GENERALIZED COVERAGE USING MULTIPLE ROBOTS: THEORY, ALGORITHMS, AND EXPERIMENTS

by

Saurav Agarwal

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2022

ABSTRACT

SAURAV AGARWAL. Generalized Coverage Using Multiple Robots: Theory, Algorithms, and Experiments. (Under the direction of DR. SRINIVAS AKELLA)

Recent technological advances have facilitated the use of mobile robots for a wide range of coverage applications such as inspection and mapping of infrastructure, precision agriculture, and disaster management. With the proliferation of these tasks comes an increasing need for autonomous systems to efficiently gather data for analyzing the state of the environment. The dissertation answers the following fundamental question: How should resource-constrained robots traverse the environment to collect data from all the relevant features? These features of interest can be represented as points, lines or curves, and areas. This dissertation unifies simultaneous coverage of all three types of features into a novel generalized coverage framework, develops algorithms for efficient coverage using multiple mobile robots, and validates them in experiments.

The dissertation first comprehensively studies the line coverage problem, i.e., coverage of one-dimensional features with multiple resource-constrained robots. We model the environment as a graph and formalize line coverage as an optimization problem using integer linear programs (ILP). The problem is NP-hard, and therefore we design approximation algorithms with provable guarantees and heuristic algorithms for large graphs, validating them extensively in simulations and experiments. Extensions to the formulation and algorithms address large-scale environments and nonholonomic robots that cannot make point turns. These formulations of the line coverage problem lay the foundation for generalized coverage. Next, we show that we can transform the area coverage problem into a line coverage problem using computational geometry techniques and then generate routes using line coverage algorithms. Existing methods have significant inefficiencies due to their use of monotone polygons. Using the line coverage transformation allows polygons with obstacles that are not monotone while minimizing the number of turns for the robots. The transformation enables algorithmic advances in line coverage to be directly applied to area coverage. Finally, we formulate the generalized coverage problem and solve it by transforming both point and area features into line features in a unified framework. The algorithms demonstrate significantly improved performance over state-of-the-art solutions while additionally incorporating battery life constraints, nonholonomic robots, and multiple home locations for large-scale environments. We evaluate the performance of the algorithms on several real-world datasets in simulations and experiments. The algorithms are very fast and generate high-quality solutions for robotics applications.

ACKNOWLEDGMENTS

The contributions made in this dissertation were possible because of the support from many people. The research stands on the shoulder of giants and is cradled by several with whom I have been fortunate to interact.

First of all, I would like to express my sincere gratitude to my advisor, Prof. Srinivas Akella, for introducing me to arc routing problems and suggesting developing algorithms for robotics applications. This idea has been fundamental to the development of the dissertation. I am indebted to him for bolstering and nurturing me to become a researcher. He spent countless hours participating in intellectual discussions, ensuring correctness, and fostering in-depth understanding, all of which have been instrumental in writing research papers and the dissertation. He has been a very supportive, patient, and caring advisor, which immensely helped me carry out my research.

I am very grateful to have David Vutetakis, Huitan Mao, Kalvik Jakkala, Sayantan Datta, and Sterling McLeod as my labmates. I thank them for their invaluable discussions and for providing great companionship throughout my life at UNC Charlotte.

I would like to thank the committee members, Prof. Erik Saule, Prof. Min Shin, Prof. Andrew R. Willis, and Prof. Artur Wolek, for reviewing the dissertation and providing valuable feedback, which has helped improve the research. I thank the Computer Science department and the University of North Carolina at Charlotte for providing a supportive and caring environment. I have been very fortunate to be supported by the National Science Foundation (NSF), the Defense Advanced Research Projects Agency (DARPA), the UNC system, and the UNC Charlotte GASP scholarship.

I also want to thank my previous advisors, Prof. Sandipan Bandyopadhyay and Prof. Shibendu Shekhar Roy, for giving me opportunities and laying the foundations for becoming a researcher.

Finally, I thank my wife, Madhyama Thakur, for being an amazing partner and providing unwavering support, love, and care. Also, I thank my son, Agastya, for bringing boundless joy and happiness into my life, thereby making the journey easier.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

In recent years there have been significant advances in the field of robotics, particularly research related to mobile robots—commonly categorized into uncrewed ground vehicles (UGVs) and uncrewed aerial vehicles (UAVs). The rapid development of small mobile robots has been the result of three main factors: (i) miniaturization of electronic components such as sensors, batteries, and micro-processors, (ii) reduction in costs, and (iii) advances in controllability (Floreano and Wood, 2015; Chung et al., 2018). These rapid developments have facilitated the use of mobile robots for a wide range of coverage applications such as inspection and mapping of infrastructure, precision agriculture, and disaster management.

The capabilities of autonomous robots are enhanced when they work as a team. Multi-robot systems offer flexibility, sensor coverage, and redundancy, making them attractive for large-scale applications. Moreover, robots have constraints on the operation time due to a limited battery capacity. Having a team of mobile robots is essential for ensuring large-scale tasks are completed within a reasonable time. However, in order to exploit the benefits of multi-robot systems, we have to address issues such as the assignment of tasks, coordination of robots, and navigation (Wagner and Choset, 2015).

1.1 Coverage Problems

In a *coverage* application, the robots are required to visit specified features in the environment. With the view of mobile robots as autonomous vehicles carrying sensors, the dissertation answers the following fundamental question: How should the robots traverse the environment to collect data from all the relevant features?

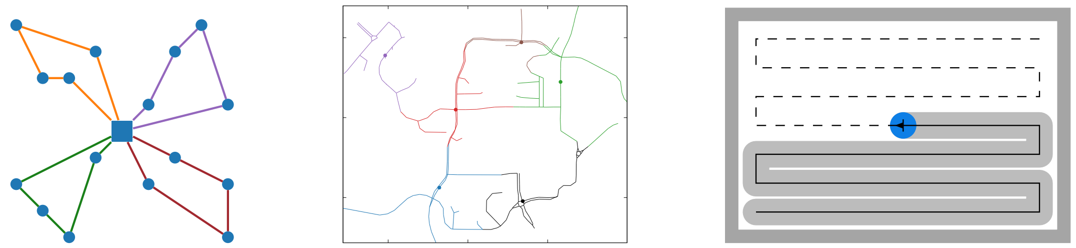


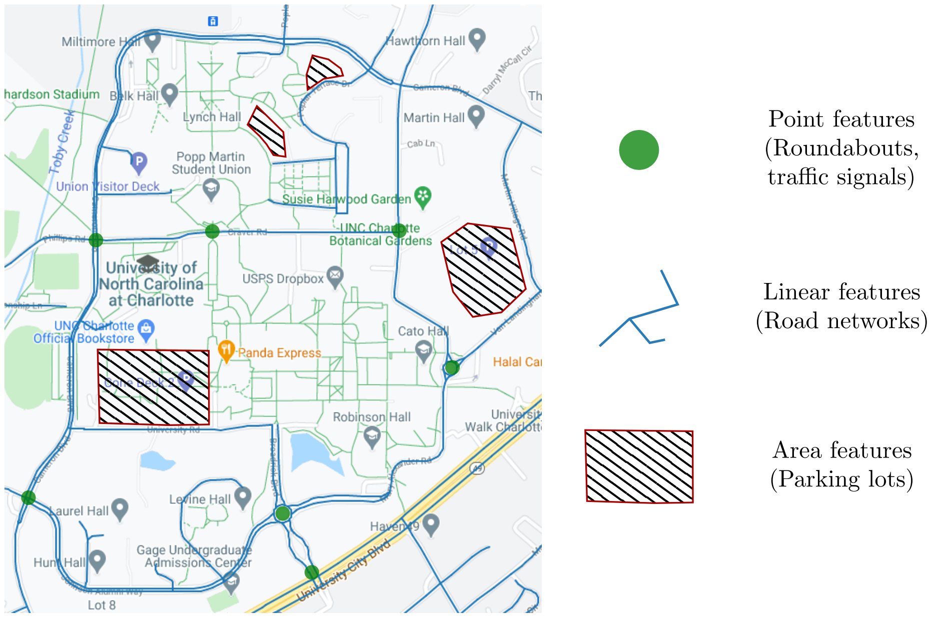
Figure 1.1: Three types of features and the corresponding coverage problems: (a) Point coverage is the coverage of zero-dimensional point features and is commonly solved using node routing algorithms. (b) Line coverage is the coverage of one-dimensional line features and belongs to the broad class of arc routing problems. (c) Area coverage is the coverage of two-dimensional regions, often solved using computational geometry techniques. The dissertation unifies the simultaneous coverage of all three types of features in a novel *generalized coverage* framework.

Environments may have features of interest that can be represented as points, lines or curves, and areas, resulting in three independent types of coverage problems as depicted in Figure 1.1.

- **Point Coverage:** Given a set of point features in an environment, point coverage is the task of efficiently visiting all the points. The environment is often modeled as a graph data structure, where the vertices of the graph represent the point features, and the edges with costs represent the traversal between vertices. Point coverage on graphs can be solved using node routing problems such as the traveling salesperson problem (TSP) and the vehicle routing problem (e.g., Macharet and Campos, 2018). Applications such as inspection of oil wells in an oil field using aerial robots can be efficiently modeled as point coverage.

- **Line Coverage:** The task of visiting a set of line features in an environment is known as line coverage. The problem can model inspection of linear infrastructure such as road networks, oil and gas pipelines, and power lines. Line coverage is related to arc routing problems commonly studied in the operations research community (see the monograph by Corberán and Laporte, 2014).

- **Area Coverage:** The coverage of two-dimensional regions is formulated as the area coverage problem. It is a classical problem in robotics and has been studied widely for various applications such as vacuuming, infrastructure inspection, and precision agriculture (see, e.g., Choset, 2001; Galceran and Carreras, 2013).



Point features (Roundabouts, traffic signals)
Linear features (Road networks)
Area features (Parking lots)
University of North Carolina at Charlotte
Popp Martin Student Union
Union Visitor Deck
USPS Dropbox
Panda Express
Robinson Hall
Levine Hall
Laurel Hall
Hunt Hall
Gage Undergraduate Admissions Center
Martin Hall
Hawthorn Hall
Lynch Hall
Belk Hall
Miltimore Hall
Cato Hall
University City Blvd
UNC Charlotte Botanical Gardens
Susie Harwood Garden
Toby Creek

Figure 1.2: Generalized coverage: simultaneous coverage of all three types of features. The roundabouts and traffic signals form the point features, the segments of the road network form the line features, and the parking lots are the area features. The robots must cover, using their sensors, all the features in the environment efficiently while respecting practical constraints such as a limited battery life.

The robotics community has been actively researching the point coverage and the area coverage problems. However, line coverage, i.e., coverage of one-dimensional features, has not received sufficient attention compared to the other two problems. Moreover, these problems are generally addressed individually. Consider the application of traffic analysis during an event: we may be interested in inspecting roundabouts and traffic signals, road networks, and parking lots, as illustrated in Figure 1.2. Such a task requires considering all three types of features concurrently. The dissertation develops the theory to unify simultaneous coverage of all three types of features into a novel *generalized coverage* framework, develops algorithms for efficient coverage using teams of multiple mobile robots, and validates them in experiments. Figure 1.3 gives an overview of various coverage problems.

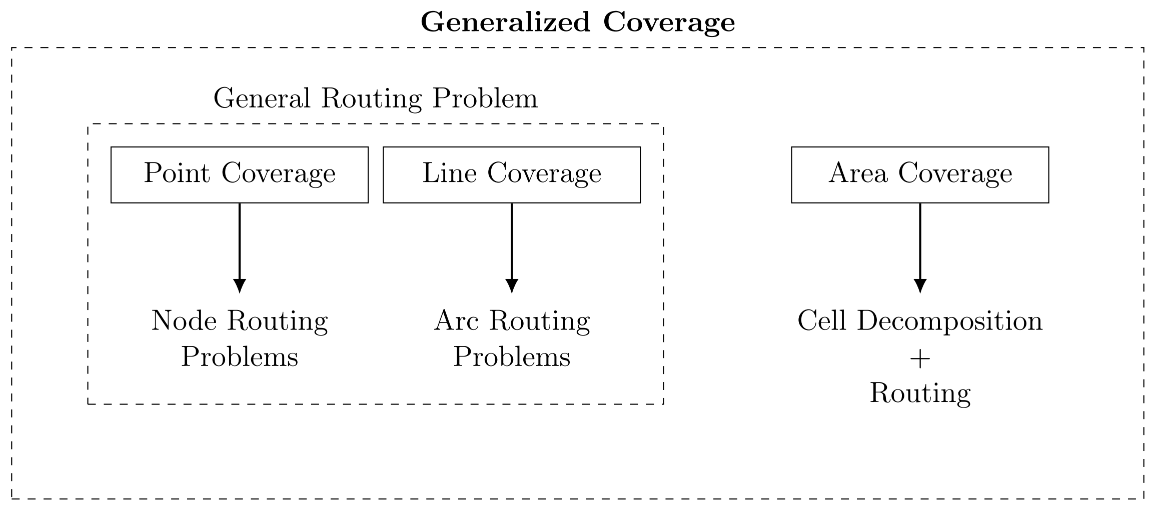


Figure 1.3: An overview of coverage problems: Point coverage and line coverage are related to node and arc routing problems, respectively. The general routing problem is aimed at solving node and arc routing problems simultaneously. The area coverage problem is typically solved using a cell decomposition method and a routing algorithm, with sometimes an intermediate step of service track generation. Generalized coverage unifies the three types of coverage problems in a single framework.

The *generalized coverage problem* can be defined as follows: Given a set of features in an environment, which may be points, lines, curves, and areas, find a set of routes for a team of robots that efficiently covers all the features while respecting practical constraints on the robots.

The dissertation comprehensively studies the line coverage problem, which lays the foundation for the generalized coverage problem. We develop algorithms to transform the point and area features into line features and use line coverage algorithms to solve generalized coverage efficiently. The algorithms substantially improve the state of the art for coverage problems while addressing several practical aspects pertinent to coverage problems.

1.2 Addressing Practical Challenges in Robotics

There are several practical challenges in deploying robots to real-world environments. Several approaches significantly abstract the coverage problems to formulate them as standard routing problems. These yield an approximation to the real-world scenario, and inaccuracies are addressed by post-processing the solutions. This dissertation addresses the following practical aspects pertinent to coverage problems directly in the formulations so that the algorithms provide high-quality solutions for real-world applications.

- **Cost models:** When a robot traverses the environment, it incurs a *cost* such as travel time. The objective of a coverage problem is to minimize the total cost of the routes for the robots while ensuring all the features are visited. A solver that gives high-quality solutions in terms of total cost in reasonable computation time is preferred. The formulations presented in the dissertation can handle arbitrary non-negative cost models, thereby generalizing to a wide range of optimization criteria.

- **Resource constraints:** Mobile robots can be severely limited in terms of the resources available to them. Aerial robots, in particular, have a short operation time due to low battery capacity and high consumption of energy. It is, therefore, important that route planning algorithms take resource constraints into account to ensure the safe recovery of the robots. As robots traverse the environment, they incur *demands* on the resources available to them, and the total demand of traversing a planned route for a robot should be less than its resource constraint. Such a limit on a resource is also referred to as the *capacity* of a robot.

- **Two travel modes:** A unique characteristic of our formulation is that we allow two modes to travel for the robots—servicing and deadheading. A robot is said to be *servicing* an environment feature when it performs task-specific actions such as collecting sensor data; otherwise, it is *deadheading*. Our formulation models different cost functions and resource demands for the two modes of travel for the robots. These modes enable the algorithms to optimize the operation time, conserve energy, and reduce the amount of sensor data.

- **Asymmetry in costs and demands:** In many robotics applications, the cost of travel and the resource demands are direction-dependent. For example, a ground robot traveling uphill can take longer and consume more energy than traveling downhill. Similarly, the costs and the resource demands of aerial robots may depend on the direction of travel due to wind conditions. Hence, we consider asymmetric cost and demand functions for servicing and deadheading. Such asymmetric functions can also model one-way streets for ground robots.

- **Turning costs and nonholonomic constraints:** Sharp turns can be very expensive for robots as they need to slow down, take the turn, and accelerate again. Similarly, nonholonomic robots such as fixed-wing UAVs cannot make point turns. It is imperative to account for the turning costs and nonholonomic constraints to ensure efficient and feasible navigation of the robots.

- **Multiple depots:** A *depot* is a location in the environment from where the robots start and end their routes. When the environment is vast, it may not be possible to service all the features from a single depot location. In such situations, it is imperative to have a multi-depot formulation where the robots have the flexibility to start and end their routes from one of several depots to optimize the routes.

1.3 Dissertation Overview

The dissertation next presents two chapters dedicated to the line coverage problem. We study single robot line coverage in Chapter 2, and line coverage with multiple robots in Chapter 3. These chapters form the foundation of generalized coverage. Area coverage is considered in Chapter 4, and generalized coverage is studied in Chapter 5.

Single Robot Line Coverage (Chapter 2): We first consider the coverage of line features using a single robot. Given a set of line features, the single robot line coverage problem is to find a route for a robot to service each of the features exactly once. We pose line coverage as an optimization problem on graphs and formulate an integer linear program (ILP), along with a proof of correctness. The ILP formulation gives an optimal solution when such a solution exists. The problem is NP-hard, and thus, we develop approximation algorithms that have a polynomial-time computation complexity with guarantees on the quality of the solution. The main algorithm partitions the problem into three cases based on the structure of the *required graph*, i.e., the graph induced by the line features. The approximation algorithms are designed for the three cases using a network-flow formulation.

Line Coverage with Multiple Robots (Chapter 3): We formulate the line coverage with multiple resource-constrained robots. The task is to find a set of routes for a team of robots to service a given set of line features while respecting the resource constraints of the robots. We extend the ILP formulation developed for single robot line coverage to multiple resource-constrained robots. We develop a constructive heuristic algorithm, Merge-Embed-Merge (MEM), that has a polynomial-time computation complexity and gives high-quality solutions. We further formulate the multi-depot version for large graphs where a route can start and end at any of the depots. Modifications are made to the MEM algorithm to handle turning costs and nonholonomic constraints.

Area Coverage (Chapter 4): The area coverage problem is to service a given set of two-dimensional regions using a team of resource-constrained robots. We incorporate the sensor field-of-view of the robots in the problem. The formulation for solving the area coverage problem consists primarily of three components: (1) Cell decomposition of the environment, (2) Service track generation for individual cells, and (3) Routing to traverse the service tracks. The central idea is to transform the area coverage problem into a line coverage problem with multiple robots. The service tracks form the line features that the robots must service, and algorithms for the line coverage problem are used to generate routes for the team of robots. Our formulation facilitates a significant generalization of the cell decomposition component to reduce the number of turns the robots must take. In particular, the cells are no longer required to be *monotone* polygons (Berg et al., 2008) with respect to the service direction. Furthermore, allowing cells to be non-monotone polygons with holes enables the additional merging of adjacent cells with the same service directions. Merging adjacent cells reduces the number of service tracks by avoiding overlapping sensor coverage regions at the common boundary of a pair of adjacent cells.

Generalized Coverage (Chapter 5): Finally, we unify the coverage of the three different types of features in the generalized coverage framework. We elucidate the transformation of point features to line features. With the transformation of point and area features to line features, the generalized coverage problem is reduced to the line coverage problem with multiple resource-constrained robots. The transformation allows us to incorporate various practical aspects of robotics into the problem, such as two travel modes, asymmetric costs and demands, turning costs and nonholonomic constraints, and multiple depot locations.

A strong motivation for the coverage problems is the application to real-world applications. Hence, we extensively evaluate the algorithms developed in the dissertation in simulations and experiments in real environments. A dataset[1] of 50 road networks from the most populous cities in the world was generated by extracting data

[1]The dataset is available at:
`https://github.com/UNCCharlotte-CS-Robotics/LineCoverage-dataset`

from OpenStreetMap[2]. The road networks represent environments with line features, and they provide widely varying graph structures for a thorough evaluation of the algorithms. We evaluate the algorithms for area coverage on two existing datasets for inspection using aerial robots and vacuuming using ground robots. We perform several experiments on the UNC Charlotte campus using commercial UAVs.

1.4 Publications

Peer-reviewed publications:

- (Agarwal and Akella, 2020). Line Coverage with Multiple Robots. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3284–3254, Paris, France.

- (Agarwal and Akella, 2021). Approximation Algorithms for the Single Robot Line Coverage Problem. In LaValle, S. M., Lin, M., Ojala, T., Shell, D., and Yu, J., editors, *Algorithmic Foundations of Robotics XIV*, pages 534–550, Cham, Germany. Springer International Publishing.

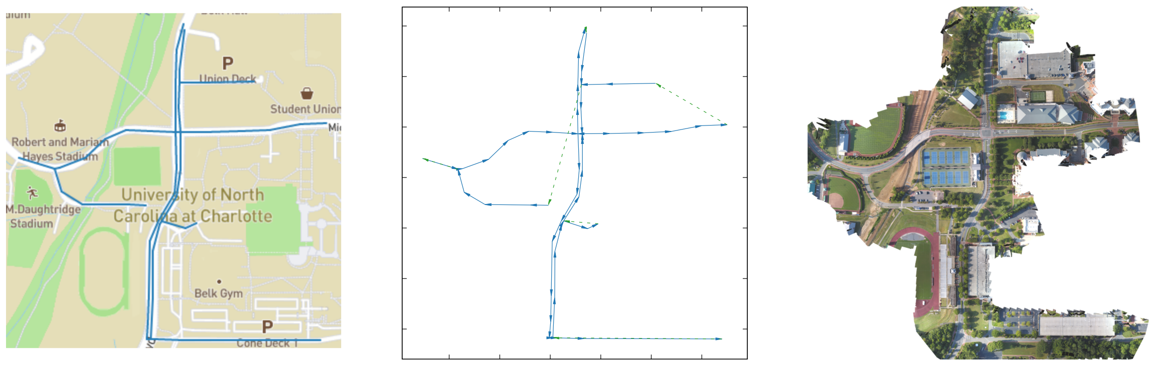
- (Agarwal and Akella, 2022b). Area Coverage With Multiple Capacity-Constrained Robots. *IEEE Robotics and Automation Letters*, 7(2):3734–3741. Selected for presentation at the *IEEE International Conference on Robotics and Automation (ICRA)*, May 2022.

[2]`https://www.openstreetmap.org/`

CHAPTER 2: SINGLE ROBOT LINE COVERAGE

Line coverage is the task of servicing a set of one-dimensional features in an environment. It is important for the inspection of linear infrastructure such as road networks, power lines, and oil and gas pipelines. This chapter addresses the single robot line coverage problem for uncrewed aerial and ground vehicles by modeling it as an optimization problem on a graph. We present an integer linear programming formulation with proofs of correctness. Using the minimum cost flow problem, we develop approximation algorithms with guarantees on the solution quality. The main algorithm partitions the problem into three cases based on the structure of the *required graph*, i.e., the graph induced by the features that require servicing. We present an optimal algorithm for Eulerian required graphs and a 2-approximation algorithm for connected required graphs. For the general case of a required graph with C connected components, the approximation algorithm uses the asymmetric traveling salesperson problem (ATSP). We give a 3-approximation algorithm when $C \in \mathcal{O}(\log(n))$, where n is the number of vertices in the input graph. The approximation factor is $\alpha(C) + 2$ in general, using an $\alpha(C)$-approximation algorithm for ATSP on a graph with C vertices. We evaluate our algorithms on road networks from the 50 most populous cities in the world. The algorithms, augmented with improvement heuristics, run within 3 s and generate solutions that are within 10% of the optimum. We experimentally demonstrate our algorithms with aerial robots on the UNC Charlotte campus road network.

2.1 Introduction


Union Deck
Student Union
Robert and Mariam Hayes Stadium
M.Daughtridge Stadium
University of North Carolina at Charlotte
Belk Gym
Cone Deck 1

(a) Input Road Network (b) Line Coverage Route (c) Orthomosaic

Figure 2.1: Line coverage of a road network by an autonomous UAV: (a) A region of the UNC Charlotte campus road network; blue lines show required edges to be serviced. Non-required edges, not shown, are straight lines between pairs of vertices. The network data was extracted using OpenStreetMap. (b) An optimal coverage tour to cover the road network is shown; dashed segments indicate deadheading travel. (c) An orthomosaic map of the road network generated from photos taken by the UAV along the required edges of the coverage tour.

Line coverage is the task of servicing linear environment features using sensors or tools mounted on a robot. The features to be serviced are modeled as one-dimensional segments (or curves); all points along the segments must be visited. Consider an application scenario, occurring due to a natural disaster such as flooding, in which an uncrewed aerial vehicle (UAV) with cameras is deployed for assessment of connectivity of a road network for emergency services. The UAV must traverse the line segments corresponding to the road network and use its cameras to capture images. It may also travel at higher speeds from one point to another while not capturing images. The following question then arises: How should a tour for the robot be planned such that it traverses each road network segment and minimizes the flight time? Figure 2.1 depicts such a scenario with an optimal tour for a UAV and an orthomosaic generated from the images collected during the flight. Power lines and oil and gas pipelines have similar linear infrastructure that requires frequent inspection. Additional applications arise in perimeter inspection and surveillance, traffic analysis of road networks, and welding operations. Line coverage algorithms can also be used as a subroutine for routing in area coverage problems where a 2D region is decomposed into line segments.

Line coverage is closely related to arc routing problems (ARPs) studied in the operations research community (Corberán and Laporte, 2014). ARPs have been studied for snow plowing, spraying salt, and cleaning road networks (Corberán et al., 2021). ARPs and their algorithms are designed specifically for human-operated vehicles. The above tasks can potentially be automated with uncrewed ground vehicles (UGVs). However, line coverage has received limited attention in the robotics community. In this chapter, we design algorithms for line coverage for autonomous systems, applicable to both UAVs and UGVs.

The line coverage problem, modeled using a graph, has two defining attributes: (1) The edges in the graph are classified as required and non-required, and (2) Robots have two modes of travel—servicing and deadheading. *Required edges* correspond to the linear features to be covered, and the *non-required edges* can be used by a robot to travel from one vertex to another to reduce cost. The vertices in the graph represent the endpoints of the edges. The robot is said to be *servicing* a required edge when it performs task-specific actions such as collecting sensor data. Each required edge needs to be serviced exactly once. The robot may also traverse an edge without performing servicing tasks to optimize the travel time, conserve energy, or reduce the amount of sensor data. This is known as *deadheading*, and both types of edges may be used any number of times for this purpose.

There is a service cost and a deadhead cost (e.g., travel time) associated with each required edge, and they are incurred each time an edge is serviced or deadheaded, respectively. Only the deadhead cost is associated with the non-required edges. The sum of the service and deadhead costs is to be minimized. As task-related actions are performed only when servicing a required edge, and not while deadheading, the service costs are considered to be more than or equal to the deadhead costs. For example, with travel time as the cost model, a UAV servicing an edge by recording images may travel slower than when deadheading to avoid motion blur. In contrast, the service and deadhead costs are usually assumed to be identical for the required edges in standard ARPs, and therefore, the line coverage problem generalizes the standard problems.

In many robotics applications, the cost of travel is direction-dependent. For example, for ground robots, the cost of traveling uphill can be significantly higher than that of traveling downhill. Similarly, for UAVs, the cost of an edge may differ along the two directions due to wind conditions. Hence, we consider the graph to have asymmetric edge costs for both servicing and deadheading. Asymmetric edges can also model one-way streets for ground robots.

The *single robot line coverage problem* is the problem of finding a coverage tour that minimizes the total travel cost while ensuring that each required edge is serviced exactly once. The practical benefits of our line coverage approach are: (1) The algorithms ensure that the required edges are covered efficiently by optimizing the total cost of the coverage tour, e.g., operation time. (2) In contrast to using area coverage, only the relevant features are inspected, thus reducing the inspection time, the amount of sensor data, and the time for data analysis. (3) Line coverage allows distinct costs for servicing and deadheading, and permits asymmetric costs.

The single robot line coverage problem is a generalization of the rural postman problem (RPP), introduced by Orloff (1974). The NP-hardness of the RPP, shown by Lenstra and Kan (1976), implies the single robot line coverage problem is NP-hard. This makes it imperative to develop approximation algorithms.

Contributions: In this chapter, we elucidate the single robot line coverage problem and develop approximation algorithms. We analyze the problem in stages—going from a simpler problem to the most general version. The problems are based on the structure of the *required graph*, i.e., the graph induced by only the required edges.

The contributions of the chapter are:

1. We pose the single robot line coverage problem as an optimization problem on graphs and develop an integer linear programming (ILP) formulation that gives an optimal solution if a solution exists. We provide formal proofs for the correctness of the formulation.

2. We develop a linear relaxation to the ILP formulation and model it using a minimum cost flow problem. The model is used to design an optimal algorithm for graphs with Eulerian required graphs. A 2-approximation algorithm is then developed for connected required graphs.

3. An $\alpha(C) + 2$ approximation algorithm is presented for the general case of a required graph with multiple connected components, where C is the number of connected components, and $\alpha(C)$ is the approximation factor for an algorithm for the asymmetric traveling salesperson problem. Proofs for the approximation guarantee are provided for all the algorithms.

4. We publish a dataset[1] consisting of road networks of the 50 most populous cities in the world and present simulation results showing that the algorithms compute solutions within 10% of the optimum. The algorithms find solutions to the instances within 3 s and are sufficiently fast for robotics applications. Experimental validation of the algorithms is performed. We will also provide an open-source implementation[2] of our algorithms.

This chapter develops a thorough theoretical analysis of the formulations and the algorithms, provides extensive simulation results, and validates the algorithms in experiments with UAVs. In particular, formal proofs for the correctness of the ILP

[1]`https://github.com/AgarwalSaurav/LineCoverage-dataset`

[2]The source code is available at:
`https://github.com/AgarwalSaurav/LineCoverage-library`

formulation are provided. Detailed theoretical analysis of the problem, with a running example, furnishes insights into the structure of the problem that has been instrumental in developing the algorithms. Further improvements to the algorithms are made using heuristic subroutines. These lead to an efficient and fast implementation that we demonstrate on a new dataset of road networks. We additionally demonstrate the application of our algorithms through two experiments on a campus road network.

Organization: The rest of the chapter is organized as follows. The related work is discussed in Section 2.2. The single robot line coverage problem is formally described in Section 2.3. The section provides the ILP formulation and the linear relaxation along with formal proofs. The approximation algorithms are developed in Section 2.4. The simulations and experiments are discussed in Section 2.5. Section 2.6 summarizes the chapter.

2.2 Related Work

The line coverage problem belongs to the broad class of arc routing problems (ARPs). A hierarchy of standard arc routing problems and the single robot line coverage problem is shown in Figure 2.2. The ARPs are usually applied to transportation problems in which servicing is related to tasks such as delivery and pick up of goods (Corberán and Laporte, 2014). Hence, the travel distances are used as costs and often have the same value whether the edge is serviced or deadheaded. Separate and asymmetric service costs are typically not considered.

