grid cells in the occupancy grid and has not moved for some time. While this step increases the time between detecting an obstacle and replanning, it is necessary to prevent faulty replanning of the coverage path as a result of LIDAR scan noise/fluctuations. This provides a sim-

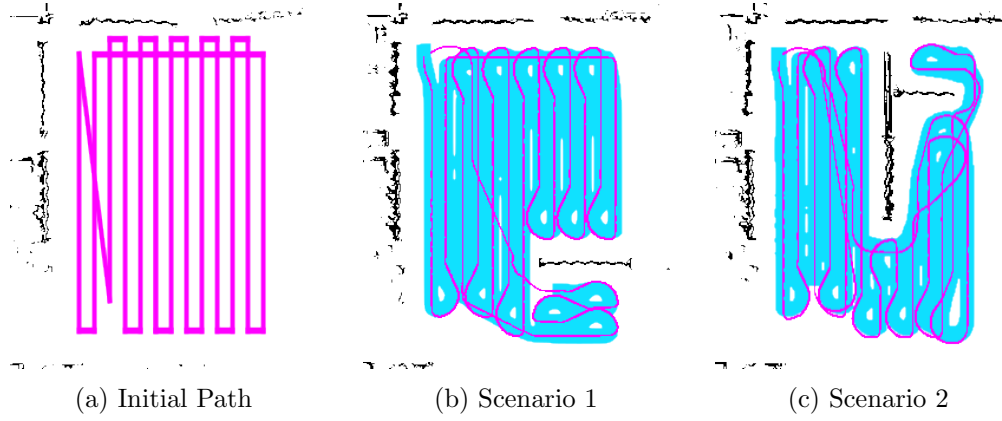


(a) Neo robot in Scenario 1



(b) Scenario 2 obstacles

Figure 4.17: Snapshots from experiments using the Avidbots Neo robot



(a) Initial Path

(b) Scenario 1

(c) Scenario 2

Figure 4.18: Robot experiment results of replanning an initial coverage path using OARP-Replan. The final robot path (pink curves) and the covered area (blue) are shown for both scenarios.

ple guideline to handle sensor noise, however more sophisticated approaches can be used instead to filter noisy sensor data.

Note that without replanning (i.e. GD replan with the local planner), the coverage path for the narrow regions will have short ranks with turns occurring in quick succession. The Neo robot cannot drive such paths without manual assistance. The proposed replanning approach aims to reduce such instances where manual assistance is required to continue coverage.

4.8 Summary

We proposed an anytime coverage replanning approach called OARP-Replan to replan coverage paths for environments with unknown static obstacles. OARP-Replan solves this problem in two stages: (i) rank replanning, and (ii) touring. The first stage replans the ranks (straight-line paths) of an obstructed coverage path, while the second stage computes a tour of the replanned ranks to obtain the new path. We proposed two integer linear programs (ILPs) that each solve the rank replanning stage so that the resulting touring stage is completed within a time budget τ . Simulations on maps of real-world environments showed that the OARP-Replan variants using each ILP achieve better coverage performance than a greedy replanner and state-of-the-art coverage planners used for online replanning [74, 88, 56]. Moreover, we also compared OARP-Replan with offline planning approaches (treating all obstacles as known) [9, 5] where we observed an improvement in total coverage time. Finally, we demonstrated the proposed approach by replanning coverage paths for an industrial-level cleaning robot.

In this work, we focus only on the case of new obstacles in the environment. The case where an obstacle is observed and later removed from the environment (including dynamic obstacles) is left for future work. This would involve detecting areas in the environment that could be free at a later time due to an obstacle removal and returning to cover them without long detours. Another future direction involves replanning using *predictions* about environment occupancy to balance coverage and exploration, which are competing objectives. To balance them, the coverage planner must determine *when* to replan the coverage path, perhaps even delaying a replan until more information about an obstacle is collected.

In the rest of the thesis, we will address two issues with the OARP-based frameworks: (i) obtaining guarantees on the computed coverage path, and (ii) removing the axis-parallel constraint on the coverage lines.

Chapter 5

CPP with Approximation Guarantees

We now shift our focus to improving the coverage replanning by analysing the length of OARP coverage paths and establishing an approximation factor for the OARP approach. We then propose an improved approximation algorithm to connect axis-parallel coverage lines into a minimum-length tour. The proposed approach can also be used lazily, i.e., without evaluating all possible transition paths. This offers a speed-up that can be useful to replan more of the coverage path upon detecting unknown obstacles.

5.1 Introduction

For general environments, CPP is an NP-hard problem [8], thus optimal planning approaches do not scale well with increasing problem complexity. So, as we discussed in Chapter 3, CPP algorithms [22, 5, 88, 74] usually follow the following framework, where they: (i) decompose the environment into sub-regions, (ii) compute an optimal set of parallel straight-line paths or *coverage lines* to cover each sub-region, and then (iii) compute a cost-minimizing tour of the coverage lines to form a coverage path. In this chapter, we refer to this approach as the *line-touring framework*. This framework is commonly used to minimize the number of turns in the coverage path as (i) turns take time to execute while adding little to no new coverage, and (ii) tools like cleaning attachments miss coverage during turns [5, 88, 74, 7].

While this framework makes the problem more tractable, it, in general, computes paths with suboptimal length. This is acceptable as long as the computed solution is “good enough” with a known approximation factor. While approximation algorithms for CPP

have been studied [8, 29], an approximation factor for the line-touring framework has not been analyzed. In this work, we study the line-touring framework applied to grid decompositions, where the environment is represented as a set of square grid cells. Each coverage line covers a *rank*, i.e., a group of grid cells aligned along an axis-parallel (horizontal/vertical) straight line. As such, one can obtain the coverage lines for a grid decomposition by partitioning it into these ranks. We refer to this variation of the line-touring framework as *rank-touring* [88, 74, 31].

In this chapter, we show that the rank-touring framework achieves an approximation factor of $(1 + 1.5\gamma)$ for path length, where $\gamma > 1$ is the approximation factor to solve the metric traveling salesman problem (metric-TSP). Following this, we propose a $(1.5 + \epsilon)$ -approximation algorithm, namely Minimum Length Coverage Approx (MLC-Approx), for minimum-length CPP in connected grid environments with holes. Here, $\epsilon \ll 1$ is a small term proportional to the number of ranks. Building upon the rank-touring framework, MLC-Approx relaxes the turn minimization objective to compute paths with near-optimal length, while still achieving fewer turns than a TSP planner.

Chapter contributions: Our specific contributions are as follows:

1. We show that the rank-touring framework for minimum-length CPP achieves an approximation factor of $(1 + 1.5\gamma)$, where $\gamma > 1$ is the approximation factor to solve metric-TSP,
2. For grid environments with holes, we propose a $(1.5 + \epsilon)$ -approximation algorithm for minimum-length CPP, which also minimizes the number of turns as a secondary objective,
3. We propose a lazy variation of the $(1.5 + \epsilon)$ -approximation algorithm that achieves the same result with improved runtime.

We also evaluate MLC-Approx using maps of real-world environments and compare against state-of-the-art approaches that employ the rank-touring [88, 74], and line-touring [9] frameworks.

Recent turn-minimizing approximate approaches have applied the rank-touring framework to minimize turns in the coverage path [88, 74]. In [74], the authors proposed the optimal axis-parallel rank partitioning (OARP) approach that guarantees the minimum number of axis-parallel ranks in the resulting coverage path. In this work, we provide an approximation factor for these approaches and propose an algorithm with a tighter guarantee.

Organization: In Section 5.2 we define CPP and the minimum-length rank-based CPP problem that we look to solve in this paper. In Section 5.3, we establish an approximation factor of $(1 + 1.5\gamma)$ for the rank-touring framework to solve the minimum-length CPP problem. In Section 5.4, we propose MLC-Approx and show that it achieves an improved approximation factor of $(1.5 + \epsilon)$, where $\epsilon \ll 1$. In Section 5.5, we introduce a lazy variation of MLC-Approx to improve CPP computation time. In Section 5.6, we provide simulation results where we compare MLC-Approx with baseline approaches that follow the same framework. Section 5.7 concludes the paper.

5.2 Problem Definition

5.2.1 Rank-touring Framework

We start by clarifying the notation for the rank-touring framework we will use in this chapter. As discussed in previous chapters, the rank-touring framework starts by partitioning the IOP $\widetilde{\mathcal{W}}$ into non-overlapping axis-parallel (i.e., horizontal/vertical) *ranks* $R = \{r_1, r_2, \dots, r_m\}$, where each rank $r_i \in R$ is a set of grid cells that can be covered in a straight-line sweep using a square tool of width l . Let P_R be the set of coverage lines that each correspond to a rank in R . For example, the coverage line of a horizontal rank r_i is a path connecting the centers of its leftmost and rightmost grid cells. The framework then computes a cost-minimizing tour of the paths in P_R to obtain a coverage path for the IOP. The resulting coverage path P is a sequence of coverage lines in P_R and collision-free *transition paths* connecting two coverage lines at their endpoints. The aim of the tour is to minimize the cost $J(P)$ associated with the transition paths.

5.2.2 Rank-based Coverage with Minimum Length

For this work, we consider the primary cost $J(P)$ of the coverage path P to be the path length $L(P)$, with the number of turns $\Theta(P)$ as a secondary cost. To minimize $\Theta(P)$, we partition $\widetilde{\mathcal{W}}$ into axis-parallel ranks R such that the number of ranks m is minimized. We refer to this partition as the *minimum rank partition* of $\widetilde{\mathcal{W}}$.

The minimum rank partition of $\widetilde{\mathcal{W}}$ can be computed using approaches in the literature [74, 88]. From this partition, one can obtain the coverage lines P_R and a coverage path P for $\widetilde{\mathcal{W}}$ using the framework. In this work, we will focus on minimizing the length $L(P)$

of this path resulting from a minimum rank partition R and develop an approximation scheme for this problem.

Problem 5.1 (Minimum-length Rank-based Coverage). *Given an approximation $\widetilde{\mathcal{W}}$ of the environment $\mathcal{W}$, a coverage tool of width l and a minimum rank partition R of $\widetilde{\mathcal{W}}$, compute a coverage path $P \in \mathcal{P}$ that solves:*

$$\begin{aligned} \min_{P \in \mathcal{P}} \quad & L(P) \\ \text{s.t.} \quad & \bigcup_{(\mathbf{x}, \theta) \in P} \mathcal{A}(\mathbf{x}, \theta) = \widetilde{\mathcal{W}}. \end{aligned}$$

Unlike the rank-touring framework, our solution approach to this problem allows changes to the ranks (e.g., breaking) to compute near-optimal solutions in the search space. Regarding the number of turns $\Theta(P)$, we focus on rank changes that add the fewest turns to the resulting P . Note that using the minimum rank partition R also helps to minimize $\Theta(P)$ as it reduces the number of coverage lines.

5.3 Approximation Factor for Rank-Touring

We first analyze the optimality of the rank-touring framework to plan minimum-length coverage paths for the environment's IOP $\widetilde{\mathcal{W}}$ (Problem 5.1). Since it is a step-wise approach, the framework may remove optimal paths from its solution space. So, we obtain an approximation factor that quantifies the length of the resulting path in comparison to the optimal coverage path length for $\widetilde{\mathcal{W}}$.

Given a coverage tool of width l , let n be the number of grid cells in $\widetilde{\mathcal{W}}$, and $R = \{r_1, \dots, r_m\}$ be any rank partition of $\widetilde{\mathcal{W}}$ (does not necessarily minimize m). Each $r_i \in R$ is a horizontal/vertical rank that contains $n_i \leq n$ grid cells. Since R is a partition, $\sum_{i=1}^m n_i = n$. Let $P_R = \{p_1, \dots, p_m\}$ be the set of coverage lines obtained from the ranks R , where $p_i \in P_R$ is the coverage line corresponding to $r_i \in R$. Since p_i is a straight line covering n_i grid cells (i.e., connecting the centers of the end grid cells in a row), the coverage line will have the length

$$L(p_i) = (n_i - 1)l. \tag{5.1}$$

Let P^* be the optimal coverage path for $\widetilde{\mathcal{W}}$ of length $L(P^*)$. To lower-bound $L(P^*)$, consider the length of a coverage path that covers every cell without any overlap (e.g., cells

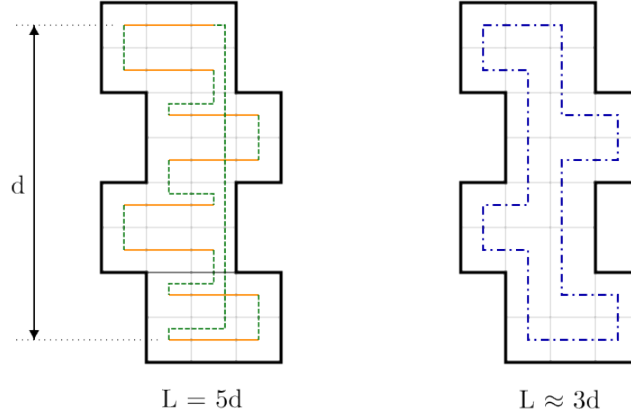


Figure 5.1: An IOP where a rank-touring algorithm (left) computes a path of length $5/3$ times the optimal coverage path length (right).

arranged in a loop). This gives

$$L(P^*) \geq \frac{\text{Area}}{l} = \frac{nl^2}{l} = nl. \quad (5.2)$$

We now derive an approximation factor for the rank-touring framework. Our derivation assumes access to an approximation algorithm to solve the metric traveling salesman problem (metric-TSP), where the edge costs of the graph satisfy the triangle inequality. Let γ be the approximation factor of this TSP solver.

Theorem 5.1. *Given an IOP $\widetilde{\mathcal{W}}$, the rank-touring framework computes a coverage path P of length $L(P)$ satisfying*

$$L(P) \leq (1 + 1.5\gamma)L(P^*),$$

where P^* is the optimal coverage path of $\widetilde{\mathcal{W}}$.