Arc Routing Problems for a Single Vehicle: The Chinese postman problem (CPP) is to find an optimal tour such that every edge in a given undirected and connected graph is traversed at least once. Edmonds and Johnson (1973) used matching and network flows to solve the CPP on undirected, directed, and Eulerian mixed graphs. They also presented an approximation algorithm for the CPP on mixed graphs that are not necessarily Eulerian. Frederickson (1979) presented a 5/3-approximation algorithm for the CPP on mixed graphs by using a combination

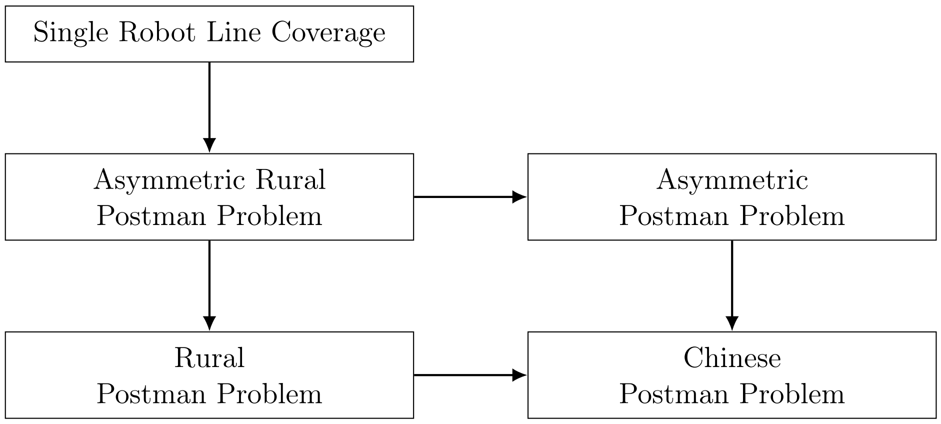

Single Robot Line Coverage
Asymmetric Rural Postman Problem
Asymmetric Postman Problem
Rural Postman Problem
Chinese Postman Problem

Figure 2.2: A hierarchy of arc routing problems with a single vehicle/robot. An arrow from problem A to problem B indicates B is a special case of A. The single robot line coverage problem generalizes all the other depicted arc routing problems. Postman problems on asymmetric graphs are also termed *windy*, for example, the windy postman problem (WPP) and the windy rural postman problem.

of two approximation algorithms. The approximation factor was later improved to 3/2 by Raghavachari and Veerasamy (1999a).

The asymmetric postman problem, also known as the windy postman problem (WPP), is the CPP with asymmetric edge costs. This problem is NP-hard, as shown by Guan (1984). Win (1989) solved the WPP for Eulerian graphs in polynomial time by modeling it as a minimum cost flow problem. Win also designed a 2-approximation algorithm for WPP on general graphs using matching (to make the graph Eulerian) and minimum cost network flow. Raghavachari and Veerasamy (1999b) gave a 3/2-approximation for the WPP. The CPP and the WPP do not allow non-required edges in the graph.

When the edges to be serviced are a subset of the edges in the graph, we have the rural postman problem (RPP). It was proved that the RPP is NP-hard by Lenstra and Kan (1976). For the RPP, Frederickson (1979) gave a 3/2-approximation algorithm similar to the algorithm by Christofides (1976) for the metric traveling salesperson problem (TSP). The asymmetric RPP considers asymmetric travel costs. This problem is also referred to as the windy rural postman problem. Bevern et al. (2017)

showed that if the n-vertex asymmetric traveling salesperson problem (ATSP), subject to the triangle inequality, is $\alpha(n)$-approximable in $t(n)$ time, then n-vertex RPP on an asymmetric and mixed graph is $(\alpha(C)+3)$-approximable in $O(t(C)+n^3\log n)$ time, where C is the number of weakly connected components in the subgraph induced by required arcs and edges. The single robot line coverage problem is closely related to the asymmetric RPP. However, in the asymmetric RPP the costs of deadheading and servicing a required edge are the same. Any instance of the asymmetric RPP can be converted to that of the line coverage problem by setting the cost of deadheading a required edge to the cost of the edge. The algorithms presented in this chapter are applicable for the asymmetric RPP. The guarantee on the approximation factor for our algorithms is an improvement over the previously best-known algorithms given by Bevern et al. (2017) for the asymmetric RPP.

Exact and metaheuristic methods have been proposed for the ARPs, and they are covered in the survey paper by Corberán and Prins (2010) and the monograph by Corberán and Laporte (2014). Exact methods include branch-and-cut with specific cutting plane procedures, branch-and-price, and column generation. One of the key techniques for these algorithms is to incorporate additional constraints that tighten the feasible space of the linear relaxation. Metaheuristic algorithms, such as scatter search, tabu search, and variable neighborhood descent, have also been used to solve the ARPs. However, these algorithms are not particularly suitable for robotics applications as they require significant computational resources. Moreover, they typically require a good initial solution as an additional input in order to upper bound the optimal cost of the instance. The algorithms presented in this chapter can be used to provide such an initial solution with a guaranteed upper bound provided by the approximation factor.

Asymmetric Traveling Salesperson Problem (ATSP): A dynamic programming algorithm that runs in $\mathcal{O}(n^2 2^n)$ computation time, where n is the number of vertices in the input graph, was given by Held and Karp (1962) and Bellman (1962). The algorithm gives optimal solutions and can efficiently solve small instances. Svensson et al. (2018) were the first to present a constant-factor approximation algorithm for the ATSP with the triangle inequality. Traub and Vygen (2020) improved the approximation ratio to $22 + \epsilon$, $\epsilon > 0$ for the ATSP. These results are relevant for the theoretical guarantees on the approximation factor of our algorithms.

Line Coverage in Robotics: Line coverage has been used in robotics for the inspection of road networks and object boundaries. Dille and Singh (2013) model the problem of road network coverage through tessellation of the road network by circles corresponding to the sensor footprint and finding a subset of the circular regions that covers the entire road network. Algorithms for node routing problems, such as the TSP and multiple TSP, are then used to find tours for the robots. A mixed integer linear programming formulation and a heuristic algorithm have been proposed for coverage of road networks by Oh et al. (2014) using Dubins curves with Euclidean distances as costs. The nearest insertion heuristic method, originally designed for the TSP, is used to find a sequence of edges to be visited while incorporating Dubins curves. The sequence is thereafter split across a team of robots using an Auction algorithm. Easton and Burdick (2005) proposed a constructive heuristic algorithm for the RPP with k vehicles for coverage of 2D object boundaries. The algorithm first groups the required edges into k clusters and computes a representative edge for each cluster. Additional edges are added to each cluster to ensure connectivity. Tours are computed for each of the clusters independently using the polynomial-time CPP algorithm proposed by Edmonds and Johnson (1973). Williams and Burdick (2006) developed algorithms for boundary inspection while considering revision to the path plan for the robots to account for the changes in the environment and in the robot team sizes. Xu and Stentz (2010) use CPP and RPP formulations for line coverage and consider the case where the prior map information may be incomplete.

They propose heuristic algorithms that can regenerate solutions rapidly when new map information is incorporated with the prior map. Xu and Stentz (2011) extended this work to multiple robots using k-means clustering to decompose networks into smaller components, similar to the algorithm presented by Easton and Burdick (2005). Campbell et al. (2018) presented application of ARPs using a single UAV, where they allow the UAV to service a required edge in parts. The UAV may service a part of a required edge, move to some other edge, and come back later to service the remaining parts of the required edge. They convert the problem into standard ARPs by discretizing each required edge. The costs are considered to be Euclidean distances. Our algorithms are directly applicable to the discretized version of this problem.

These papers illustrate various applications of the line coverage problem. However, they do not consider asymmetric edge costs or distinct service and deadhead costs. Moreover, the heuristic algorithms do not provide theoretical guarantees on the quality of the solutions. The formulation and the algorithms presented in this chapter address these shortcomings of the prior work.

Arc Routing Problems in Area Coverage: Arc routing problems have been used in robotics primarily as a subroutine in area coverage problems to generate efficient routes for a robot. Arkin et al. (2000) use an algorithm similar to the one given by Edmonds and Johnson (1973) for the CPP to find routes for a robot for the *milling problem*, a variant of the area coverage problem wherein the tool is constrained within the workspace. Mannadiar and Rekleitis (2010) formulate the area to be covered in terms of edges in a Reeb graph. Optimal solutions to the CPP were used to compute an Euler tour for coverage of available free space while minimizing the path length. Karapetyan et al. (2017) use the CPP formulation for k robots to find routes for multiple robots on a Reeb graph. They used the CPP to compute a large Euler tour and then break it into smaller tours using the algorithm given by Frederickson et al. (1976). Our algorithms for the single robot line coverage problem are directly applicable to the above techniques to generate routes for the area coverage problem.

2.3 Problem Statement

We now model the single robot line coverage problem as an optimization problem on a graph. We are given a connected undirected graph $G = (V, E, E_r)$, where V is the set of vertices, E is the set of edges, and $E_r \subseteq E$ is the set of required edges. The set E can contain parallel edges between two vertices, i.e., we allow for G to be a multigraph. The service and deadhead costs are given as inputs along with the graph. The *single robot line coverage problem* is to find a coverage tour that minimizes the total cost of travel on the graph, such that each of the required edges in E_r is serviced exactly once.

For each edge e in E we associate two directional arcs a_e and $\bar{a}_e$ that are opposite in direction to one another. If a robot *services* a required edge $e \in E_r$ in the direction a_e, then a service cost $c_s(a_e)$ is incurred; similarly for the direction $\bar{a}_e$. If a robot traverses an edge without servicing it, the robot is said to be *deadheading*; for example, this occurs when a robot is traveling from a vertex of an edge to that of another edge using a non-required edge. Both required and non-required edges may be deadheaded. Deadhead costs for an edge e are denoted by $c_d(a_e)$ and $c_d(\bar{a}_e)$. We use $c_s(A)$ and $c_d(A)$ to denote the corresponding sums of the service and deadhead costs for a set of arcs A. We denote by $\bar{A}$ the set of arcs oppositely directed to the arcs in A.

We consider the edge costs, for both servicing and deadheading, to be direction dependent, i.e., the graph is asymmetric. For example, $c_s(a_e)$ may differ from $c_s(\bar{a}_e)$. The service and deadhead costs can be arbitrary positive numbers, with the constraint that the service cost of an edge is no less than the deadhead cost in the same direction. The costs, such as travel time, appear in the objective function of the problem. Since we allow edge costs to be asymmetric, the model allows both directed and mixed graphs. This can be achieved by setting the cost in the direction opposite to an arc to a very large constant. We also allow multiple copies of the edges and can model repeated servicing of segments.

2.3.1 Preliminaries

Let $G = (V, E, E_r)$ be a connected undirected graph for the line coverage problem, such that $E_r \subseteq E$. The subgraph $G_r = (V_r, E_r)$ induced by the set of required edges E_r is called the *required graph* of G; $V_r \subseteq V$ is the set of vertices that have at least one edge in E_r incident on them. The set of non-required edges is denoted by $E_n = E \backslash E_r$. We define the set of all arcs to be $\mathcal{A} = \bigcup\{a_e, \bar{a}_e\},\ \forall e \in E$. Similarly, $\mathcal{A}_r$ is defined for the set of required edges. If an arc a represents the travel direction from vertex u to vertex v, then the vertices u and v are called the tail $t(a)$ and head $h(a)$ of a, respectively. We denote by $H(\mathcal{A}, v)$ all the arcs $a \in \mathcal{A}$ that have v as the head. Similarly, $T(\mathcal{A}, v)$ is defined for the tail. The *degree* of a vertex $v \in V$ is the number of edges incident on v. A *walk* in a graph G is a non-empty alternating sequence $v_1 e_1 v_2 e_2 \dots e_k v_{k+1}$ of vertices in V and edges in E such that the tail of e_i is v_i and head of e_i is v_{i+1} for all $1 \leq i \leq k$. A *closed walk* is a walk with the same start and end vertex, i.e., $v_1 = v_{k+1}$. An *Euler tour* is a closed walk such that every edge in the graph is traversed exactly once. A graph that has an Euler tour is called *Eulerian*. It is well established that an undirected graph is Eulerian if and only if every vertex has an even degree, see e.g., Papadimitriou and Steiglitz (1982).

Let $D = (V, A)$ be a *directed graph* (digraph) with V as the set of vertices and A as the set of (directed) arcs. The digraph is strongly connected if there exists a directed path from any vertex in V to any other vertex in V. Analogous to the undirected graph, $T(A, v)$ and $H(A, v)$ are defined for the arc set A and a vertex $v \in V$. The *indegree* of a vertex $v \in V$, denoted by $\mathrm{indeg}(v)$, is the number of arcs entering the vertex v. Similarly, the *outdegree* of a vertex $v \in V$, denoted by $\mathrm{outdeg}(v)$, is the number of arcs going out of the vertex v. A digraph is Eulerian if and only if the graph is strongly connected and *balanced*, i.e., $\mathrm{indeg}(v) = \mathrm{outdeg}(v), \forall v \in V$.

Imbalance $\mathcal{I}(A, v)$ for the arc set A at a vertex v is given by $\text{outdeg}(v) - \text{indeg}(v) = |T(A, v)| - |H(A, v)|$. Analogous to the undirected graph, a *diwalk* is a sequence $v_1 a_1 v_2 a_2 \dots a_k v_{k+1}$ of vertices and arcs in a digraph $D = (V, A)$ such that the tail of a_i is v_i and head of a_i is v_{i+1}. A *closed diwalk* is a walk with the same start and end vertices. An *Eulerian tour* on an Eulerian digraph is a closed diwalk such that each arc is traversed exactly once. An Euler tour can be constructed from an Eulerian graph (or digraph) in $\mathcal{O}(|A|)$ computational time, see e.g., Papadimitriou and Steiglitz (1982).

Definition 2.3.1. Coverage Tour:
Given a connected graph $G = (V, E, E_r)$, a *coverage tour* is a closed walk in the graph G such that each required edge $e \in E_r$ is serviced exactly once.

Note that in a coverage tour, a required or a non-required edge may be used multiple times for deadheading. We define the following variables:

$$\begin{aligned} &s_{a_e}, s_{\bar{a}_e} \in \{0, 1\}, \text{ and } s_{a_e} + s_{\bar{a}_e} = 1 \quad \forall e \in E_r \\ &d_{a_e}, d_{\bar{a}_e} \in \mathbb{N} \cup \{0\} \quad \forall e \in E \end{aligned} \tag{2.1}$$

The variables s_{a_e} and $s_{\bar{a}_e}$ represent the two opposite directions of servicing the edge e; exactly one of the two can be equal to 1 for a valid coverage tour. The variables d_{a_e} and $d_{\bar{a}_e}$ represent the number of times an edge is deadheaded in the corresponding direction. The cost of a coverage tour τ is to be minimized and is given by:

$$c(\tau) = \sum_{e \in E_r} \left[c_s(a_e) s_{a_e} + c_s(\bar{a}_e) s_{\bar{a}_e}\right] + \sum_{e \in E} \left[c_d(a_e) d_{a_e} + c_d(\bar{a}_e) d_{\bar{a}_e}\right] \tag{2.2}$$

For a valid coverage tour τ we can create an Eulerian digraph $D_e = (V, A_e)$ from these variables by adding each arc as many times as the value of its corresponding variable. The digraph will have the same total cost as the coverage tour, i.e., $c(\tau) = c(A_e)$. A closed diwalk can be obtained from D_e in $\mathcal{O}(|A_e|)$ time. We will often use this equivalent Eulerian digraph representation of a coverage tour in the rest of the chapter.

Definition 2.3.2. $\mathcal{O}$-Notation: Cormen et al. (2009)
For a given function $g(n)$, we denote by $\mathcal{O}(g(n))$ the set of functions

$$O(g(n)) = \big\{ f(n) \,|\, \text{there exist positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \big\}.$$

Definition 2.3.3. Approximation Algorithm: Williamson and Shmoys (2011)
An α-approximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the value of an optimal solution.

In the following subsection, we present an integer linear programming (ILP) formulation for the single robot line coverage problem. The ILP formulation allows us to formally define the problem in the form of an objective and a set of constraints. The ILP formulation provides an optimal solution, if one exists. In subsequent subsections, we provide a linear programming (LP) relaxation of the ILP formulation and relate the LP to a network flow model. The network flow model forms the basis of our approximation algorithms in Section 2.4.

2.3.2 ILP Formulation

The standard ILP formulations for arc routing problems involve an exponential number of constraints for ensuring connectivity (Corberán and Laporte, 2014) and are usually difficult to incorporate into standard ILP solvers. Therefore, we adopt the formulation presented by Gouveia et al. (2010) as later specialized by Agarwal and Akella (2020).

SRLC-ILP (single robot line coverage ILP formulation)

Minimize:

$$c(\tau) = \sum_{e \in E_r} \left[c_s(a_e) s_{a_e} + c_s(\bar{a}_e) s_{\bar{a}_e} \right] + \sum_{e \in E} \left[c_d(a_e) d_{a_e} + c_d(\bar{a}_e) d_{\bar{a}_e} \right] \tag{2.3}$$

subject to:

$$\sum_{a_1 \in H(\mathcal{A}_r, v)} s_{a_1} + \sum_{b_1 \in H(\mathcal{A}, v)} d_{b_1} - \sum_{a_2 \in T(\mathcal{A}_r, v)} s_{a_2} - \sum_{b_2 \in T(\mathcal{A}, v)} d_{b_2} = 0, \quad \forall v \in V \tag{2.4}$$

$$s_{a_e} + s_{\bar{a}_e} = 1, \quad \forall e \in E_r \tag{2.5}$$

$$\sum_{a \in H(\mathcal{A}, v)} z_a - \sum_{a \in T(\mathcal{A}, v)} z_a = \sum_{a \in H(\mathcal{A}_r, v)} s_a, \quad \forall v \in V \setminus \{v_0\} \tag{2.6}$$

$$\sum_{a \in T(\mathcal{A}, v_0)} z_a = \sum_{a \in \mathcal{A}_r} s_a = |E_r| \tag{2.7}$$

$$z_a \leq |E_r|(d_a + s_a), \quad \forall a \in \mathcal{A}_r \tag{2.8}$$

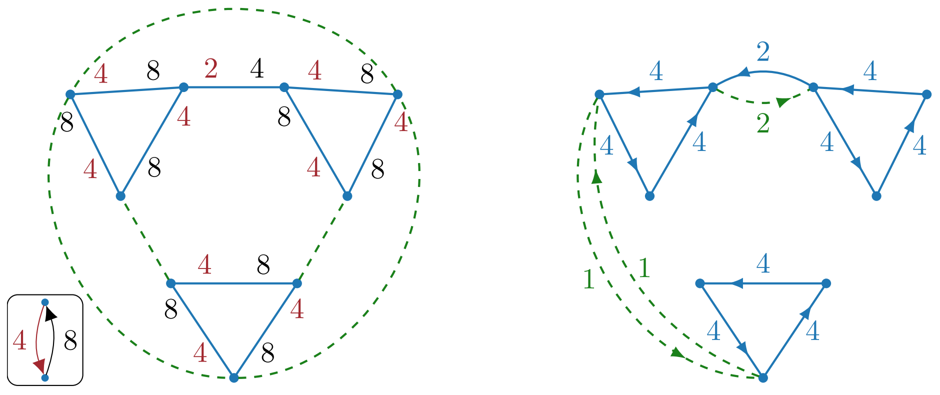
$$z_a \leq |E_r| d_a, \quad \forall a \in \mathcal{A} \setminus \mathcal{A}_r \tag{2.9}$$

$$s_{a_e}, s_{\bar{a}_e} \in \{0, 1\}, \quad \forall e \in E_r \tag{2.10}$$

$$d_{a_e}, d_{\bar{a}_e} \in \mathbb{N} \cup \{0\}, \quad \forall e \in E \tag{2.11}$$

$$z_{a_e}, z_{\bar{a}_e} \in \mathbb{N} \cup \{0\}, \quad \forall e \in E \tag{2.12}$$

The objective function (2.3) minimizes the cost of a coverage tour. The *balance* (or symmetry) constraints (2.4) state that for each vertex, the number of arc traversals into a vertex should be equal to the number of arc traversals out of the vertex. This ensures that the resulting digraph has an Euler tour. The *servicing* constraints (2.5) ensure that each required edge is serviced exactly once and in only one direction. Constraints (2.6)–(2.9) are *connectivity* constraints, with v_0 an arbitrary vertex in V_r. These are a type of generalized flow constraints. The vertex v_0, also referred to as the depot, is a source of a flow equal to the number of required edges $|E_r|$ as given by constraints (2.7). Constraints (2.6) states that a flow of one unit is absorbed each time a required edge is serviced. Building on the analysis by Gouveia et al. (2010), we show that these constraints ensure that the solution digraph is connected in Lemma 1. The variables z_a represent the flow across the edges and are illustrated further in Theorem 2. The *integrality* constraints, (2.10)–(2.12) ensure integrality of the variables. An input graph with its optimal tour is shown in Figure 2.3.



(a) Input graph $G = (V, E, E_r)$

(b) Optimal coverage tour represented as an Eulerian digraph

Figure 2.3: In the input graph (a), the blue solid lines and the green dashed lines represent required and non-required edges, respectively. The service costs in the two directions are shown in the input graph—the costs are closer to the head of the corresponding arc. Deadhead costs for the required edges are here half of the service costs in the respective directions. Non-required edges are set to have a unit cost in both directions. Deadheading costs are not shown. In the optimal coverage tour (b), the servicing arcs are shown as blue solid arcs, while the deadheadings are shown as green dashed arcs. The numbers in Figure (b) indicate the costs of the arcs in the final solution. The cost of the coverage tour is 42.

We now prove the correctness of the ILP formulation by showing that the formulation gives an optimal coverage tour (Theorem 2). There are two components to the proof: (1) The feasible solution space of the SRLC-ILP formulation consists of Eulerian digraphs that correspond to feasible coverage tours (Lemma 1), and (2) Any feasible coverage tour has an equivalent feasible solution to the ILP (part of Theorem 2). Using the above two statements and the fact that the ILP formulation is an optimization problem with the cost of the coverage tour as the objective function, it follows that the optimal solution to the formulation has an equivalent optimal coverage tour.

Lemma 1. *Given a connected graph* $G = (V, E, E_r)$*, a solution to the SRLC-ILP formulation has an equivalent Eulerian digraph.*

Proof. A digraph is Eulerian if (1) each vertex in the digraph is balanced, and (2) the digraph is strongly connected. Given a solution to the SRLC-ILP formulation, we create a digraph $D_e = (V, A_e)$ with the same vertex set as the input graph and an arc set A with $d_a + s_a$ copies of each arc a in $\mathcal{A}_r$ and d_a copies for each arc in $\mathcal{A} \setminus \mathcal{A}_r$. Note that the digraph D_e satisfies indeg = outdeg for each vertex because of balance constraints (2.4).

It remains to show that the digraph D_e is strongly connected. In particular, we will show that any arc $a \in A_e$ with $s_a > 0$ and/or $d_a > 0$ is connected to an arbitrary vertex $v_0 \in V_r$, where V_r is the vertex set corresponding to the required graph. The vertex v_0 must be traversed by a feasible solution of the SRLC-ILP formulation as the vertex v_0 is connected to at least one of the required edges in E_r.

For our proof by contradiction, assume that D_e is not connected, i.e., it has more than one connected component. This means that there is a subset of arcs, with nonzero s_a and/or d_a, forming a smaller closed walk that is not connected to the vertex v_0. We will assume that this closed walk contains at least one arc that is being serviced, for otherwise, it is a closed walk of deadheading arcs only and can be eliminated without an increase in cost. Let $S \subset V$ be the set of vertices corresponding to this closed walk such that $v_0 \notin S$. Define $\bar{S} = V \setminus S$. Note that $v_0 \in \bar{S}$.

Summing the constraints (2.6) over all the vertices in S gives the following equation:

$$\sum_{v \in S} \left(\sum_{a \in H(\mathcal{A}, v)} z_a - \sum_{a \in T(\mathcal{A}, v)} z_a \right) = \sum_{v \in S} \left(\sum_{a \in H(\mathcal{A}_r, v)} s_a \right) \tag{2.13}$$

For the purposes of this proof define the following for any pair of sets $F, G \subseteq V$:

$$\begin{aligned}
\delta(F,G) &= \{a \in A_e \mid t(a) \in F, h(a) \in G\} \\
R(F,G) &= \sum_{a \in \delta(F,G)} s_a, \quad N(F,G) = \sum_{a \in \delta(F,G)} d_a \\
Z(F,G) &= \sum_{a \in \delta(F,G)} z_a
\end{aligned} \tag{2.14}$$

Then (2.13) can be written as:

$$Z(\bar{S},S) - Z(S,\bar{S}) = R(S,S) + R(\bar{S},S) \tag{2.15}$$

Note that $R(S,S) > 0$ and $R(\bar{S},S) = 0$ from our assumption for contradiction. The term $Z(S,\bar{S})$ is non-negative. This implies that $Z(\bar{S},S)$ is strictly positive.

Summing the constraints (2.8) and (2.9) over all arcs $a \in \delta(\bar{S},S)$ gives:

$$Z(\bar{S},S) \leq |E_r| \left(R(\bar{S},S) + N(\bar{S},S)\right) \tag{2.16}$$

Since $Z(\bar{S},S) > 0$ and $R(\bar{S},S) = 0$, it must be that $N(\bar{S},S)$ is strictly positive. Thus, there is at least one arc with tail in $\bar{S}$ and head in S that is being deadheaded. There must also be a deadheading arc with tail in S and head in $\bar{S}$ because of the balance constraints (2.4). Hence, S and $\bar{S}$ are connected, leading to a contradiction.

Thus, the digraph $D_e = (V, A_e)$ is balanced and connected, i.e., the digraph is Eulerian. A coverage tour can be obtained from the Eulerian digraph by computing an Eulerian diwalk in $\mathcal{O}(|A_e|)$ computation time. The cost of an Eulerian diwalk, and the corresponding coverage tour, on the digraph D_e is $c(A_e) = c(\tau)$, where $c(\tau)$ is the value of the objective function for a solution to the SRLC-ILP formulation. Each feasible solution of the SRLC-ILP formulation has a corresponding feasible coverage tour. $\square$

Theorem 2. *Given a connected graph $G = (V, E, E_r)$, the SRLC-ILP formulation gives an optimal coverage tour, if a solution exists.*

Proof. We first prove that any feasible coverage tour τ has a corresponding feasible solution for the SRLC-ILP formulation with the same cost. In other words, the feasible solution space of the SRLC-ILP formulation includes all the feasible coverage tours. Represent the given coverage tour as a closed diwalk $v_1a_1 \dots a_{i-1}v_ia_i \dots v_ka_kv_1$, along with information on whether an arc in the closed walk is serviced or deadheaded. For each variable $s_a, \forall a \in \mathcal{A}_r$ assign its value according to the direction in which the arc is serviced, and for each variable $d_a, \forall a \in \mathcal{A}$ assign its value equal to the number of times the arc is deadheaded. Also, let $D_e = (V, A_e)$ be the corresponding digraph. Note that $c(\tau) = c(A_e)$. Since in a connected closed diwalk indeg = outdeg, constraints (2.4) are satisfied. The diwalk must service each edge exactly once, satisfying (2.5).

Assume, without loss of generality, that the first arc a_1 in the diwalk corresponds to a required edge; we can always rotate the diwalk to obtain an equivalent diwalk satisfying the assumption. Set $v_0 = v_1$, i.e., the first vertex of the diwalk. Assume that the coverage tour visits v_0 only once; we will generalize this later. This implies that there are exactly two arcs in A_e that are connected to v_0, one leaving and one entering. Set $z_a = 0$ for each arc in $\mathcal{A}$. Set $z_{a_1} = |E_r|$, satisfying constraint (2.7). Now traverse the edges in the sequence and direction given by the diwalk. Following the notation for a diwalk, arc a_i leaves vertex v_i. During the coverage tour traversal, if a vertex v_i is traversed for the first time then set z_{a_i} to be $z_{a_{i-1}}$ minus the number of servicing arcs entering v_i. If v_i has already been traversed by an arc whose tail is v_i, then set $z_{a_i} = z_{a_i} + z_{a_{i-1}}$. The value of z_a remains zero for the arcs that were not traversed by the closed walk. Thus, constraints (2.8) and (2.9) are satisfied for all the arcs. The constraints (2.6) will be satisfied at each vertex, other than v_0, by construction.

Now we address the case when the coverage tour traverses the vertex v_0 multiple times. This will result in l loops at v_0, which we index by $j = 1, \ldots, l$. We first perform the same procedure to assign the values of z_a as in the preceding paragraph. Let the first arc (leaving v_0) and last arc (entering v_0) for a loop j be a_{j_1} and a_{j_k}, respectively. Now for each loop j, reduce the value of z for all arcs in the loop j by the value of z_a for the last arc in the loop $a = a_{j_k}$. Finally, increase the value of z for all arcs in any one of the loops by $|E_r| - \sum_{j \in \{1,\ldots,l\}} z_{a_{j_1}}$, in order to satisfy (2.7). An example is shown in Figure 2.4.

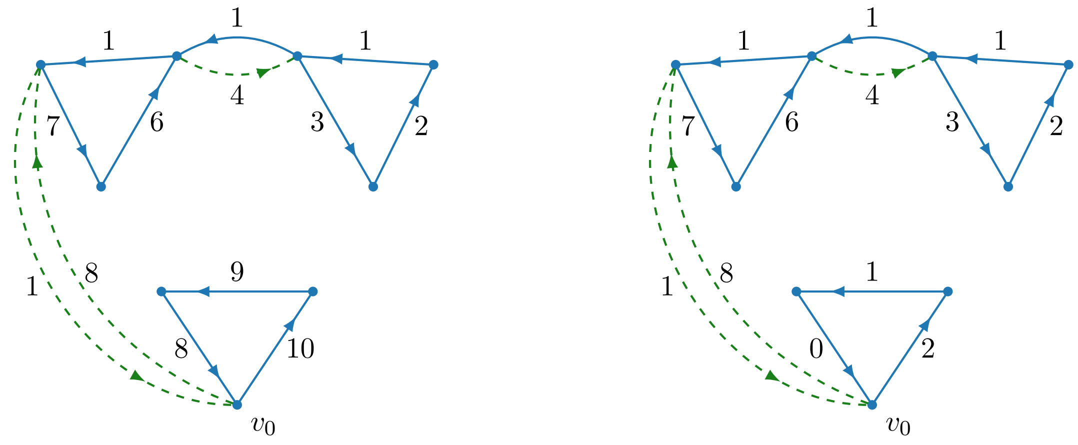


Figure 2.4: Two stages of assigning values of z_a to arcs from a given feasible coverage tour, as discussed in the proof of Theorem 2. The service and deadhead arcs are represented by solid and dashed lines, respectively. The arrows indicate the direction of travel. There are two loops connected to the depot vertex v_0. The numbers in the figures indicate values of z_a in the two stages, with the bottom figure showing the final values. The z values of the service arcs in the bottom triangle loop are reduced by 8. Note that $z_a = 0$ for all other arcs not shown in the digraph.

Hence, we can compute a feasible solution to the SRLC-ILP formulation from a feasible connected closed diwalk, with the same cost, i.e., the feasible solution space has all the feasible closed diwalks (coverage tours). Combining with Lemma 1, any feasible solution to the SRLC-ILP formulation gives a feasible connected closed diwalk of the same cost. Thus, the feasible solution space of the ILP corresponds to exactly the space of the feasible coverage tours—there is no feasible coverage tour that is not part of the feasible solution space of the ILP, and there is no feasible solution to the ILP that is not a coverage tour. As the objective function of the SRLC-ILP formulation corresponds to the cost of a coverage tour, an optimal feasible solution to the formulation will also be an optimal coverage tour for the single robot line coverage problem. □

2.3.3 Linear Relaxation of SRLC-ILP

We consider a linear relaxation of the SRLC-ILP formulation that is closely related to the minimum cost flow problem. The relaxation, as we shall see later, gives insights into the structure of the problem and enables development of the approximation algorithms. The key idea is to first build a min-cost digraph, which is not necessarily balanced, and then use an LP formulation, modeled as a minimum cost flow problem, to reverse some of the service arcs in the digraph and add deadheading arcs such that the resulting digraph is balanced. We will use the flow problem to develop approximation algorithms for the different cases of the single robot line coverage problem based on the structure of the required graph.