Proof. Let $R = \{r_1, \dots, r_m\}$ be a rank partition of $\widetilde{\mathcal{W}}$, and let $P_R = \{p_1, \dots, p_m\}$ be the resulting coverage line set. The rank-touring framework computes a coverage path P by connecting the coverage lines in P_R using a set of transition paths P_T . The resulting length of P is

$$L(P) = L(P_R) + L(P_T) = \sum_{i=1}^m L(p_i) + L(P_T).$$

We bound $L(P)$ by bounding the above sum of path lengths. Following Eqs. (5.1) and (5.2), we have

$$\sum_{i=1}^m L(p_i) = (n - m)l \leq nl \leq L(P^*).$$

Now, consider a tour T_e of length $L(T_e)$ over the *endpoints* of P_R computed by the γ -approximation metric-TSP solver using collision-free paths in $\widetilde{\mathcal{W}}$. Note that T_e does not necessarily connect the coverage line set P_R to form a coverage path. Let T_e^* be the optimal TSP tour of these endpoints. Then, we have

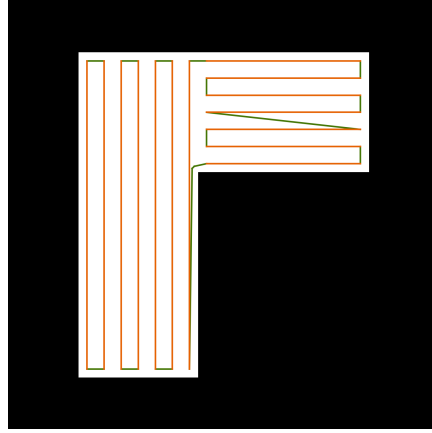
$$L(T_e) \leq \gamma L(T_e^*) \leq \gamma L(P^*).$$

The second inequality follows as T_e^* covers fewer grid cells than P^* (covers all cells). Consider the weighted graph $\bar{G}$ induced by T_e , where the edge costs are the path lengths between endpoints. To this, we add edges corresponding to the coverage lines P_R . The resulting $\bar{G}$ has a total edge cost $\leq (1 + \gamma)L(P^*)$ and all of its vertices have odd degree. However, the edges of T_e form two perfect matchings of the endpoints of P_R . Let M^* be the matching with cost $\leq L(T_e)/2$. Adding the edges of M^* to $\bar{G}$ gives a graph with all even degree vertices containing an Eulerian cycle of cost $\leq (1 + 1.5\gamma)L(P^*)$. This proves the theorem, as this cycle's cost is a bound on $L(P)$. $\square$

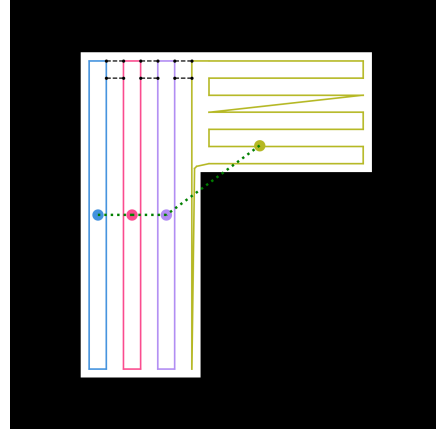
Obtaining a tight example for this bound is non-trivial without considering environments with unrealistic perturbations. However, we provide an algorithm for which this factor must be at least $5/3$.

Proposition 5.1. *There exists an IOP $\widetilde{\mathcal{W}}$ for which a rank-touring algorithm computes a coverage path of length $5/3$ times the optimal coverage path length.*

Proof. Consider the IOP in Fig. 5.1, where d is its height measured between the center points of the top and bottom grid cells. We can stack copies of this IOP vertically to obtain a new IOP with an arbitrarily large d . Now, consider a rank-touring algorithm that (i) partitions the IOP into the minimum number of *horizontal* ranks, and (ii) computes a minimum-length tour connecting the coverage lines. The resulting coverage path, shown in Fig. 5.1 (left), has length $5d$ for a large d . However, the IOP allows a Hamiltonian cycle (right of Fig. 5.1) with length $3d + 3l \approx 3d$, i.e., the optimal coverage path. Therefore, the algorithm computes a coverage path of length $5/3$ times the optimal. $\square$



(a) Coverage ranks (orange) with matching edges (green)



(b) Sub-tour merging with spanning tree

Figure 5.2: MLC-Approx stages: (a) obtain coverage lines (orange) from the minimum rank partition and compute min-length perfect matching paths (green), and (b) identify the resulting sub-tours (colored paths) and evaluate possible merges (black lines) as determined by an ordering (dotted green lines).

5.4 A $(1.5 + \epsilon)$ -Approximation Algorithm

We now present a $(1.5 + \epsilon)$ -approximation algorithm for Problem 5.1, called the Minimum Length Coverage Approx (MLC-Approx) algorithm. MLC-Approx modifies the touring step in the rank-touring framework to improve the guarantee on the path length. Instead of connecting ranks at their endpoints, MLC-Approx allows the breaking and merging of neighboring ranks while minimizing the path length added to the final coverage path. We consider that the IOP is *connected*, i.e., there exists a path between every pair of grid cells with *axis-parallel* motions.

5.4.1 General Framework

A pseudocode of the coverage planning approach is given in Algorithm 2. We start by computing the *minimum* rank partition R of the IOP $\mathcal{W}$ and obtain the resulting coverage lines P_R for the IOP. For this, we use the approach proposed in [74], which utilizes a linear program (LP) to compute the optimal partition in polynomial time. Fig. 5.2(a) shows the resulting set of coverage lines for an example IOP.

Let V be the set of all the rank endpoints, where each $v \in V$ corresponds to the center of a grid cell at one end of a rank. We compute a *perfect matching* of V that minimizes the length of the paths connecting the endpoints in the matching (Line 1). For this, we use a transition graph $G = (V, E)$ where the edges E represent collision-free *transition paths* between two endpoints with the edge weights being the lengths of the transition paths. The result is a set M of matching paths connecting pairs of rank endpoints. Together, the matching paths M and the coverage lines P_R form a set $\mathcal{T}$ of *sub-tours*, i.e., closed paths that collectively cover all cells in the IOP, where $|\mathcal{T}| \geq 1$ (Line 2). Fig. 5.2 shows an example set of sub-tours, where the coverage lines (orange) and matching paths (green) together form a set of sub-tours (colored paths in Fig. 5.2(b)) covering the IOP.

Algorithm 2 MLC-Approx

Input: Environment IOP $\widetilde{W}$, Coverage lines P_R from rank partition, Transition Graph $G = (V, E)$ of rank endpoints

Output: Coverage Path P

- 1: $M \leftarrow$ A min-length perfect matching of V using G
 - 2: $\mathcal{T} \leftarrow$ Sub-tours obtained from combining P_R and M
 - 3: $S_{\mathcal{T}} \leftarrow \text{SUBTOURORDERING}(\mathcal{T})$ $\triangleright$ Ordering of adjacent sub-tour pairs to be merged
 - 4: **for** $(S_i, S_j) \in S_{\mathcal{T}}$ **do**
 - 5: Merge S_i with S_j $\triangleright$ See Section 5.4.2
 - 6: **end for**
 - 7: $P \leftarrow$ Final tour after merging all sub-tours
 - 8: **return** P
-

We now look to merge the sub-tours in $\mathcal{T}$ into a single coverage path P . For this, we use the connectivity of the IOP and the resulting adjacency of the sub-tours.

Definition 5.1 (Adjacent Sub-tours). *Two sub-tours are adjacent if they each cover grid cells that are axis-parallel (horizontal/vertical) neighbors of each other.*

Since the IOP is connected, every sub-tour in $\mathcal{T}$ is adjacent to at least one other sub-tour. Therefore, we use a sub-routine named $\text{SUBTOURORDERING}(\mathcal{T})$ to determine an ordering of adjacent sub-tour pairs that can be merged to obtain P (Line 3). In our implementation of $\text{SUBTOURORDERING}(\mathcal{T})$, we compute a *spanning tree* in an adjacency graph $G' = (V', E')$, where each sub-tour is represented by a vertex $v \in V'$ and edges $e \in E'$ connect two adjacent sub-tours. Fig. 5.2(b) shows the spanning tree of sub-tours in an example environment. We use the edges of this spanning tree to determine pairs of adjacent sub-tours that can be merged in any order to obtain P (Lines 4-7). In the next sub-section, we address how two sub-tours are merged in MLC-Approx.

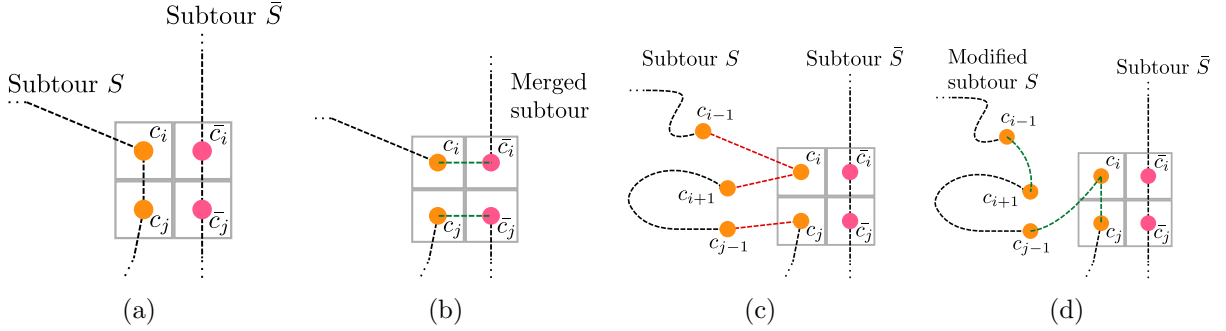


Figure 5.3: Corner junction merging: (a) Corner junction formed by 4 grid cells. (b) Edge swap to merge two sub-tours at the junction. (c & d) Modifying sub-tour S to allow an edge swap by removing the red edges and adding the green edges.

5.4.2 Sub-tour Merging

We propose two strategies to merge adjacent sub-tours, namely *double edge* merging and *corner junction* merging. In both strategies, we exploit the layout of the adjacent (neighboring) grid cells covered by two sub-tours to determine a merging strategy that limits the path length added. For the rest of this section, let the two sub-tours to be merged be represented by S and $\bar{S}$. With some abuse of notation, we say that a grid cell $c_i \in S$ if sub-tour S covers the cell c_i .

Double Edge Merging: A simple strategy to merge sub-tours is to add a *double edge* between the two adjacent grid cells $c_i \in S$ and $\bar{c}_i \in \bar{S}$. Specifically, we add a traversal path from c_i to $\bar{c}_i$ and another path from $\bar{c}_i$ to c_i . During coverage, the coverage tool follows this double edge, where it moves from sub-tour S to sub-tour $\bar{S}$, covers $\bar{S}$, and returns to finish covering S . Given that the coverage tool width is l , the length added to the coverage path using this strategy is $2l$. Additionally, since we consider the IOP to be connected, all sub-tours arising from the perfect matching can be merged using double edges to form a single tour.

Corner Junction Merging: We now consider a strategy to reduce the length added during the merge. Consider the case where two adjacent grid cells $c_i, c_j \in S$ share a corner with two adjacent grid cells $\bar{c}_i, \bar{c}_j \in \bar{S}$. We refer to these four grid cells as a *corner junction* (Fig. 5.3). Now, we evaluate a set of possible merges within this junction that limits the path length added. First, consider the case where sub-tour S has an edge (c_i, c_j) and sub-tour $\bar{S}$ has an adjacent and parallel edge $(\bar{c}_i, \bar{c}_j)$ connecting the corresponding pairs of cells (Fig. 5.3(a)). In this case, we simply perform an *edge swap*, where we replace these edges with new ones $(c_i, \bar{c}_i)$ and $(c_j, \bar{c}_j)$ that merge the sub-tours (Fig. 5.3(b)). This swap

adds no path length to the resulting merged sub-tour.

Now, consider the case where sub-tour S has no edge (c_i, c_j) parallel to $(\bar{c}_i, \bar{c}_j)$. Instead, let c_{i-1}, c_{i+1} be the cells covered by sub-tour S before and after c_i respectively. Also, let c_{j-1} be the cell covered by sub-tour S before c_j . Fig. 5.3(c) shows an example junction. For now, assume that an edge $(\bar{c}_i, \bar{c}_j)$ exists in sub-tour $\bar{S}$ (we will remove this assumption later). To merge the sub-tours, we first modify sub-tour S by removing three edges, namely (c_{i-1}, c_i) , (c_i, c_{i+1}) , and (c_{j-1}, c_j) , and replacing them with three new edges (c_{i-1}, c_{i+1}) , (c_{j-1}, c_i) , and (c_i, c_j) (Fig. 5.3(d)). The path resulting from these edge swaps is still a closed sub-tour, but more importantly, there is now an edge connecting c_i and c_j in the sub-tour. We now merge the sub-tours using the edge swap from before. For edge (c_{j-1}, c_i) , let $L(c_{j-1}, c_i)$ be the length of the edge (the shortest obstacle free path between the cells). WLOG, we assume that $L(c_{j-1}, c_i) \leq L(c_{i+1}, c_j)$. Specifically, between two choices for the above sub-tour modification, we chose the one with the least added length. While this added length can be computed after the sub-tour modification is made, we speed up this process by obtaining a bound on the added length that is easier to compute.