As before, let the input graph be $G = (V, E, E_r)$ with a required graph $G_r = (V_r, E_r)$. Modify the SRLC-ILP formulation as follows:

1. Remove the connectivity constraints (2.6)–(2.9) from the SRLC-ILP formulation. When the required graph G_r is connected, the ILP formulation is still valid.

2. Generate a digraph $D_m = (V, A_m)$, using the algorithm MinCostDigraph, which selects the arc with the minimum service cost for each required edge. The min-cost digraph D_m for an input graph is shown in Figure 2.5.

3. Introduce a *reverse* variable r_a for each arc $a \in A_m$, to represent the reversal of service direction of the arc a. The reverse variables r_a take values from $\{0, 2\}$; $r_a = 0$ when the service direction is not changed and $r_a = 2$ when the direction is reversed.

4. Relax the integrality constraints (2.11) so that the deadheading variables $d_{a_e}, d_{\bar{a}_e}$ and the new reverse variables r_a are now continuous.

If an arc's service direction is reversed from a to $\bar{a}$, $r_a = 2$, the imbalance changes by 2, and the total cost changes by $c_s(\bar{a}) - c_s(a)$. We assign *reversal cost* $c_r(a)$ for each arc $a \in A_m$ and set $c_r(a)$ to $\frac{c_s(\bar{a})-c_s(a)}{2}$.

Algorithm 1: MinCostDigraph

```
Input  : Graph $G = (V, E, E_r)$
Output : Minimum cost digraph $D_m = (V, A_m)$
$A_m \leftarrow \emptyset$ ;
for $e \in E_r$ do
    if $c_s(a_e) \leq c_s(\bar{a}_e)$ then
        $a_e$.SERVICE $\leftarrow$ TRUE ;
        $A_m$.INSERT($a_e$) ;
    else
        $\bar{a}_e$.SERVICE $\leftarrow$ TRUE ;
        $A_m$.INSERT($\bar{a}_e$) ;
```

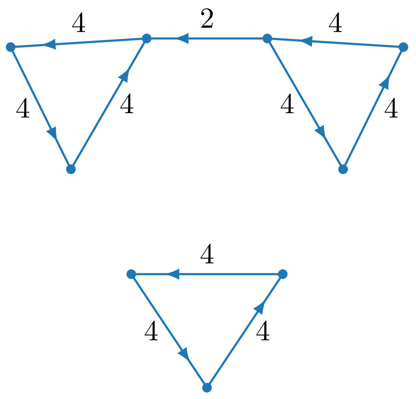


Figure 2.5: The min-cost digraph $D_m = (V, A_m)$ for the input graph given in Figure 2.3. Note that the graph is neither balanced nor connected.

We now state the linear relaxation of the SRLC-ILP formulation.

SRLC-LP (single robot line coverage linear programming formulation)

Minimize:

$$c_s(A_m) + \sum_{a \in A_m} c_r(a) r_a + \sum_{e \in E} \left[c_d(a_e) d_{a_e} + c_d(\bar{a}_e) d_{\bar{a}_e} \right] \tag{2.17}$$

subject to:

$$\sum_{a_1 \in H(A_m,v)} -r_{a_1} + \sum_{b_1 \in H(\mathcal{A},v)} d_{b_1} + \sum_{a_2 \in T(A_m,v)} r_{a_2} - \sum_{b_2 \in T(\mathcal{A},v)} d_{b_2} = \mathcal{I}(A_m, v), \quad \forall v \in V \tag{2.18}$$

$$0 \leq r_a \leq 2, \quad \forall a \in A_m \tag{2.19}$$

$$d_{a_e}, d_{\bar{a}_e} \geq 0, \quad \forall e \in E \tag{2.20}$$

Expression (2.17) is the modified objective function. The first term in the objective function $c_s(A_m)$ is the sum of the service costs of all the arcs in the digraph D_m and is independent of the variables. The imbalance in the digraph D_m at a vertex v is represented by $\mathcal{I}(A_m, v)$. The conditions (2.18) ensure that the digraph corresponding to a feasible solution will be balanced, i.e., the indegree will equal the outdegree at every vertex. The constraints (2.19) state that the variable r_a, corresponding to reversal of service direction, is between 0 and 2; if $r_a = 0$ then the direction of travel is the same as that of the arc in A_m, and if $r_a = 2$ then the direction of the arc is reversed, thereby reversing the service direction.

2.3.4 A Network Flow Graph Model

Arc routing problems are often solved by modeling them as network flow graphs and finding a minimum cost flow. Using this approach, algorithms for the CPP and the WPP were presented by Edmonds and Johnson (1973) and Win (1989), respectively. Inspired by such techniques, we present a network flow graph model for solving the linear programming formulation SRLC-LP and establish its equivalence. We will then use the model to develop approximation algorithms in Section 2.4.

Let $G = (V, E, E_r)$ be the input graph. First, generate a min-cost digraph $D_m = (V, A_m)$ using the algorithm MinCostDigraph(G). Now construct a network flow graph $D_f = (V, A_f)$, in $\mathcal{O}(|V| + |E|)$ time (Algorithm 2), as follows:

1. For each service arc $a \in A_m$, add three arcs a, $\bar{a}$, and a' to A_f with the costs

per unit flow $c_f(\cdot)$ and capacities as given in Table 2.1, which defines the Flow Model. The direction of arc a is same as that in A_m, whereas the direction of arcs $\bar{a}$ and a' are opposite to that of the corresponding arc in A_m. In the flow digraph D_f, the arcs a and $\bar{a}$ will represent deadheadings and the arc a' will represent service reversal.

2. Similarly, for each $b \in \mathcal{A}_n$ where $\mathcal{A}_n = \mathcal{A} \setminus \mathcal{A}_r$, add two arcs b and $\bar{b}$ to A_f, with the costs per unit flow and capacities in Table 2.1. These two arcs, b and $\bar{b}$, represent deadheadings of a non-required edge in the two directions.

Table 2.1: Flow Model (FM): arc costs and capacities. Three arcs $(a, \bar{a}, a')$ are added for each required edge and two arcs $(b, \bar{b})$ for each non-required edge.

Arc	Description	Unit Flow Cost $c_f(\cdot)$	Capacity
a	Forward deadheading	$c_d(a)$	∞
$\bar{a}$	Backward deadheading	$c_d(\bar{a})$	∞
a'	Service reversal	$c_r(a) = \big(c_s(\bar{a}) - c_s(a)\big)/2$	2
b	Non-required forward deadheading	$c_d(b)$	∞
$\bar{b}$	Non-required reverse deadheading	$c_d(\bar{b})$	∞

3. For each vertex $v \in V$, assign the following node flow demand based on the degree of v in D_m:

$$d(v) = \text{outdeg}(v) - \text{indeg}(v) = \mathcal{I}(A_m, v) \tag{2.21}$$

4. Let f_a be the flow along the arc $a \in A_f$, and let the *flow vector* be $\mathbf{f} = [f_a \mid a \in A_f]$. The cost $c(\mathbf{f})$ of a flow $\mathbf{f}$ is:

$$c(\mathbf{f}) = \sum_{a \in A_f} c_f(a) f_a \tag{2.22}$$

A flow digraph D_f is shown in Figure 2.6 for the input graph of Figure 2.3 with the min-cost digraph shown in Figure 2.5.

```
Algorithm 2: ConstructFlowDigraph
Input  : Graph $G = (V, E, E_r)$,
         Digraph $D_m = (V, A_m)$
Output : Flow Digraph $D_f = (V, A_f)$
$A_f \leftarrow \emptyset$ ;
for $a \in A_m$ do
    Insert arcs $a$, $\bar{a}$ and $a'$ into $A_f$, with costs and capacities from to Table 2.1 ;
for $e \in E \setminus E_r$ do
    Let $b_e$ and $\bar{b}_e$ be the arcs corresponding to $e$ ;
    Insert arcs $b_e$ and $\bar{b}_e$ into $A_f$, with costs and capacities from Table 2.1 ;
for $v \in V$ do
    $d(v) \leftarrow \mathcal{I}(A_m, v)$ ;
```

We now formulate a minimum cost flow problem for the network flow graph $D_f = (V, A_f)$ and show that it models the linear relaxation SRLC-LP.

Definition 2.3.4. Minimum Cost Flow Problem

Let $D_f = (V, A_f)$ be a given flow digraph, along with costs, capacities, and node flow demands. Then the minimum cost flow problem is to find a feasible flow $\mathbf{f}$ such that:

1. the cost of flow $c(\mathbf{f})$ is minimized, and

2. demand $d(v)$ is satisfied for all $v \in V$.

Theorem 3. *Let $G = (V, E, E_r)$ be an input graph for the single robot line coverage problem, with minimum cost digraph $D_m = (V, A_m)$ and flow digraph $D_f = (V, A_f)$. The minimum cost flow problem for network flow digraph D_f models the linear relaxation SRLC-LP of the SRLC-ILP formulation for the single robot line coverage problem.*

Proof. Observe that any feasible solution to the minimum cost flow problem is a feasible solution to the linear programming formulation SRLC-LP, and vice versa, using the following relation between the variables:

$$\begin{aligned} &f_a = d_a,\ f_{a'} = r_a,\ f_{\bar{a}} = d_{\bar{a}}, && \forall a \in A_m \\ &f_{a_e} = d_{a_e},\ f_{\bar{a}_e} = d_{\bar{a}_e}, && \forall e \in E \setminus E_r \end{aligned} \tag{2.23}$$

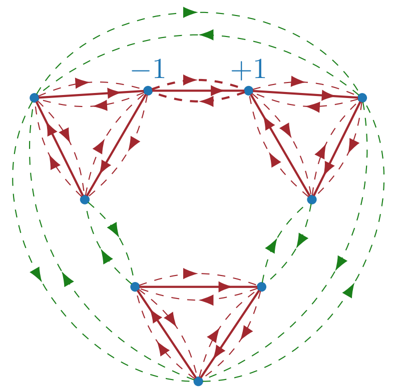
As the capacity of the reversal arcs $r_{a'}$ is set to 2, the flow $f_{a'}$ across any such arc will be no greater than 2, and the constraints (2.19) are satisfied. The flow problem resolves the demand $d(v) = \mathcal{I}(A_m, v)$ for each vertex $v \in V$, and thus, satisfies the balance constraints (2.18). The objective of the minimum cost flow problem $c(\mathbf{f})$ summed with $c_s(A_m)$ is then exactly the objective function (2.17) of SRLC-LP. Thus, the theorem follows. $\square$

Remark. The minimum cost flow, for a graph $G = (V, E)$, can be computed in time $\mathcal{O}\big((m \log n)(m + n \log n)\big)$, as shown by Orlin (1993), where $m = |E|$ and $n = |V|$. Optimal solutions to the minimum cost flow algorithms are integers as the imbalance at each vertex is also an integer (see e.g., Papadimitriou and Steiglitz (1982)), thus $f_{a'} \in \{0, 1, 2\}$. When we set $r_a = f_{a'}$ we may have $r_a = 1$. This in turn violates the integrality constraints (2.10), and the corresponding service variables are half-integral: $s_a = s_{\bar{a}} = 0.5$. For any coverage tour there exists an equivalent network flow digraph with a corresponding flow. However, not all solutions to the network flow digraph have an equivalent coverage tour.

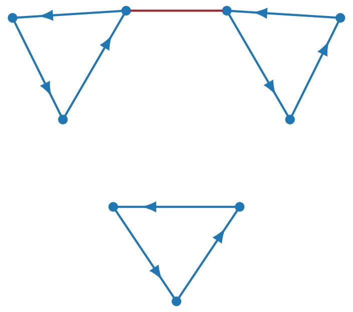
2.4 Approximation Algorithms

We now present approximation algorithms for the single robot line coverage problem by partitioning it into three different cases, as illustrated in Figure 2.7. The cases are based on the structure of the required graph, i.e., the subgraph induced by only the required edges:

1. Eulerian required graph: The algorithm LP-Solve, derived from the network flow model, gives an optimal solution.



(a) Flow digraph $D_f = (V, A_f)$



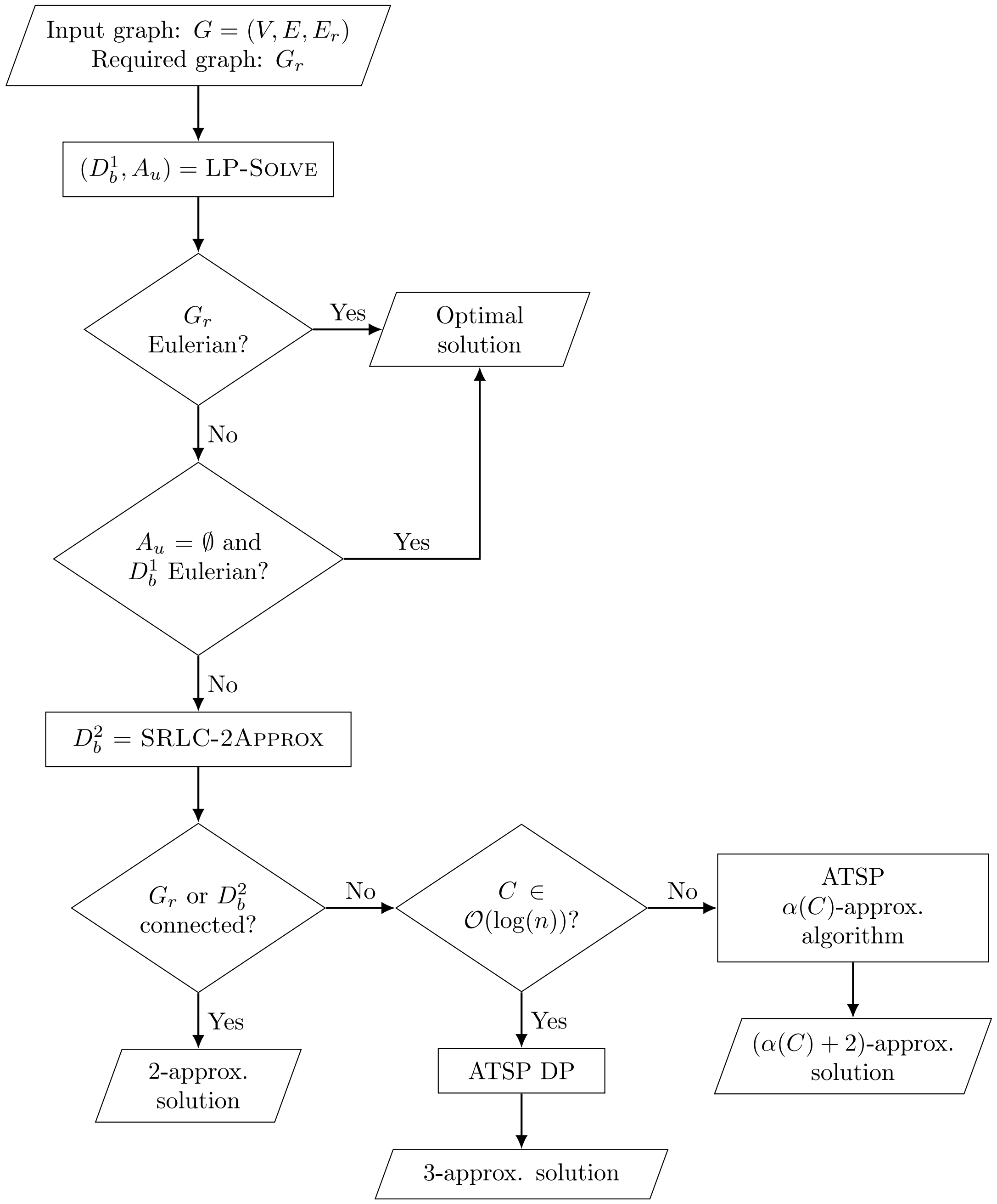
(b) Optimal solution to SRLC-LP with ambiguous (undirected) edges

Figure 2.6: Flow digraph and an optimal solution to the SRLC-LP problem for the input graph G in Figure 2.3 with the min-cost digraph shown in Figure 2.5. (a) The red (darker) solid arcs correspond to service reversal arcs, and the red dashed arcs correspond to the deadheading of required edges. The dashed green (lighter) arcs are flow arcs corresponding to deadheading across non-required edges. The nonzero imbalances for the vertices are shown. (b) The set of arcs corresponding to ambiguous edges A_u—those that remain undirected after solving the flow problem—is shown in red (darker).

2. Connected required graph: The 2-approximation algorithm SRLC-2Approx gives a solution with cost at most twice the optimal cost.

3. General required graph: The $(\alpha(C) + 2)$-approximation algorithm, where C is the number of connected components in the required graph, and $\alpha(C)$ is the approximation factor for the ATSP on a graph with C vertices, gives a solution with cost at most $\alpha(C) + 2$ times the optimal cost. When the number of connected components in the required graph is small, i.e., $C \in \mathcal{O}(\log(n))$, a 3-approximation solution is obtained.

We recast the theoretical results of the previous section in algorithmic form as Algorithm 3, which computes a digraph by solving the linear relaxation to the single robot line coverage problem and creates a balanced digraph $D_b = (V, A_b)$. Given a graph $G = (V, E, E_r)$, we first compute the min-cost digraph $D_m = (V, A_m)$ (line 1) and construct the flow digraph $D_f = (V, A_f)$ from D_m (line 2). We then compute the minimum cost flow $\mathbf{f}$ for the flow digraph (line 3). If the optimal flow across an arc a is 0 or 2, we set the service direction of the corresponding edge to the direction of a or $\bar{a}$, respectively, and add the corresponding arc to the arc set A_b (lines 6–11). For some of the required edges, the optimal flow through the reversal arcs may be 1, i.e., $r_a = 1$. These arcs, denoted by A_u, correspond to the edges whose service direction remains *ambiguous* in the solution to the flow problem (lines 12–13). Finally, we add deadheading arcs to the arc set A_b according to the corresponding optimal flow through the deadheading arcs of the flow digraph (lines 14–20). The digraph $D_b = (V, A_b)$ and the set of ambiguous arcs A_u are the output of the algorithm. Note that the digraph D_b is balanced but may have multiple components, each of which is Eulerian. An output of LP-Solve for the input graph given in Figure 2.3 is shown in Figure 2.6(b). The arcs corresponding to the digraph D_b are shown in blue, whereas the ambiguous edge, corresponding to the edge set A_u, is shown in dark red.



Input graph: $G = (V, E, E_r)$
Required graph: G_r
$(D_b^1, A_u) = $ LP-Solve
G_r Eulerian?
Yes
Optimal solution
No
$A_u = \emptyset$ and D_b^1 Eulerian?
Yes
No
$D_b^2 = $ SRLC-2Approx
G_r or D_b^2 connected?
No
$C \in \mathcal{O}(\log(n))$?
No
ATSP $\alpha(C)$-approx. algorithm
Yes
2-approx. solution
Yes
ATSP DP
3-approx. solution
$(\alpha(C) + 2)$-approx. solution

Figure 2.7: Flowchart illustrating the different cases of the single robot line coverage problem based on the structure of the required graph G_r. The approximation factors of the algorithms depend on the structure of G_r.

Algorithm 3: LP-Solve

```
Input  : Graph $G = (V, E, E_r)$
Output : Digraph $D_b = (V, A_b)$ and undirected arc set $A_u$
$D_m = (V, A_m) \leftarrow$ MinCostDigraph($G$) ;
$D_f = (V, A_f) \leftarrow$ ConstructFlowDigraph($G, D_m$) ;
Compute the minimum cost flow $\mathbf{f}$ for $D_f$ ;
/* Generate digraph $D_b$                                        */
$A_b \leftarrow \emptyset$ ;
for $a \in A_m$ do
    if $f_{a'} = 0$ then
        $a$.service $\leftarrow$ True ;
        $A_b$.insert($a$) ;
    else if $f_{a'} = 2$ then
        $\bar{a}$.service $\leftarrow$ True ;
        $A_b$.insert($\bar{a}$) ;
    else if $f_{a'} = 1$ then
        $A_u$.insert($a$) ;
    $A_b$.insert($f_a$ copies of $a$) ;
    $A_b$.insert($f_{\bar{a}}$ copies of $\bar{a}$) ;
for $e \in E \setminus E_r$ do
    $A_b$.insert($f_{a_e}$ copies of $a_e$) ;
    $A_b$.insert($f_{\bar{a}_e}$ copies of $\bar{a}_e$) ;
```

2.4.1 Eulerian Required Graph

We first consider the case where the required graph $G_r = (V_r, E_r)$, for the input graph $G = (V, E, E_r)$, is Eulerian, i.e., the subgraph is connected and the degree of each vertex in V_r is even. It should be noted that this special case is not the same as the Eulerian graphs for the asymmetric/windy postman problem (WPP) considered by Win (1989) and by Raghavachari and Veerasamy (1999b). This is because non-required edges are permitted, and the deadheading costs for the required edges can differ from their service costs. Thus, an Eulerian tour on the required graph does not ensure a coverage tour with minimum cost. The following lemma shows that the algorithm LP-Solve gives an optimal solution for this case in running time that is polynomial in the number of edges and vertices.

Theorem 4. *Let the input graph be $G = (V, E, E_r)$ such that the required graph $G_r = (V_r, E_r)$ is Eulerian. Then algorithm* LP-Solve *gives an optimal feasible solution for* *the single robot line coverage problem in polynomial time. In particular,* $r_a = f_{a'} \in \{0, 2\}\ \ \forall a \in A_m$ *and* $d_{a_e} = f_{a_e},\ d_{\bar{a}_e} = f_{\bar{a}_e} \in \mathbb{N} \cup \{0\}\ \ \forall e \in E$*.*

Proof. Let the min-cost digraph be $D_m = (V, A_m)$ and the flow digraph be $D_f = (V, A_f)$, for the input graph $G = (V, E, E_r)$. Since the required graph G_r is Eulerian, the number of edges incident at a vertex in G_r is even. The number of outgoing arcs and incoming arcs, at a vertex in D_m, will both be even or will both be odd. Therefore, the node flow demand for a vertex $v \in V$ computed for the digraph D_m will be even. The capacities defined in Table 2.1 are either 2 or ∞. Furthermore, the minimum cost flow algorithm gives integral solutions. Hence, the optimal flow is also even for each arc $a_f \in A_f$ and in particular $r_a = f_{a'} \in \{0, 2\}$. This result can also be derived by establishing total unimodularity of the constraint matrix obtained from the balance constraints (2.18) by replacing each r_a by $2\tilde{r}_a$, d_a by $2\tilde{d}_a$, and $d_{\bar{a}}$ by $2\tilde{d}_{\bar{a}}$. Note that the imbalance $\mathcal{I}(A_m, v)$ is even for each vertex $v \in V$. Thus, we can divide the entire equation by 2, and the constraints will still have integral coefficients and constants.

$$\sum_{a_1 \in H(A_m, v)} -\tilde{r}_{a_1} + \sum_{b_1 \in H(\mathcal{A}, v)} \tilde{d}_{b_1} + \sum_{a_2 \in T(A_m, v)} \tilde{r}_{a_2} - \sum_{b_2 \in T(\mathcal{A}, v)} \tilde{d}_{b_2} = \frac{\mathcal{I}(A_m, v)}{2} \quad \forall v \in V$$

The constraint matrix for the above equation corresponds to an incidence matrix with integers, and thus is totally unimodular. The substituted variables will be integral, and the original variables will all be even. As the required graph is connected, the digraph D_b obtained from the algorithm will also be connected, and thus will be Eulerian.

Let $m = |E|$ and $n = |V|$. The min-cost digraph and the flow digraph can be constructed in $\mathcal{O}(n)$ and $\mathcal{O}(m + n)$ time, respectively. Thus, the running time of the algorithm is dependent on the algorithm for solving the minimum cost flow problem, i.e., $\mathcal{O}\big((m \log n)(m + n \log n)\big)$. $\square$

2.4.2 Connected Required Graph

We now consider the case where the required graph $G_r = (V_r, E_r)$, for the input graph $G = (V, E, E_r)$, is connected but not necessarily Eulerian, i.e., the degree of each vertex in G_r might not be even. The created flow digraph $D_f = (V, A_f)$ may have vertices with odd flow demands (2.21). As a result, the optimal flow values need not be even. While it is not a problem if d_a or $d_{\bar{a}}$ is odd for some arc $a \in \mathcal{A}$, we need to assign service directions to the edges for which the reverse variables r_a is 1 for arcs $a \in A_m$ and potentially add deadheading arcs to make the corresponding digraph Eulerian.

Let A_u be the set of arcs for which the flow for the arc corresponding to the reversal of service direction is 1, i.e., $A_u = \{a \mid r_a = f_{a'} = 1, a \in A_m\}$, as given by algorithm LP-Solve, and let E_u be the corresponding edge set. We first check for cycles in the graph (V, E_u). Such cycles will have a sequence of edges such that the total cost of the edges if oriented in the clockwise direction is the same as the total cost of the edges if oriented in the anti-clockwise direction since the flow obtained is optimal. We can orient such cycles in either clockwise or anti-clockwise order without changing the cost of the solution. Now, let us say we service the edge corresponding to some $a \in A_u$ in the same direction as a. This creates an imbalance of $+1$ at the tail $t(a)$ and -1 at the head $h(a)$. We can resolve this by adding a shortest path deadheading from $h(a)$ to $t(a)$, the cost of which is denoted by $c_d(h(a), t(a))$. The total cost due to traversals of this edge will be the sum of the service cost of a and this deadheading cost. Similarly, we can consider servicing in the direction of $\bar{a}$, and then consider deadheading from $h(\bar{a})$ to $t(\bar{a})$. Of the two combinations, the one that has a lower total cost is selected. This is done for each ambiguous edge corresponding to $a \in A_u$. This idea is described concretely in algorithm SRLC-2Approx, and we will show that it is a 2-approximation algorithm.

Let the optimal flow be $\mathbf{f}$, and the cost of the optimal tour be c^*. Then the optimal

Algorithm 4: SRLC-2Approx

```
Input   : Graph $G = (V, E, E_r)$
Output : Digraph $D_b = (V, A_b)$
$(D_b^{\mathrm{LP}} = (V, A_b^{\mathrm{LP}}),\ A_u) \leftarrow$ LP-Solve$(G)$ ;
/* Let $E_u$ be the edge set corresponding to $A_u$                    */
$A_b \leftarrow A_b^{\mathrm{LP}}$ ;
Find cycles in the graph $(V, E_u)$ ;
Orient cycles in anti-clockwise orientation and add arcs to $A_b$ ;
for $a \in A_u$ do
    $p \leftarrow$ shortest deadheading path from $h(a)$ to $t(a)$ ;
    $\bar{p} \leftarrow$ shortest deadheading path from $h(\bar{a})$ to $t(\bar{a})$ ;
    if $c_s(a) + c_d(p) \leq c_s(\bar{a}) + c_d(\bar{p})$ then
        $a$.service $\leftarrow$ True ;
        $A_b$.insert$(a)$ ;
        $A_b$.insert$(p)$ ;
    else
        $\bar{a}$.service $\leftarrow$ True ;
        $A_b$.insert$(\bar{a})$ ;
        $A_b$.insert$(\bar{p})$ ;
```

value of the linear relaxation z^* of the formulation SRLC-LP is:

$$z^* = c_s(A_m) + c(\mathbf{f}) = c_s(A_m \setminus A_u) + c_s(A_u) + c(\mathbf{f}) \leq c^* \tag{2.24}$$

Let A_d denote the set of arcs corresponding to the service direction and deadheading decided unambiguously by the optimal flow. Then,

$$c(A_d) = c_s(A_m \setminus A_u) + c(\mathbf{f}) - c_r(A_u) \tag{2.25}$$

Thus,

$$z^* = c(A_d) + c_s(A_u) + c_r(A_u) \tag{2.26}$$

Substituting the value of $c_r(A_u) = \frac{c_s(\bar{A}_u) - c_s(A_u)}{2}$ in (2.26) and using (2.24) we have,

$$2c(A_d) + c_s(A_u) + c_s(\bar{A}_u) \leq 2c^*$$

$$\text{or, } c(A_d) + c_s(A_u) + c_s(\bar{A}_u) \leq 2c^* - c(A_d) \tag{2.27}$$

Theorem 5. *Let the input graph be $G = (V, E, E_r)$ such that the required graph $G_r = (V_r, E_r)$ is connected. Then algorithm* SRLC-2Approx *generates a coverage tour with cost at most twice the cost of the optimal coverage tour in polynomial time.*

Proof. Let A_s be the set of arcs corresponding to A_u with final service directions as oriented by the algorithm SRLC-2Approx. The total cost of the solution digraph $D_b = (V, A_b)$ generated by the algorithm is:

$$\begin{aligned} c(A_b) &= c(A_d) + c_s(A_s) + \sum_{a \in A_s} c_d(h(a), t(a)) \\ &\leq c(A_d) + c_s(A_u) + c_d(\bar{A}_u) \end{aligned}$$

Note that the inequality is true because we selected the service and deadheading directions to minimize the sum of the costs for individual arcs in A_u.

Furthermore, $c_d(a) \leq c_s(a)$ for $a \in \bar{A}_u$ because the deadheading cost is assumed to be no greater than the corresponding service cost. Hence,

$$c(A_b) \leq c(A_d) + c_s(A_u) + c_s(\bar{A}_u) \tag{2.28}$$

Combining (2.28) with (2.27):

$$c(A_b) \leq 2c^* - c(A_d) \leq 2c^*$$

As the required graph G_r is connected, the solution digraph D_b is also connected. The digraph D_b is also balanced, as discussed previously. Hence, a coverage tour can be generated by computing an Eulerian diwalk on D_b with the same cost as that of A_b. Thus, we obtain a coverage tour of cost at most twice the cost of the optimal tour.

The complexity of the algorithm is determined by the algorithm LP-Solve, which can be solved in $\mathcal{O}\big((m \log n)(m + n \log n)\big)$ time, where $m = |E|$ and $n = |V|$. Depending on the structure of the instance, one may need to compute the shortest deadheading paths between all pairs of vertices. This can be done using the Floyd-Warshall algorithm in $\mathcal{O}(n^3)$ computation time, see e.g., Dasgupta et al. (2006). $\square$

2.4.3 General Required Graph

We now consider input graphs for which the required graph G_r, induced by the required edges, may have multiple connected components. For such graphs, algorithm SRLC-2Approx may output a digraph that is disconnected even though the individual connected components are Eulerian. For the graph given in Figure 2.3, with the flow digraph given in Figure 2.6, the output of algorithm SRLC-2Approx is shown in Figure 2.8. Note that the digraph has multiple connected components even though the individual components are Eulerian.

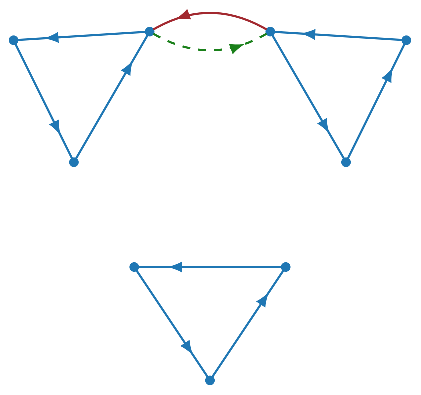


Figure 2.8: Digraph computed by the algorithm SRLC-2Approx for the graph in Figure 2.3 with the flow digraph shown in Figure 2.6. Note that the digraph has multiple connected components, each of which is Eulerian.