Proposition 5.2. *Given two sub-tours S and $\bar{S}$ with a corner junction as defined above, merging the sub-tours at the corner junction when there is no edge (c_i, c_j) results in added length of at most $L(c_{j-1}, c_i)$.*

Proof. The length of the edges added by the modification is:

$$L^+ = L(c_{i-1}, c_{i+1}) + L(c_{j-1}, c_i) + L(c_i, c_j).$$

Similarly, the length removed by the modification is:

$$L^- = L(c_{i-1}, c_i) + L(c_i, c_{i+1}) + L(c_{j-1}, c_j).$$

However, following the triangle inequality, we also have:

$$L(c_{i-1}, c_{i+1}) \leq L(c_{i-1}, c_i) + L(c_i, c_{i+1}).$$

Subtracting L^- from L^+ , we have the added path length:

$$\begin{aligned} L^+ - L^- &= L(c_{i-1}, c_{i+1}) + L(c_{j-1}, c_i) + L(c_i, c_j) - \\ &\quad L(c_{i-1}, c_i) - L(c_i, c_{i+1}) - L(c_{j-1}, c_j). \end{aligned}$$

Following the inequality and that $L(c_{j-1}, c_j) \geq L(c_i, c_j) = l$, we can rewrite the above as:

$$L^+ - L^- \leq L(c_{j-1}, c_i),$$

which proves the proposition. $\square$

Following this, we identify a sub-tour modification that minimizes the added length by checking the length of the “*cross path*” (i.e., the corresponding $L(c_{j-1}, c_i)$) added. If $L(c_{j-1}, c_i) > 2l$, we use the double edge strategy for the merge using one of two grid cell pairs in the corner junction. For the case when sub-tour $\bar{S}$ also does not have an edge $(\bar{c}_i, \bar{c}_j)$ between its adjacent cells, we modify $\bar{S}$ as above to create an edge at the junction. We check if the *total* length added by modifying both subtours is greater than $2l$, at which point we simply defer to the double edge strategy.

The above strategies enable the merging of two sub-tours at any set of adjacent cells or corner junctions, while ensuring that the path length added by each merge does not exceed $2l$. In Line 5 of Algorithm 2, we choose the merge with the least (potentially zero) added path length.

5.4.3 Approximation on Path Length

We now show that the above approach yields the following approximation factor on the resulting coverage path length.

Theorem 5.2. *Given a connected IOP with n grid cells, and a rank partition R containing m ranks, the coverage path P obtained using MLC-Approx has length $L(P)$ satisfying*

$$L(P) \leq (1.5 + m/n)L(P^*),$$

where P^* is the optimal coverage path of the IOP.

Proof. Let $L(P_R)$ be the total length of the coverage lines P_R obtained from the ranks R . Recall that

$$L(P_R) = (n - m)l.$$

Let $L(M)$ be the total length of the matching paths M computed by MLC-Approx to form the sub-tours. Now, let T_V^* be the optimal TSP tour of the rank endpoints V . Since M is a minimum-length perfect matching of V , we have

$$L(M) \leq 0.5L(T_V^*) \leq 0.5L(P^*).$$

Now, we consider the length added to the path from merging adjacent sub-tours. We have the number of sub-tours to be at most m , i.e., the number of ranks. Since we limit the

length added from each merge to $2l$ (double edge strategy), the total path length added is $2ml$. Putting it all together, we have the following for the total path length:

$$\begin{aligned} L(P) &= L(P_R) + L(M) + 2ml \\ &\leq (n - m)l + 0.5L(P^*) + 2ml \\ &= nl + 0.5L(P^*) + ml = (1 + m/n)nl + 0.5L(P^*). \end{aligned}$$

Following that $L(P^*) \geq nl$ (Eq. 5.2), we obtain

$$L(P) \leq (1.5 + m/n)L(P^*),$$

which proves the theorem. □

In Theorem 5.2, we observe an additive m/n term in the approximation factor. This term has little effect as (i) using a minimum rank partition (i.e., minimizing m), we observe that $m/n \ll 1$ is a small number, and (ii) many of the sub-tour merges can be performed with the corner junction strategy, which adds little to no additional length. With this, we see that the approximation guarantee is closer to 1.5.

5.4.4 Turn Minimization

While the use of a minimum rank partition minimizes the number of ranks in the coverage path, we must also minimize the turns added during the sub-tour merges. To do this, we use a *weighted* sub-tour adjacency graph to compute the ordering in Line 3 of MLC-Approx (Algorithm 2), where the edge weights are given by the minimum number of turns added by merging two adjacent sub-tours. We then compute a *minimum* spanning tree (MST) in this graph to determine the sub-tour pairs to be merged. When merging two adjacent sub-tours (as per MST edge), we pick the merge strategy that minimizes the number of added turns. Typically, this leads to merges occurring at the ends of ranks, as they add fewer additional turns to the final coverage path.

5.5 MLC-Approx with Lazy Evaluations

For large maps, computing a fully-connected transition graph $G = (V, E)$ to perform perfect matching of all rank endpoints can be computationally expensive, as it requires planning collision-free paths between all combinations of endpoints ($O(V^2)$ operations). However,

Table 5.1: Statistics for dataset maps.

Statistic	Min	Max	Mean	Std Dev
Grid Cells	610	3262	2079	953
Minimum Number of Ranks	39	290	147	73
TSP Path Length (m)	489	2640	1675	776

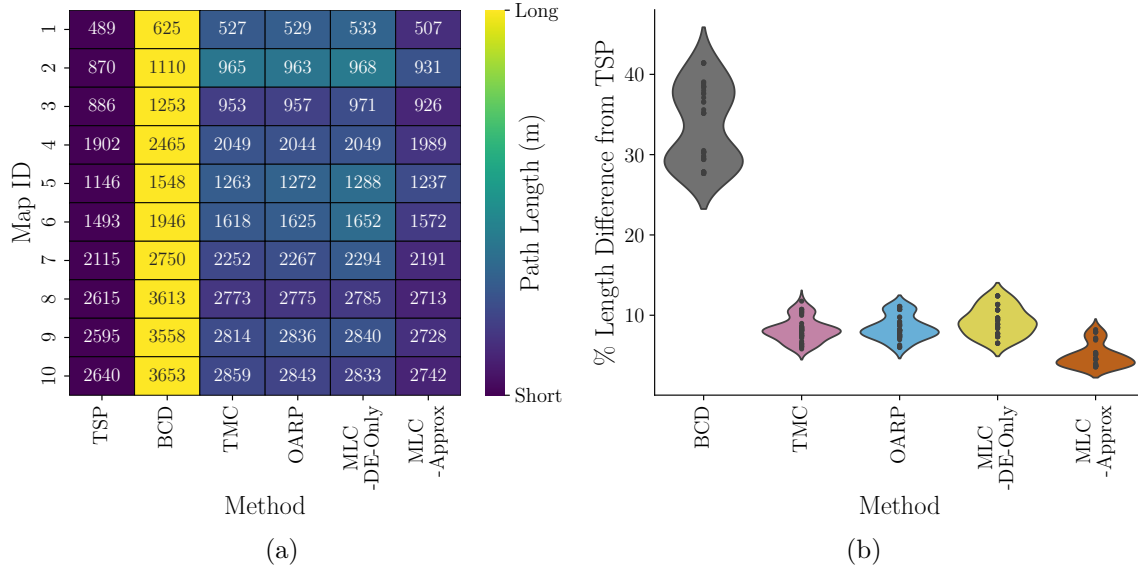


Figure 5.4: Path length comparisons. (a) Average path lengths for each map. (b) % length difference from TSP across all maps.

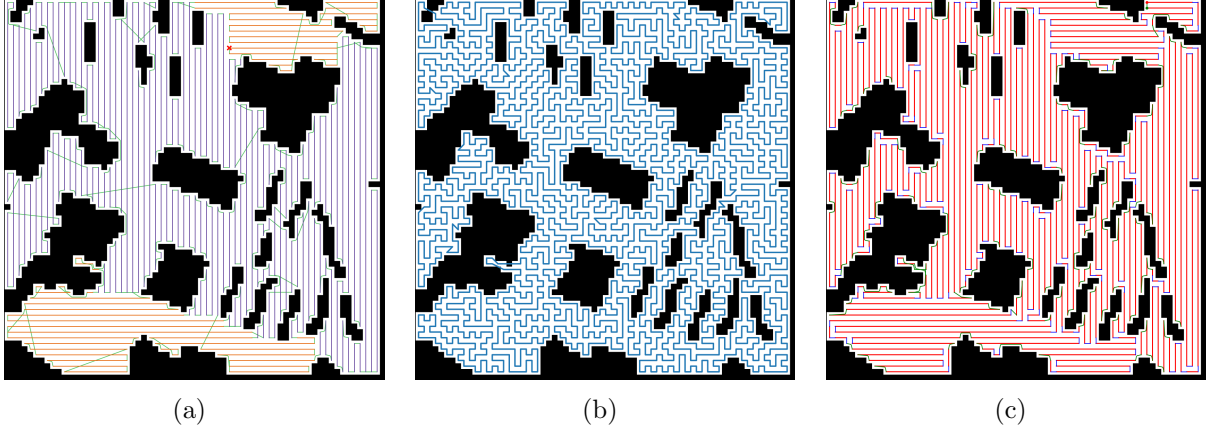


Figure 5.5: Coverage paths for an example environment generated using (a) OARP, (b) TSP, and (c) the proposed MLC-Approx.

we exploit the use of perfect matching to employ a *lazy* strategy, where we do not require exploring all the paths in the transition graph.

We start with a fully-connected transition graph $\tilde{G} = (V, \tilde{E})$, where $\tilde{E}$ is the set of *Euclidean* edges (may have collisions) between each pair of endpoints in V . We then replace Line 1 in MLC-Approx with the following. We iteratively compute a matching $\tilde{M}$ using $\tilde{G}$, where $\tilde{M}$ may contain both Euclidean and collision-free edges. For all Euclidean edges in $\tilde{M}$, the corresponding edge in $\tilde{E}$ is replaced with a collision-free edge, and the edge cost is updated accordingly. This process is repeated until $\tilde{M}$ has no Euclidean edges, i.e., $\tilde{M} = M$. While the lazy approach may still require $O(V^2)$ edge replacements, we observe that this number is far less in practice. Since the length of the Euclidean edge is a lower bound on the length of the collision-free path, the final matching M is optimal.

For some complex environments, one may additionally use a *difference threshold* heuristic to reduce the number of matching iterations. If the length difference between a Euclidean edge and its replacement exceeds this threshold, we preemptively replace all other incident edges. This still only replaces a fraction of the edges while reducing the number of iterations, improving runtime.

5.6 Simulation Results

While the approximation factor is the focus of this paper, we explore how MLC-Approx performs in practice compared to state-of-the-art approaches. We conduct simulations us-

ing a dataset of 10 real-world environments for which the statistics are given in Table 5.1. The dataset, which is available at the link <https://github.com/megalpian/MLC-Maps>, includes a combination of street maps from [83] and anonymized maps of real-world environments provided by our industry partner, Avidbots. To implement MLC-Approx, we obtain the minimum number of axis-parallel ranks for each map’s IOP by solving the LP from [74]. The map dataset is sorted in increasing order of complexity as determined by the number of ranks computed by the LP for each map. We compute the minimum-length perfect matching using the lazy approach (Section 5.5). We then use the strategies from Sections 5.4.2 and 5.4.4 to merge the resulting sub-tours into a coverage path while minimizing turns added to the path.

5.6.1 Path Length Comparisons

We compare MLC-Approx against three baseline approaches: (i) the OARP approach from [74], (ii) the turn-minimizing coverage (TMC) approach from [88], and (iii) the boustrophedon cell decomposition (BCD) approach from [9]. OARP and TMC are turn-minimizing CPP approaches that use the rank-touring framework. TMC computes ranks using an iterative heuristic that can be run for a long time to find high-quality solutions. To standardize the comparison, the runtime of this heuristic is limited to that of OARP’s rank partitioning step. All approaches are tested on each dataset map for 5 trials. For each map, the grid-based approaches (MLC-Approx, OARP, and TMC) must cover the same grid approximation as determined by a tool width $l = 0.8$ m (to match our industry partner’s robot). For OARP and TMC, a GTSP solver [80] is used to compute a minimum-length tour connecting the ranks, where the solver is given a fully-connected transition graph. In addition to these baselines, we also perform an ablation analysis by comparing the full MLC-Approx approach with a variant that only uses double edge (DE) merging. We will refer to this as MLC-DE-Only.

To compare against an optimal planner, we compute a coverage path for each map by solving the Euclidean Travelling Salesman Problem (TSP) of the grid cells. The TSP path will always have the shortest length (does not minimize turns), so the length difference to TSP gives the gap to the optimal path length. The LKH3 solver [41] was used to solve TSP with the Euclidean distance between grid cells as the edge costs (may have collisions). Fig. 5.4(a) reports the average path lengths across trials for each approach and map, and Fig. 5.4(b) compares the *% length difference* from TSP for all approaches. MLC-Approx outperforms all benchmarks and comes closest to the TSP length. We attribute this to the sub-tour merging strategies (Section 5.4.2) that re-use path length instead of adding length to connect ranks. We see this effect in Fig. 5.5, where the MLC-Approx path has fewer

overlapping transition paths (green lines) than the OARP path, resulting in an overall shorter path. MLC-Approx also outperforms BCD, mainly because BCD paths have many overlapping sections as it aims to cover 100% of the environment. However, we observe that grid-based approaches achieve an average coverage of 96% of the environments in our dataset. So, while MLC-Approx does not cover the entire environment, it covers most of the reachable regions.

MLC-Approx path lengths are also closer to TSP than MLC-DE-Only, supporting our earlier claim that most sub-tours are merged with corner junction merging, which adds little to no length. As such, MLC-Approx path lengths are closer to the optimal than the $(1.5 + m/n)L(P^*)$ bound.

5.6.2 Turns Comparisons

We now compare the number of turns in the coverage paths generated by MLC-Approx. For the ablation analysis, we also include a variant of MLC-Approx without turn minimization (Section 5.4.4), which we refer to as MLC-NoTurnMin. Fig. 5.6(a) shows the number of turns for each CPP approach to cover each map, averaged across trials. While MLC-Approx has fewer turns compared to MLC-NoTurnMin, MLC-Approx has more turns than the other turn-minimizing approaches. This is because sub-tour merges often add turns to the path as a result of breaking ranks to re-use path length. We analyze this tradeoff between length and turns by comparing the number of turns per 100 meters of path length, i.e., a measure of path "straightness". Fig. 5.6(b) shows this comparison, where MLC-Approx paths have comparable straightness to the turn-minimizing baselines. In comparison to TSP, MLC-Approx paths have significantly fewer turns despite having comparable path lengths. While this comparison is not entirely fair, we use it to show that MLC-Approx approaches TSP-like path lengths with comparable straightness (turns/length tradeoff) to that of turn-minimizing approaches. While one can also compare against angular-metric TSP [6], we found that, while it provides good solutions for smaller maps of ~ 100 grid cells, it scales poorly for the large maps in our dataset.