Our approach is to generate a tour through the connected components by solving the ATSP problem on an auxiliary graph whose vertices correspond to the components. We combine the tour with the arcs generated in each of the components. We develop an $(\alpha(C) + \beta)$-approximation algorithm where C is the number of connected components in G_r and β is the approximation factor for the single robot line coverage problem on graphs with a connected required graph. The α approximation factor depends on the approximation algorithm for the asymmetric traveling salesperson problem (ATSP), and a β of 2 was discussed in the previous subsection using the SRLC-2Approx algorithm. Constant factor approximation algorithms for ATSP were recently given by Svensson et al. (2018) and by Traub and Vygen (2020).

The output digraph D_b of the SRLC-2Approx algorithm is processed to find strongly connected components. Note that the number of strongly connected components in D_b will be no greater than the number of connected components C of the required graph G_r of G. We then create an auxiliary graph $G_0 = (V_0, E_0)$ with $V_0 \subseteq V_r$ consisting of one arbitrary vertex from each of the connected components in D_b such that a vertex $v \in V_0$ corresponds to a required edge. G_0 is a complete graph with E_0 consisting of edges between all pairs of vertices in V_0. Each edge in E_0 has two weights corresponding to the shortest deadhead cost paths in the two directions. An ATSP algorithm is then used to find a tour connecting all the vertices in G_0. The arcs in the ATSP tour are then added to the disconnected diwalk generated from the SRLC-2Approx algorithm to obtain a connected coverage tour. In the following theorem, we prove the approximation factor for our algorithm for general graphs. The key ideas for the proof of the theorem are motivated by Bevern et al. (2017).

Theorem 6. *The single robot line coverage problem can be solved in polynomial time with an approximation factor of $\alpha(C)+\beta$, where $\alpha(C)$ is the approximation factor for an algorithm for the asymmetric traveling salesperson problem with C vertices, and β is the approximation factor for line coverage on graphs with a connected required graph.*

Proof. Let the optimal coverage tour be τ^* and digraph D_b be the output of the SRLC-2Approx algorithm. Note that D_b may contain multiple strongly connected components. However, the number of strongly connected components in the digraph

D_b will be no more than the number of connected components C in the required graph G_r. As the linear relaxation does not consider the connectivity constraints, the solution D_b is an approximation result to a relaxation of the original problem. Hence, $c(D_b) \leq \beta\, c(\tau^*)$, where $\beta = 2$ for the SRLC-2Approx algorithm.

Let $V_0 \subseteq V_r$ be a set of vertices with one arbitrary vertex from each of the connected components in D_b. Then $|V_0| \leq C$. Any coverage tour must visit each of the vertices in V_0 because each vertex in V_0 lies on a required edge. For the graph G_0, let T^* be the optimal ATSP tour and T be the ATSP tour returned by the $\alpha(C)$-approximation algorithm. Then $c(T) \leq \alpha(C)\, c(T^*) \leq \alpha(C)\, c(\tau^*)$.

Let τ be the final coverage tour obtained by adding the arcs from T to the digraph D_b and generating an Eulerian tour. Then $c\,(\tau) = c\,(D_b) + c\,(T) \leq \beta\ c(\tau^*) + \alpha(C)\, c(\tau^*) = (\alpha(C) + \beta)\ c(\tau^*)$. □

Corollary 7. *Combining Theorem 5 and Theorem 6, and noting that the number of connected components C is usually small in practice, as stated by Bevern et al. (2017), we observe:*

1. *The single robot line coverage problem has an $\alpha(C) + 2$ approximation factor. This also improves the previously best-known approximation result of $\alpha(C) + 3$ for the asymmetric rural postman problem given by Bevern et al. (2017).*

2. *If $C \in \mathcal{O}(\log n)$, an $\mathcal{O}(C^2 2^C)$ dynamic programming algorithm gives the optimal ATSP solution in polynomial time, giving a 3-approximation algorithm for the single robot line coverage problem.*

Remark. For the special case when we have two connected components, we do not have an ATSP tour. In such a scenario, we can duplicate one of the vertices $v \in V_0$ and add a zero cost edge from the duplicated vertex to v. The edge set E_0 will then be created on these three vertices.

The final coverage tour for the example graph, given in Figure 2.3, is shown in Figure 2.9. For this example, the cost of the tour obtained using the approximation algorithm is optimal.

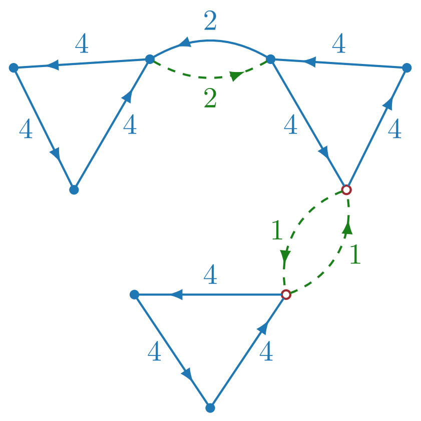


Figure 2.9: The final coverage tour, in the form of an Eulerian digraph, obtained for the input graph in Figure 2.3. The algorithm generates an ATSP tour on the solution from SRLC-2Approx, shown in Figure 2.8. As there were two connected components, the algorithm first creates an auxiliary vertex, by duplicating one of the vertices corresponding to a connected component. The arbitrary vertices from each of the components are shown as red unfilled circles. A different choice for these vertices can result in a different coverage tour, potentially of a different cost. The total cost of the coverage tour is 42, which is the same as the optimal cost.

2.4.4 Improvements and Extensions

We now provide two heuristics that improve the quality of the solutions generated by the algorithms discussed in the chapter. We explore the use of the generalized traveling salesperson problem (GTSP) instead of the ATSP. We discuss the practical aspects of implementing the algorithms. Finally, we discuss the application of our algorithm to the line coverage problem with multiple robots.

Short-circuiting: Our first heuristic is to improve the final coverage tour by replacing a sequence of consecutive deadheading edges with the shortest path from the first vertex of the sequence to the last vertex of the sequence.

2-opt heuristic: The quality of the solution can be further improved by employing a simple 2-change local neighborhood search, also known as 2-opt, similar to that for the TSP, see e.g., Roughgarden (2020). Since it is an *anytime* heuristic, i.e., it always maintains a feasible solution, we can ensure that the total number of constant-time local moves is restricted to n^3 to maintain the $\mathcal{O}(n^3)$ running time of the algorithm. We discuss the computational costs and the improvement in the solutions on a dataset of 50 road networks in Section 2.5.

GTSP based algorithm: In the algorithm for the case of general required graphs, an arbitrary vertex was selected for each of the connected components to create an auxiliary graph G_0 required as input to the ATSP algorithm. Since the choice of the vertices may affect the cost of the tour, an alternative technique is to formulate the problem as a generalized traveling salesperson problem (GTSP). Each connected component forms a cluster, and the vertices in the connected component form nodes in the cluster. The cost between any pair of nodes corresponds to the shortest deadheading path between the nodes. The GTSP is then to compute a minimum cost tour such that at least one of the nodes in each cluster is visited. A GTSP instance can be solved by converting it to an instance of the ATSP with n vertices, where n is the total number of nodes in the GTSP instance as given by Noon and Bean (1993). Such a solution has an approximation factor of $\alpha(n) + 2$, where $\alpha(n)$ is the approximation factor of an algorithm for the ATSP on a graph with n vertices. In principle, a GTSP based algroithm can provide better solutions as we no longer select an arbitrary vertex from each connected component. However, in practice, the GTSP based algorithm may require a longer computation time, and therefore is not always suitable for robotics applications that require rapid computation of the coverage tours for the robots.

Practical considerations: The $(22 + \epsilon)$-approximation algorithm for the ATSP given by Traub and Vygen (2020) is not very practical for robotics applications because of its running time and challenging implementation. However, it is very relevant for providing a constant-factor approximation guarantee. As the number of connected components is usually very small, the dynamic programming based algorithm of Held and Karp (1962) works well in practice. The algorithm runs in $\mathcal{O}(n^2 2^n)$, where n is the number of vertices. Techniques for bitwise operations by Knuth (2011) are used to run through the 2^n combinations. We also use a state-of-the-art solver for ATSP from Helsgaun (2000) for instances with a larger number of connected components. The results are discussed in Section 2.5.

Multiple robots: Our approximation algorithms have implications for the algorithms for arc routing problems with multiple robots. In the capacitated arc routing problem (CARP), the edges of the graph have a demand associated with them, and the robots have a capacity Q, see e.g., Corberán and Laporte (2014). The task is to find a set of tours for a team of k robots such that the total demand for any of the robots does not exceed its capacity Q. The objective function is the total cost of all the tours for the robots. One of the strategies for the CARP and its variants is to first find a large tour ignoring the demand constraints by employing algorithms for the single robot problem. Then the tour is split into smaller components that respect the capacity constraints (Wøhlk, 2008). Thus, any improvements to the algorithms for the single robot version improve the quality of the solution for the version with multiple robots. A tour-splitting algorithm was given by Bevern et al. (2017) for the CARP on mixed and windy graphs. The algorithm has an approximation factors of $8\alpha(C+1)+27$ in general, and an approximation factor of 35 when the number of connected components C is small, i.e., $C \in \log(n)$. Our results immediately improve these approximation factors to $8\alpha(C+1)+19$ and 27, respectively. We intend to explore the tour-splitting strategy for the line coverage problem with multiple robots using the results of this chapter.

2.5 Simulations and Experiments

We now establish the efficiency and efficacy of the presented algorithms for the single robot line coverage problem through simulations and experiments.

The algorithms are implemented in `C++` and executed on a desktop computer with an Intel Core i9-7980XE processor. We take advantage of the advances in linear programming solvers and use Gurobi Optimization (2021) to obtain solutions rapidly for the minimum cost flow problem. Smaller instances of the ATSP are solved using the dynamic programming algorithm given by Held and Karp (1962), while larger instances are solved using the LKH solver by Helsgaun (2000). The GLKH solver by Helsgaun (2015) is used to solve the GTSP instances. The algorithm for the RPP on mixed and windy graphs by Bevern et al. (2017) is also adapted for the single robot line coverage problem for comparison with the algorithms presented in this chapter. The short-circuiting based tour improvement routine is applied to the solutions generated from each algorithm as it replaces consecutive deadheadings with the shortest deadheading paths. The SRLC-ILP formulation is solved using Gurobi and run on a cluster node with 48 cores. The output from our algorithm is used to provide an initial solution to the ILP formulation, which helps in upper bounding the branch-and-bound algorithms and obtaining solutions faster. Solving an ILP to obtain an optimal solution can take a long time; it took around 20 hours for one of the instances with 635 vertices and 730 required edges.

2.5.1 Simulation Results on Road Networks

An important application of the single robot line coverage problem is the mapping, inspection, and surveillance of road networks. We generated a dataset[3] consisting of road networks from the 50 most populous cities in the world. These road networks differ considerably in structure from one another, and thus allow testing of the algorithms on a variety of graphs. The data was obtained from OpenStreetMap contributors (2022) by selecting a bounding polygon of 0.5 to 2.0 km^2 area using a web-based tool. The dataset consists of road networks with 75 to 635 vertices and 93

[3]The dataset and a tool for extracting road network data are available at: `https://github.com/AgarwalSaurav/LineCoverage-dataset`.

to 730 required edges. As UAVs can fly from one location to another, non-required edges are added between each pair of vertices, resulting in tens of thousands of non-required edges. In the case of no-fly zones, the corresponding non-required edges can be pruned in practice. The servicing and deadheading speeds are set to $7\,\mathrm{m{\cdot}s^{-1}}$ and $10\,\mathrm{m{\cdot}s^{-1}}$, respectively. A wind of $2\,\mathrm{m{\cdot}s^{-1}}$ is simulated from the south-west direction, i.e., $\pi/4$ radians from the horizontal axis. These parameters are set based on real-world experiments discussed in the following subsection.

Denote the speed of the UAV by v and the wind speed by w. For traversal of an edge from a tail vertex t to a head vertex h, let the travel vector $\mathbf{t}$ denote the vector from t to h. Let ϕ be the angle between the wind vector and the travel vector $\mathbf{t}$ for an edge. Then the effective speed of the UAV is given by:

$$v_{\text{eff}} = w\cos\phi + \sqrt{v^2 - w^2\sin^2\phi} \tag{2.29}$$

The cost function is defined as the time taken for the UAV to traverse an edge:

$$c(t,h) = \frac{\|\mathbf{t}\|_2}{v_{\text{eff}}} \tag{2.30}$$

Here, $\|\mathbf{t}\|_2$ is the Euclidean distance from t to h. The speed v of the UAV is set to the servicing or deadheading speed according to its travel mode. Note that the cost function is asymmetric due to wind.

We use the following notation for brevity:

1. β2-ATSP: Algorithm SRLC-2Approx along with the dynamic programming algorithm for the ATSP.
2. β2-GTSP: Algorithm SRLC-2Approx along with the GLKH solver for the generalized ATSP.
3. β2-ATSP-2opt: Algorithm SRLC-2Approx along with the dynamic program-

ming algorithm for the ATSP and 2-opt heuristic.

4. β3-ATSP: Algorithm given by Bevern et al. (2017) along with the dynamic programming algorithm for the ATSP.

Figure 2.10 shows four of the fifty road networks in the dataset, along with the coverage tours obtained from the ILP formulation and the β2-ATSP-2opt algorithm presented in the chapter.

A cost comparison of the solutions obtained from β2-ATSP, β2-ATSP-2opt and β3-ATSP is shown in Figure 2.11. The y-axis shows the percentage difference in cost with respect to the optimal solution, i.e., $\frac{c-c^*}{c^*} \times 100$, where c is the cost of the coverage tour given by the corresponding algorithm, and c^* is the optimal solution obtained using the ILP formulation. The solutions obtained by our final algorithm β2-ATSP-2opt are within 10% of the optimal solution. Algorithm SRLC-2Approx with the DP algorithm for ATSP, denoted by β2-ATSP, generally performs better than the algorithm given by Bevern et al. (2017), denoted by β3-ATSP. There seems to be no perceptible trend in cost difference percentage with the increase in the size of the instance.

The computation times, shown in Figure 2.12, were obtained by averaging over 100 runs. The algorithm β2-ATSP is comparable in running time to β3-ATSP, while providing better solutions, in general. For the β2-ATSP-2opt algorithm, additional time is spent to run the 2-opt heuristic, which runs very fast for smaller instances and takes up to an additional 2 s for some of the larger instances. All the 50 instances were solved within 3 s with a mean time of 0.83 s and median time of 0.73 s, using the β2-ATSP-2opt algorithm.

Figure 2.13 shows comparisons of costs and computation time for the β2-ATSP and the β2-GTSP algorithms. The comparison is performed for the instances with at least three connected components in the required graph. Using the GTSP gives better solutions in general as the algorithm has the flexibility to select any vertex in

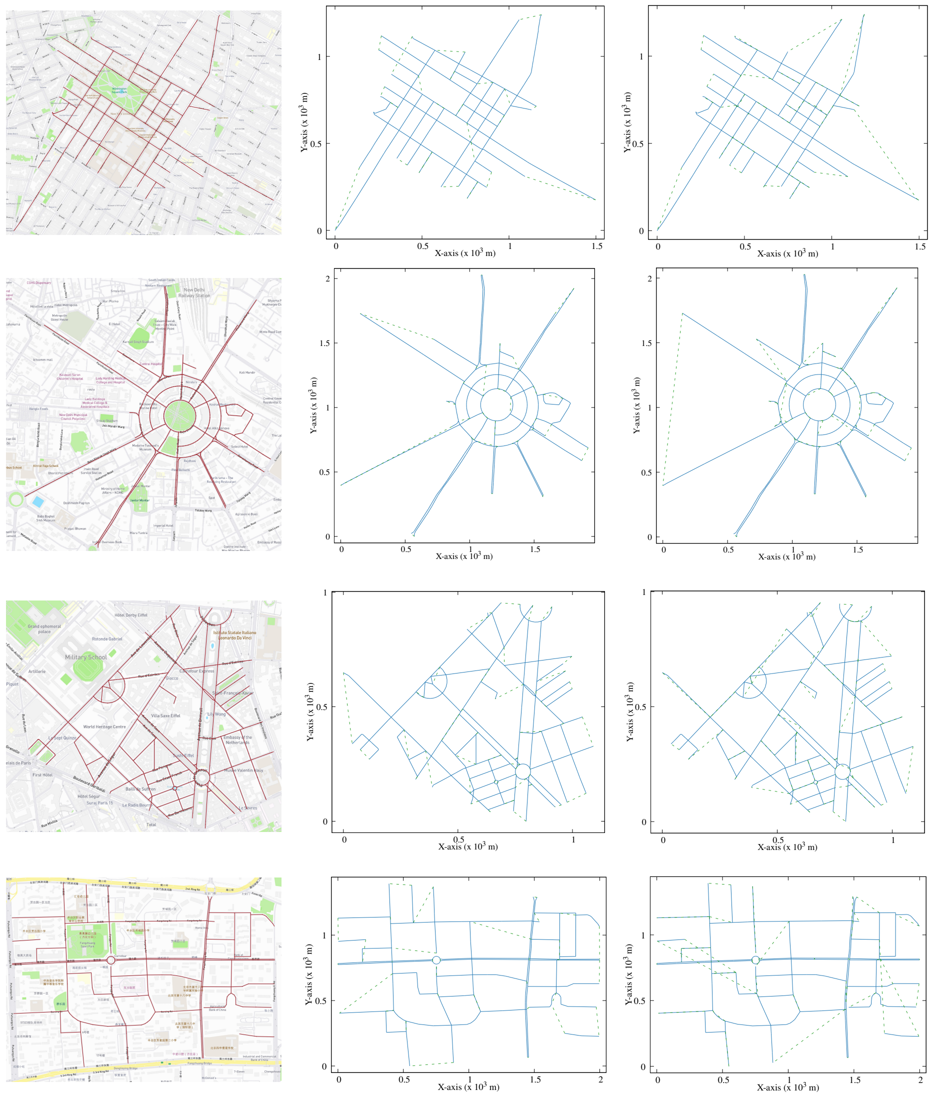


Figure 2.10: Four of the fifty sample road networks obtained from the most populous cities: The first column is the map representing the input graph, the second column is the optimal solution obtained using the SRLC-ILP formulation, and the third column is the final result of the algorithm (β2-ATSP-2opt) presented in this chapter. The road networks, from top to bottom, are from (a) New York, (b) Delhi, (c) Paris, and (d) Beijing. Only the required edges representing the road network are shown in the map. There is a non-required edge for each pair of vertices in the graph. For example, the New York dataset has 379 vertices, 402 required edges, and 71, 631 non-required edges. The solid blue lines represent servicing while the dashed green lines represent deadheading travel.

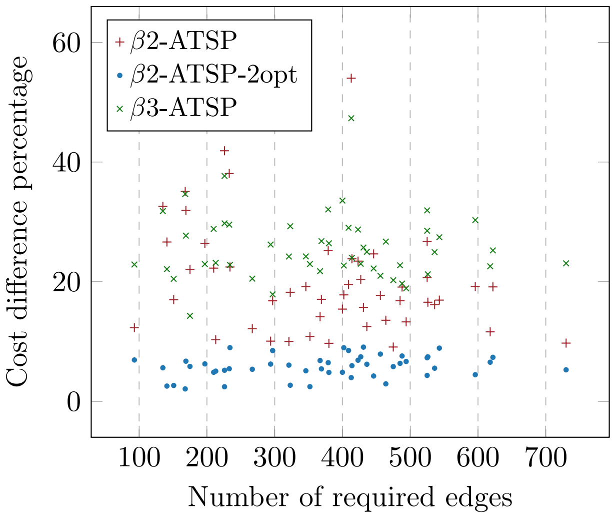


Figure 2.11: Cost comparison of the algorithms for the 50 road network dataset: The β2-ATSP algorithm, shown by red plus + markers, generally performs better than the β3-ATSP algorithm given by Bevern et al. (2017), shown by green cross ×. The solutions obtained by the β2-ATSP-2opt algorithm, shown by blue dots ●, are consistently within 10% of the optimal.

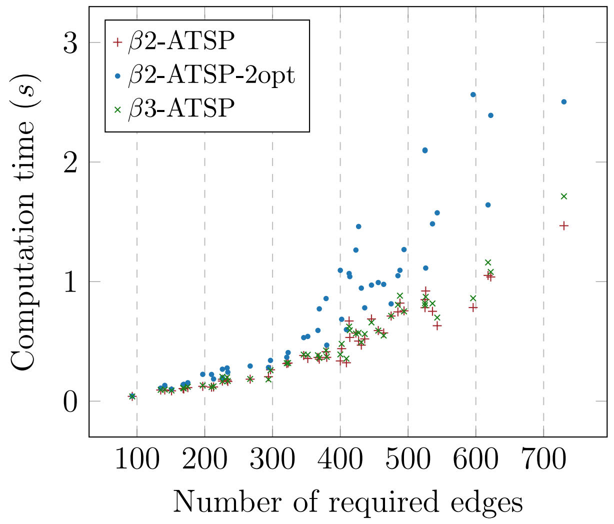
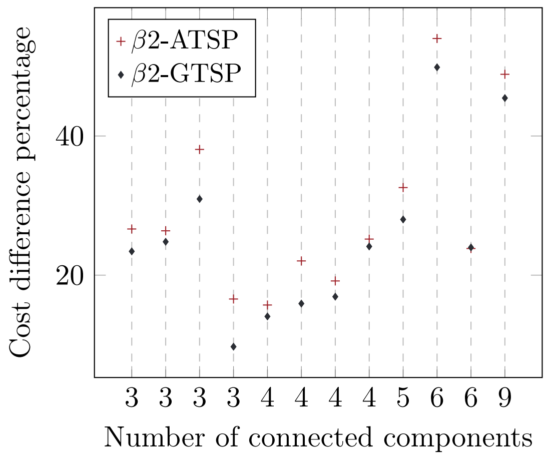


Figure 2.12: Computation time comparison of various algorithms in the chapter: The β2-ATSP algorithm takes time comparable to the β3-ATSP algorithm. The 2-opt heuristic improvement over the β2-ATSP algorithm takes very small additional time for small instances and up to 2 s for larger instances. The running times are averaged over 100 runs.



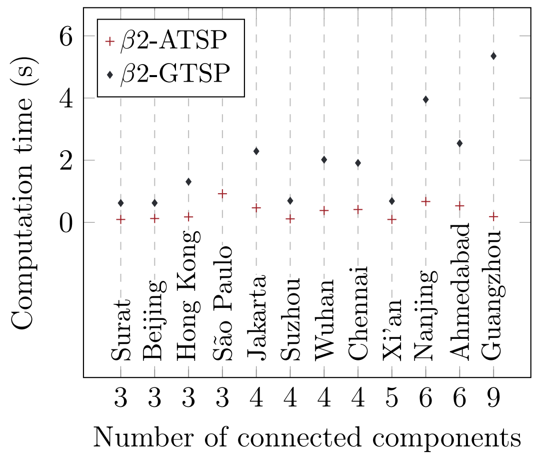


Figure 2.13: Computation time and cost comparisons for the β2-ATSP and the β2-GTSP algorithms: The β2-GTSP algorithm gives better solutions, in general. However, the β2-ATSP algorithm is computationally much more efficient. The comparison is performed for the instances with at least three connected components in the required graph.

each of the connected components. In contrast, an arbitrary vertex is selected for each connected component for the β2-ATSP algorithm. In only one of the instances (Ahmedabad), the final coverage tour obtained using the DP algorithm for the ATSP resulted in slightly better results than that for using the GTSP, and this is because the ATSP based solution was more favorable for the short-circuiting routine and resulted in a better solution overall. The β2-ATSP algorithm is computationally much more efficient.

The simulation results indicate that all the ATSP based algorithms are sufficiently fast. With a small additional computational cost for the 2-opt heuristic, the β2-ATSP-2opt algorithm computes high-quality solutions.

2.5.2 Experiments on a Campus Road Network

We performed line coverage on two different portions of the UNC Charlotte road network using a DJI Phantom 4 quadrotor UAV. Figure 2.1 shows a portion of the road network and Figure 2.14 shows a network of lanes on a set of parking lots. The experiments were performed with two different sets of operating conditions. The servicing and deadheading speeds, along with wind speeds and directions are specified in Table 2.2. The cost functions are based on the time to traverse the respective edge; the wind conditions make the costs asymmetric, as specified by Equations (2.29) and (2.30). The computed line coverage tour costs using the SRLC-ILP formulation and the β2-ATSP-2opt algorithm are provided in Table 2.2. The table also provides the actual flight times. Figures 2.15 and 2.16 show the computed coverage tours using the β2-ATSP-2opt algorithm, the actual flight paths, and orthomosaics for the two datasets. We generated orthomosaics from the images collected during the flights. The images are taken only during servicing (and not during deadheading) leading to a smaller number of images, and thus reduced the time to compute the orthomosaic.

We have the following observations from our experiments.

1. The actual flight time differs from the computed flight time. Since we use a commercial mobile application to fly the UAV autonomously along the coverage tour, we do not have access to a model of the controller. As our formulation allows arbitrary cost functions, a high fidelity model of the trajectory controller and wind effects can be incorporated to get better results. Another aspect is that we do not model turning costs, and UAVs need to slow down to take sharper turns. This increases the actual flight time, and indicates the importance of modeling the effect of turns in the objective function in the future.

2. Since we flew the UAV at a relatively high altitude (compared to the distance between parallel required edges and the sensor field of view), the generated orthomosaic provides an area coverage of the parking lots. Line coverage can in fact be used as a subroutine for area coverage, and that is a direction we are pursuing.

These experiments demonstrate the use of our line coverage formulation and the algorithms to generate efficient coverage tours for linear infrastructure. The two modes of travel—servicing and deadheading—can be conveniently modeled in the formulation allowing lower operation times. Furthermore, allowing deadheading reduces the amount of sensor data required for analysis.

Table 2.2: Operating conditions, computed coverage tour costs, and actual flight times for experiments with a quadrotor UAV

	Road network	Parking lots
Service speed	$7.00\,\text{m·s}^{-1}$	$3.33\,\text{m·s}^{-1}$
Deadheading speed	$10.00\,\text{m·s}^{-1}$	$5.00\,\text{m·s}^{-1}$
Wind speed	$2.00\,\text{m·s}^{-1}$	$1.34\,\text{m·s}^{-1}$
Wind direction	$45.00°$	$67.50°$
SRLC-ILP cost	485 s	1,123 s
SRLC-ILP flight time	502 s	925 s
β2-ATSP-2opt cost	492 s	1,172 s
β2-ATSP-2opt flight time	527 s	1,023 s

2.6 Summary

Motivated by coverage applications for linear infrastructure such as road networks, power lines, and oil and gas pipelines, we addressed the single robot line coverage problem for autonomous aerial and ground robots. The linear features are modeled as required edges in a graph that the robot must service. Additional non-required edges, which do not require servicing, provide flexibility for a robot to select its path. The two modes of travel—servicing and deadheading—permit better modeling of real-world scenarios where a robot needs to perform task-specific actions such as

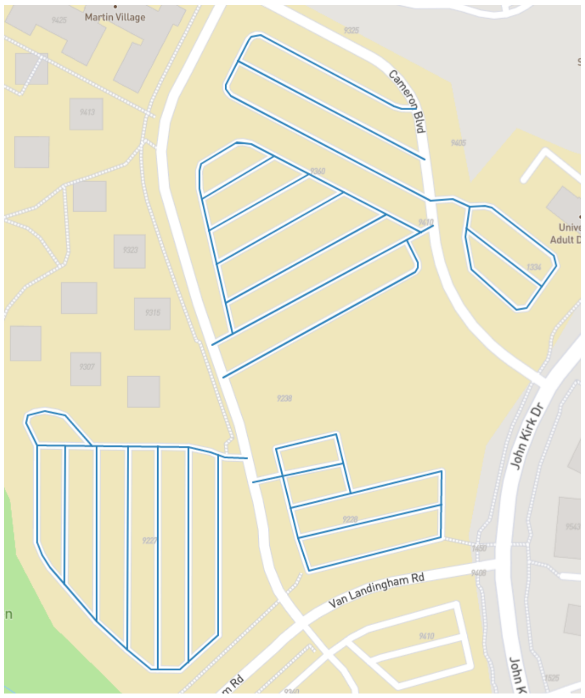
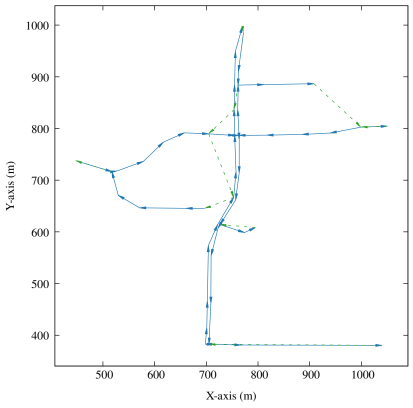


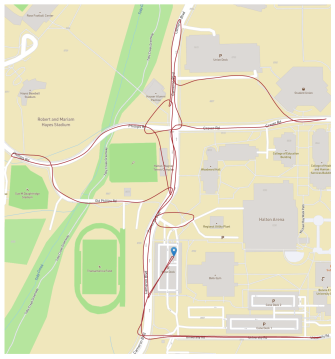
Figure 2.14: A network of lanes specified on a set of parking lots: The total length of the lanes is 2,982 m. There are 90 vertices, 104 required edges, and 4,005 non-required edges. The required edges form four connected components.

taking images only along specified features. This reduces the workload of the robot, permits further optimization of the travel cost, and decreases the amount of sensor data that needs to be analyzed. Our formulation models asymmetric cost functions and permits multiple copies of edges. This enables one-way streets and repeated servicing of segments.

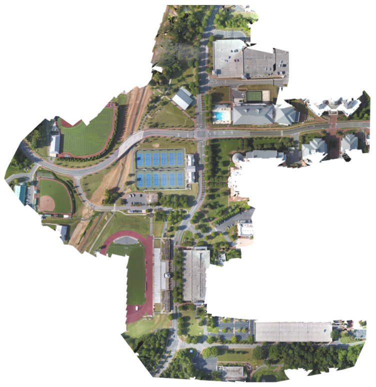
We formulated the single robot line coverage problem as an optimization problem and developed an ILP formulation that gives optimal solutions. Formal proofs establish the correctness of the formulation. As the problem is NP-hard, we developed approximation algorithms that have a guarantee on the quality of the solutions. Studying the structure of the required graph—the graph induced by the linear features—provided us insights into the problem, which were used to develop the approximation algorithms. The algorithms were developed in stages going from a simple version of the problem to the most general one. First, an optimal algo-



(a) Coverage route

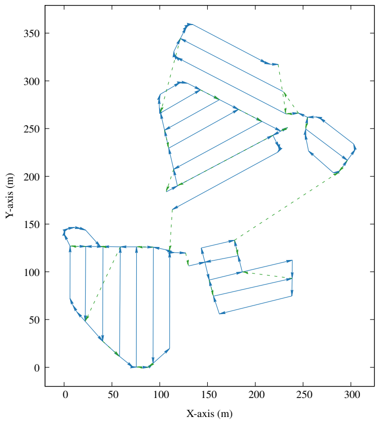


(b) Actual flight path

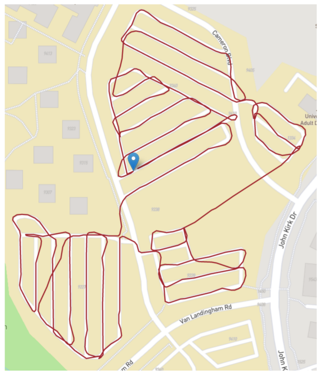


(c) Orthomosaic

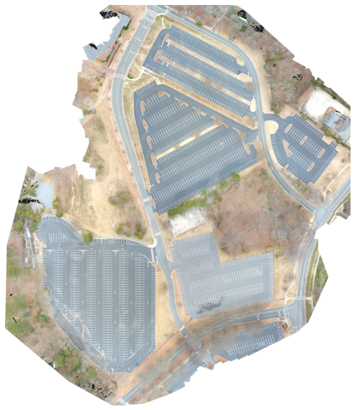
Figure 2.15: Coverage of a portion of the UNC Charlotte road network: The required graph has one connected component. The road network has a length of 2,658 m with 48 vertices and 48 required edges. There are 1,128 non-required edges formed by each pair of vertices. (a) Coverage tour generated using the $\beta 2$-ATSP-2opt algorithm. The cost of the solution is 492.48 s. The servicing travel is denoted by blue solid lines, while green dashed lines denote the deadheading travel. The arrowheads indicate the direction of travel. (b) The actual flight path of a UAV executing the coverage tour autonomously. The blue marker denotes the home location for the UAV. (c) Orthomosaic generated from the images collected during servicing travel along the coverage tour. Collecting images only during servicing reduces the number of images that need to be processed for mapping and analysis.