BCD achieves the fewest turns for some maps due to the presence of non-orthogonal boundaries in these maps (e.g., walls oriented at different angles). Unlike grid-based approaches, BCD is not limited to covering in axis-parallel directions, and so its paths align better with the environment. A future improvement of MLC-Approx is allowing multiple coverage directions while retaining its guarantees.

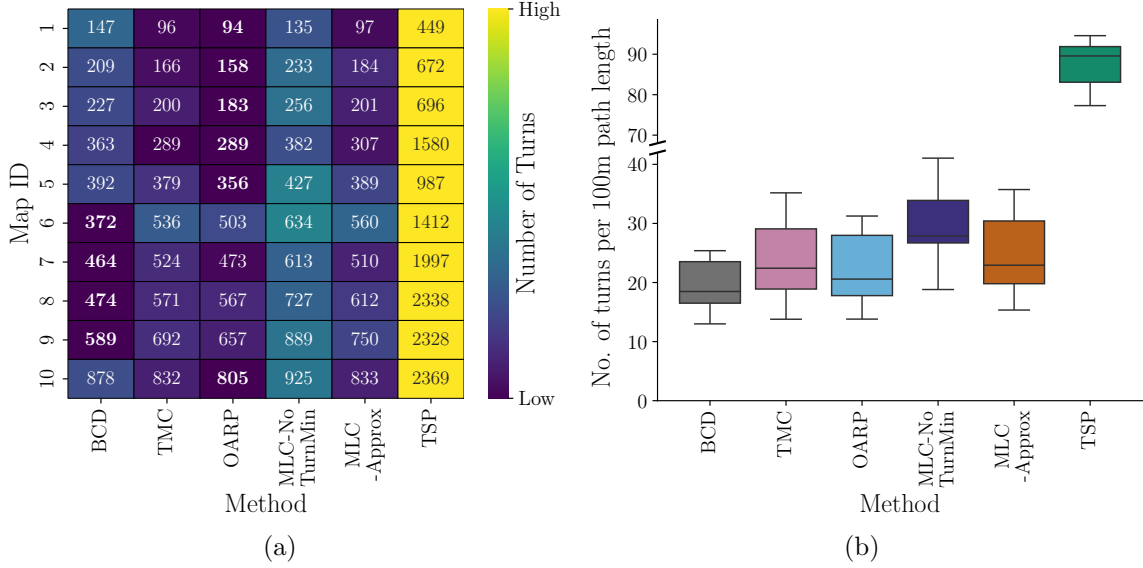


Figure 5.6: Turn comparisons. (a) Number of turns (bold - fewest turns). (b) Coverage path straightness as denoted by the number of turns per 100 meters of path length.

5.6.3 Runtime Improvements with Lazy Evaluations

We demonstrate the runtime improvements from using MLC-Approx with lazy evaluations. For the lazy method, we used a difference threshold of 8 meters (10 times l) for all maps to standardize the comparison. Since perfect matching can typically be computed efficiently, the runtime bottleneck for MLC-Approx is the number of edge *evaluations*, i.e., computing collision-free paths connecting ranks. We plan these paths using a visibility graph planner [69]. At worst, both lazy MLC-Approx and non-lazy approaches [88, 74] evaluate all $O(V^2)$ edges in the transition graph. However, we plot the number of edge evaluations for lazy and non-lazy MLC-Approx in Fig. 5.7(a), where we observe that lazy MLC-Approx evaluates a small fraction of all edges. This results in a runtime performance boost compared to non-lazy approaches. Fig. 5.7(b) shows the results of comparing the runtimes of each approach, where lazy MLC-Approx achieves the fastest runtimes among all baselines. As discussed in Section 5.5, lazy MLC-Approx computes an optimal matching of rank endpoints, so it does not sacrifice the path quality.

Here, the TSP planner appears to be faster than non-lazy approaches while having more edges to evaluate. This is because we used Euclidean distances for the TSP costs, which are quicker to compute than visibility graph paths. Using a visibility planner, the runtime to compute a complete graph for TSP rises to a prohibitive amount (multiple hours).

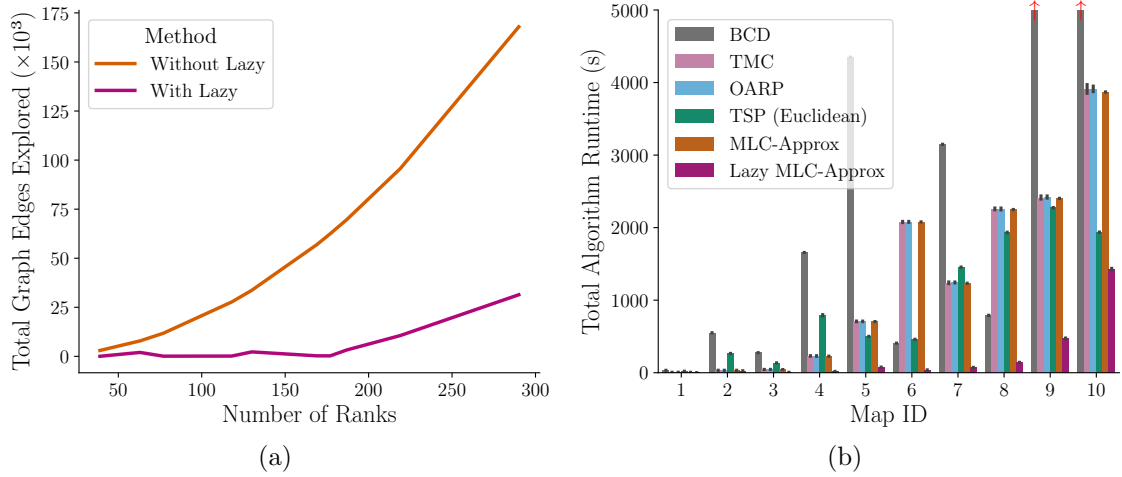


Figure 5.7: Runtime comparisons. (a) Number of edge evaluations (i.e., obstacle-free paths planned) by MLC-Approx with/without lazy. (b) Runtimes for each map (red arrow - exceeds 5000s).

5.7 Summary

This chapter studied minimum-length coverage path planning (CPP) for robots using grid decompositions of environments. The rank-touring framework introduced by recent CPP approaches [7, 88, 74] guarantees a coverage path of length $(1 + 1.5\gamma)$ times the optimal, where $\gamma \geq 1$ is the approximation factor for metric-TSP. To improve this bound, we proposed the MLC-Approx algorithm, which modifies the rank-touring framework to achieve an approximation factor of $(1.5 + \epsilon)$, where $\epsilon \ll 1$ is proportional to the number of ranks (straight-line paths). Empirically, MLC-Approx plans coverage paths with near-optimal lengths in comparison with state-of-the-art CPP approaches [74, 88, 9]. MLC-Approx paths also have comparable “straightness” to turn-minimizing CPP planners, which is desirable for path tracking. A lazy variation of MLC-Approx was also proposed for efficient computation of these paths.

The core techniques of MLC-Approx (matching and sub-tour merging) are applicable to any rank-touring approach that minimizes ranks. For real-world deployment, MLC-Approx paths should be tracked using a local planner similar to the setup from [74]. Moreover, for replanning coverage paths, the lazy MLC-Approx approach offers a runtime speed-up in comparison with OARP to compute coverage paths. In combination with OARP-Replan, one may be able to replan more of the coverage path using MLC-Approx as a replacement for the GTSP transition planner. A future improvement to MLC-Approx is to enable a configurable trade-off between path length and number of turns.

Chapter 6

Multi-Directional CPP with Approximate Decompositions

Finally, we address the problem of removing the axis-parallel constraint on the coverage lines. We propose an approximate decomposition that allows coverage lines to be placed along multiple directions in the environment, which is better suited for non-orthogonal environments.

6.1 Introduction

While at the cost of solution quality, coverage planning approaches in literature use the following framework to make CPP more tractable: (i) decomposing the environment into sub-regions that can each be individually covered, and (ii) computing a visitation order or a tour of the sub-regions. The environment decomposition step is critical as it determines the majority of the coverage path, and also affects the cost of the tour connecting the sub-regions. Therefore, it is important to compute good decompositions that account for these effects on the path.

Recall that, in the literature, CPP algorithms are grouped into exact or approximate approaches depending on the choice of decomposition. Exact approaches usually decompose the environment into convex or monotone sub-regions called cells or “*sectors*” [9, 46, 5]. In this chapter, we refer to these sub-regions as sectors. Each sector is then covered by a lawnmower path (i.e. using parallel coverage lines) along an optimal orientation as determined by the geometry of the sector. However, when applied to complex environments

with irregular boundaries, exact CPP approaches are susceptible to *over-decomposition* where some sectors are thinner than the coverage tool. This is a consequence of covering the entire environment and results in many instances of *double coverage* where the robot returns to areas it has already covered.

In previous chapters, we had proposed CPP approaches that minimize double coverage by minimizing the number of coverage lines (i.e. straight-line paths). This falls under the broad category of approximate CPP approaches that use grid decompositions to cover areas that are reachable by the coverage tool. The decomposition is composed of non-overlapping (usually square) grid cells of size equal to the coverage tool [57, 47, 74, 75]. However, using grid cells constrains the robot to cover the environment along axis-parallel (horizontal or vertical) orientations, leading to sub-optimal “staircase-like” coverage paths. We seek a decomposition that addresses the shortcomings of both exact and approximate approaches by (i) preventing over-decomposition and (ii) removing axis-parallel constraints.

In this chapter, we propose a decomposition of the environment into possibly overlapping rectangular sectors. We choose rectangles for the following reasons: (i) the lawnmower path to cover a rectangle can be optimally oriented along the longest edge [46], and (ii) one can efficiently identify the largest rectangle in an environment [60, 27]. We refer to this as the *sector decomposition* problem. Inspired by works in sensor coverage [25, 61], we leverage the *submodularity* property of covering the environment with sectors to design our solution approach. Specifically, we show that the sector decomposition problem is an instance of the *submodular set cover* (SSC) problem, which has been studied extensively in the literature [91, 48, 19]. In our proposed approach, we use the result that the greedy algorithm for SSC has a well-known approximation guarantee [91].

Chapter contributions: Our specific contributions are as follows:

1. We formulate the sector decomposition problem that aims to minimize the number of sectors and show that this is an instance of the submodular set cover problem.
2. We propose a greedy algorithm to solve the sector decomposition problem, with an approximation guarantee on the number of sectors following that of [91].
3. We present simulation results using maps of real-world environments and compare against state-of-the-art coverage planning approaches.

Organization: In Section 6.2, we introduce the sector decomposition problem. In Section 6.3, we motivate minimizing the number of sectors and show that the sector decomposition problem is an instance of SSC. In Section 6.4, we present our greedy approach to decompose

the environment into sectors. In Section 6.5, we describe how we compute the coverage path using the decomposition. Finally, in Section 6.6, we provide the results of generating coverage paths for real-world environments using the proposed approach.

6.2 Problem Definition

Consider an indoor environment as represented by a set $\mathcal{W} \subseteq \mathbb{R}^2$. We look to cover the environment using the robot's coverage tool, which we assume to be a square of width l . This assumption follows from other works in coverage planning including [8, 74]. Specifically, we look to decompose $\mathcal{W}$ into (possibly overlapping) rectangular subsets called *sectors*. Let $\mathcal{Q}$ denote a set of candidate sectors where each sector $Q_i \in \mathcal{Q}$ represents a rectangle in the environment (i.e., $Q_i \subseteq \mathcal{W}$) with the longest edge oriented along an angle $\theta_i \in [0, \pi)$. Note that the set $\mathcal{Q}$ can be uncountably infinite. Let $|Q|$ denote the area of a set $Q \subseteq \mathbb{R}^2$, and define the function $a : 2^{\mathcal{Q}} \rightarrow \mathbb{R}$ for a subset $\mathcal{S} \subseteq \mathcal{Q}$ as

$$a(\mathcal{S}) = \left| \bigcup_{Q_i \in \mathcal{S}} Q_i \right|. \quad (6.1)$$

We consider a set of candidate sectors $\mathcal{Q}$ such that its union covers $\mathcal{W}$, i.e. $a(\mathcal{Q}) = |\mathcal{W}|$ (i.e., total environment area). Hence, there exists a subset $\mathcal{S}$ that covers all of $\mathcal{W}$ if needed.

We are now ready to present the main problem of this chapter. Given the environment $\mathcal{W}$, we look to decompose $\mathcal{W}$ into the minimum number of rectangular sectors. However, we also look to prevent cases of over-decomposition, where the decomposition may contain small sectors covering inaccessible areas in the environment. To do this, we seek to find a decomposition such that the area within $\mathcal{W}$ covered by the sectors is at least $\gamma|\mathcal{W}|$, where $\gamma \in (0, 1]$ is a design parameter we call the *minimum coverage ratio*. The value of γ can be chosen by the user based on the percentage of the total area the user would like to cover (for 95% coverage, $\gamma = 0.95$). We comment on the relationship between γ and the decomposition later in this chapter.

Problem 6.1 (Sector Decomposition). *Given an environment $\mathcal{W}$, a set of candidate rectangular sectors $\mathcal{Q}$ and a minimum coverage ratio γ , compute a set of sectors $\mathcal{S} \subseteq \mathcal{Q}$ that solves:*

$$\begin{aligned} \min_{\mathcal{S} \subseteq \mathcal{Q}} \quad & |\mathcal{S}| \\ \text{s.t.} \quad & a(\mathcal{S}) \geq \gamma|\mathcal{W}|. \end{aligned}$$

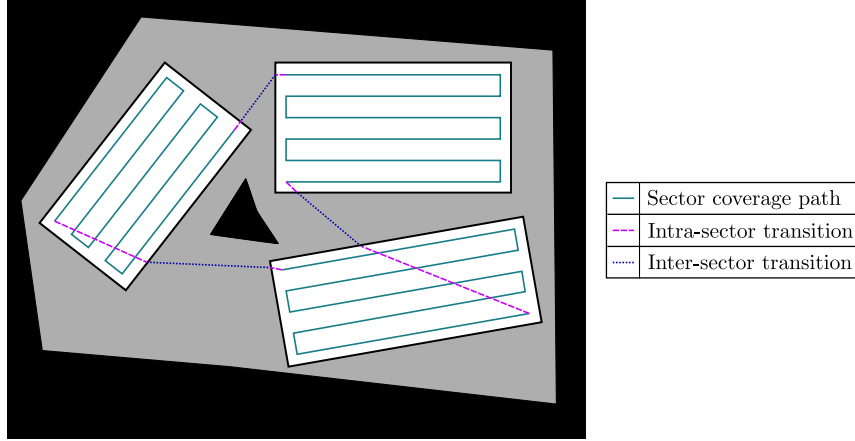


Figure 6.1: Coverage path resulting from a sector decomposition containing three sectors. Each sector is covered by a lawnmower path and the robot transitions between sectors using (i) intra-sector paths, and (ii) inter-sector paths.