(a) Coverage route



(b) Actual flight path



(c) Orthomosaic

Figure 2.16: Line coverage of lanes specified on a set of parking lots: The required graph has four connected components. (a) Coverage tour generated using the β2-ATSP-2opt algorithm. The cost of the solution is 1,172 s. The servicing travel is denoted by blue solid lines, while green dashed lines denote the deadheading travel. The arrowheads indicate the direction of travel. (b) The actual flight path of a UAV executing the coverage tour autonomously. The blue marker denotes the home location for the UAV. The actual flight took 1,023 s. (c) Orthomosaic computed from images collected during the flight.

rithm, based on the minimum cost flow problem, was discussed for the case where the required graph is Eulerian. For the case where the required graph is connected but not necessarily Eulerian, a 2-approximation algorithm was developed. Finally, an $(\alpha(C) + 2)$-approximation algorithm was given for the general case of a required graph with C components, where $\alpha(C)$ is the approximation factor for an algorithm for the ATSP. Proofs for the approximation factor were provided for each of the algorithms. Heuristics that improve the quality of the solutions were incorporated into the algorithm, and a GTSP based alternative was discussed.

Simulation results on a road network dataset of the 50 most populous cities in the world show that our main algorithm computes high-quality solutions that are within 10% of the optimum in less than 3 s. The algorithms are sufficiently fast for robotics applications. Experiments with a commercial aerial robot were performed on a portion of the UNC Charlotte road network and on lanes of a set of parking lots.

As the images were collected only during servicing, and not while deadheading, a smaller number of images of only the features of interest were collected. Orthomosaic maps were generated using these images.

CHAPTER 3: LINE COVERAGE WITH MULTIPLE ROBOTS

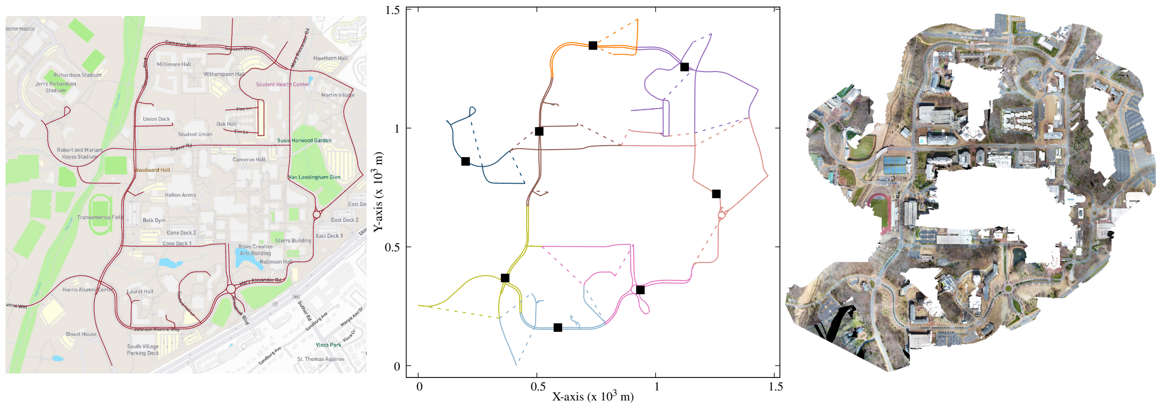
The line coverage problem is to find efficient routes for coverage of linear environment features by one or more resource-constrained robots. Linear features model environments such as road networks, power lines, and oil and gas pipelines. We define two modes of travel for the robots: servicing and deadheading. A robot, such as an uncrewed aerial vehicle (UAV), services a feature if it performs task-specific actions, e.g., taking images, as it traverses the feature; otherwise it is deadheading. Traversing the environment incurs costs (e.g., travel time) and demands on resources (e.g., battery life). Servicing and deadheading can have different cost and demand functions, and we further permit them to be asymmetric in the two directions of a line feature. We model line coverage as an optimization problem on a graph using an integer linear program (ILP); the task is to find a set of routes that minimizes the total cost of travel such that all the features are serviced within the resource limit for each robot. The problem is NP-hard, and hence we develop a fast and efficient heuristic algorithm, Merge-Embed-Merge (MEM). The algorithm is versatile and can be modified to incorporate robot constraints; we illustrate this for two variants of the line coverage problem. We formulate the multi-depot version for large graphs where a route can start and end at any of the depots. We then extend the MEM algorithm to handle turning costs and nonholonomic constraints. We benchmark the algorithm with the optimal ILP approach on a dataset of road networks from the 50 most populous cities in the world. We demonstrate the application of the algorithm using commercial aerial robots on road networks.

3.1 Introduction

Line coverage is the task of servicing linear environment features using sensors or tools mounted on robots. The features to be serviced are modeled as one-dimensional segments (or curves), and all points along the segments must be visited. Consider a natural disaster scenario such as flooding. A team of uncrewed aerial vehicles (UAVs) is deployed to assess the accessibility of a road network for emergency services. The UAVs must traverse the line segments corresponding to the road network and capture images for analysis. This chapter seeks to answer the following question: How should efficient routes for UAVs be planned so that each road network segment is serviced? Mobile robots are often resource-constrained—they must come back to their *depot* or launch location before they exhaust their resources, such as battery life. Figure 3.1 shows a large road network, routes for eight UAVs covering the entire road network, and an orthomosaic generated from the images taken by the team of UAVs. Power lines and gas pipelines have similar infrastructure that requires frequent inspection. Additional applications arise in perimeter inspection and surveillance, traffic analysis of road networks, and welding and 3D printing operations.

Line coverage is closely related to arc routing problems (ARPs) studied in the operations research community (see monograph by Corberán and Laporte, 2014). ARPs have been used to generate routes for snow plowing, spraying salt, and cleaning road networks (Corberán et al., 2021). The ARPs and their algorithms are designed specifically for human-operated vehicles. Although the above tasks can potentially be automated with ground robots, line coverage has received limited attention in the robotics community. In this chapter, we design algorithms for line coverage using autonomous systems applicable to aerial, ground, and underwater robots.

The line coverage problem with multiple robots, modeled using a graph, has three defining attributes: (1) The edges in the graph are classified as required and non-required, (2) Robots have two modes of travel—servicing and deadheading, and



(a) Input Road Network (b) Line Coverage Routes (c) Orthomosaic

Figure 3.1: Line coverage of the UNC Charlotte road network using a team of resource-constrained UAVs. (a) The input road network of length 13 km, spanning an area of 1.5 km^2. The road network is modeled as a graph comprising 842 vertices and 865 required edges representing the road segments. As the UAVs can fly from one vertex to another, we add a non-required edge between each pair of vertices, resulting in 353,196 non-required edges in the graph. (b) Routes for eight UAVs, distinguished by different colors, generated using the Merge-Embed-Merge (MEM) algorithm with multiple depots developed in this chapter. The algorithm computes depot locations, shown by black squares, from where the UAVs start and end their routes. The solid lines represent servicing, while the dashed lines represent deadheading. The UAVs can fly faster while deadheading, thereby optimizing total flight time. (c) An orthomosaic of the road network generated from the images taken by the UAVs flown autonomously along the computed routes.

(3) Robots have constraints on the resources available to them. Required edges in the graph correspond to the line features to be covered, and a robot can use the non-required edges to travel from one vertex to another to reduce cost. The vertices in the graph represent the endpoints of the edges. A robot is said to be *servicing* a required edge when it performs task-specific actions such as collecting sensor data. Each required edge needs to be serviced exactly once by any robot. Robots may also traverse an edge without performing servicing tasks to optimize the travel time, conserve energy, or reduce the amount of sensor data. This mode of travel is known as *deadheading*, and both types of edges may be used any number of times for this purpose. The operation time of a robot is constrained by resources such as battery life available to them. Hence, the routes should be planned such that the total demand

for the resources is within the resource limit.

A service cost and a deadhead cost (e.g., travel time) are associated with each required edge, and they are incurred each time an edge is serviced or deadheaded, respectively. Only the deadheading cost is associated with the non-required edges. The total sum of the service and deadhead costs over all routes for the robots is to be minimized. Moreover, traversing an edge results in the consumption of resources such as battery life. These are modeled as *demands* on the edges, and the total demand of a route should be less than the given *capacity* of the robots. The costs and demands for servicing an edge are usually more than those for deadheading as task-related actions are only performed while servicing. For example, a robot servicing an edge by recording images may travel slower to avoid motion blur, resulting in a longer travel time.

In many robotics applications, the cost of travel and the resource demands are direction-dependent. For example, a ground robot traveling uphill can take longer and consume more energy than when traveling downhill. Similarly, the costs and demands of aerial robots may differ in two directions due to wind conditions. Hence, we consider the graph to have asymmetric cost and demand functions for servicing and deadheading. Such asymmetric functions can also model one-way streets for ground robots.

When the network is vast, it may not be possible to service the entire network from a single *depot* location. In such situations, it is imperative to have a multi-depot formulation where the robots have the flexibility to start and end their routes from one of several depots to optimize the routes. Another practical consideration is taking sharp turns which can be very expensive as a robot has to slow down, take a turn, and then accelerate. Similarly, *nonholonomic* robots such as fixed-wing UAVs and underwater vehicles cannot make immediate turns. The algorithms presented in this chapter account for both the multi-depot formulation and the turning costs for nonholonomic robots.

The *line coverage problem with multiple robots* is the task of computing efficient coverage routes for a set of line features such that the total cost of travel is minimized while respecting the capacity constraints. The practical benefits of our line coverage approach are: (1) The algorithms ensure that the required edges are covered efficiently by optimizing the total cost of the coverage routes. (2) In contrast to using area coverage, only the relevant line features are serviced, thus reducing the operation time, the amount of sensor data, and the time for data analysis. (3) Distinct and asymmetric costs and demands allow for further optimization of coverage routes.

The contributions of the chapter are:

1. We develop an integer linear programming (ILP) formulation for the multi-robot line coverage problem.

2. The line coverage problem is NP-hard in general. Thus, we develop a fast and efficient constructive heuristic algorithm, the Merge-Embed-Merge (MEM) algorithm.

3. We develop a multi-depot formulation for instances with graphs spanning over a large area. The MEM heuristic algorithm is extended to solve the multi-depot line coverage problem efficiently.

4. Turns can be expensive for robots and incorporating their costs and demands for routes is essential for ensuring efficient and safe routes. We present extensions of the MEM algorithm to incorporate turning costs. Furthermore, we use Dubins curves to model turns for nonholonomic robots and incorporate them into the MEM algorithm. These show that, with little modification, we can incorporate several practical aspects of deploying mobile robots for line coverage into the versatile MEM algorithm.

5. We demonstrate the algorithm on a dataset[1] consisting of road networks from the 50 most populous cities in the world. Illustrative experiments with UAVs are also presented. We provide an open-source implementation[2] of our algorithms.

The rest of the chapter is organized as follows. The related work is discussed in Section 3.2. The multi-robot line coverage problem is formally described in Section 3.3, along with an ILP formulation. Section 3.4 develops the constructive heuristic algorithm and extends it to two variants of the problem. The simulations and experiments are discussed in Section 3.5. Section 3.6 summarizes the chapter.

3.2 Related Work

This section first discusses the related work on arc routing problems (ARPs). We discuss the various relevant problems in ARPs, briefly describe solution methods, and contrast our approach with existing ones. Next, we discuss work related to line coverage and area coverage in robotics.

3.2.1 Arc Routing Problems

The line coverage problem belongs to the broad class of arc routing problems (ARPs). The ARPs are usually applied to transportation problems in which servicing is related to tasks such as delivery and pick-up of goods (Corberán and Laporte, 2014). In such problems, the costs are associated with the travel distances, and they have the same value for servicing and deadheading. Furthermore, the demands are associated with the load the vehicles can carry, e.g., the capacity of a garbage collection truck. Therefore, the deadheading of an edge does not affect the capacity. In contrast, the capacity in the line coverage problem is associated with battery life, and demands are incurred in both modes of travel.

The Chinese postman problem (CPP), the simplest of the ARPs, is to find an op-

[1]The dataset is available at:
`https://github.com/UNCCharlotte-CS-Robotics/LineCoverage-dataset`
[2]The source code is available at:
`https://github.com/UNCCharlotte-CS-Robotics/LineCoverage-library`

timal tour such that every edge in an undirected and connected graph is traversed at least once. Edmonds and Johnson (1973) used matching and network flow to solve the CPP on undirected, directed, and Eulerian mixed graphs. In the rural postman problem (RPP), the task is to service a subset of edges in a graph. Frederickson (1979) gave a 3/2-approximation algorithm, based on the algorithm given by Christofides (1976) for the traveling salesperson problem (TSP). The asymmetric RPP, i.e., RPP on a graph with asymmetric edge costs, is much harder than the symmetric counterpart. The formulation for the single robot line coverage problem, presented in Chapter 2 and Agarwal and Akella (2021), can be used to solve asymmetric RPP. We developed algorithms based on the structure of the required graph—the graph induced by the set of required edges. A 2-approximation algorithm for the case of a connected required graph and a $(\alpha(C) + 2)$-approximation algorithm for a required graph with C connected components were developed. Here, $\alpha(C)$ is the approximation factor for an algorithm for the TSP on asymmetric graphs with triangle inequality.

Easton and Burdick (2005) introduced kRPP, the RPP with k vehicles. They modeled coverage of 2D object boundaries as a kRPP and presented a cluster first and route-second heuristic. The kRPP does not consider the capacity of the robot (vehicle). The capacitated arc routing problem (CARP), introduced by Golden and Wong (1981), considers k vehicles of a given capacity Q. The required edges have a demand associated with them, and the total demand of a route should be less than the capacity. The CARP is a special case of the line coverage problem—there are no deadheading demands, and all costs and demands are symmetric. Since the problem is NP-hard (Golden and Wong, 1981), several heuristic algorithms have been developed for the CARP (Corberán and Laporte, 2014). Wøhlk (2008) presented a 7/2-approximation algorithm for the problem. Several variants of the ARP partially or individually consider a subset of the characteristics of the line coverage problem. Gouveia et al. (2010) presented a lower bound approach for the CARP on mixed graphs using a mixed integer linear program (MILP). The CARP-DD, introduced by Kirlik and Sipahioglu (2012), considers deadheading demands for the CARP.

In the multi-depot capacitated arc routing problem (MDCARP), a set of depot locations are given as input. The vehicles can start and end their route at any depot, and usually, the requirement is to return to the start depot at the end of the route. The MDCARP is especially relevant in large-scale applications where servicing a network may not be optimal or even feasible from a single location. Given the low battery life of robots, compared to fueled vehicles, the MDCARP becomes relevant even for moderately sized graphs. One common technique is to cluster the input graph into smaller subgraphs and assign a single depot for each subgraph. The algorithms for the CARP are then used on each of the subgraphs independently. Such techniques have also been described as districting by Muyldermans et al. (2003) and as sectoring by Mourão et al. (2009). The line coverage problem is a generalization of all these ARPs, as shown in Table 3.1.

Table 3.1: The line coverage problem with its special cases

Literature	Service		Deadheading	
	Cost	Demand	Cost	Demand
kRPP Easton and Burdick (2005)	S	$-$	$=$	$-$
CARP Golden and Wong (1981)	S	S	$=$	$-$
MDCARP Muyldermans et al. (2003)	S	S	$=$	$-$
CARP-DD Kirlik and Sipahioglu (2012)	S	S	$=$	S
Mixed-CARP Gouveia et al. (2010)	A	A	A	$-$
Line coverage problem	A	A	A	A

S: symmetric, A: asymmetric, and $-$: not considered.
$=$: The deadheading costs are equal to the service costs.

Several exact and metaheuristic algorithms have been proposed for the ARPs. They are covered in the survey paper by Corberán and Prins (2010) and the monograph by Corberán and Laporte (2014). Exact approaches include branch-and-bound with cutting planes, branch-and-price, and column generation. The problem is NP-hard, and these exponential-time algorithms are not suitable for large-scale robotics applications. Metaheuristic algorithms, such as scatter search, tabu search, and variable neighborhood descent have been used for ARPs. Similar to the exact methods, these can require significant computation resources. Moreover, they typically require a good initial solution as an additional input to upper bound the optimal cost. The heuristic algorithms presented in this chapter can provide such an initial solution and be used as a fast sub-routine to generate intermediate solutions.

Algorithms based on minimum-cost perfect matching and minimum-cost flow problems are used widely to solve ARPs. They generate a set of Eulerian digraphs, and a robot route can be generated by computing an Eulerian tour for each such digraph. Incorporating multiple depots, turning costs, and nonholonomic constraints is not trivial using these approaches. In contrast, this chapter develops a constructive heuristic algorithm, Merge-Embed-Merge (MEM), which maintains a set of routes for the robots and constructively merges pair of routes to form larger routes. The algorithm is very fast and gives solutions of high quality. Furthermore, the versatile nature of the algorithm allows us to extend the algorithm to several variants of the problem relevant to robotics.

3.2.2 Line Coverage in Robotics

There has been little work on using robots for line coverage tasks. Dille and Singh (2013) presented algorithms to perform coverage of a road network using a single aerial robot with kinematic constraints. They modeled the problem through the tessellation of the road segments by circles of radius corresponding to the sensor footprint and finding a subset of the circular regions covering the entire road network. Algorithms for node routing problems, such as the TSP and the multiple TSP, are then used to find the routes for the robots. Oh et al. (2014) proposed an MILP formulation and a heuristic algorithm for the coverage of road networks. The nearest insertion heuristic, originally designed for TSP, finds a sequence of edges to be visited while incorporating Dubins curves for nonholonomic robots. The sequence is split across a team of robots using an auction algorithm. Algorithms for kRPP were developed for boundary inspection with multiple robots by Easton and Burdick (2005). Williams and Burdick (2006) developed algorithms for boundary inspection while considering revisions to the path plan for the robots to account for environmental changes. Xu and Stentz (2010) use CPP and RPP formulations for environmental coverage and consider the case of incomplete prior map information. They extended this work to multiple robots (Xu and Stentz, 2011) using k-means clustering to decompose the environment into smaller components, similar to the approach by Easton and Burdick (2005). Campbell et al. (2018) presented an application of ARPs to cover road networks using a single UAV. They discretize the required edges to allow the UAV to service an edge in parts.

The above work illustrates the line coverage applications in robotics. However, line coverage has not been studied as extensively as the area coverage problem or node routing problems. Current work does not usually consider the battery life of the robots and thus may not be suitable for large-scale applications. In contrast, we model the battery life as capacity (resource) constraints and consider multiple depots, enabling solutions for large networks. Furthermore, we allow demands on resources while deadheading and asymmetric functions for costs and demands.

3.2.3 Arc Routing Problems in Area Coverage

ARPs have been used in area coverage problems as a subroutine to generate efficient robot routes. Arkin et al. (2000) used the CPP to find a route for the milling problem, a variant of the area coverage problem wherein the tool is constrained within the workspace. Mannadiar and Rekleitis (2010) performed a cell decomposition and computed a Reeb graph. The edges of the graph correspond to the cells in the decomposition. The problem of visiting the cells was then formulated as the CPP. Karapetyan et al. (2017) used the CPP to compute a large Euler tour and then decomposed the tour into smaller ones using an algorithm given by Frederickson et al. (1976). We present a new approach in Chapter 4 for the area coverage problem by generating service tracks in the environment after performing cell decomposition. These service tracks were modeled as required edges for the line coverage problem, and the MEM algorithm was used to generate efficient routes for multiple capacity-constrained robots. Using the above technique, the algorithms presented in this chapter for multiple depots and nonholonomic robots apply to the area coverage problem.

3.3 Problem Statement

We pose the line coverage problem as an optimization problem on a graph. The environment comprises linear features (line segments or curves) that need to be serviced by a homogeneous team of robots. It is modeled as an undirected and connected graph $G = (V, E, E_r)$, where $E_r \subseteq E$ is the set of *required edges* representing the linear features. The graph may have edges that do not require servicing, and the robots can use them to optimize their routes; these are the *non-required edges* given by $E_n = E \setminus E_r$. The set E can contain parallel edges between two vertices, i.e., we allow for G to be a multigraph. The set of vertices V consists of edge endpoints, edge intersections, and depot locations. The *depots* $V_d \subseteq V$ are a subset of vertices at which the robots start and end their routes.

For each edge e in E we associate two directional arcs a_e and $\bar{a}_e$ that are opposite in direction to one another. There are two modes of travel for a robot—servicing and deadheading. A robot is said to be *servicing* an edge when it performs task-specific actions such as taking images along the edge. We associate two binary variables $s^k_{a_e}$ and $s^k_{\bar{a}_e}$ with servicing an edge $e \in E_r$ by robot k: if a robot k services edge e in the direction a_e, then $s^k_{a_e}$ is 1 and 0 otherwise; similarly for the direction $\bar{a}_e$. Each required edge is required to be serviced exactly once:

$$\sum_{k=1}^{K} \left(s^k_{a_e} + s^k_{\bar{e}} \right) = 1, \quad \forall e \in E_r \tag{3.1}$$

If a robot traverses an edge without servicing it, the robot is said to be *deadheading*, e.g., this occurs when a robot travels from its depot to an edge to be serviced. We associate two non-negative integer variables $d^k_{a_e}$ and $d^k_{\bar{a}_e}$ with deadheading an edge $e \in E$ by robot k. The edges can be deadheaded any number of times. The task is to compute a set of routes $\Pi = \{\pi^1, \ldots, \pi^K\}$ for K robots.

Servicing an edge $e \in E_r$ in the direction a_e incurs a service cost $c_s(a_e)$; similarly for the direction $\bar{a}_e$. Analogously, deadheading an edge $e \in E$ incurs deadhead costs $c_d(a_e)$ and $c_d(\bar{a}_e)$. A robot may need to travel slower while performing task-specific actions resulting in higher costs for servicing than deadheading. Thus, the two cost functions can differ. The costs are associated with a minimization objective function of the optimization problem, such as total travel time. The cost of a route $\pi^k \in \Pi$ is given as:

$$c(\pi^k) = \sum_{e \in E_r} \left[c_s(a_e)\, s^k_{a_e} + c_s(\bar{a}_e)\, s^k_{\bar{a}_e} \right] + \sum_{e \in E} \left[c_d(a_e)\, d^k_{a_e} + c_d(\bar{a}_e)\, d^k_{\bar{a}_e} \right] \tag{3.2}$$

Each robot is constrained by a resource such as operation time, total travel distance, or battery life. Such a constraint is represented by a *budget* or *capacity* Q for each robot. The consumption of resources is modeled by demand functions $q_s(a_e)$ for servicing and $q_d(a_e)$ for deadheading an edge $e \in E$ in the direction a_e. The total demand incurred by a robot for a route π^k must be less than the capacity:

$$q(\pi^k) = \sum_{e \in E_r} \left[q_s(a_e)\, s^k_{a_e} + q_s(\bar{a}_e)\, s^k_{\bar{a}_e} \right] + \sum_{e \in E} \left[q_d(a_e)\, d^k_{a_e} + q_d(\bar{a}_e)\, d^k_{\bar{a}_e} \right] \leq Q \tag{3.3}$$

We consider the edge costs and demands for both servicing and deadheading to be direction-dependent, i.e., the graph is asymmetric. For example, $c_s(a_e)$ can differ from $c_s(\bar{a}_e)$. Asymmetry in graphs can occur when modeling wind for aerial robots or terrain for ground robots. It also allows us to model one-way streets, directed graphs, and mixed graphs in general. This can be achieved by setting the cost and demand in the prohibited direction to be a large constant. The traversal of edges can be modeled using any function such as constant velocity or cubic trajectories, and travel time can be used as the cost function. Similarly, demands and capacity can be specified in terms of battery life. Such functions can also incorporate wind and terrain information (Yongguo Mei et al., 2006; Franco and Buttazzo, 2015). In general, the costs and demands are allowed to be arbitrary non-negative constants. If we also have point features V_f in the environment, we add an artificial edge (v, v) for each point feature $v \in V_f$. The cost and the resource demand for servicing such an artificial edge would be the same as that of servicing the point feature, and the cost and demand for deadheading is set to zero. This transformation allows modeling both the point and the line features in the same formulation.

Definition: Given an undirected and connected graph $G = (V, E, E_r)$, the *line coverage problem* is to find a set of coverage routes Π for K robots that services each required edge in E_r exactly once and minimizes the total cost of the routes, while respecting the resource constraints.

3.3.1 Integer Linear Programming Formulation

An integer linear program (ILP) is a formulation for optimization problems with integer variables, a linear objective function, and a set of linear constraints. An ILP formulation provides a concise mathematical description of the problem, and solving the ILP gives an optimal solution to an instance of the problem if the instance has feasible solutions.

3.3.1.1 Variables

We have the following variables for the ILP.

- Binary service variables $s^k_{a_e}, s^k_{\bar{a}_e} \in \{0,1\}$ for each required edge $e \in E_r$ and each robot k.
- Integer deadheading variables $d^k_{a_e}, d^k_{\bar{a}_e} \in \mathbb{N} \cup \{0\}$ for each edge $e \in E$ and each robot k.
- Integer flow variables $z^k_{a_e}, z^k_{\bar{a}_e} \in \mathbb{N} \cup \{0\}$ for each edge $e \in E$ and each robot k. The flow variables are used in connectivity constraints to ensure that routes are connected to the depots.

For now, we assume that all the robots start and end their routes at the same depot location $v_0 \in V$. We will generalize the formulation to multiple depots in the following subsection.

3.3.1.2 Routing Constraints

The routing constraints ensure connectivity of a robot route to the depot and eliminate sub-tours. For ease of notation, we define the following sets:

$$\mathcal{A} = \bigcup_{e \in E} \{a_e, \bar{a}_e\}, \quad \mathcal{A}_r = \bigcup_{e \in E_r} \{a_e, \bar{a}_e\},$$
$$H(\mathcal{A}, v) = \text{arcs in } \mathcal{A} \text{ with } v \text{ as the head, and}$$
$$T(\mathcal{A}, v) = \text{arcs in } \mathcal{A} \text{ with } v \text{ as the tail.}$$

Here, $\mathcal{A}$ is the set of all arcs, and $\mathcal{A}_r$ is the set of arcs corresponding to required edges.

We have the following set of *routing constraints* for each robot $k \in \{1, \ldots, K\}$:

$$\sum_{a \in T(\mathcal{A}, v_d^k)} z_a^k = \sum_{a \in \mathcal{A}_r} s_a^k \tag{3.4}$$

$$\sum_{a \in H(\mathcal{A}, v)} z_a^k - \sum_{a \in T(\mathcal{A}, v)} z_a^k = \sum_{a \in H(\mathcal{A}_r, v)} s_a^k, \quad \forall v \in V \setminus \{v_d^k\} \tag{3.5}$$

$$z_a^k \leq \sum_{a \in \mathcal{A}_r} s_a^k, \quad \forall a \in \mathcal{A} \tag{3.6}$$

$$z_a^k \leq |E| d_a^k, \quad \forall a \in \mathcal{A} \setminus \mathcal{A}_r \tag{3.7}$$

$$z_a^k \leq |E|(s_a^k + d_a^k), \quad \forall a \in \mathcal{A}_r \tag{3.8}$$

$$\sum_{a \in H(\mathcal{A}_r, v)} s_a^k + \sum_{a \in H(\mathcal{A}, v)} d_a^k = \sum_{a \in T(\mathcal{A}_r, v)} s_a^k + \sum_{a \in T(\mathcal{A}, v)} d_a^k, \quad \forall v \in V \tag{3.9}$$

The constraints (3.4) to (3.8) are generalized flow constraints that together ensure the connectivity of the route to the depot and prohibit any sub-tours. The variables z_a^k are flow variables for each edge direction. Constraint (3.4) define the amount of flow being released from the depot vertex $v_d^k = v_0$, which acts as a source of the flow. For any vertex v (other than the depot vertex), a flow equal to the number of servicing arcs, with v as the head, is absorbed by the vertex. This is expressed in constraints (3.5). The amount of flow through an arc is limited by constraints (3.6) to (3.8). An edge has a positive flow if and only if it is traversed. Finally, the vertex symmetry constraints (3.9) ensure that the number of arcs entering a vertex is same as the number of arcs leaving it.

ILP Formulation: The objective function of the line coverage problem is to minimize the total cost of the routes. We can now pose the line coverage problem as an optimization problem formulated as an ILP:

Minimize:

$$\sum_{k=1}^{K} c(\pi^k) = \sum_{k=1}^{K} \Big[\sum_{a \in \mathcal{A}_r} c_s(a) s_a^k + \sum_{a \in \mathcal{A}} c_d(a) d_a^k \Big] \tag{3.10}$$

subject to:

$$\sum_{k=1}^{K}\left(s^k_{a_e}+s^k_{\bar{a}_e}\right)=1, \quad \forall e \in E_r \tag{3.11}$$

$$q(\pi^k)=\sum_{a\in\mathcal{A}_r} q_s(a)\, s^k_a+\sum_{a\in\mathcal{A}} q_d(a)\, d^k_a \leq Q, \quad \forall k \tag{3.12}$$

$$\text{Routing constraints (3.4) to (3.9) for each robot } k \tag{3.13}$$

$$s^k_a \in \{0,1\}, \quad \forall a \in \mathcal{A}_r \quad \forall k \tag{3.14}$$

$$d^k_a, z^k_a \in \mathbb{N}\cup\{0\}, \quad \forall a \in \mathcal{A}, \quad \forall k \tag{3.15}$$

3.3.2 Multi-Depot Formulation

In graphs spanning over a large area, it may be inefficient or even infeasible to service all the locations from a single location. Hence, we develop a multi-depot formulation, wherein we are given a set of potential depot locations $V_d \subseteq V$, and a route can be assigned to any of the depots to start and end the route. The problem can be formulated by adding assignment constraints—each route needs to be assigned exactly one depot location from V_d.

In general, the entire vertex set V could be the set of potential depot locations. However, this would increase the complexity of the problem significantly. Instead, a smaller subset of vertices, ideally of size K or less, can help in reducing the complexity while also providing high-quality solutions. These locations can be selected from the field of operation based on terrain data or by clustering the vertices or the edges.

$$\sum_{d\in V_d} x^k_d = 1, \quad \forall k \tag{3.16}$$

$$v^k_d=\sum_{d\in V_d} v_d\, x^k_d, \quad \forall k \tag{3.17}$$

$$x^k_d \in \{0,1\}, \quad \forall v_d \in V_d, \forall k \tag{3.18}$$

We introduce the binary variable x_d^k, which is 1 when the depot $v_d \in V_d$ is assigned to route k, and 0 otherwise. Constraints (3.16) ensure that exactly one depot is assigned to each route. The assignment of the depots to the routes in done by constraints (3.17).

Routing constraints (3.4) and (3.5) depend on the assigned depot v_d^k. The constraint (3.4) is active only for the assigned depot, whereas constraints (3.5) are active for all the vertices except the assigned depot. These can be resolved by multiplying the variable x_d^k to both sides of the constraint (3.4), and the expression $(1 - x_d^k)$ to both sides of constraints (3.5).

However, this will result in a quadratic set of constraints, and the problem will become non-linear. Although such quadratic constraints can be converted to linear constraints by introducing additional binary variables and large constants, it would significantly increase the size of the problem. As the motivation for the multi-depot problem is to solve large instances, it is not beneficial to formulate the problem as an ILP, which becomes harder to solve for instances with a large number of variables and constraints. The difficulty of solving such variants of the problem further motivates the development of versatile constructive heuristic algorithms that can efficiently solve the problem for large instances.

3.4 Heuristic Algorithms for Line Coverage

The line coverage problem and its variants are NP-hard problems. Computing optimal solutions, e.g., using an ILP formulation, is usually feasible only for small instances. This motivates us to develop heuristic algorithms to compute high-quality solutions for large instances. Moreover, the algorithm is constructive—it maintains a set of feasible routes and iteratively merges pair of routes to form a new larger route. The constructive nature allows algorithm modification for several variants of the problem for robotics applications, such as incorporating turning costs and nonholonomic constraints.

3.4.1 Merge-Embed-Merge: A Constructive Heuristic

This section develops a new algorithm, Merge-Embed-Merge (MEM), for the line coverage problem. The underlying concept is to maintain a set of feasible routes; initially, a route is created for each required edge. Subsequently, routes are merged together greedily to form a smaller set of routes. This concept of merging was first presented by Clarke and Wright (1964) for the capacitated vehicle routing problem, and later adopted in the *Augment-Merge* heuristic by Golden et al. (1983) for the CARP. However, the Augment-Merge algorithm cannot handle the asymmetric costs and demands and the deadheading demands of the line coverage problem. Furthermore, the heuristic degrades rapidly with instance size, especially when the set of required edges is much smaller than the entire edge set (Corberán and Laporte, 2014). We improve this by including an *embed* step in our algorithm.

The MEM algorithm, given in Algorithm 5, comprises four components: (1) *initialization* of routes, (2) computation of *savings*, (3) *merging* two routes to form a new route, and (4) *embedding* the newly merged route. A max-heap data structure is used to greedily decide the two routes to be merged and embed new routes.

3.4.1.1 Representation of Routes

A route is represented by a sequence of required arcs that are to be serviced by the robot traversing the route. As the costs and the demands can be direction-dependent, arcs are used instead of the corresponding required edges. The robot starts at the depot, travels to the starting vertex of the first required arc, services the sequence of required arcs, and returns to the depot. Consider a route R_p, as shown in Figure 3.2: the vertex i is the starting vertex of the first required arc a_s, and the vertex j is the end vertex of the last required arc a_l in the sequence given by R_p. The robot will deadhead from the depot v_0 to vertex i, service the required arcs starting from i up to the last required arc ending at vertex j, and then deadhead back to the depot v_0. Note that the paths $v_0 \to i$ and $j \to v_0$ are the shortest paths and may contain several arcs, all deadheaded. If two successive required arcs a_t and a_{t+1} are not adjacent, the robot deadheads along the shortest path from the ending vertex of a_t to the starting vertex of a_{t+1}. Thus, the sequence $i \to j$ may involve deadheading between the constituent required arcs. The cost of the route R_p is given as:

$$c(R_p) = c_d(v_0, i) + c_s(R_p) + c_d(j, v_0) + \lambda \tag{3.19}$$

The shortest path gives the costs of deadheading from and to the depot. The cost of servicing the sequence of required arcs in R_p is given by $c_s(R_p)$, which may include deadheadings between non-adjacent required arcs. An additional constant cost λ representing route setup cost is added to the route cost. The setup cost helps reduce the number of routes during the merging process.

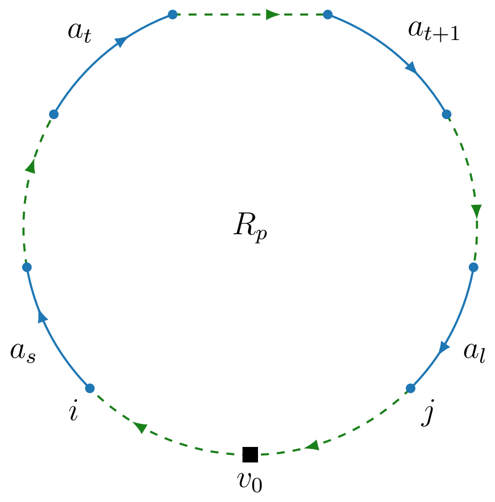


Figure 3.2: Representation of a route R_p as a sequence of arcs corresponding to required edges: The required arcs are shown as solid blue lines, and deadheadings are shown as dashed green lines. The route internally includes deadheadings given by shortest paths to and from the depot v_0 (black square). The route may have deadheadings between two non-adjacent required arcs.

Algorithm 5: Merge-Embed-Merge (MEM) algorithm

```
Input   : $G = (V, E, E_r)$, costs, demands, capacity $Q$
Output : Coverage routes $\mathcal{R}$, where each tour $R \in \mathcal{R}$ is a sequence of required edges
/* Initialization of routes                                          */
1  $\mathcal{R} \leftarrow \emptyset$; $k \leftarrow 1$;
2  for $e \in E_r$ do                                   // $a_e$ and $\bar{a}_e$ are arcs for $e$
3      $i \leftarrow \text{tail}(a_e)$; $j \leftarrow \text{head}(a_e)$;
4      $c \leftarrow c_d(v_0, i) + c_s(a_e) + c_d(j, v_0) + \lambda$;
5      $\bar{c} \leftarrow c_d(v_0, j) + c_s(\bar{a}_e) + c_d(i, v_0) + \lambda$;
6      if $c \leq \bar{c}$ then $R_k \leftarrow a_e$ else $R_k \leftarrow \bar{a}_e$;
7      $\mathcal{R}.\text{push}(R_k)$; $k \leftarrow k + 1$;
/* Compute savings                                                   */
8  $S \leftarrow \emptyset$;
9  foreach pair of tours $R_p, R_q$ do
10     Compute saving $s_{pq}$ for $R_p \uplus R_q$;
11     if $s_{pq} \geq 0$ and $\text{demand}(R_p \uplus R_q) \leq Q$ then
12         $S.\text{push}\big((p, q, s_{pq})\big)$;
13 make_heap($S$);                                                 // max-heap
/* Repeated Merge and Embed                                          */
14 while $S \neq \emptyset$ do
15     $(p, q, s) \leftarrow S.\text{extract-max}(\ )$;
16     if $R_p \neq \emptyset$ and $R_q \neq \emptyset$ then
           /* Merge                                                     */
17         $R_k \leftarrow R_p \uplus R_q$;
18         $\mathcal{R}.\text{push}(R_k)$; $k \leftarrow k + 1$;
19         $R_p \leftarrow \emptyset$; $R_q \leftarrow \emptyset$;
           /* Embed                                                     */
20         foreach tour $R_i$ with $i \neq k$ and $R_i \neq \emptyset$ do
21             Compute saving $s_{ki}$ for $R_k \uplus R_i$;
22             if $s_{ki} \geq 0$ and $\text{demand}(R_k \uplus R_i) \leq Q$ then
23                 $S.\text{insert}\big((k, i, s_{ki})\big)$;
24 Remove empty routes from $\mathcal{R}$;
```

3.4.1.2 Initialization of Routes

The MEM algorithm (Algorithm 5) constructs a route for each required edge in the initialization step (lines 2–7). Each edge $e \in E_r$ has two arcs a_e and $\bar{a}_e$ associated with it, representing the two travel directions. Let i and j denote the tail and the head of the arc a_e, respectively (line 3). Then the route for servicing a_e comprises the shortest path from the depot v_0 to the tail vertex i, servicing of arc a_e, and the shortest path from the head vertex j to the depot. In the other direction, the route comprises the shortest path from the depot to the head vertex, servicing of arc $\bar{a}_e$, and the shortest path from the tail vertex to the depot. Since the costs are asymmetric, of the two routes, the one with the lower cost is selected (line 6). It is assumed that the demand for the initial routes is less than the capacity, for else, the instance does not have a feasible solution. A constant route setup cost λ is added to the route cost. The routes are stored in the list $\mathcal{R}$, and there are $m = |E_r|$ routes initially. The particular case where an edge can be serviced in only one direction can also be handled appropriately in the initialization step.

3.4.1.3 Computation of Savings

Consider two routes R_p and R_q, with tail and head vertices given by t_p, h_p and t_q, h_q, as potential candidates for merging. There are eight possible permutations to merge the two routes, of which four are shown in Figure 3.3. The remaining four ways consist of routes in the reverse directions. The first merged route R_{pq} and its cost $\text{cost}(R_{pq})$ are:

$$R_{pq} := R_p \uplus R_q := v_0 \rightarrow t_p \xrightarrow{R_p} h_p \rightarrow t_q \xrightarrow{R_q} h_q \rightarrow v_0 \tag{3.20}$$

$$\text{cost}(R_{pq}) = c_d(v_0, t_p) + c_s(R_p) + c_d(h_p, t_q) + c_s(R_q) + c_d(h_q, v_0) + \lambda \tag{3.21}$$

Such a merge can have potential *saving* in cost since we no longer require $h_p \rightarrow v_0$ and $v_0 \rightarrow t_q$. There is also a cost-saving of a route setup cost λ as we have a single route instead of two. Thus, the net saving in cost s_{pq} for merging routes R_p and R_q is given by $\mathrm{cost}(R_p) + \mathrm{cost}(R_q) - \mathrm{cost}(R_{pq})$. However, the cost savings are affected by the direction of the edges due to asymmetry in the costs. Hence, we need to consider all eight permutations for merging two routes. Some of these permutations might not satisfy the capacity constraints, i.e., the total demand of the merged route may be more than the capacity Q. We denote by $R_p \uplus R_q$ the merged route with the maximum cost saving and satisfies the capacity constraint. If no such combination exists, then $R_p \uplus R_q = \emptyset$ and saving $s_{pq} = -\infty$.

For each pair of routes that yield a feasible merged route, the maximum saving in cost is computed and stored as a tuple (p, q, s_{pq}), where p and q correspond to the routes considered and s_{pq} is the corresponding saving (lines 9–12). These $m(m-1)/2$ tuples are stored in a binary max-heap data structure S, which can be built in $\mathcal{O}(m^2)$ computation time (line 13).

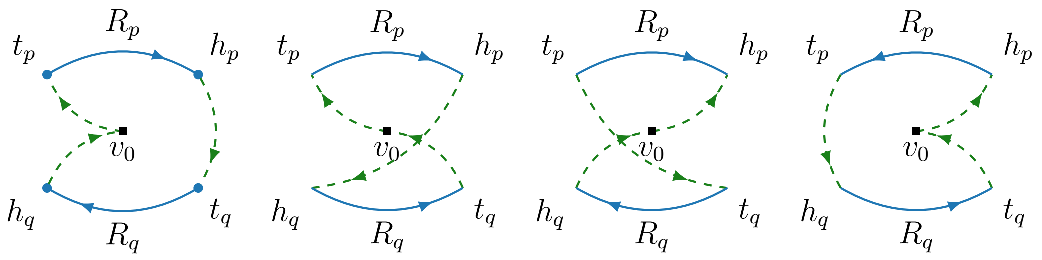


Figure 3.3: The figure shows four of the eight permutations to merge two routes R_p and R_q. The remaining four permutations consist of the shown permutations in the reverse directions. The tail and the head vertices for R_p are t_p and h_p, respectively. Similarly, t_q and h_q are defined for R_q. The first merged route is $v_0 \to t_p \xrightarrow{R_p} h_p \to t_q \xrightarrow{R_q} h_q \to v_0$ and its reverse direction route is $v_0 \to h_q \xrightarrow{\overline{R}_q} t_q \to h_p \xrightarrow{\overline{R}_p} t_p \to v_0$. The saving for merging two routes comes from potentially reduced deadheading to and from the depot.

Next, the merge and embed steps are executed repeatedly until no further merges are possible (lines 14–23).

3.4.1.4 Merge

The pair of routes with maximum cost savings is selected to form a new merged route, thereby maximizing immediate savings. The maximum element from the max-heap S is extracted (line 15), and the constituent routes are merged if neither is empty (lines 16–17). The merged route R_k is added to the list of routes $\mathcal{R}$ (line 18). The constituent routes are set to $\emptyset$ so that they are no longer considered for future merges (line 19). The complexity of the merge step is $\mathcal{O}(\log|S|)$, where $|S|$ is the number of elements in S.

3.4.1.5 Embed

We consider merging existing non-empty routes with the newly merged route R_k in the embed step (lines 20–23). Potential cost savings are computed for the new route R_k if merged with the other non-empty routes in the list $\mathcal{R}$. New tuples (k, i, s_{ki}) are generated and inserted into the max-heap S, if merging satisfies the capacity constraint and the cost saving is non-negative (lines 22–23). As there are $|\mathcal{R}| - 1$ such new tuples, the embed step has a computational complexity of $\mathcal{O}\left(|\mathcal{R}| \log\left(|S| + |\mathcal{R}|\right)\right)$.

The merge and embed components are executed until no further merges are possible, i.e., $S = \emptyset$. The maximum number of routes in the list $\mathcal{R}$ is upper bounded by $2m$, with at most m non-empty routes at any iteration. Here, $m = |E_r|$ is the number of required edges. The maximum number of elements in the max-heap S is $\mathcal{O}(m^2)$. Thus the complexity of the repeated merge-embed component over all possible merges is $\mathcal{O}(m^2 \log m)$, which is also the overall complexity of the algorithm. Depending on the structure of the instance, one may need to compute the shortest deadheading paths between all pairs of vertices. This can be done using the Floyd-Warshall algorithm in $\mathcal{O}(|V|^3)$ computation time (Dasgupta et al., 2006).

There are two essential characteristics of the algorithm that have practical benefits: (1) The algorithm maintains a feasible set of routes. These routes can be extracted at any point in the algorithm, giving it the *anytime* property. (2) The number of routes generated by the algorithm does not depend on the number of robots, i.e., the algorithm is agnostic to the number of robots available. It generates the number of routes required to cover the entire environment completely. Thus if a small fleet of robots is available, one can execute multiple routes for some of the robots by replacing or recharging batteries.

3.4.2 Multi-Depot Line Coverage for Large Graphs

In the above MEM algorithm, we considered a single depot location, where all the robots start and end their routes. However, it may not be efficient or feasible to service the required edges representing the linear features from a single location for environments spanning over a large area. Thus, we extend the MEM algorithm to enable multiple depots for the line coverage problem. We are given as input a set of depot locations $V_d \subseteq V$. These locations can be specified based on operator ease and field constraints, e.g., an operator may prefer to launch aerial robots from high vantage points. Alternatively, the locations can be determined using the k-medoids clustering algorithm.

Two modifications need to be made to the MEM algorithm given in Algorithm 5 to solve the multi-depot line coverage problem: (1) The initialization of the routes, and (2) The computation of savings for merging two routes.

3.4.2.1 Initialization of routes

The initialization process for the multi-depot line coverage problem is given in the procedure Initialize-MD. For each required edge, we iterate over all the depot locations and compute the cost of servicing the edge in the two directions. The number of such computations is $2|V_d|$. The one with the lowest cost is selected, provided the demand is less than the capacity. It is assumed that a required edge can be serviced from at least one of the depots in at least one direction. The complexity of the initialization step changes from $\mathcal{O}(|E_r|)$ to $\mathcal{O}(|V_d||E_r|)$.

```
Procedure Initialize-MD
Input  : $G = (V, E, E_r)$, depots $V_d$, costs, demands, capacity $Q$
Output : Initialized coverage routes $\mathcal{R}$ with assigned depots
1  $\mathcal{R} \leftarrow \emptyset$; $k \leftarrow 1$;
2  for $e \in E_r$ do                                        // $a_e$ and $\bar{a}_e$ are arcs for $e$
3      $R_k \leftarrow \emptyset$; $R_k$.cost $\leftarrow \infty$;
4      for $v \in V_d$ do                                    // Iterate over depots
5          $i \leftarrow$ tail($a_e$); $j \leftarrow$ head($a_e$);
6          $c \leftarrow c_d(v, i) + c_s(a_e) + c_d(j, v) + \lambda$;
7          $d \leftarrow q_d(v, i) + q_s(a_e) + q_d(j, v)$;
8          if $d \leq Q$ and $c < R_k$.cost then
9              $R_k \leftarrow a_e$; $R_k.v_0 \leftarrow v$;       // Form route with depot $v_0$
10         $\bar{c} \leftarrow c_d(v, j) + c_s(\bar{a}_e) + c_d(i, v) + \lambda$;
11         $\bar{d} \leftarrow q_d(v, j) + q_s(\bar{a}_e) + q_d(i, v)$;
12         if $\bar{d} \leq Q$ and $\bar{c} < R_k$.cost then
13             $R_k \leftarrow \bar{a}_e$; $R_k.v_0 \leftarrow v$;  // Form route with depot $v_0$
14     $\mathcal{R}$.push($R_k$); $k \leftarrow k + 1$;
```

3.4.2.2 Computation of Savings

In the single depot version of the MEM algorithm, we considered eight permutations for merging two routes, four of which are shown in Figure 3.3. The cost saving for merging two routes R_p and R_q is given by cost(R_p) + cost(R_q) − cost(R_{pq}), where R_{pq} is the merged route. In the multi-depot formulation, the cost and the demand of the merged route depend on the depot v_0—the merged route can be assigned any of the depots, independent of the constituent routes. However, the choice of the depot affects the saving in cost that can be achieved by merging two routes. Thus, we iterate over all the depots for the multi-depot line coverage problem and check all the eight merging permutations for each depot. This gives us $8|V_d|$ computations, and the one with the maximum saving is selected provided it satisfies the capacity constraint.

The most expensive component of the MEM algorithm is the embed step. This step involves computation of savings of a newly merged route with the others in the current set of routes $\mathcal{R}$ (lines 20–23 in Algorithm 5). Thus, the overall complexity of the algorithm changes to $\mathcal{O}(|V_d|m^2 \log m)$. As the computational cost depends on the number of depot locations, keeping it as small as necessary is advantageous. In practice, making the number of depots equal the number of robots available gives good results.

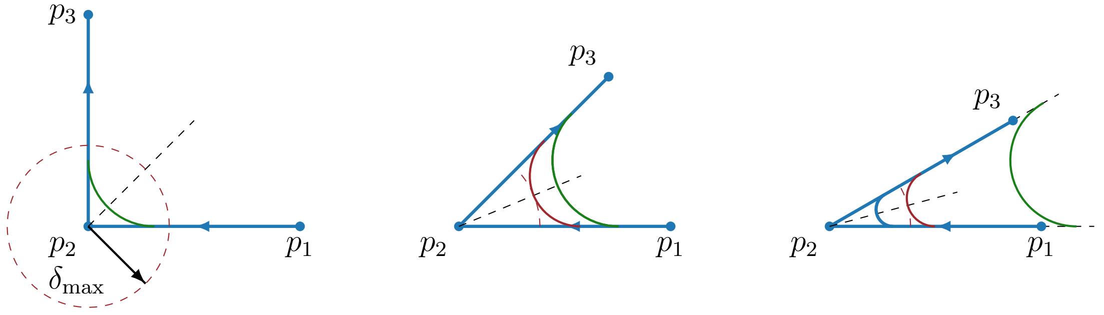
A common approach for solving multi-depot problems is to cluster the required edges and create subgraphs (Muyldermans et al., 2003; Mourão et al., 2009). Each subgraph is then treated as an instance of the single-depot problem. However, there are two issues with the process: (1) The final routes become dependent on the clustering algorithm, which is generally non-deterministic, and requires multiple runs of the entire algorithm to get high-quality solutions on average. (2) As the generated clusters are treated independently, sometimes multiple routes are generated for a cluster with one serving only a small number of edges, i.e., the robots are not using their capacity to the full potential. The algorithm does not have a choice to exchange or transfer some of these edges to nearby clusters. This often leads to inefficient solutions. These limitations are resolved in the multi-depot version of the MEM algorithm by directly considering the depots in the routing process. Although the clustering may be used for assigning depot locations, it is not used to cluster the required edges. Furthermore, compared to the cost of running the whole algorithm for different seeds of the clustering algorithm, the additional cost of iterating over the depots in the initialization and the computation of savings is much lower, leading to an efficient solver for the multi-depot line coverage problem.

3.4.3 Line Coverage with Turning Costs and Nonholonomic Constraints

Until now, we have considered the costs and the demands for traversing the edges to be arbitrary non-negative constants. However, they do not model turning costs and nonholonomic constraints. A smooth trajectory is desired in several robotics applications. Even for differential drive robots, which can perform turns in place, a sharp turn is undesirable as the robot will need to slow down, take a turn, and then accelerate. Smooth paths are often computed as a post-processing step after path planning (Ravankar et al., 2018). Such path smoothing techniques can be integrated with the MEM algorithm so that the costs and the demands of routes are closer to the actual values. This would require modification of the initialization procedure and the computation of savings, similar to the previous section for the multi-depot line coverage problem. However, path smoothing can be an expensive process, and doing so $8|V_d|$ times for the embed step in each iteration may not be practical for applications that require rapid solutions. This section illustrates constant-time procedures to generate routes with smooth turns and paths that respect nonholonomic constraints, and integrates them with the MEM algorithm. Furthermore, the costs and the demands for taking turns are incorporated into the MEM algorithm. We use Dubins curves for nonholonomic robots, which assume instantaneous steering.

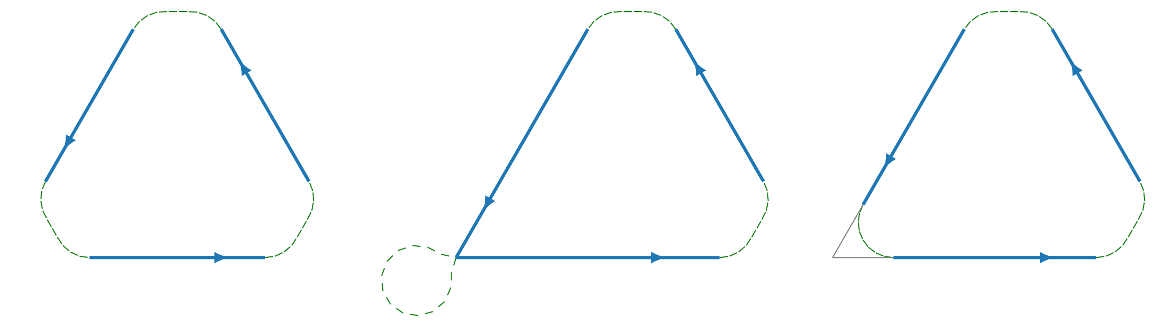
3.4.3.1 Smooth Turns and Turning Costs

We first consider generation of routes with smooth turns for holonomic robots. We will later see that such smooth turns are useful for nonholonomic robots as well. For a specified linear velocity when a robot is in motion, the minimum turning radius for a robot is given by the ratio of its linear and angular velocities; if the turning radius is small, the robot can take sharper turns. When two adjacent edges do not have a sharp corner, the robot can take a smooth turn without deviating too much from the edges, as shown by the green arc in Figure 3.4(a). The arc is tangential to the two edges. A user-defined parameter $\delta_{\max}$ determines the maximum allowable deviation from the edges, as shown by the red dashed circle. The parameter can be set based on the sensor field-of-view to ensure coverage of every point on the corresponding linear features. However, if the turn is sharp, the robot needs to slow down so that the maximum deviation is within $\delta_{\max}$, as shown by the red arc in Figure 3.4(b). This requires the robot to decelerate to achieve the required turning radius by decreasing the linear velocity. We enforce that the robot can start decelerating only after it has reached the middle of the edge to modularize the computation of the turning costs and to avoid a cascading effect in computing the cost and the demand for a route. When a turn is very sharp, shown in Figure 3.4(c), the robot may not have enough length available to decelerate to the required velocity for the red arc corresponding to δ_{max}. This requires solving a quadratic equation to determine a time-optimal turning arc, which corresponds to the innermost blue arc in the figure. In the worst case of a 180 degree turn, the robot may need to come to a complete stop. As the accelerations of the robots are usually high, and the lengths of the edges are comparatively large in practical applications, we can assume that the robot can come to a complete stop from its full speed within half the length of the edge; otherwise the robot may not be able to take 180 degree turns. Note that the formulation does not restrict having different speeds for servicing and deadheading. These three cases allow the modeling of smooth turns between two adjacent edges in constant time.



(a) Wide turn (b) Sharp turn (c) Very sharp turn

Figure 3.4: Smooth turns for a robot traversing two adjacent edges from p_1 to p_3 through p_2. The minimum turning radius is given by the ratio of its linear and angular velocities, and the corresponding arc is shown in green. (a) When the corner is wide, the robot can turn smoothly without changing its velocity and deviating too much from the original path. (b) However, if the turn is sharp, the robot needs to slow down so that the deviation is within the permitted limit δ_{max}. The optimal turning arc is shown in red. (c) If the turn is very sharp, the robot may not have enough distance to decelerate to the required velocity for the red arc. In the worst case of a turn of π radians, it may have to come to a complete stop. The innermost blue arc shows the optimal turning arc based on the maximum deceleration and the turning angle.



(a) No adjacent edges, Dubins curves

(b) Adjacent edges, Dubins curves

(c) Adjacent edges, Dubins curves and smooth turns

Figure 3.5: Modeling of deadheading paths for nonholonomic robots. (a) The graph does not contain sharp turns and Dubins curves give optimal paths for deadheading. (b) When we have adjacent edges, common in road networks, Dubins curves can create several circular arcs to orient the robot along the edges. (c) We introduce smooth turns with Dubins curves to generate efficient deadheading paths between required edges.

3.4.3.2 Nonholonomic Robots

Several commonly used robots, such as car-like robots, fixed-wing UAVs, and underwater robots, have nonholonomic constraints, i.e., the robots cannot take turns in place. We use a unicycle model of the robots (Lynch and Park, 2017) and incorporate Dubins curves for the motion of the robots. Dubins curves are often used to determine optimal paths from one pose of the robot to the other (LaValle, 2006), where the pose comprises the position and the orientation of the robot. An example of a coverage route using Dubins curves to determine optimal deadheadings is given in Figure 3.5(a). A Reeds-Shepp model can also be used to determine turns for nonholonomic robots.

The Dubins curves can be inefficient when we have adjacent required edges, as shown in Figure 3.5(b), as we may have to take extra turns to align the heading of the robot with the subsequent edge. Instead, we leverage the turning cost model, described in the previous subsection, to allow the robots to deviate from their path within a given limit $\delta_{\max}$. This eliminates most of the extra turns generated by just using Dubins curves. An example is shown in Figure 3.5(c). In the case of fixed-wing UAVs that have a lower bound on the minimum speed, the algorithm can choose to follow Dubins curves for very sharp turns.

Procedure Initialize-MD-Turns

```
Input  : $G = (V, E, E_r)$, depots $V_d$, costs, demands, capacity $Q$
Output : Initialized coverage routes $\mathcal{R}$ with assigned depots
$\mathcal{R} \leftarrow \emptyset$; $k \leftarrow 1$;
for $e \in E_r$ do                                          // $a_e$ and $\bar{a}_e$ are arcs for $e$
    $R_k \leftarrow \emptyset$; $R_k$.cost $\leftarrow \infty$;
    for $v \in V_d$ do                                      // Iterate over depots
        $c \leftarrow c_d(v, a_e) + c_s(a_e) + c_d(a_e, v) + \lambda$;
        $d \leftarrow q_d(v, a_e) + q_s(a_e) + q_d(a_e, v)$;
        if $d \leq Q$ and $c < R_k$.cost then
            $R_k \leftarrow a_e$; $R_k.v_0 \leftarrow v$;           // Form route with depot $v_0$
        $\bar{c} \leftarrow c_d(v, \bar{a}_e) + c_s(\bar{a}_e) + c_d(\bar{a}_e, v) + \lambda$;
        $\bar{d} \leftarrow q_d(v, \bar{a}_e) + q_s(\bar{a}_e) + q_d(\bar{a}_e, v)$;
        if $\bar{d} \leq Q$ and $\bar{c} < R_k$.cost then
            $R_k \leftarrow \bar{a}_e$; $R_k.v_0 \leftarrow v$;           // Form route with depot $v_0$
    $\mathcal{R}$.push($R_k$); $k \leftarrow k + 1$;
```

Given an initial and a final heading angle at the depots, we require three kinds of cost functions for deadheading: (1) from a depot to any required arc $c(v_0, a)$, (2) from one required arc to another $c(a_1, a_2)$, and (3) from a required arc to a depot $c(a, v_0)$. Here, v_0 is a depot, and $a, a_1, a_2 \in \mathcal{A}_r$ are required arcs. For environments with cost functions that satisfy the triangle inequality, these cost functions can be computed in constant time using the above procedures for smooth turns and Dubins curves. When the triangle inequality is not satisfied, we use the technique of line graphs[3] Winter (2002); Geisberger and Vetter (2011) to compute the shortest deadheading paths and cost functions, which can be computed in $\mathcal{O}(|E|^3)$ time, where E is the number of edges in the graph. The procedure Initialize-MD-Turns shows the modified initialization using multiple depots and turning costs. Similarly, the cost savings for merging can be modified to incorporate turning costs. The running time

[3]Winter (2002); Geisberger and Vetter (2011) refer to line graphs as dual graphs, which should not be confused with dual graphs in graph theory.

of the MEM algorithm does not change and is given by $\mathcal{O}(|V_d|m^2 \log m)$.

3.5 Simulations and Experiments

We developed the Merge-Embed-Merge (MEM) heuristic algorithm for the line coverage problem with multiple resource-constrained robots to obtain high-quality solutions rapidly. We further developed algorithms for large-scale graphs using a multi-depot formulation and for nonholonomic robots. This section empirically validates these assertions about the algorithm through simulations and experiments. We analyze the performance of the MEM algorithm, in terms of computation time and the solution quality, on a dataset of 50 road networks. We perform experiments with aerial robots on the UNC Charlotte campus road network. Finally, we illustrate the line coverage problem with nonholonomic robots.

We use the DJI Phantom 4 quadrotor in our experiments, with the following cost model for traversing the edges. Denote the speed of the UAV by v and the wind speed by w. Let the travel vector $\mathbf{t}$ denote the traversal of an edge from its tail vertex v_t to the head vertex v_h. Let ϕ be the angle between the wind vector and the travel vector $\mathbf{t}$ for an edge. Then the effective speed of the UAV is given by:

$$v_{\text{eff}} = w \cos\phi + \sqrt{v^2 - w^2 \sin^2\phi}$$

The cost function is defined as the time taken for the UAV to traverse an edge:

$$c(v_t, v_h) = \frac{\|\mathbf{t}\|_2}{v_{\text{eff}}}$$

Here, $\|\mathbf{t}\|_2$ is the Euclidean distance from v_t to v_h. We use different speeds for servicing and deadheading, and the value of v is set accordingly based on the travel mode. Note that the cost function is direction-dependent due to wind, and hence, the graph is asymmetric.

For convenience, we use travel time defined by the cost function as the demand function and specify the capacity in terms of the allowable flight time for the UAVs. The cost and the demand functions need not be the same in practice, and functions that model battery consumption can be used instead. Our open-source implementation allows for any cost and demand functions with a non-negative cost and demand for the edges.