Solving Problem 6.1, we obtain sectors and the coverage lines (i.e., straight-line paths) forming lawnmower paths for each sector. A sector has *two* possible paths depending on the connections between the coverage lines. From this, we plan a coverage path for the environment using a *sector touring* approach where: (i) we pick a lawnmower path for each sector, and (ii) connect the lawnmower paths using a cost-minimizing tour between the path endpoints. An example coverage path for a set of three sectors is shown in Fig. 6.1. We provide details on sector touring in Section 6.5.

Remark 6.1. *While we consider rectangular sectors in this chapter, one could more generally consider sectors that are monotone with a well-defined orientation for the lawnmower path. However, in our solution approach, we leverage existing approaches [60] that efficiently compute the largest inscribed rectangle within the environment at an orientation.*

6.3 Link to Submodular Set Cover

In this section, we establish that Problem 6.1 is an instance of SSC. We motivate minimizing the number of sectors by deriving a bound on the worst-case coverage path length resulting from a decomposition of rectangular sectors. We then prove the submodularity of the area function a , which, based on [91], implies that the greedy algorithm provides a polynomial-time approximation algorithm.

6.3.1 Coverage Path Optimality

We now motivate minimizing the number of sectors in the decomposition by considering the length of the resulting coverage path after sector touring. Giving an exact characterization of the relationship between the sector decomposition and the coverage path length is challenging. Thus, our approach is to bound the length of the coverage path obtained from sector touring and minimizing this bound. For convex environments, this may be done using a single sector covering the environment. However, for non-convex environments, such as those considered in this chapter, obtaining a similar bound requires more in-depth analysis.

Given a sector decomposition of the environment, the lawnmower path length to optimally cover each rectangular sector can be tightly bounded using the rectangle dimensions and the coverage tool width l . Consider a rectangular sector Q of width w and height h . Without loss of generality, let $w \geq h$ and consider a lawnmower path with coverage lines along the longest edge (i.e. width) of the rectangle. Now, let P_Q be the path to cover Q and let $L(P_Q)$ be its length.

Lemma 6.1. *Given a rectangular sector Q of width w and height h , the coverage path P_Q has a length $L(P_Q)$ satisfying*

$$L(P_Q) \leq w \left\lceil \frac{h}{l} \right\rceil.$$

Proof. The length of each coverage line in the lawnmower path is $w - l$ as the tool can stay at a distance of $l/2$ away from the perimeter of the rectangle to cover it. To connect each coverage line to an adjacent line, we require a *connecting path* of length at most l . So each line-connection pair has a length of w .

If h is a multiple of l , then the number of line-connection pairs is exactly h/l , and the bound is tight. If h is not a multiple of l , i.e., $h \neq cl$ for some $c \in \mathbb{N}$. The number of coverage lines needed to cover Q is $\lceil h/l \rceil$. Therefore, the total path length is upper-bounded by $w \lceil h/l \rceil$, which proves the lemma. $\square$

Given a set $\mathcal{S}$ of rectangular sectors that cover the environment $\mathcal{W}$, i.e. $a(\mathcal{S}) = |\mathcal{W}|$, let P , with length $L(P)$, be the coverage path obtained by connecting the lawnmower paths using *sector touring*. Since the sectors can overlap, the sum of sector areas is given by

$$\sum_{Q \in \mathcal{S}} |Q| = (1 + \alpha) |\mathcal{W}|,$$

where α is the ratio of $|\mathcal{W}|$ double-covered by sectors in $\mathcal{S}$. Let w' represent the longest width of all sectors in $\mathcal{S}$. We now bound $L(P)$ with respect to the length $L(P^*)$ of the optimal coverage path P^* for $\mathcal{W}$ (unconstrained by sectors).

Proposition 6.1 (Coverage Path Length Optimality). *Given an environment $\mathcal{W}$ and a coverage tool of width l , let P^* be an optimal coverage path of $\mathcal{W}$, with length $L(P^*)$. Consider a decomposition of $\mathcal{W}$ into rectangular sectors $\mathcal{S}$, and let α be the ratio of sector overlap and w' be the longest sector width. Then, the length $L(P)$ of the coverage path P resulting from sector touring satisfies:*

$$L(P) \leq (2 + \alpha)L(P^*) + (\sqrt{2} + 1)w'|\mathcal{S}|.$$

Proof. Since the area covered by a tool of width l along any path of length L is at most lL , we have

$$L(P^*) \geq \frac{|\mathcal{W}|}{l}, \tag{6.2}$$

with equality if the path is a straight line. The path P is obtained by (i) covering each sector individually using lawnmower paths, and (ii) defining the connections between the lawnmower paths, called *transition paths*. Let L_s be the total length of the lawnmower paths. Following Lemma 6.1,

$$L_s \leq \sum_{Q \in \mathcal{S}} \left(\frac{|Q|}{l} + w' \right) \leq \frac{\sum_{Q \in \mathcal{S}} |Q|}{l} + |\mathcal{S}|w'$$

Following the definition of α , we get the following ratio between the sum of sector areas and l :

$$\frac{\sum_{Q \in \mathcal{S}} |Q|}{l} = \frac{(1 + \alpha)|\mathcal{W}|}{l} \leq (1 + \alpha)L(P^*).$$

We now bound the length of the transition paths L_t that connect the lawnmower paths. We split the transition paths into two path segments: (i) inter-sector (path between sectors), and (ii) intra-sector (path traveled within each sector). Fig. 6.1 illustrates these path segments in an example coverage path. To bound L_t , we consider computing transition paths as follows: (i) we use a TSP tour that visits a single corner of each sector (inter-sector), and (ii) connect each ends of the lawnmower path to the corner (intra-sector).

To bound the inter-sector path segments, let $\text{TOUR}(\mathcal{S})$ be the length of the TSP tour of the sector corners. Note that the optimal coverage path P^* for the environment visits each sector at least once. So, we have that

$$\text{TOUR}(\mathcal{S}) \leq L(P^*).$$

We now bound the intra-sector path segments for each sector. A lawnmower path for a rectangle starts close to one sector corner and ends at a different corner. Therefore, the cumulative path length to reach any corner from the endpoints of a lawnmower path is at most the diameter of the sector (length of the diagonal), which is at most $\sqrt{2}w'$. As a result, we have that

$$L_t \leq L(P^*) + \sqrt{2}|\mathcal{S}|w'.$$

Adding all the path lengths, we arrive at the final result:

$$\begin{aligned} L(P) &= L_s + L_t \\ &\leq (1 + \alpha)L(P^*) + (1 + \sqrt{2})|\mathcal{S}|w' + \text{TOUR}(\mathcal{S}), \\ &\leq (2 + \alpha)L(P^*) + (1 + \sqrt{2})|\mathcal{S}|w' \end{aligned}$$

which proves our proposition. $\square$

Following Proposition 6.1, we observe two factors of the sector decomposition affecting the length of the worst-case coverage path: (i) the number of sectors $|\mathcal{S}|$, and (ii) sector overlap α . Therefore, reducing the number of sectors in the decomposition reduces the second term in the bound. We address (ii) in our approach (Section 6.4) by constraining sector overlap when computing the decomposition.

6.3.2 Submodularity of Sector Coverage

We now show that the coverage function a is a submodular function defined on the set of all candidate sectors $\mathcal{Q}$. This is not immediately obvious as $\mathcal{Q}$ is uncountably infinite.

Proposition 6.2. *The coverage function $a : 2^{\mathcal{Q}} \rightarrow \mathbb{R}$ as defined in Eq. (6.1) is normalized, monotone and submodular.*

Proof. Recall that for a to be normalized, $a(\emptyset) = 0$ for the empty set $\emptyset$. As such, normalization holds trivially as the coverage of $\mathcal{W}$ is 0 if no sectors are picked from $\mathcal{Q}$. Similarly,

recall that a is monotone if for all $A \subseteq B \subseteq \mathcal{Q}$, $a(A) \leq a(B)$. This holds trivially, since for any set $A \subseteq \mathcal{Q}$, each sector in A must cover a non-negative amount of area in $\mathcal{W}$.

To show the submodularity of a , consider sets $A \subseteq B \subset \mathcal{Q}$ and a sector $Q \in \mathcal{Q} \setminus B$. From the definition of submodularity in Chapter 2, the following must hold:

$$a(A \cup \{Q\}) - a(A) \geq a(B \cup \{Q\}) - a(B).$$

For the set B , let $\delta_Q(B) = a(B \cup \{Q\}) - a(B)$ represent the marginal return of adding Q to the set B . First, notice that

$$\delta_Q(B) = \left| Q \setminus (\cup_{Q_B \in B} Q_B) \right|.$$

Now, manipulating this expression we get

$$\begin{aligned} \delta_Q(B) &= \left| Q \setminus (\cup_{Q_B \in B} Q_B) \right| \\ &= \left| Q \setminus \left((\cup_{Q_A \in A} Q_A) \cup (\cup_{Q_B \in B \setminus A} Q_B) \right) \right| \\ &= \left| \left(Q \setminus (\cup_{Q_A \in A} Q_A) \right) \cap \left(Q \setminus (\cup_{Q_B \in B \setminus A} Q_B) \right) \right| \\ &\leq \left| Q \setminus (\cup_{Q_A \in A} Q_A) \right| \\ &= \delta_Q(A), \end{aligned}$$

where the equality in the third line follows from an application of DeMorgan's Laws (i.e., $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$). This proves the proposition. $\square$

Following Proposition 6.2, we have that Problem 6.1 is an instance of SSC which allows us to use the result from [91] to propose a greedy decomposition approach.

6.4 Sector Decomposition Approach

We now describe our solution approach for sector decomposition. We look to greedily add large rectangular sectors to the decomposition that cover new areas of the environment. We first describe the greedy step that leverages computational geometry algorithms to identify the largest candidate sector in the environment (or its subset). We then present the greedy algorithm that computes a sector decomposition while constraining the overlap between the sectors. For the case where the overlap is not constrained, we provide an approximation factor for the greedy algorithm following [91].

6.4.1 Sector Identification

We first look to identify the largest candidate sector (i.e. the rectangle of maximum area) for a given environment $\mathcal{W}$ (or its subset). One can efficiently compute the largest area *axis-parallel* rectangle in the environment [60, 27]. Also, for coverage planning, one may constrain the robot’s coverage directions using the orientations of the environment’s edges (boundary and holes), so that the robot covers along long walls and moves predictably [46, 9, 5].

To identify the largest sector, we use a polygonal representation of $\mathcal{W}$ and extract a set of candidate orientations Θ using the polygon’s edges. We limit our candidate sectors $\mathcal{Q}$ to rectangles oriented along every $\theta \in \Theta$. We then obtain the largest axis-parallel rectangular sector in $\mathcal{W}$ for each θ using existing approaches [60]. Repeating this process for every θ , we form the set $\bar{\mathcal{Q}}$ of candidate sectors for the greedy step.

6.4.2 Greedy Decomposition

We propose an algorithm that decomposes the environment $\mathcal{W}$ by greedily choosing rectangular sectors that cover the largest remaining uncovered area. The pseudocode is given in Algorithm 3. Until the minimum coverage of the environment is achieved (Line 3), we compute candidate sectors $\bar{\mathcal{Q}}$ for each orientation $\theta \in \Theta$ (Line 4). We then pick the candidate sector $Q \in \bar{\mathcal{Q}}$ covering the largest additional area, add it to our decomposition $\mathcal{S}$ (Lines 5-6). We then remove the area it covers from the environment to avoid overlapping coverage with future candidate sectors (Line 8).

In some cases, a small amount of overlap between sectors is useful to have better fitting neighbouring sectors. To do this, we introduce a parameter called sector erosion radius, β , which is used to compute the Minkowski difference $\bar{Q}$ between Q and a disk of radius β . Only $\bar{Q}$ is removed from the environment so that future sectors may overlap with Q . Setting β to 0 leads to disjoint sectors, while arbitrarily large values lead to unconstrained overlapping.

Remark 6.2. *For unconstrained overlap, the candidate sectors must be computed using an alternate approach, as the convex skull method may return the same rectangle repeatedly. One may use a checkerboard decomposition like in [88] to identify sectors at each orientation. The candidate sectors only need to be computed once for this case.*

Algorithm 3 Greedy Sector Decomposition (G-Sect)

Input: Environment $\mathcal{W}$, Candidate orientations Θ , Minimum coverage ratio γ , Sector erosion radius β

Output: Sector decomposition $\mathcal{S}$

```
1:  $\mathcal{S} \leftarrow \emptyset$  ▷ Initialize to empty set
2:  $\bar{\mathcal{W}} \leftarrow \mathcal{W}$  ▷ Copy environment
3: while  $a(\mathcal{S}) < \gamma|\mathcal{W}|$  do
4:    $\bar{\mathcal{Q}} \leftarrow$  Candidate sectors for  $\bar{\mathcal{W}}$  for all angles in  $\{\theta\}$ 
5:    $Q \leftarrow \arg \max_{Q \in \bar{\mathcal{Q}}} a(\mathcal{S} \cup \{Q\}) - a(\mathcal{S})$ 
6:    $\mathcal{S} \leftarrow \mathcal{S} \cup \{Q\}$ 
7:    $\bar{Q} \leftarrow$  Erode  $Q$  by a radius of  $\beta$ 
8:    $\bar{\mathcal{W}} \leftarrow \bar{\mathcal{W}} \setminus \bar{Q}$ 
9: end while
10: return  $\mathcal{S}$ 
```

6.4.3 Analysis

In the case of overlapping sectors (large β), we obtain an approximation factor for the G-Sect algorithm as a function of the minimum coverage ratio γ . We derive this factor following that of Wolsey's analysis of the greedy algorithm in [91]. Given a set of candidate orientations Θ , let $\mathcal{S}^*$ be the minimum sector decomposition of the environment, i.e. $|\mathcal{S}^*| \leq |\mathcal{S}|$ for any $\mathcal{S}$.