3.5.1 Simulation Analysis on a Dataset of 50 Road Networks

The analysis is performed on a dataset of 50 road networks from the most populous cities around the world. The road networks are representative of environments with linear features, and they provide widely varying graph structures for a thorough evaluation of the algorithm. These networks are represented by line segments, which form the set of required edges. The endpoints and the intersections of the road segments form the vertex set in the graph. As UAVs that fly at high altitudes can travel from one vertex to another, for each pair of vertices, we add a non-required edge if a required edge does not exist between the vertices. The MEM algorithm is implemented in C++ and executed on a standard laptop with an Intel Core i7-1195G7 processor on a single core. We compare the quality of the solutions computed using the MEM algorithm with the solutions from solving the ILP formulation. The ILP formulation is solved using Gurobi Optimization (2021), and the C++ API was used to interface with the solver. The ILP formulation is executed on a cluster node with an Intel Xeon Gold 6248R processor using 16 cores in parallel for each road network. Additionally, the solutions obtained using the MEM algorithm are used to provide an initial solution to the solver, which helps in upper bounding the branch-and-bound algorithm used by ILP solvers. As the problem is NP-hard, the ILP formulation can take a very long time to reduce the gap between the current best solution and the lower bound, even though a powerful cluster node is used and an initial solution is provided. Thus, we limit the execution time to 24 hours.

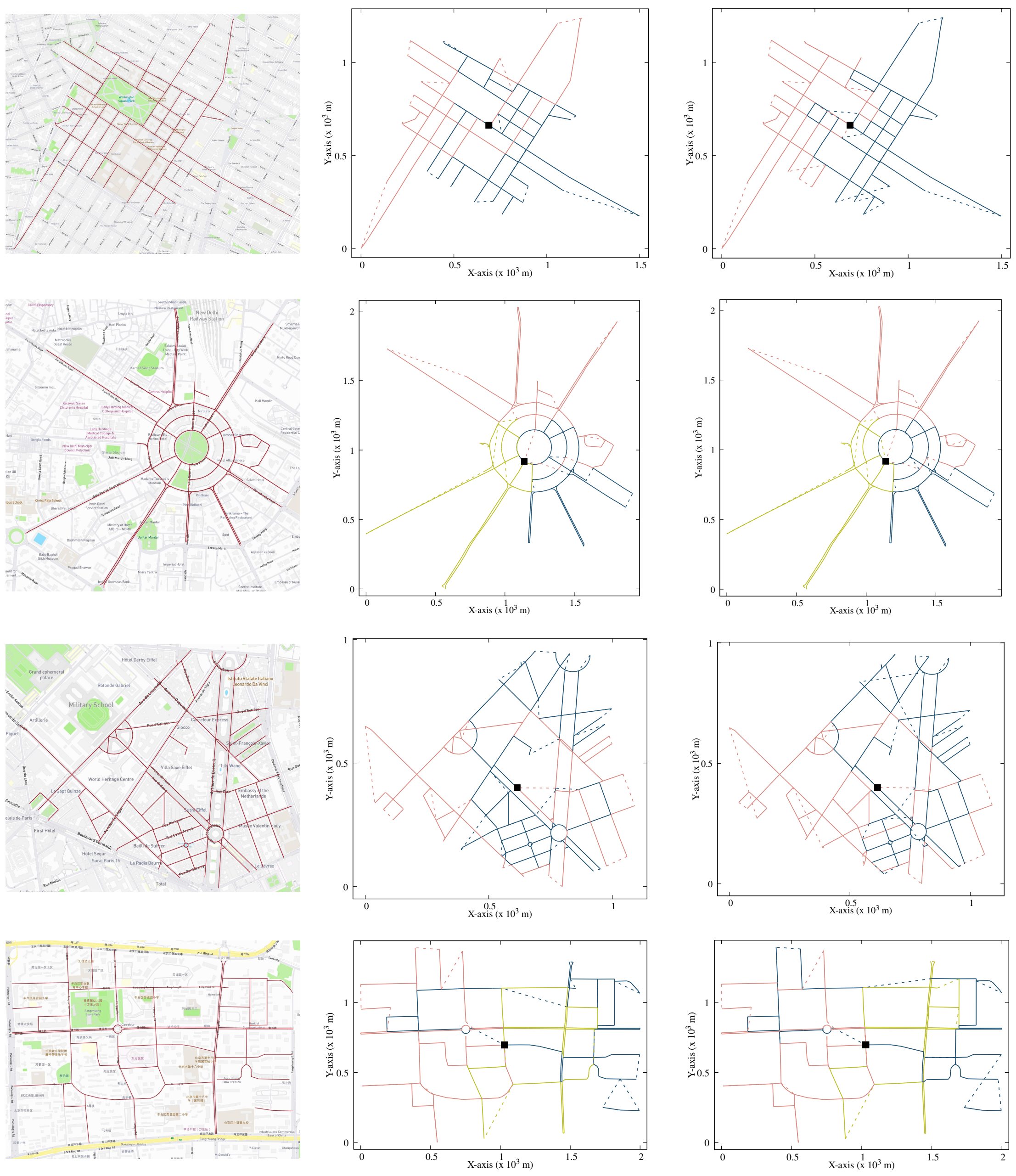


Figure 3.6: Four of the 50 sample road networks obtained for the most populous cities: The first column is the map with the input graph, the second column is the solution obtained using the ILP formulation, and the third column is the solution obtained using the MEM algorithm. The road networks, from top to bottom, are from (a) New York, (b) Delhi, (c) Paris, and (d) Beijing. Only the required edges are shown in the input graph, and there is a required edge for each pair of vertices in the graph. For example, the New York graph has 379 vertices, 402 required edges, and 71,361 non-required edges. The solid lines represent servicing travel mode, and the dashed lines represent deadheading travel mode.

We set the servicing and the deadheading speeds to $7\,\mathrm{m{\cdot}s^{-1}}$ and $10\,\mathrm{m{\cdot}s^{-1}}$, respectively. A wind of $2\,\mathrm{m{\cdot}s^{-1}}$ is simulated from the south-west direction, i.e., $\pi/4$ radians from the horizontal axis. For the first set of experiments, we set the capacity of the robot to 20 minutes (1,200 s). Figure 3.6 shows four of the fifty road networks, along with the routes obtained using the ILP formulation and the MEM algorithm. We consider a single depot location for these simulations, shown by a black square. The depot is set to be the vertex closest to the mean of all the vertices in the graph. The road networks provide a rich set of graph structures, as illustrated by the four sample graphs. The solutions from the MEM algorithm look similar to that of the ILP formulation, and the amount of deadheading is minimal.

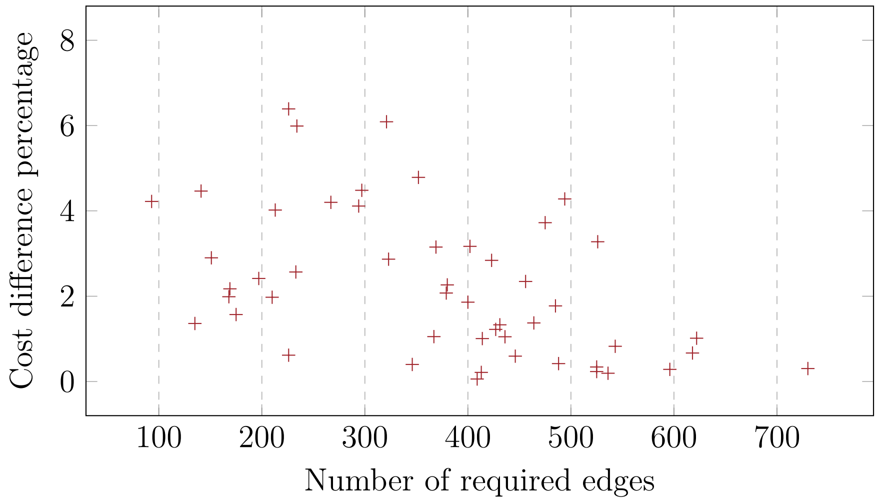


Figure 3.7: Comparison of the solutions generated using the MEM algorithm with the solutions from ILP formulation for the road network dataset. The cost difference percentage is computed as $100\frac{c-c^*}{c^*}$. The MEM solutions are within 7% of the ILP solutions. The number of required edges correspond to the number of linear segments in the road networks.

Figure 3.7 shows the cost difference percentage, computed as $100\frac{(c-c^*)}{c^*}$, where c and c^* are the costs of solutions computed using the MEM algorithm and the ILP formulation, respectively. Note that the difference in cost does not deviate significantly as the number of required edges increases. The performance of the ILP decreases rapidly for large graphs as the number of variables increases, as the ILP is not able to converge to an optimal solution within the computation time limit of 24 hours. Furthermore, graphs with a large total network length require more routes, increasing the number of variables in the ILP formulation. However, the MEM algorithm generates high-quality solutions even for large graphs.

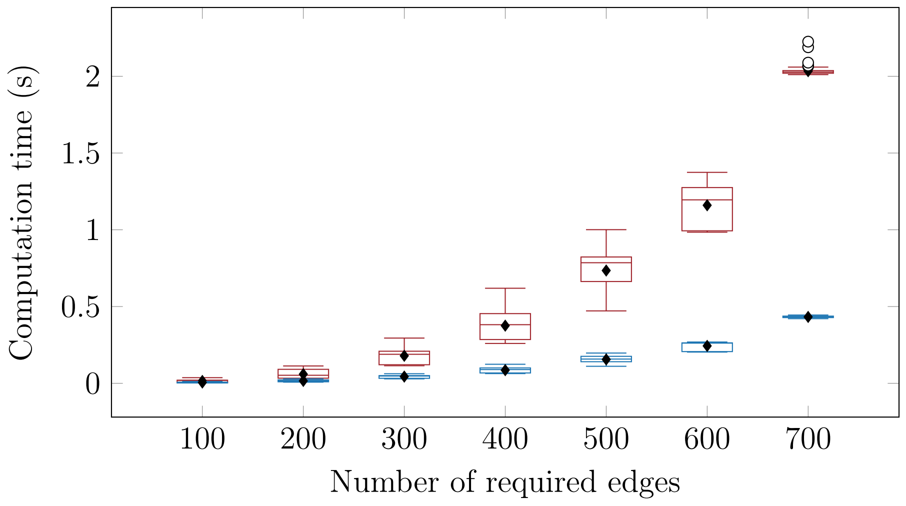


Figure 3.8: Computation time for the road network dataset using the MEM algorithm: Each road network is placed in bins of size 100 according to the number of required edges. The MEM algorithm is executed 100 times for each road network. The results are shown as a boxplot: the red boxes represent the total computation time, while the blue boxes represent the time taken by the MEM algorithm. Diamond markers show the average, and the circles are outliers.

Figure 3.8 shows the computation time required to obtain solutions for the road network dataset. Most of the computation is spent processing the road network and creating the graph. The largest road network with 730 required edges is solved in about 2 s, of which the MEM algorithm takes less than 0.5 s. For graphs with less than 200 required edges, the solutions are generated within 0.1 s. The analysis of computation time confirms our assertions that the algorithm is very fast for robotics applications. It can also be used in real-time, where the graph structure and the remaining battery life are updated as robots traverse the environment.

Next, we analyze the performance of the MEM algorithm with different robot capacities. For each road network, the capacity is set as a fraction of the minimum

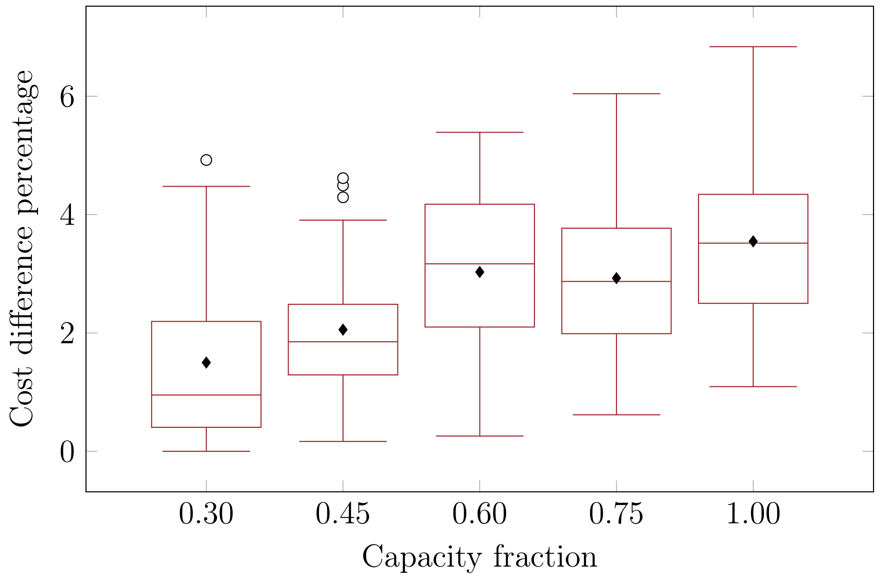


Figure 3.9: Cost difference between the solutions generated using the MEM algorithm and the ILP solution for different capacities. For each instance, the capacity is set as a fraction of the route cost for a single robot with infinite capacity. The results are shown as boxplots, circles show the outliers, and the diamond markers show the average. The average difference in the cost between the solutions generated by the MEM algorithm and the ILP formulation is 2.61%.

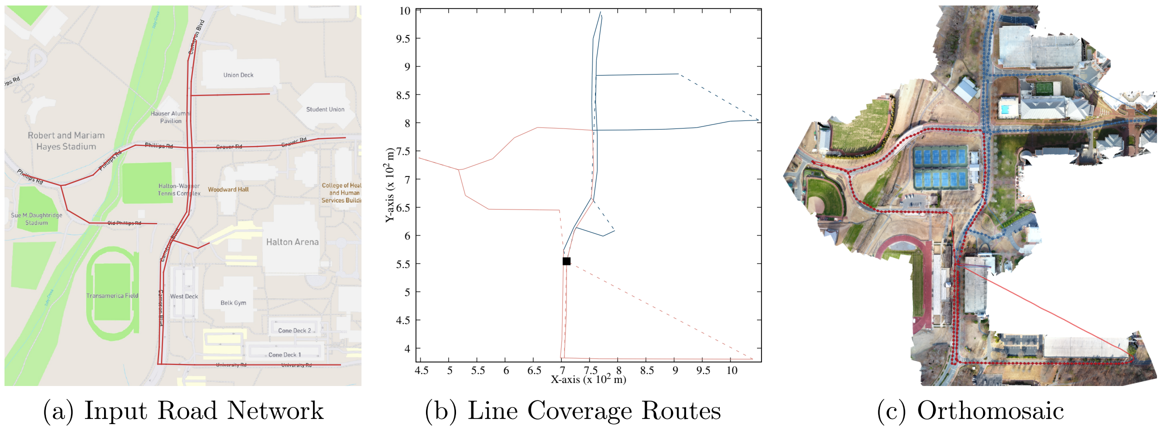
cost required to cover the entire network using a single robot. Figure 3.9 shows the cost difference percentage between the solutions obtained using the ILP formulation and the MEM algorithm with varying capacity fractions. The performance of the ILP formulation worsens as the capacity decreases because more routes are required to cover the entire road network. The performance of the MEM algorithm remains consistent. The capacity does not have a significant effect on the running time, and the algorithm requires fewer merges as the capacity decreases.

The simulations on the road network dataset show that the MEM algorithm generates solutions with costs comparable to that of the bounded-computation-time ILP formulation, i.e, within 7%, and with a maximum computation time of around 2 s. We modeled a fixed depot location, cost and demand functions, capacity constraints, and wind conditions within the same framework. These results show that the algorithm is suitable for deploying robots, in particular aerial robots, for coverage of linear infrastructure.

3.5.2 Experiment on a Road Network with a Single Depot

We performed experiments with a UAV on a portion of the UNC Charlotte campus road network, shown in Figure 3.10. The service and the deadhead speed were set to $3.33\,\mathrm{m{\cdot}s^{-1}}$ and $5\,\mathrm{m{\cdot}s^{-1}}$, and a wind speed of $0.89\,\mathrm{m{\cdot}s^{-1}}$ from the west was incorporated based on the wind conditions on the day of the experiment. A conservative capacity of 600 s was selected. Coverage solutions were generated using both the ILP formulation and the MEM algorithm. The ILP formulation gave the optimal solutions for the road network. The computed and the actual flight times are shown in Table 3.2. The actual flight times include take-off and landing and are close to the computed cost.

The experiment shows that the line coverage problem is well-suited to model coverage of linear infrastructure such as a road network. The problem models two different modes of travel and incorporates wind conditions in the formulation. Furthermore, activating the sensors only during servicing helps in reducing the amount of data required that needs to be processed for analysis and creating orthomosaic maps.



(a) Input Road Network (b) Line Coverage Routes (c) Orthomosaic

Figure 3.10: Line coverage of a portion of the UNC Charlotte road network with two routes and a single depot. (a) The input road network has a length of 2,658 m with 48 vertices and 48 required edges (shown as red lines). There are 1,128 non-required edges formed by pairs of vertices (not shown). (b) Two routes distinguished by different colors are computed using the MEM algorithm. The algorithm computes a depot location, shown by the black square, from where the UAVs start and end their routes. The solid lines represent servicing, while the dashed lines represent deadheading. (c) An orthomosaic generated from the images taken by the UAVs flown autonomously along the computed routes. The lines show the actual flight path, and the dots are the locations where the images were taken.

Table 3.2: Comparison of computed and actual flight times

	Computed Cost (s)		Actual Flight Time (s)	
	Route 1	Route 2	Route 1	Route 2
ILP	462	554	527	642
MEM	473	599	548	688

3.5.3 Experiment on a Large-Scale Road Network

We used the multi-depot MEM algorithm for the UNC campus road network spanning an area of 1.5 km^2 and a length of 12 km, shown in Figure 3.1. The road network consists of 842 vertices, 865 required edges, and 353,196 non-required edges. k-medoids clustering was used to obtain eight depot locations, shown as black squares in the figure. The depots are well distributed in the road network such that no road segment is far away from all the depots. The multi-depot MEM algorithm computed eight routes, and no two routes share the same depot. Note that the k-medoids algorithm is only used to compute the depot locations, and we do not cluster the edges. A service speed of $5\,\mathrm{m{\cdot}s^{-1}}$ and a deadhead speed of $8\,\mathrm{m{\cdot}s^{-1}}$ were set for the experiments. The computed routes and the orthomosaic generated from the images collected during flights are shown in Figure 3.1. Table 3.3 gives computed costs, actual flight times, and the number of images collected for each route. For these experiments we did not incorporate the wind conditions, which could be a reason for a higher deviation between the computed cost and the actual flight time.

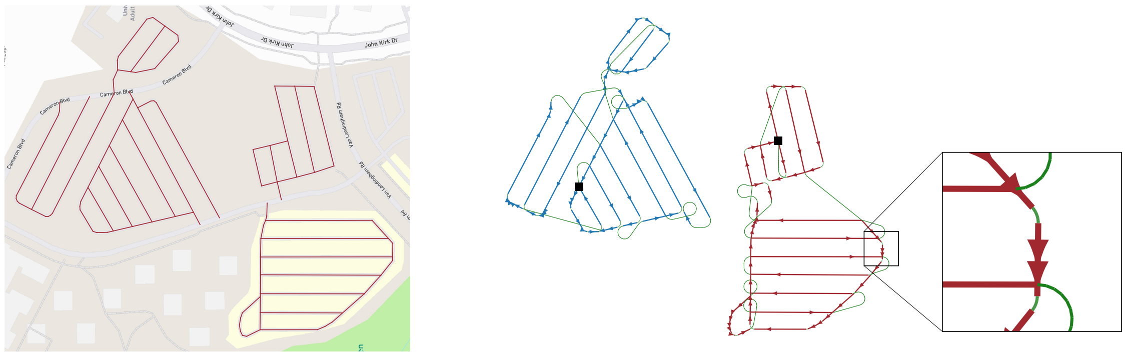
Table 3.3: Data of flights for the UNC Charlotte Road Network

Computed Cost (s)	Flight time (s)	Number of images
215	346	87
458	616	180
407	496	170
538	657	221
371	493	140
449	599	192
481	627	210
277	393	120

It can be observed from the routes that the individual routes primarily consist of road segments that are close to each other and are connected, indicating that the MEM algorithm can efficiently distribute the line features among routes. Furthermore, the routes are assigned to the closest depot, showing that the multiple depot formulation of the MEM algorithm is well-suited for the line coverage problem with large graphs.

3.5.4 Nonholonomic Robots

We combined the multi-depot formulation and the nonholonomic formulation for a network of lanes on a set of parking lots as an illustrative example. The input network and the planned routes are shown in Figure 3.11. The service and the deadhead speeds were set to $3.33\,\mathrm{m{\cdot}s^{-1}}$ and $5.00\,\mathrm{m{\cdot}s^{-1}}$. A maximum angular velocity of $\pi/4\,\mathrm{rad{\cdot}s^{-1}}$ and maximum acceleration of $3.00\,\mathrm{m{\cdot}s^{-2}}$ were set. A deviation limit $\delta_{\max}$ of $4\,\mathrm{m}$ was selected to allow smooth turns for adjacent required edges. Note that the acceleration is only used for generating smooth turns for adjacent edges with very sharp corners. It is not required for generating Dubins curves. Two depots were obtained using clustering. The routes were generated for a flight time of 600 s. The deadheadings are composed of Dubins curves for non-adjacent required edges and smooth turns for adjacent required edges. Note that the algorithm computed routes that cover separated regions which reduces the amount of deadheading, thereby optimizing the total cost of the routes.



(a) Input Road Network (b) Line Coverage Routes

Figure 3.11: Line coverage of a network of lanes in a set of parking lots: (a) The total length of the lanes in the input graph is 2,982 m. There are 90 vertices, 104 required edges, and 4,005 non-required edges. (b) Coverage routes using two depots and nonholonomic robots. There are two routes distinguished by different colors. The green lines show deadheadings composed of Dubins curves and smooth turns. The arrows indicate the direction of travel. The inset shows an enlarged view of smooth turns for adjacent required edges.

3.6 Summary

This chapter presented the line coverage problem with multiple resource-constrained robots for coverage of linear features in an environment. The features can be line segments or any one-dimensional curves. The environment was modeled as a graph with the linear features forming a set of required edges, and the vertices of the graph comprising endpoints and intersections of the linear features. Additionally, the formulation permits non-required edges in the graph that the robots need not cover but can be used for faster travel between vertices. The edges have non-negative costs (e.g., travel time) and resource demands (e.g., battery life) associated with them. The formulation minimizes the total costs of the routes under the constraint that each route's total demand is within the available resource capacity. A unique characteristic of our formulation is that we allow two modes to travel for the robots—servicing and deadheading. The formulation models different cost functions and resource demands for the two modes of travel for the robots. These modes enable the algorithms to optimize the operation time, conserve energy, and reduce the amount of sensor data. Furthermore, the cost and the demand functions can be direction-dependent, i.e., the graph is asymmetric. This facilitates the modeling of wind conditions, uneven terrain, and one-way streets.

We posed line coverage as an optimization problem on graphs and formulated it as an integer linear program (ILP). The ILP provides an optimal solution for instances that have feasible solutions. However, the problem is NP-hard, and solving the ILP even for moderate-sized graphs can be computationally expensive. We, therefore, designed a heuristic algorithm, Merge-Embed-Merge (MEM), which has a polynomial-time complexity of $\mathcal{O}(m^2 \log m)$, where m is the number of linear features in the environment. The algorithm maintains a set of feasible routes and iteratively merges pairs of routes to form new larger ones. Eight possible ways of merging two routes dictate the savings in merging routes. The algorithm is constructive in the sense that new routes are constructed as the algorithm progresses, in contrast to other graph procedures where a digraph is used as a proxy to the routes. The constructive nature of the algorithm allows us to extend the algorithm to several variants of the line coverage problem.

When the environment is large, it may not be possible to cover the entire network from a single depot location. Thus, we formulate the line coverage problem with multiple depots and extend the MEM algorithm to solve the problem. This is the first fast and efficient algorithm for the line coverage problem with multiple depots and applies to related arc routing problems commonly used for human-driven vehicles. Taking turns can be very expensive for robots. Similarly, nonholonomic robots, such as fixed-wing UAVs and car-like ground robots, cannot make point turns. A common technique is to post-process the routes to obtain smooth trajectories. However, this can violate the capacity constraints and may lead to inefficient routes. Our formulation incorporates smooth turns and nonholonomic constraints into the algorithm by including costs and demands on resources due to the turns. Therefore, the routes are both efficient and safe with respect to resource constraints.

We evaluated the MEM algorithm on a dataset of 50 road networks from the most populous cities around the world. The networks have varying graph structures allowing a thorough assessment of the algorithm. The evaluation shows that the MEM algorithm computes high-quality solutions with costs within 7% of the costs of the ILP solutions. The MEM algorithm also performs similarly when the capacity of the robots is varied. The evaluation of the computation times shows that the algorithm is very fast as it solves the largest of graphs within 0.5 s, and the entire procedure, including creating graphs, takes around 2 s. For graphs with less than 200 required edges, the solutions are generated within 0.1 s.

We illustrated the algorithm in experiments on the UNC Charlotte road network. For the first experiment, two routes from a single depot location were autonomously executed by a commercial UAV to cover a portion of the road network. In the second experiment, we used the formulation for the line coverage problem with multiple depots to obtain eight depots and routes for the road network. These routes were then executed autonomously using a UAV. Next, we illustrated the line coverage algorithm with multiple depots and nonholonomic robots on lanes of a set of parking lots. These experiments show that the algorithm models coverage of real-world linear infrastructure well and can be conveniently used in commercial applications. The algorithm is also very fast and can be used for computing routes online on the robots for an environment with linear features that need to be updated.

CHAPTER 4: AREA COVERAGE

Area coverage is the task of efficiently servicing a given two-dimensional surface using sensors mounted on robots such as uncrewed aerial vehicles (UAVs) and uncrewed ground vehicles (UGVs). We present a novel formulation for generating coverage routes for multiple capacity-constrained robots, where capacity can be specified in terms of battery life or flight time. Traversing the environment incurs demands on the robot resources, which have capacity limits. The central aspect of our approach is transforming the area coverage problem into a line coverage problem (i.e., coverage of linear features), and then generating routes that minimize the total cost of travel while respecting the capacity constraints. We define two modes of travel: (1) servicing and (2) deadheading, which correspond to whether a robot is performing task-specific actions or not. Our formulation allows separate and asymmetric travel costs and demands for the two modes. Furthermore, the cells computed from cell decomposition, aimed at minimizing the number of turns, are not required to be monotone polygons. We develop new procedures for cell decomposition and generation of service tracks that can handle non-monotone polygons with or without holes. We establish the efficacy of our algorithm on a ground robot dataset with 25 indoor environments and an aerial robot dataset with 300 outdoor environments. The algorithm generates solutions whose costs are 10% lower on average than state-of-the-art methods. We additionally demonstrate our algorithm in experiments with UAVs.

4.1 Introduction

This chapter addresses the *area coverage problem*—the task of efficiently servicing a given planar surface. There are several applications of the area coverage problem; these include mapping and inspection of large regions using a team of aerial robots (i.e., UAVs), and vacuuming, lawn mowing and harvesting with ground robots (i.e., UGVs). The area coverage problem also applies to CNC-based machining operations, as illustrated by Arkin et al. (2000). The problem is widely studied in the robotics literature (see recent suverys by Galceran and Carreras, 2013; Cabreira et al., 2019). However, relatively few approaches for the area coverage problem consider multiple robots. Furthermore, practical constraints such as limited battery capacity and the effect of wind or uneven terrain are usually not considered. The chapter presents a method for area coverage that addresses these challenges. Even when these constraints are not modeled, the algorithms in this chapter, in comparison to recent work, generate higher quality solutions.

We consider two modes of travel for a robot. A robot is said to be *servicing* when it performs task-specific actions, such as taking images or vacuuming using its sensors or tools. A robot may travel from one location to another at faster speeds without performing task-specific actions—referred to as *deadheading*—to optimize the mission time, conserve energy, or reduce the amount of sensor data for analysis. The robots usually have a finite amount of resources, such as battery charge, which can be specified in terms of energy or a time limit, referred to hereafter as *capacity.* The robots must return to their home location before the resource consumed exceeds the capacity. The goal is to find efficient routes for a team of robots such that the entire environment is serviced while respecting the capacity constraints. Figure 4.1 shows an example environment and routes generated for capacity-constrained robots.

Our formulation for solving the area coverage problem consists primarily of three components: (1) *Cell decomposition* of the environment, (2) *Service track generation*

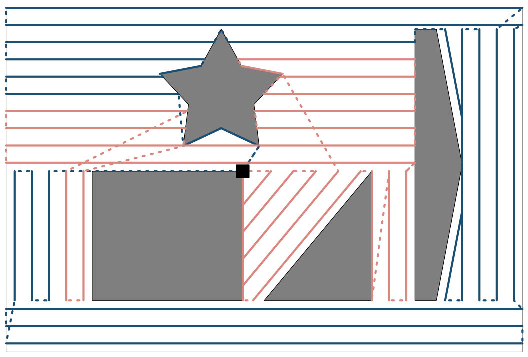


Figure 4.1: Area coverage of an environment with a team of capacity-constrained robots. The grey regions represent obstacles in the environment. The black square (near the center) represents the depot location—the robots start and end their routes at the depot. The solution consists of two routes, shown in dark blue and light red. The solid lines represent the service tracks. The dashed lines represent deadheading travel—the robots can turn off the sensors and travel at faster speeds along these line segments.

for individual cells, and (3) *Routing* to traverse the service tracks. The central aspect is to transform the area coverage problem into a line coverage problem—the coverage of linear features in an environment discussed in Chapter 3 and Agarwal and Akella (2020). The service tracks form the linear features that the robots must service, and an efficient algorithm for the line coverage problem is used to generate routes for the team of robots. This allows us to model the cost of travel (e.g., time), the demands on resources (e.g., battery), and asymmetric costs and demands for travel due to wind or uneven terrain. Our formulation facilitates a significant generalization of the cell decomposition component to reduce the number of turns that the robots must take. In particular, the cells are no longer required to be *monotone* polygons (Berg et al., 2008) with respect to the service direction. This generalization enables additional service directions for the cells to minimize the number of turns. Furthermore, allowing cells to be non-monotone polygons with holes enables the additional merging of adjacent cells with the same service directions. Merging adjacent cells reduces the number of service tracks by avoiding overlapping sensor coverage regions at the common boundary of a pair of adjacent cells. Additionally, we observe that a simple back-and-forth (i.e., boustrophedon) pattern does not always guarantee complete coverage. We mitigate this issue in the new service track generation algorithm.

The contributions of the chapter are:

1. A cell decomposition algorithm that allows non-monotone polygons and optimizes the number of turns that the robots need to take.

2. A new service track generation algorithm capable of handling non-monotone polygons with obstacles. The algorithm improves coverage of the environment over the traditional boustrophedon pattern.

3. A new formulation to transform an instance of the area coverage problem into that of the line coverage problem.

4. We minimize the total cost of coverage routes for multiple robots while respecting their capacity constraints.

5. An open-source implementation[1] of our algorithms.

This is the first method for cell decomposition and service track generation that minimizes the number of turns while allowing non-monotone polygons with holes. Furthermore, this is the first approach for the area coverage problem that allows two modes of travel, capacity constraints, and asymmetric travel costs and demands.