Proposition 6.3 (Approximation factor for G-SectD). *Given an environment $\mathcal{W}$ and a minimum coverage ratio of γ , let $\mathcal{S}^*$ be the minimum sector decomposition given candidate orientations Θ . Then, the G-Sect algorithm computes a sector decomposition $\mathcal{S}$ satisfying*

$$\frac{|\mathcal{S}|}{|\mathcal{S}^*|} \leq 1 + \ln \left(\frac{1}{1 - \gamma} \right).$$

Proof. We first prove this proposition for a *finite* set of candidate sectors $\mathcal{Q}$ that cover the environment $\mathcal{W}$. Let T be the number of iterations run by the greedy algorithm, and let $\mathcal{S}^r$ for $1 \leq r \leq T$ be the subset that is picked by the greedy algorithm after r iterations. Following Wolsey's analysis of the greedy algorithm for SSC over a finite set [91], we have the following approximation factor:

$$\frac{|\mathcal{S}|}{|\mathcal{S}^*|} \leq 1 + \ln \left(\frac{a(\mathcal{Q}) - a(\emptyset)}{a(\mathcal{Q}) - a(\mathcal{S}^{T-1})} \right). \quad (6.3)$$

Since a is normalized, we have $a(\emptyset) = 0$. Since $\mathcal{Q}$ covers $\mathcal{W}$, we have that $a(\mathcal{Q}) = |\mathcal{W}|$. Given that G-Sect terminates after a minimum coverage of $\gamma|\mathcal{W}|$, we have that $a(\mathcal{S}^{T-1}) \leq \gamma|\mathcal{W}|$. Substituting these values in Eq. (6.3), we get

$$\frac{|\mathcal{S}|}{|\mathcal{S}^*|} \leq 1 + \ln \left(\frac{|\mathcal{W}|}{|\mathcal{W}| - \gamma|\mathcal{W}|} \right) = 1 + \ln \left(\frac{1}{1 - \gamma} \right) \quad (6.4)$$

We now show that this bound also applies when $\mathcal{Q}$ is infinite. Let $\mathcal{S}^* = \{Q_1^*, Q_2^*, \dots, Q_k^*\}$ be the minimum sector decomposition of $\mathcal{W}$ from all of $\mathcal{Q}$. Now, let $\mathcal{S} = \{Q_1, Q_2, \dots, Q_T\}$ be the sector decomposition obtained by G-Sect given γ . Using these sets, we construct a finite set of candidate sectors $\mathcal{Q}' = \mathcal{S} \cup \mathcal{S}^*$. On this finite set $\mathcal{Q}'$, we can run G-Sect and for each greedy iteration r , we have the following for all $Q_i^* \in \mathcal{S}^*$:

$$a(\mathcal{S}^r \cup Q_{r+1}) - a(\mathcal{S}^r) \geq a(\mathcal{S}^r \cup Q_i^*) - a(\mathcal{S}^r).$$

This follows from the correctness of the greedy step for a set of candidate orientations Θ . As a result, the approximation factor holds on the finite set $\mathcal{Q}'$. However, since $\mathcal{Q}'$ contains the choices of G-Sect from $\mathcal{Q}$ and the optimal set from $\mathcal{Q}$, the approximation factor also holds on the infinite set $\mathcal{Q}$. This proves the proposition. $\square$

A key observation from Proposition 6.3 is that the approximation factor increases as γ increases. For $\gamma = 0.95$ (95% coverage), the approximation factor is close to 4. Obtaining a similar approximation factor for minimally overlapping sectors (small β) is a non-trivial task. However, decompositions using small β values better suit coverage planning as they reduce instances of double coverage. In Section 6.6, we provide simulation results of planning coverage paths using minimally overlapping sectors.

6.4.4 Local Sector Merges

After a decomposition is computed using G-Sect, we merge neighbouring sectors to further reduce the number of sectors. Specifically, we conduct merges where the lawnmower path of one sector can be extended to cover a neighbouring sector. Given the sector decomposition and corresponding sector orientations, we start by counting the number of coverage lines required to cover each sector. We then iterate through the sectors in the order of increasing sector area (i.e. smallest to largest) to check for possible merges. For each sector Q , we look for an adjacent sector Q_{adj} with a coverage orientation θ_{adj} such that the number of coverage lines to cover $Q \cup Q_{\text{adj}}$ along the orientation θ_{adj} is less than or equal to the cumulative number of lines required to cover each sector individually. The reduction in

Table 6.1: Robot parameters used for experiments

Parameter	Value
Coverage tool width l	0.8 m
Maximum linear velocity	1 m/s
Linear acceleration	$\pm 0.5\text{ m/s}^2$

coverage lines indicates that the lawnmower path of Q_{adj} can be extended to cover Q . Any ties are broken by choosing Q_{adj} with the largest area. The merged sector is added back to the set of sectors and the process is repeated until all possible merges have been completed. The resulting sector decomposition forms our final solution to Problem 6.1.

Note that these merges may result in sectors that are not necessarily rectangular, i.e. they do not strictly satisfy Problem 6.1. However, they still satisfy the key restriction that each sector is covered by a lawnmower path. To obtain a strictly rectangular decomposition, this step may be skipped. In practice, a similar strategy may also be used to cover areas that are uncovered by the sector decomposition.

6.5 Sector Touring

We now describe our approach to plan a coverage path using the sector decomposition from Section 6.4. Similar to [9], we formulate a generalized travelling salesman problem (GTSP) that aims to minimize the time to cover all sectors by determining (i) the visitation order of the sectors in the decomposition, and (ii) entry and exit points for each sector. We create an auxiliary graph where the vertices are grouped into sets. Each set in the graph represents the coverage of a sector and consists of at most four vertices that represent the directed paths to cover each sector, i.e. two possible lawnmower paths per sector where each path can be traversed in two directions. The chosen path and traversal direction determine the entry and exit points for each sector. The graph edges represent the transition paths to travel between sectors. The edge costs are given by the travel time along an obstacle-free path from one end of a lawnmower path to another determined using a visibility graph planner [69]. We compute this travel time using a piecewise constant acceleration robot model where the robot travels along a straight line with constant acceleration until a maximum velocity is reached. The parameters used for this model is given in Table 6.1. We do not directly penalize the turning time of the robot, but the above model penalizes the stop time preceding an upcoming turn. The GTSP tour in the resulting graph gives

the coverage path for the environment as a series of connected sectors.

6.6 Simulation Results

In this section, we present simulation results for the proposed sector decomposition approach using maps of real-world environments. Specifically, our dataset includes 2 street maps of New York and Shanghai cities from [57] and 2 anonymized environments obtained from combining maps of real-world environments obtained from our industry partner Avidbots. We compare the coverage planning performance of our approach against two state-of-the-art approaches: (i) an exact decomposition approach called boustrophedon cell decomposition (BCD) [9], and (ii) our previous approximate (grid) decomposition approach called Optimal Axis-Parallel Rank Partitioning (OARP) approach [74].

For our simulations, we set the desired coverage ratio γ to be 95% of the environment, after which the greedy algorithm is terminated. This choice helps eliminate sectors that cover small inaccessible areas and effectively reduces the number of sectors in the decomposition. We first identify the environment’s walls using Probabilistic Hough Transform and obtain the set of candidate coverage orientations Θ . The coverage tool width l for the simulations is configured as in Table 6.1. For the BCD and sector decomposition approaches, we ignore small obstacles (obstacle area $< 4l^2$) as they do not affect the coverage orientation and can be trivially evaded during coverage. For OARP, some small obstacles are removed during the construction of the grid decomposition.

Pertaining to sector overlap, we motivate our configuration of G-Sect by decomposing a simple map using two values of the sector erosion radius β : $l/4$ and $2l$. Figure 6.2 shows the decompositions, where we observe that increasing overlap reduces the number of sectors but also results in overlapping coverage lines (i.e., double coverage). However, by using a small β along with local sector merges, G-Sect computes a minimal decomposition with non-overlapping coverage lines (Figure 6.2 (c)). Therefore, we conduct the rest of the experiments with $\beta = l/4$ and conduct sector merges.

Figure 6.3 shows the decompositions computed using G-Sect for our test maps. We observe that sector decomposition results in intuitive coverage lines, especially in narrow regions where the coverage lines are oriented along the edges of the environment. Table 6.2 shows the results of comparing G-Sect’s decompositions against those of BCD [9]. We observe that sector decomposition approach covers the environment with fewer sectors, predominantly due to (i) our choice of γ , and (ii) sector merging. In contrast, BCD results in over-decomposition due to the large number of reflex vertices present in the test environments. G-Sect is less influenced by these vertices for the chosen γ .

Table 6.2: Comparison of number of sectors in each decomposition

Map	BCD [9]	Our Approach
Anonymous 1	55	32
Anonymous 2	75	43
New York	136	73
Shanghai	86	44

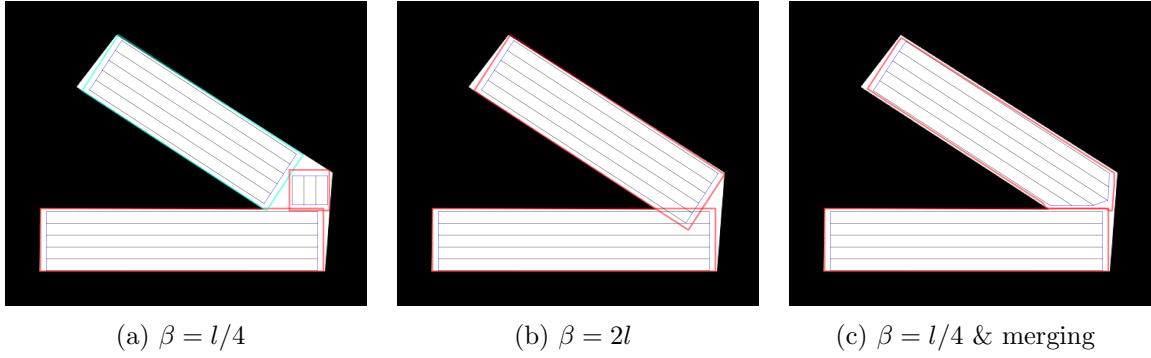
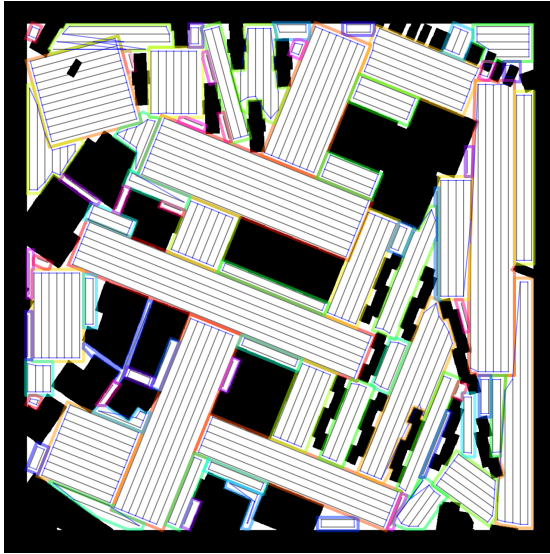


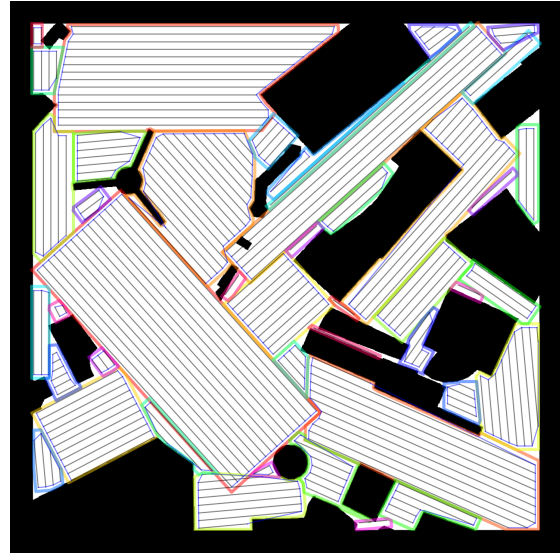
Figure 6.2: G-Sect results with varying sector overlap and merging.

Table 6.3: Comparison of coverage planning performance for the test maps.

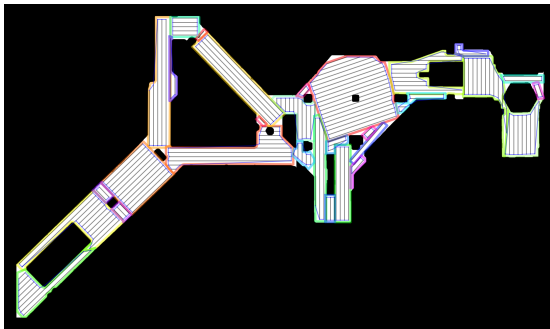
	BCD [9]			OARP [74]			Sector Decomposition		
	# Lines	% Area	Cost (s)	# Lines	% Area	Cost (s)	# Lines	% Area	Cost (s)
Anonymous 1	187	100%	2297	169	95.4%	1850	158	95%	1714
Anonymous 2	204	100%	2566	130	94.2%	2425	125	95%	2255
New York	300	100%	4692	219	94.3%	3880	267	95.1%	3874
Shanghai	235	100%	4540	187	97%	3713	221	95%	3687



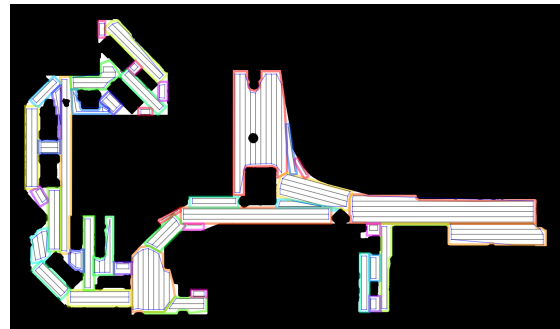
(a) New York



(b) Shanghai



(c) Anonymous 1



(d) Anonymous 2

Figure 6.3: Sector decomposition results for the test maps.

We now compare the coverage planning performance of the proposed approach with that of BCD and OARP. We generate an auxiliary graph as described in Section 6.5 with the parameters from Table 6.1, and solve the GTSP using the open source GLNS solver [80]. Table 6.3 shows the results of comparing (i) number of coverage lines, (ii) percent area covered, and (iii) coverage path cost. For every map, we run each approach for 5 trials and present the average for each metric. We observe that the proposed approach achieves the lowest path cost for all test maps. This is due to G-Sect computing coverage lines that align with the walls of the environment. For the New York and Shanghai maps, we observe that the proposed approach outperforms OARP despite having more coverage lines. This is because OARP constrains the robot’s motion to axis-parallel orientations creating long transition paths (where the robot performs no coverage). However, for environments with axis-parallel boundaries, we expect the OARP approach to perform better as it minimizes the number of coverage lines (fewer transitions). A future direction we look to explore is a combination of G-Sect and OARP [74] that minimizes the number of coverage lines in the path while covering along multiple orientations. Meanwhile, the BCD approach covers the environment entirely but plans relatively expensive paths as a result of over-decomposition.

6.7 Summary

We proposed an environment decomposition approach for robot coverage planning called *sector decomposition*. The approach decomposes the environment into the minimum number of rectangular sectors, where each sector is covered using a lawnmower path oriented along its longest edge. We showed that sector decomposition is an instance of the sub-modular set cover (SSC) problem, and proposed a greedy algorithm with a guarantee on the number of sectors in the decomposition. Finally, we present coverage planning results using the proposed approach for real-world environments and show that it improves upon existing coverage planning approaches [9, 74].

While we use lawnmower paths to cover each sector, one may use OARP in conjunction with the sectors to compute the minimum number of ranks oriented at multiple angles. A future direction of this work is such a combination to minimize the number of turns in the coverage path generated using sector decomposition, for cases when robot turning is detrimental to coverage time and quality.

Chapter 7

Conclusions and Future Work

Coverage path planning (CPP) is the task of planning a robot’s path such that the tool/sensor being carried by the robot covers all points in its environment. Additionally, CPP aims to minimize the cost associated with the path, such as length, turns, and/or coverage time. While CPP is NP-Hard for known environments [8], the complexity of the problem increases further when unknown obstacles are introduced into the environment. In this thesis, we studied the problem of coverage path planning for partially unknown static environments.

In Chapter 3, we proposed the optimal axis-parallel rank partitioning (OARP) approach to compute minimum-turn coverage paths for known non-convex environments with obstacles. The key to OARP is that we can partition the environment efficiently into the minimum number of axis-parallel (horizontal/vertical) thin rectangles called *ranks* that can each be covered along a straight-line path. We then compute a generalized traveling salesman problem (GTSP) tour of the ranks to compute a coverage path such that the transition time between the rank endpoints is minimized. We presented simulation results with two datasets of maps and compared against state-of-the-art CPP approaches in the literature [88, 9]. We also showcased an industrial cleaning robot executing a path generated using OARP for an example environment in a high-fidelity robot operating system (ROS) simulation.

In Chapter 4, we extended OARP to replan coverage paths upon detecting new static obstacles in the environment. We aim to conduct path replans that minimize the robot’s drive time and *stoppage time* resulting from the robot not having a safe path to continue coverage, i.e., from computationally expensive replans. We proposed OARP-Replan, an anytime coverage replanning approach, that uses the time available before reaching the

first newly detected obstruction to replan as much of the interrupted coverage plan optimally (few turns as possible). To incorporate the time budget, we modified OARP to consider the coupling between the rank partitioning and the touring steps. We showed that OARP-Replan computed optimal replans of the coverage path if given a large enough time budget, i.e., enough time to reach the new obstacle. We showcased this approach using extensive simulations and real-world experiments with an industrial cleaning robot, where we observed an improved performance against several baseline approaches, specifically resulting from achieving fewer turns in the coverage path with little to no stoppage resulting from the replanning. Moreover, in our robot experiments, OARP-Replan enabled a large cleaning robot to reach and cover narrow areas in the environment caused by new obstacles.

We then focused on two improvements to the planning approaches proposed in the previous chapters. In Chapter 5, we proposed a modification to OARP that computes minimum-length coverage paths with approximation guarantees, while minimizing turns as a secondary objective. We started by showing that the rank-touring framework (e.g., OARP) to compute coverage paths achieves an approximation factor of $(1 + 1.5\gamma)$ on the coverage path length, where γ is the approximation factor to solve the metric-TSP. We then proposed an improved $(1.5 + \epsilon)$ -approximation algorithm called MLC-Approx that can replace the GTSP step in OARP. Here, $\epsilon \ll 1$ is proportional to the number of ranks in the environment’s coverage path. In practice, we show that MLC-Approx computes paths with near-optimal lengths and comparable “straightness” to minimum-turn CPP approaches like OARP. Moreover, we also proposed a lazy variation of MLC-Approx that computes paths quicker without losing the approximation guarantees. This lazy variant offers a large speedup in path computation and can enable faster replans of coverage paths.

Finally, in Chapter 6, we remove the axis-parallel constraint introduced in OARP by computing an approximate decomposition that allows coverage along multiple directions. We propose decomposing the environment into the minimum number of *sectors*, which are possibly overlapping rectangular sub-regions. Each sector can be covered in parallel straight-line paths oriented along its longest edge. We showed that the sector decomposition problem is an instance of the submodular set cover (SSC) problem, for which the greedy algorithm achieves the best-known approximation guarantee. Following this, we proposed a greedy algorithm for sector decomposition that incorporates techniques from computational geometry to identify sectors in the environment [27, 60]. Given a minimum desired coverage ratio of γ (i.e., fraction of environment area covered), we obtained an approximation guarantee of $1 + \ln(\frac{1}{1-\gamma})$ for the greedy approach. In simulations, we obtained intuitive decompositions of non-orthogonal complex environments resulting in shorter coverage times than axis-parallel paths.

7.1 Future Work

While this thesis studies algorithms to compute near-optimal coverage paths for partially unknown environments, there are still some challenges that are yet to be solved. Specifically, we identify the following two topics for future work in this domain.

Obstacle Predictions: The coverage replanning approach proposed in Chapter 4 registers a new obstacle using sensor readings collected before a replan. However, for a large obstacle that is not visible at once, the replanning is done in parts, where each replan uses parts of the obstacle that have been observed by the sensor at different replans. This may result in a replanned path with some sub-optimal detour paths that may be prevented if more information about the obstacle was available at the first replan. To mitigate this, one may predict the shape and size of a partially-observed obstacle, which can then be used to compute replanned paths closer to the optimal. This approach can be especially useful if there are recurring obstacles across multiple coverage planning missions. A possible direction towards this predictive coverage replanning approach is to use modern mechanisms like deep neural networks [42] to generate predictions about the obstacles or whole environments that can be incorporated in a replanning approach like OARP-Replan.

Early Environment Exploration: Consider a cleaning robot exploring an unknown warehouse environment using a long range LIDAR scanner while covering with a smaller vacuum attachment. In this case, the sensor has a much larger footprint than the vacuum tool, so a coverage path for the vacuum tool will guarantee exploration of the environment with the sensor. However, when covering cluttered environments with unknown obstacles, one may want to detect new obstacles earlier in the coverage path so more time is available to replan the coverage path such that the overall coverage time is minimized. In the cleaning robot example, early exploration of a large obstacle in a occluded section of the warehouse may allow enough time for the robot to replan a cost-minimizing coverage path that avoids it. For CPP applications where robustness is an issue (i.e., dead-locks due to no safe paths), early exploration of uncovered environments may help identify areas where the robot is likely to get stuck. Therefore, future work in this area may balance planning optimal coverage paths with planning *informative* paths that encourage early exploration.

Multi-robot extensions: While we focus on single-robot coverage planning in this thesis, many of these concepts can be potentially used to design planning and replanning algorithms for multi-robot coverage. For example, one may modify the MLC-Approx algorithm proposed in Chapter 5 to compute coverage subtours for a swarm of n robots, instead of a single tour. For multi-robot coverage replanning, OARP-Replan may be modified to

replan multiple coverage paths at once, where any changes to the coverage ranks can be addressed by replanning the paths for a subset of robots in the swarm. A possible challenge to address is to compute multi-robot paths with each robot being subject to a capacity constraint, which may also be explored as future work.

Another consideration when replanning coverage paths with multiple robots is respecting communication constraints as determined by the swarm network. Obstacles seen by one robot in the swarm may affect a section of the path that was assigned to be covered by other robots in the swarm. In such cases, a replanning algorithm may additionally re-assign sections of the path to balance the coverage load.

References

- [1] Ercan U. Acar and Howie Choset. Critical point sensing in unknown environments. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3803–3810 vol.4, April 2000.
- [2] Ercan U. Acar and Howie Choset. Sensor-based Coverage of Unknown Environments: Incremental Construction of Morse Decompositions. *The International Journal of Robotics Research*, 21(4):345–366, April 2002.
- [3] Ercan U. Acar, Howie Choset, and Ji Yeong Lee. Sensor-based coverage with extended range detectors. *IEEE Transactions on Robotics*, 22(1):189–198, February 2006.
- [4] Tobias Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009.
- [5] Saurav Agarwal and Srinivas Akella. Area Coverage With Multiple Capacity-Constrained Robots. *IEEE Robotics and Automation Letters*, 7(2):3734–3741, April 2022.
- [6] Alok Aggarwal, Don Coppersmith, Sanjeev Khanna, Rajeev Motwani, and Baruch Schieber. The Angular-Metric Traveling Salesman Problem. *SIAM Journal on Computing*, 29(3):697–711, January 2000.
- [7] Esther M. Arkin, Michael A Bender, Erik D Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia. Optimal Covering Tours with Turn Costs. *SIAM Journal on Computing*, 35(3):531–566, 2005.
- [8] Esther M. Arkin, Sándor P. Fekete, and Joseph S.B. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1-2):25–50, 2000.

- [9] Rik Bähneemann, Nicholas Lawrance, Jen Jen Chung, Michael Pantic, Roland Siegwart, and Juan Nieto. Revisiting Boustrophedon Coverage Path Planning as a Generalized Traveling Salesman Problem. In *Field and Service Robotics*, pages 277–290. Springer, 2021.
- [10] Iago Z. Biundini, Milena F. Pinto, Aurelio G. Melo, Andre L. M. Marcato, Leonardo M. Honório, and Maria J. R. Aguiar. A Framework for Coverage Path Planning Optimization Based on Point Cloud for Structural Inspection. *Sensors*, 21(2):570, 2021.
- [11] Stanislav Bochkarev and Stephen L. Smith. On minimizing turns in robot coverage path planning. In *IEEE International Conference on Automation Science and Engineering*, pages 1237–1242, 2016.
- [12] Richard Bormann, Florian Jordan, Joshua Hampp, and Martin Hägele. Indoor Coverage Path Planning: Survey, Implementation, Analysis. In *IEEE International Conference on Robotics and Automation*, pages 1718–1725, May 2018.
- [13] Amanda Bouman, Joshua Ott, Sung-Kyun Kim, Kenny Chen, Mykel J. Kochenderfer, Brett Lopez, Ali-akbar Agha-mohammadi, and Joel Burdick. Adaptive Coverage Path Planning for Efficient Exploration of Unknown Environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 11916–11923, October 2022.
- [14] Lauren Bramblett, Rahul Peddi, and Nicola Bezzo. Coordinated Multi-Agent Exploration, Rendezvous, & Task Allocation in Unknown Environments with Limited Connectivity. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 12706–12712, October 2022.
- [15] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23, March 1986.
- [16] Stanley Brown and Steven L. Waslander. The constriction decomposition method for coverage path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3233–3238, October 2016.
- [17] Tauã M. Cabreira, Carmelo Di Franco, Paulo R. Ferreira, and Giorgio C. Buttazzo. Energy-Aware Spiral Coverage Path Planning for UAV Photogrammetric Applications. *IEEE Robotics and Automation Letters*, 3(4):3662–3668, October 2018.

- [18] Kyle Cesare, Ryan Skeelee, Soo-Hyun Yoo, Yawei Zhang, and Geoffrey Hollinger. Multi-UAV exploration with limited communication and battery. In *IEEE International Conference on Robotics and Automation*, pages 2230–2235, May 2015.
- [19] Wenjing Chen and Victoria Crawford. Bicriteria Approximation Algorithms for the Submodular Cover Problem. *Advances in Neural Information Processing Systems*, 36:72705–72716, December 2023.
- [20] SeungYoon Choi, SeungGwan Lee, Hoang Huu Viet, and TaeChoong Chung. B-Theta*: An Efficient Online Coverage Algorithm for Autonomous Cleaning Robots. *Journal of Intelligent & Robotic Systems*, 87(2):265–290, August 2017.
- [21] Howie Choset. Coverage for robotics – A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1):113–126, October 2001.
- [22] Howie Choset and Philippe Pignon. Coverage Path Planning: The Boustrophedon Cellular Decomposition. *Field and Service Robotics*, pages 203–209, 1998.
- [23] Nicos Christofides. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. *Operations Research Forum*, 3(1):1–4, March 2022.
- [24] Micah Corah and Nathan Michael. Efficient Online Multi-robot Exploration via Distributed Sequential Greedy Assignment. In *Robotics: Science and Systems XIII*. Robotics: Science and Systems Foundation, July 2017.
- [25] Micah Corah and Nathan Michael. Distributed Submodular Maximization on Partition Matroids for Planning on Large Sensor Networks. In *IEEE Conf. on Decision and Control (CDC)*, pages 6792–6799, December 2018.
- [26] Tung Dang, Marco Tranzatto, Shehryar Khattak, Frank Mascarich, Kostas Alexis, and Marco Hutter. Graph-based subterranean exploration path planning using aerial and legged robots. *Journal of Field Robotics*, 37(8):1363–1388, 2020.
- [27] Karen Daniels, Victor Milenkovic, and Dan Roth. Finding the largest area axis-parallel rectangle in a polygon. *Computational Geometry*, 7(1-2):125–148, January 1997.
- [28] Arun Das, Michael Diu, Neil Mathew, Christian Scharfenberger, James Servos, Andy Wong, John S. Zelek, David A. Clausi, and Steven L. Waslander. Mapping, Planning, and Sample Detection Strategies for Autonomous Exploration. *Journal of Field Robotics*, 31(1):75–106, 2014.

- [29] Sándor P. Fekete and Dominik Krupke. Covering Tours and Cycle Covers with Turn Costs: Hardness and Approximation, April 2019.
- [30] Sándor P. Fekete and Dominik Krupke. What Goes Around Comes Around: Covering Tours and Cycle Covers with Turn Costs. *Theory of Computing Systems*, 68(5):1207–1238, 2024.
- [31] Sándor P. Fekete and Dominik Krupke. What Goes Around Comes Around: Covering Tours and Cycle Covers with Turn Costs. *Theory of Computing Systems*, 68(5):1207–1238, October 2024.
- [32] Yoav Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):77–98, 2001.
- [33] Yoav Gabriely and Elon Rimon. Spiral-STC: An on-line coverage algorithm of grid environments by a mobile robot. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 954–960, 2002.
- [34] Yoav Gabriely and Elon Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry*, 24(3):197–224, 2003.
- [35] Douglas W. Gage. Randomized search strategies with imperfect sensors. In *Mobile Robots VIII*, volume 2058, pages 270–279. SPIE, February 1994.
- [36] Enric Galceran, Ricard Campos, Narcís Palomeras, David Ribas, Marc Carreras, and Pere Ridao. Coverage Path Planning with Real-time Replanning and Surface Reconstruction for Inspection of Three-dimensional Underwater Structures using Autonomous Underwater Vehicles: Coverage Path Planning with Real-time Replanning and Surface Reconstruction. *Journal of Field Robotics*, 32(7):952–983, October 2015.
- [37] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013.
- [38] Simone Godio, Stefano Primates, Giorgio Guglieri, and Fabio Dosis. A Bioinspired Neural Network-Based Approach for Cooperative Coverage Planning of UAVs. *Information*, 12(2):51, February 2021.
- [39] I. A. Hameed. Intelligent coverage path planning for agricultural robots and autonomous machines on three-dimensional terrain. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 74(3-4):965–983, 2014.

- [40] Keld Helsgaun. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, October 2000.
- [41] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.
- [42] Cherie Ho, Seungchan Kim, Brady Moon, Aditya Parandekar, Narek Harutyunyan, Chen Wang, Katia Sycara, Graeme Best, and Sebastian Scherer. MapEx: Indoor Structure Exploration with Probabilistic Information Gain from Global Map Predictions. *arXiv:2409.15590*, 2024.
- [43] Alan J. Hoffman and Joseph B. Kruskal. Integral Boundary Points of Convex Polyhedra. In *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 49–76. Springer, Berlin, Heidelberg, 2010.
- [44] Christian Hofner and Günther Schmidt. Path planning and guidance techniques for an autonomous mobile cleaning robot. *Robotics and Autonomous Systems*, 14(2):199–212, 1995.
- [45] Yafei Hu, Junyi Geng, Chen Wang, John Keller, and Sebastian Scherer. Off-Policy Evaluation With Online Adaptation for Robot Exploration in Challenging Environments. *IEEE Robotics and Automation Letters*, 8(6):3780–3787, June 2023.
- [46] Wesley H. Huang. Optimal line-sweep-based decompositions for coverage algorithms. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 27–32 vol.1, 2001.
- [47] Aleksandr Ianenkov, Alexander Artamonov, Georgii Sarapulov, Alexey Safaraleev, Sergey Bogomolov, and Dong-ki Noh. Coverage Path Planning with Proximal Policy Optimization in a Grid-based Environment. In *IEEE Conf. on Decision and Control (CDC)*, pages 4099–4104, December 2020.
- [48] Rishabh K Iyer and Jeff A Bilmes. Submodular Optimization with Submodular Cover and Submodular Knapsack Constraints. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.
- [49] Wei Jing, Di Deng, Zhe Xiao, Yong Liu, and Kenji Shimada. Coverage Path Planning using Path Primitive Sampling and Primitive Coverage Graph for Visual Inspection. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1472–1479, 2019.

- [50] Xinyue Kan, Hanzhe Teng, and Konstantinos Karydis. Multi-robot Field Exploration in Hex-Decomposed Environments for Dubins Vehicles. In *IEEE International Conference on Robotics and Biomimetics*, pages 449–455, December 2019.
- [51] Xinyue Kan, Hanzhe Teng, and Konstantinos Karydis. Online Exploration and Coverage Planning in Unknown Obstacle-Cluttered Environments. *IEEE Robotics and Automation Letters*, 5(4):5969–5976, October 2020.
- [52] Athanasios Ch. Kapoutsis, Savvas A. Chatzichristofis, and Elias B. Kosmatopoulos. DARP: Divide Areas Algorithm for Optimal Multi-Robot Coverage Path Planning. *Journal of Intelligent & Robotic Systems*, 86(3-4):663–680, June 2017.
- [53] Nare Karapetyan, Jason Moulton, Jeremy S. Lewis, Alberto Quattrini Li, Jason M. O’Kane, and Ioannis Rekleitis. Multi-robot Dubins Coverage with Autonomous Surface Vehicles. In *IEEE International Conference on Robotics and Automation*, pages 2373–2379, May 2018.
- [54] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.
- [55] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics. Springer-Verlag, Berlin Heidelberg, third edition, 2006.
- [56] Tushar Kusnur and Maxim Likhachev. Complete, Decomposition-Free Coverage Path Planning. In *IEEE International Conference on Automation Science and Engineering*, pages 1431–1437, August 2022.
- [57] Junjie Lu, Bi Zeng, Jingtao Tang, Tin Lun Lam, and Junbin Wen. TMSTC*: A Path Planning Algorithm for Minimizing Turns in Multi-Robot Coverage. *IEEE Robotics and Automation Letters*, 8(8):5275–5282, August 2023.
- [58] Chaomin Luo and Simon X. Yang. A Bioinspired Neural Network for Real-Time Concurrent Map Building and Complete Coverage Robot Navigation in Unknown Environments. *IEEE Transactions on Neural Networks*, 19(7):1279–1298, July 2008.
- [59] Andrew O. Makhorin. GLPK - GNU Project - Free Software Foundation (FSF). <https://www.gnu.org/software/glpk/>.
- [60] Zahraa Marzeh, Maryam Tahmasbi, and Narges Mirehi. Algorithm for finding the largest inscribed rectangle in polygon. *Journal of Algorithms and Computation*, 51(1):29–41, June 2019.

- [61] Negar Mehr and Roberto Horowitz. A Submodular Approach for Optimal Sensor Placement in Traffic Networks. In *American Control Conference (ACC)*, pages 6353–6358, June 2018.
- [62] Petar Milošević. *Creative Commons BY-SA 4.0, via Wikimedia Commons*. (<https://creativecommons.org/licenses/by-sa/4.0>).
- [63] Weimin Mo and Zhiyun Lin. MDSTC: A Dynamic Approach to Multi-Robot Coverage Path Planning. In *International Conference on Control, Automation, Robotics and Vision*, pages 1100–1105, December 2024.
- [64] David Morilla-Cabello, Luca Bartolomei, Lucas Teixeira, Eduardo Montijano, and Margarita Chli. Sweep-Your-Map: Efficient Coverage Planning for Aerial Teams in Large-Scale Environments. *IEEE Robotics and Automation Letters*, 7(4):10810–10817, October 2022.
- [65] M. A. Viraj J. Muthugala, S. M. Bhagya P. Samarakoon, and Mohan Rajesh Elara. Toward energy-efficient online Complete Coverage Path Planning of a ship hull maintenance robot based on Glasius Bio-inspired Neural Network. *Expert Systems with Applications*, 187:115940, 2022.
- [66] B. Nasirian, M. Mehrandezh, and F. Janabi-Sharifi. Efficient Coverage Path Planning for Mobile Disinfecting Robots Using Graph-Based Representation of Environment. *Frontiers in Robotics and AI*, 8:4, 2021.
- [67] George L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, New York, 1999.
- [68] Charles E. Noon and James C. Bean. An Efficient Transformation Of The Generalized Traveling Salesman Problem. *INFOR: Information Systems and Operational Research*, 31(1):39–44, 1993.
- [69] K. J. Obermeyer and Contributors. VisiLibity: A C++ Library for Visibility Computations in Planar Polygonal Environments. <http://www.VisiLibity.org>, 2008.
- [70] Jordi Palacín, Tomás Palleja, Ignasi Valganón, Ramón Pernia, and Joan Roca. Measuring Coverage Performances of a Floor Cleaning Mobile Robot Using a Vision System. In *IEEE International Conference on Robotics and Automation*, pages 4236–4241, April 2005.

- [71] Liam Paull, Sajad Saeedi, Mae Seto, and Howard Li. Sensor-Driven Online Coverage Planning for Autonomous Underwater Vehicles. *IEEE/ASME Transactions on Mechatronics*, 18(6):1827–1838, December 2013.
- [72] Leonidas Pitsoulis, Konstantinos Papalamprou, Gautam Appa, and Balázs Kotnyek. On the representability of totally unimodular matrices on bidirected graphs. *Discrete Mathematics*, 309(16):5024–5042, 2009.
- [73] Azarakhsh Rafiee, Eduardo Dias, Steven Fruijtier, and Henk Scholten. From BIM to Geo-analysis: View Coverage and Shadow Analysis by BIM/GIS Integration. *Procedia Environmental Sciences*, 22:397–402, January 2014.
- [74] Megnath Ramesh, Frank Imeson, Baris Fidan, and Stephen L. Smith. Optimal Partitioning of Non-Convex Environments for Minimum Turn Coverage Planning. *IEEE Robotics and Automation Letters*, 7(4):9731–9738, October 2022.
- [75] Megnath Ramesh, Frank Imeson, Baris Fidan, and Stephen L. Smith. Anytime Re-planning of Robot Coverage Paths for Partially Unknown Environments. *IEEE Transactions on Robotics*, 40:4190–4206, 2024.
- [76] Megnath Ramesh, Frank Imeson, Baris Fidan, and Stephen L. Smith. Approximate Environment Decompositions for Robot Coverage Planning using Submodular Set Cover. In *IEEE Conference on Decision and Control*, pages 7528–7533, December 2024.
- [77] Mitul Saha, Tim Roughgarden, Jean-Claude Latombe, and Gildardo Sánchez-Ante. Planning Tours of Robotic Arms among Partitioned Goals. *The International Journal of Robotics Research*, 25(3):207–223, March 2006.
- [78] Haydar Sahin and Levent Guvenc. Household robotics: Autonomous devices for vacuuming and lawn mowing [Applications of control]. *IEEE Control Systems Magazine*, 27(2):20–96, April 2007.
- [79] K.S. Senthilkumar and K.K. Bharadwaj. Multi-robot exploration and terrain coverage in an unknown environment. *Robotics and Autonomous Systems*, 60(1):123–132, January 2012.
- [80] Stephen L. Smith and Frank Imeson. GLNS: An effective large neighborhood search heuristic for the Generalized Traveling Salesman Problem. *Computers & Operations Research*, 87:1–19, 2017.

- [81] Junnan Song and Shalabh Gupta. ε^* : An online coverage path planning algorithm. *IEEE Transactions on Robotics*, 34(2):526–533, 2018.
- [82] Soohwan Song, Daekyum Kim, and Sungho Jo. Online coverage and inspection planning for 3D modeling. *Autonomous Robots*, 44(8):1431–1450, 2020.
- [83] Nathan R. Sturtevant. Benchmarks for Grid-Based Pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.
- [84] Bing Sun, Daqi Zhu, Chen Tian, and Chaomin Luo. Complete Coverage Autonomous Underwater Vehicles Path Planning Based on Glasius Bio-Inspired Neural Network Algorithm for Discrete and Centralized Programming. *IEEE Transactions on Cognitive and Developmental Systems*, 11(1):73–84, March 2019.
- [85] Chee Sheng Tan, Rosmiwati Mohd-Mokhtar, and Mohd Rizal Arshad. A Comprehensive Review of Coverage Path Planning in Robotics Using Classical and Heuristic Algorithms. *IEEE Access*, 9:119310–119342, 2021.
- [86] Jingtao Tang and Hang Ma. Large-Scale Multi-Robot Coverage Path Planning via Local Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17567–17574, March 2024.
- [87] Jingtao Tang, Chun Sun, and Xinyu Zhang. MSTC*:Multi-robot Coverage Path Planning under Physical Constrains. In *IEEE International Conference on Robotics and Automation*, pages 2518–2524, May 2021.
- [88] Isaac Vandermeulen, Roderich Groß, and Andreas Kolling. Turn-minimizing multi-robot coverage. In *IEEE International Conference on Robotics and Automation*, pages 1014–1020, 2019.
- [89] Hoang Huu Viet, Viet-Hung Dang, SeungYoon Choi, and Tae Choong Chung. BoB: An online coverage approach for multi-robot systems. *Applied Intelligence*, 42(2):157–173, March 2015.
- [90] Hoang Huu Viet, Viet-Hung Dang, Md Nasir Uddin Laskar, and TaeChoong Chung. BA*: An online complete coverage algorithm for cleaning robots. *Applied Intelligence*, 39(2):217–235, September 2013.
- [91] Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, December 1982.

- [92] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 146–151. IEEE Comput. Soc. Press, 1997.
- [93] Alexander Zelinsky and Shin’ichi Yuta. A unified approach to planning, sensing and navigation for mobile robots. In *Experimental Robotics III*, Lecture Notes in Control and Information Sciences, pages 444–455, Berlin, Heidelberg, 1994. Springer.