4.2 Related Work

Area coverage has a large body of work that has been covered extensively in recent survey papers by Galceran and Carreras (2013); Cabreira et al. (2019). The area coverage problem is related to the lawn mowing problem, which was shown to be NP-hard by Arkin et al. (2000). Consequently, several approximation and heuristic algorithms have been proposed. The approaches for area coverage problems can be broadly classified into approximate and exact methods.

[1]Source code available at:
`https://github.com/UNCCharlotte-CS-Robotics/AreaCoverage-library`.

Grid-based approaches, which fall under approximate methods, were some of the earliest techniques for solving the area coverage problem (Gabriely and Rimon, 2001). These methods typically discretize the environment into small cells based on a given resolution. Thus, the quality of the results depends on the resolution (Wei and Isler, 2018). Moreover, the computational complexity increases rapidly with environment size. Vandermeulen et al. (2019) address coverage with multiple robots using turn minimization as the objective for cell decomposition. The environment is contracted into a rectilinear polygon with integer side lengths. This new polygon is then decomposed into rectangles of unit width (called *ranks*) such that the sum of altitudes is minimized. An m-TSP algorithm is used to find paths for the robots. While the algorithm works well for rectilinear environments, it is not designed for complex non-rectilinear environments.

There has been recent interest in learning-based strategies. However, they are not yet generalizable to large complex environments. Usually, very small grid sizes are used to benchmark the results—a 16x16 grid was used by Apuroop et al. (2021) and a 7x7 grid by Theile et al. (2020). Retraining of the neural network was required for each environment in the approach by Theile et al. (2020). Moreover, these do not consider multiple robots. In contrast to the above grid-based approaches, our formulation can handle environments with non-rectilinear boundaries and obstacles. We also allow capacity constraints, asymmetric costs and demands, and two different modes of travel.

In this chapter, we focus our attention on exact methods. These methods typically use computational geometry and graph theory. A common approach is to decompose the environment into cells, known as cell decomposition. Choset (2000) presented the widely used boustrophedon cell decomposition (BCD), an efficient way to decompose a given environment with obstacles. The key idea is to generate *monotone polygons* (Berg et al., 2008) with respect to a given service direction using a sweep-line based algorithm.

In most mobile robotics applications, it is desirable to have long paths with as few turns as possible. Turns can be very expensive both in terms of time and battery consumption, as the robot may need to slow down, take a turn, and then accelerate again. Huang (2001) presented a minimum sum of altitudes (MSA) formulation. The MSA corresponds to the number of turns required for a robot to service the environment. For both convex and non-convex polygons, the service track orientation that minimizes the number of turns is parallel to one of the polygon edges. A dynamic programming algorithm with an exponential running time was presented to compute an optimal decomposition. In contrast, the BCD is computationally very efficient but does not consider the number of turns. Hence, several heuristic algorithms have been developed that trade off optimizing the number of turns and computational efficiency.

A trapezoidal decomposition was used by Oksanen and Visala (2009) to obtain an initial set of cells that are then merged to reduce the number of cells. Service directions are determined by using a bisection search. A sweep-line based algorithm, similar to BCD, was presented by Yu and Hung (2015) to obtain an initial decomposition of the environment. A service direction is determined independently for each cell. Adjacent cells that have the same service direction are merged if they remain monotone even after merging. Nielsen et al. (2019) obtain an initial decomposition of the environment by extending interior edges of concave vertices. An integer programming formulation and a heuristic algorithm are proposed to obtain solutions efficiently. An approach based on vehicle routing problems is used to route multiple robots. All these techniques require a set of monotone polygons. In contrast, our approach removes this requirement, enabling us to improve the cell decomposition procedure.

In the approaches by Mannadiar and Rekleitis (2010) and Xu et al. (2014), a Reeb graph is generated from the BCD, where the cells are represented by edges and the connectivity of cells is represented using vertices. An algorithm for the Chinese postman problem (CPP) finds a tour on the Reeb graph, which provides a sequence of cells to be visited by the robot. A single service direction is assumed for the entire environment, determined by simple heuristics such as longest edge or wind direction. This work was extended by Karapetyan et al. (2017) to multiple robots using clustering and the k-CPP algorithm for routing. These methods do not consider the minimization of the number of turns. Furthermore, each cell is treated as a unit. Assuming a fixed path for individual cells or treating each cell as a unit can be very restrictive, especially for capacity-constrained robots. The robots might not be able to cover multiple cells or even a single large cell, resulting in a large number of inefficient routes.

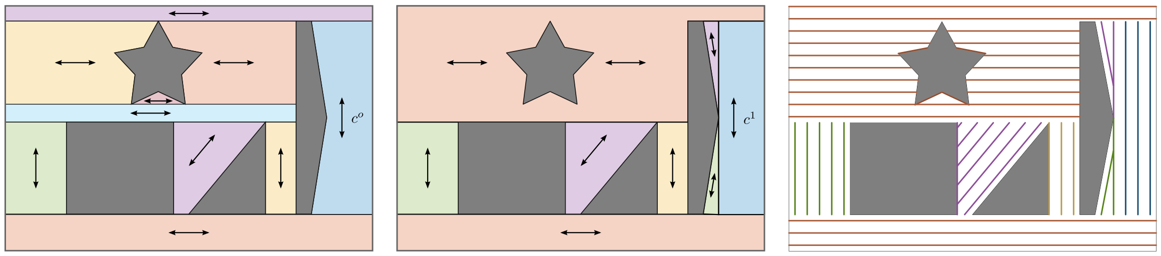
Algorithms for the generalized traveling salesperson problem (GTSP) are often used for computing routes. In the approach by Lewis et al. (2017), the BCD is used to obtain a set of cells with the same service direction. A GTSP instance is generated with two vertices for each service track for the two travel directions, forming a cluster. A GTSP algorithm generates a tour such that a single vertex is traversed from each cluster. In Bochkarev and Smith (2016), the cell decomposition starts with any convex decomposition of the polygon and is improved by adding cuts at reflex vertices. Finally, the GTSP is used to generate a tour on an auxiliary graph, similar to the approach by Lewis et al. (2017). In a recent paper by Bähnemann et al. (2021), the BCD is computed for each edge direction, and cells are assigned independent service directions. The BCD that has the least MSA is selected. For each cell and edge direction, four patterns for servicing are provided based on where the robot starts and ends. Each pattern forms a vertex in the GTSP instance graph, and vertices corresponding to the same cell are grouped in a cluster. A visibility graph is used to form edges between vertices. A GTSP tour then traverses a vertex, representing a pattern, from each cluster. In GTSP (Lewis et al., 2017; Bochkarev and Smith, 2016) and m-TSP (Vandermeulen et al., 2019) based approaches, the cells are usually not treated as a unit (except for Bähnemann et al., 2021) and thus, are more efficient. These procedures do not consider capacity constraints and are designed for a single robot. Algorithms for vehicle routing problems (VRP) allow capacity constraints and multiple robots (Toth and Vigo, 2014). Although the costs of the edges in both VRP and GTSP graphs can be asymmetric, the service tracks are represented as nodes, which do not have costs or demands. Thus the nodes cannot model asymmetric costs and demands of the service tracks. Our approach uses the line coverage problem to closely model the area coverage problem, with the edges in the graph representing the service tracks. This enables modeling of capacity constraints, asymmetric costs and demands, and two modes of travel.

4.3 Solution Approach for the Area Coverage Problem

Given a region $\mathcal{R} \subset \mathbb{R}^2$, the area coverage problem is to find a set of routes for a team of robots such that the total cost of the routes is minimized, and all the points in the region are serviced by the robots. Limited battery life is one of the most critical restrictions on mobile robots, especially for aerial robots. Thus, in our formulation, we incorporate an additional constraint that the total *demand* on resources for each route should not exceed a given capacity for the robots. The capacity can be specified in terms of energy, time limit, or travel length. We have two modes of travel for the robots: (1) servicing and (2) deadheading. A robot is said to be servicing if it performs task-specific actions, such as taking images, as it traverses a path. A robot may travel from one location to another while not performing servicing tasks, such as returning to the home/depot location. Such travel is known as deadheading. Functions for service and deadhead costs and demands are given as input to the problem. Our formulation can handle separate and asymmetric costs and demands.

We model the environment with a set of polygons. The environment may have obstacles or sub-regions that are not required to be serviced. These sub-regions are referred to as *holes*. Depending on the application, the robots may be permitted to travel across the holes, e.g., an aerial robot flying at a high altitude may optimize its path by flying over a hole representing a building. We treat the robots as point robots, unless otherwise specified. For finite-sized robots, we compute the free workspace using techniques for computing configuration space, such as the Minkowski sum (Berg et al., 2008).

We now describe our approach to solve the area coverage problem with multiple capacity-constrained robots. We break the problem into three components: (1) Cell decomposition, (2) Service track generation, and (3) Routing.



(a) Initial cell decomposition (b) Final cell decomposition (c) Service tracks

Figure 4.2: An environment with four obstacles. The double head arrows indicate the service directions. (a) In the initial decomposition with ten cells, the cell c^o has an optimal service direction for which it is not monotone. (b) The final decomposition, with eight cells, is obtained after the greedy improvement and then merging adjacent cells with the same service direction. The cells in the final decomposition are also not necessarily monotone polygons. (c) The service tracks are generated for each cell independently, and overlapping segments are removed. Service tracks include those belonging to the two scenarios described in Figure 4.3, e.g., the vertical purple and green tracks on the left edge of cell c^1 and some of the edges of the star-shaped obstacle.

4.3.1 Cell Decomposition

The primary motivation for the cell decomposition component is to minimize the number of turns that the robots need to take. This is done by decomposing the environment into smaller polygons, referred to as cells, and computing a *service direction* that minimizes the number of turns for each cell independently. Such an optimal direction is related to the minimum sum of altitudes (MSA) of a polygon and is parallel to one of the edges of the boundary or hole of the cell (Huang, 2001). The service tracks are generated parallel to the corresponding service direction for a cell. Our cell decomposition method is a culmination of experimentation with various existing methods. However, we deviate significantly in one crucial aspect—we allow the cells to be non-monotone with respect to the direction perpendicular to the service direction, i.e., the intersection of a cell (its interior) and a line parallel to the service direction need not be a connected line segment. Allowing non-monotone cells increases the feasible solution space, enabling cell decompositions that can potentially reduce the number of turns.

Our cell decomposition method is composed of three steps (1) initial decomposition, (2) greedy improvement, and (2) cell merging.

Initial Decomposition: We use an approach similar to that given by Bähnemann et al. (2021) for the initial decomposition.

1. Obtain a set of directions corresponding to the edges of the environment, i.e., edges of the outer boundary and the holes. Parallel directions are ignored. Let $\mathcal{V}$ denote the set of all such directions.

2. For each direction $v \in \mathcal{V}$:

    (a) Perform BCD with a line parallel to v and sweeping perpendicular to itself. Let $\mathcal{C}$ denote the set of cells obtained from the BCD.

    (b) For each cell $c \in \mathcal{C}$: Compute the MSA a_c and the corresponding service direction u_c, even if the cell c is non-monotone with respect to the direction perpendicular to u_c.

    (c) Compute the total MSA for the direction v:
    $\alpha_v = \sum\limits_{c \in \mathcal{C}} a_c$.

3. Select the decomposition that gives the minimum sum of altitudes, i.e., $\alpha^* = \min\limits_{v \in \mathcal{V}} \alpha_v$.

Figure 4.2(a) shows an initial decomposition consisting of ten cells for an environment with four obstacles. The double arrows indicate the optimal service direction. Note that the right cell marked c^o is non-monotone with respect to the direction perpendicular to the optimal service direction. Such directions would have been eliminated in the approach by Bähnemann et al. (2021).

The running time for this step is $\mathcal{O}(n^2 \log n)$, where n is the number of vertices in the environment, and it dictates the overall complexity for cell decomposition.

Greedy Improvement: Improvements to the decomposition have been shown by further decomposition of the cells by splitting them along edges corresponding to a non-convex vertex of a polygon (Huang, 2001; Bochkarev and Smith, 2016; Nielsen et al., 2019). Thus, we apply a greedy strategy to split the cells in the initial decomposition further. We identify the non-convex vertices for a cell and compute a set of splitting lines $\mathcal{L}_s$. There are two types of splitting lines: (1) The set of lines corresponding to an edge adjacent to a non-convex vertex, and (2) The set of lines parallel to an edge of the cell and passing through a non-convex vertex such that both its adjacent edges lie on the same side of the line. We ensure that the splitting lines are neither parallel nor anti-parallel to each other. For each line $l \in \mathcal{L}_s$, we split the cell polygon to obtain a new set of polygons $\mathcal{C}_l$. Now obtain the total MSA for $\mathcal{C}_l$. If this total MSA is less than the MSA for the original cell, then the line l is a valid candidate for splitting. Of all the splitting lines $l \in \mathcal{L}_s$, we greedily select the one that gives the least total MSA after splitting. The new polygons are then recursively improved using the same procedure. The cell marked c^o in Figure 4.2(a) has been split further by a vertical line of Type 2 to create three new cells, shown in Figure 4.2(b). Note that the splitting line does not lie within the *cone of bisection* described by Bochkarev and Smith (2016). As the objective is to reduce the number of turns, the greedy improvement may split a non-monotone cell into monotone cells if doing so reduces the number of turns. We used a greedy approach instead of a dynamic programming approach to reduce computation costs.

Cell Merging: Adjacent cells that have the same (or similar) service directions can be merged to reduce the total number of cells (Oksanen and Visala, 2009; Yu and Hung, 2015). Merging adjacent cells reduces the overlapping regions of sensor coverage. This decreases the number of service tracks and the number of turns. We allow the merging of adjacent cells even when the resulting cell is non-monotone and contains holes; this significantly reduces the number of cells. Figure 4.2(b) shows the final decomposition with eight cells. The total length of the service tracks for the initial decomposition is 2,146 m. The greedy improvement reduces it to 2,103 m, and cell merging reduces it further to 2,003 m, an improvement of 6.7% over the initial decomposition. The most significant reduction comes from merging the cells around the star-shaped obstacle into a non-monotone cell with a hole.

4.3.2 Service Track Generation

The next step is to generate the service tracks for each cell. Since the cells are not required to be monotone polygons with respect to the service direction, we develop a new algorithm to generate the service tracks. Existing approaches usually generate a path, in the form of a boustrophedon or lawn mower pattern, within each cell that a robot must follow. We identify two scenarios wherein a boustrophedon pattern does not guarantee complete coverage, as shown in Figure 4.3. Assuming a square sensor field-of-view, these can happen (1) when an edge is oriented at a very small angle with the service direction, and (2) when an edge is inclined at an angle smaller than $\pi/4$ with the service direction and intersects the service tracks. These scenarios can be extended to other types of sensor field-of-view as well. The second scenario was discussed for disc-shaped sensors by Vandermeulen et al. (2019).

We now discuss a new algorithm, based on the sweep-line algorithm (Berg et al., 2008), for generating the service tracks for a cell obtained from the cell decomposition step. The algorithm can handle non-monotone cells with holes and resolves the issues

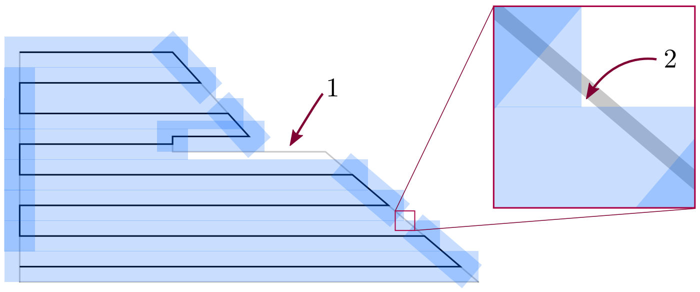


Figure 4.3: Two scenarios, identified by the red arrows, for which a boustrophedon pattern does not always guarantee a complete coverage even when the polygon is monotone with respect to the service direction (horizontal here). The blue shaded region represents coverage of the environment with a square sensor field-of-view.

shown in Figure 4.3. Without loss of generality, we assume the service direction is parallel to the X-axis. The sweep line is parallel to the service direction and sweeps vertically from the lowest to the highest vertex while keeping track of the edges it encounters. The vertices of the edges correspond to the events in the sweep-line algorithm. We assume a square sensor field-of-view for clarity.

Initialization:

- Represent a cell vertex u by its coordinates (u_x, u_y), and represent each cell edge by a pair of vertices (u, v) such that $u_y \leq v_y$, i.e., the lower vertex appears first.

- Let $\mathcal{E}$ be a list of all the edges in the cell. Sort the edges in $\mathcal{E}$ in ascending order by u_y, the y-coordinate of the lower vertex u, for faster computation in Step (2).

- Let $\mathcal{E}_c$ represent the set of current edges used for generating service tracks parallel to the service direction.

- Let $\mathcal{E}_s$ represent the set of special edges of the cell that correspond to scenario 1 shown in Figure 4.3.

- Let $\mathcal{S}$ represent the set of service tracks.

- Initially, the sets $\mathcal{E}_c$, $\mathcal{E}_s$, and $\mathcal{S}$ are all empty.

- Let the sweep line L be the line parallel to the service direction (X-axis) and passing through the lowest vertex $u^l = (u^l_x, u^l_y)$ of the first edge in the sorted list $\mathcal{E}$.

- Set the offset $o = u^l_y + f/2$, where f is the lateral sensor field-of-view.

Iteration: while the edge list $\mathcal{E}$ is not empty

(1) Offset the line L to be at a distance o from the X-axis.

(2) For each edge $e \in \mathcal{E}$ with $u_y \leq o$, remove e from $\mathcal{E}$:

- if $v_y \leq o$, add e to $\mathcal{E}_s$,

- else, add e to the set of current edges $\mathcal{E}_c$. If the edge subtends an angle smaller than $\pi/4$, add e to the list of service tracks $\mathcal{S}$ as well (scenario 2).

(3) Compute the intersections of the line L with the edges in $\mathcal{E}_c$. Sort these intersection points in ascending order of the x-coordinate. Generate service tracks using the intersection points such that the tracks do not overlap with the interior of the obstacles. Add the service tracks to the set $\mathcal{S}$.

(4) For each edge $e \in \mathcal{E}_s$, check if e lies within the field-of-view of any of the service tracks generated in the current and the previous iterations. If not, add e to the set of service tracks $\mathcal{S}$. Remove all edges from $\mathcal{E}_s$.

(5) Offset the sweep line by f and let $o = o + f$.

The set $\mathcal{S}$ gives the set of service tracks for a cell. We compute the service tracks for each cell in the decomposition. Finally, overlapping components of any pair of service tracks for the entire environment are iteratively removed. Figure 4.2(c) shows the generated service tracks. The running time complexity is $\mathcal{O}(n_c\text{MSA}_c)$, where n_c and MSA_c are the number of vertices and the MSA for a cell c, respectively.

4.3.3 Routing

Once the service tracks are computed, the problem can be transformed into the line coverage problem—the coverage of linear features. Here the service tracks correspond to the linear features in the environment. We addressed the line coverage problem and presented an efficient algorithm for multiple capacitated robots in Chapter 3 and Agarwal and Akella (2020). We use a graph as the underlying data structure for the transformation.

Vertices V: The set of vertices V consists of:

- the endpoints of the service tracks;
- the vertices of the environment polygons;
- the depot: A special vertex in the graph from where the routes start and end. For aerial robots, it corresponds to the home location for take off and landing.

Required Edges E_r: A required edge is an edge that needs to be serviced exactly once, and therefore the set of required edges E_r is precisely the set of service tracks. There is a cost and a demand associated with each of the required edges. Furthermore, the edges are considered to have asymmetric costs and demands. Techniques given by Yongguo Mei et al. (2006); Franco and Buttazzo (2015); Cabreira et al. (2018) can be used to obtain demands on battery life. Demands can also be specified in terms of time. Our algorithm can handle arbitrary non-negative input values for costs and demands.

Non-required Edges E_n: We add a non-required edge between each pair of vertices $u, v \in V$ such that the line segment (u, v) does not pass through any of the obstacles and remains within the interior of the outer boundary. We compute a visibility graph to determine if a pair of vertices forms a valid edge for travel. In applications where the robots may travel across the holes, such as aerial robots flying at high altitudes, we do not need to check if the line segment crosses the holes. We may still need to compute the visibility graph as the outer boundary may be non-convex. The robots are not required to traverse the non-required edges. They may, however, use these edges to travel quickly from one vertex to another. A robot is said to be deadheading when traveling along a non-required edge. There is a cost and demand associated with deadheading also. As task-specific actions such as taking images need not be performed, the robots may travel faster than when servicing. Thus, the deadheading costs and demands can differ from those for servicing. Energy-efficient operating speeds for servicing and deadheading can be obtained through experiments (Franco and Buttazzo, 2015); these show that as the speed increases, the power consumption first decreases and then increases rapidly as the speed approaches the upper limit.

Capacity Q: A fixed capacity such as battery life or maximum flight time is specified for the robots. The total demand on the resources accumulated from traversing required and non-required edges along each route should not exceed the capacity Q. This constraint is critical for safe operations, particularly for UAVs, as the battery life can be very limited.

Let $G = (V, E, E_r)$ be the graph created from the area coverage problem, where $E = E_r \cup E_n$ is the set of all the edges. The line coverage problem is then to find a set of routes such that the total cost of travel is minimized and each of the required edges is serviced, while ensuring that the total demand for each route is less than the capacity of the robots, as discussed in Chapter 3. We use the Merge-Embed-Merge (MEM) algorithm to solve the line coverage problem as it is fast and efficient for robotics applications.

The MEM algorithm is composed of four elements: (1) *initialization* of routes, (2) computation of *savings*, (3) *merging* of two routes to form a new route, and (4) *embedding* the new merged route. A route is initialized for each of the required edges (i.e., service tracks) by deadheading from the depot to one of the vertices of the edge, servicing the required edge, and then deadheading back to the depot via the other vertex. Since the costs are asymmetric, the service direction that gives a lower route cost is selected. Two routes can be merged to form a new route with potentially lower cost than the sum of the costs of the two routes. The difference in the costs by performing such a merge is called *savings*. There are eight possible ways of merging two routes, and only the ones that satisfy the capacity constraint are considered. For each pair of initial routes, the optimal savings that respects the capacity constraint is inserted into a max-heap data structure. The routes with the maximum savings are extracted from the max-heap and are merged to form a new route. The new route is inserted into the set of routes, and the individual routes are set to invalid. This new route is then embedded into the max-heap by computing savings with the other valid routes. The merging and the embedding operations are performed iteratively until no further valid merge is possible. Since the capacity constraints are checked before creating a new route, the algorithm always maintains a set of feasible routes. The running time of the algorithm is $\mathcal{O}(m^2 \log m)$, where m is the number of required edges (i.e., service tracks).

Using the line coverage problem, instead of a node routing problem such as the GTSP or the vehicle routing problem (Toth and Vigo, 2014), allows direct modeling of the service tracks as graph edges. More importantly, asymmetric costs and demands for the edges can be modeled in the line coverage problem, along with the capacity constraints. The MEM algorithm for the line coverage problem rapidly computes routes of high quality.

The service track generation and the routing components are independent modules. The modular nature of these components allows making independent improvements. Tracks can be generated by other decomposition methods, and a line coverage algorithm can be used for routing.

4.4 Simulations and Experiments

The algorithms for the area coverage problem are implemented in `C++`. We use the computational geometry algorithms library (CGAL) (The CGAL Project, 2021) for precise numerical computations and geometry functionalities. The program is executed on a desktop with an Intel 7thGen Core i9-7980XE processor.

4.4.1 Dataset with 25 Indoor Environments

Simulation results on a dataset[2] with 25 large indoor environments for vacuuming robots were presented in Vandermeulen et al. (2019). The environments are primarily rectilinear in structure. The robots have a tool width of 0.1 m. We use the path length for both the cost and demand functions for direct comparison. We compute the free workspace by taking a Minkowski sum of the obstacle polygons with the square robot geometry (Berg et al., 2008). The robots must graze the boundaries to vacuum thoroughly. Thus, the entire boundary of the free workspace is added to the set of service tracks. Thereafter, we run the three components of our algorithm to obtain the coverage routes for the robots.

The cumulative lengths and number of turns for the 25 environments are presented in Table 4.1. Figure 4.4 shows our solution with 2 and 4 robots for the largest environment.

4.4.2 Dataset with 300 Outdoor Environments

To benchmark our results for outdoor coverage with aerial robots, we use the dataset (provided with the source code) given by Bähnemann et al. (2021) consisting of 300 unique environments with 1 to 15 holes derived from buildings. The outer boundary has an area of 10,000 m^2, and an aerial robot with a 3 m sided square sensor field-of-view is used. The trajectories of the robots are defined by a velocity ramp model with a maximum acceleration $a_{\max}$ and velocity $v_{\max}$ of $1\,\mathrm{m{\cdot}s^{-2}}$ and $3\,\mathrm{m{\cdot}s^{-1}}$,

[2]The datasets and our detailed results are available at: `https://github.com/UNCCharlotte-CS-Robotics/AreaCoverage-dataset`.

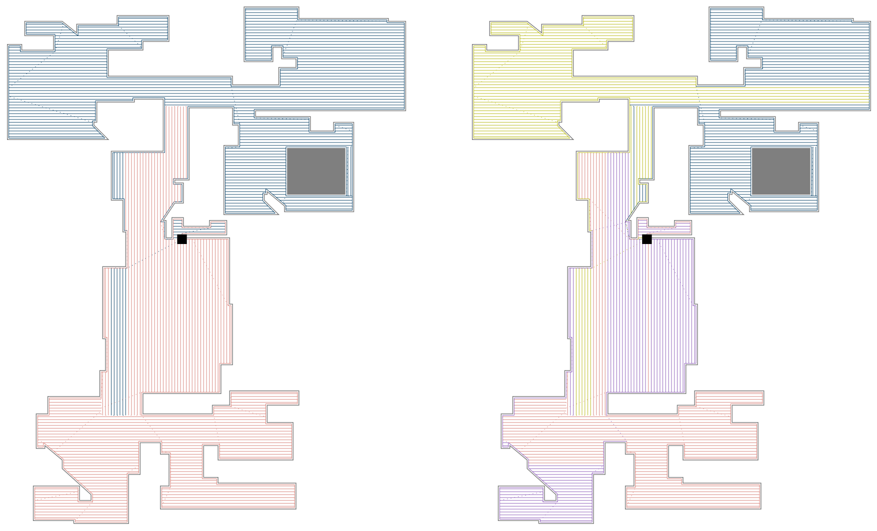


Figure 4.4: Coverage of a large indoor environment with 107 m^2 area and 151 vertices, given in Vandermeulen et al. (2019). The left figure shows our solution for two robots with a tool width of 0.1 m and a capacity of 700 m, with tour costs 616 m and 620 m. The right figure shows our solution for four robots with a capacity of 320 m, with tour costs 312 m, 313 m, 318 m, and 300 m.

respectively. The travel time t is used as the cost and demand functions and is given in terms of segment length d as:

$$t = \begin{cases} \sqrt{\frac{4d}{a_{\max}}}, & \text{if } d < d_a \\ \frac{v_{\max}}{a_{\max}} + \frac{d}{v_{\max}}, & \text{if } d \geq d_a \end{cases}, \text{ where } d_a = \frac{v_{\max}^2}{a_{\max}}$$

We ran the simulations for two scenarios: (1) infinite capacity, representing coverage with a single robot, and (2) capacity of the robots set to 20 minutes (1,200 s). The comparison of the total cost of the routes is shown in Figure 4.5. Our algorithm generates lower cost solutions than that of Bähnemann et al. (2021) for both single and multiple robots and for all the instances, with an average improvement of 10% and a standard deviation of 4%. The total cost for the multiple robot solutions is the sum of the costs of the individual routes, and the solutions are better than Bähnemann et al. (2021), even though a limited battery capacity reduces the feasible space considerably. The computation time is shown in Figure 4.6, and is similar for both single and

Table 4.1: Cumulative results for the 25 indoor environments dataset

r	Capacity	Vandermeulen et al. (2019)		This chapter		Improvement (%)	
		l (m)	η	l (m)	η	l	η
1	∞	15,195	11,377	14,781	10,183	2.72	10.49
2	0.75	15,303	11,380	14,793	10,191	3.33	10.45
3	0.50	15,461	11,533	14,823	10,211	4.13	11.46
4	0.30	15,564	11,586	14,939	10,274	4.02	11.32
5	0.25	15,715	11,663	15,030	10,308	4.36	11.62

The first column r indicates the number of robots. The length and the number of turns are denoted by l and η, respectively. The capacity is set as a fraction of the route cost for a single robot. The average computation time is 0.42 s, over all 25 environments and over 100 runs. The computation time does not vary much with the number of robots.

multiple robots; the only difference is in the running time of the MEM algorithm, which converges faster as the capacity decreases.

4.4.3 Outdoor Experiment with Aerial Robots

We selected a 19,000 m^2 area in the UNC Charlotte campus, shown in Figure 4.7, for coverage with UAVs. An appropriate launch site was assigned as a depot. A subregion corresponding to the footprint of a building was selected as a hole, consisting of 45 vertices, in the environment. As aerial robots can fly at high altitudes, we allow non-required edges that cross the hole. The servicing and deadheading speeds were set to $3.33\,\text{m}\cdot\text{s}^{-1}$ and $5\,\text{m}\cdot\text{s}^{-1}$, respectively. A wind of $1.39\,\text{m}\cdot\text{s}^{-1}$ at an angle of 225 degrees (from NE), for the day of the experiment, was incorporated into the cost and demand functions, making the edges asymmetric in the two directions of travel. The costs and the demands are based on the edge travel times. A conservative capacity of 600 s was set, and two routes, computed in 2.4 s, were obtained. A DJI Phantom 4 drone was used to autonomously fly the two routes sequentially. The routes and the orthomosaic obtained from the collected images are shown in Figure 4.7. The computed costs for the routes, shown in blue and red in Figure 4.7, were 336 s and 394 s, and the flight times were 317 s and 369 s.

Discussion: In our simulations and experiments, we considered three types of sce-

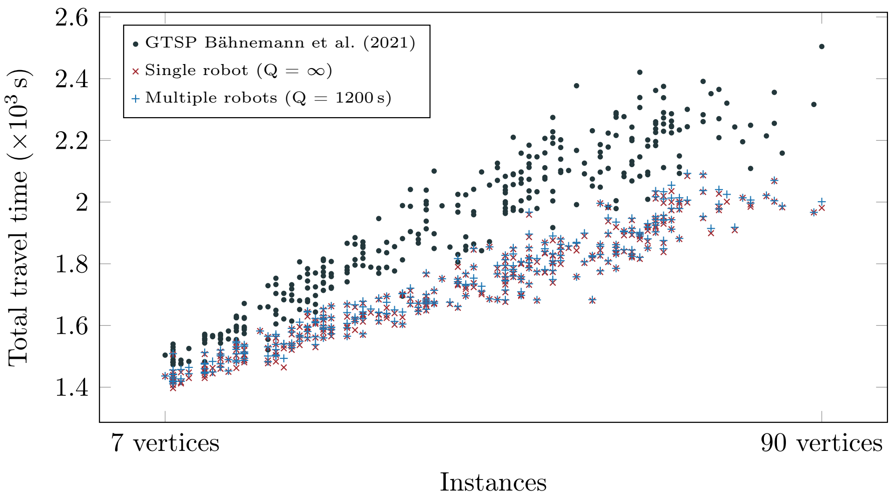


Figure 4.5: Comparison of total travel time cost of routes computed by our algorithm and the GTSP based algorithm given by Bähnemann et al. (2021). The instances are arranged in increasing order of the number of vertices in the environments. We compute the results for two cases: (1) Single robot with infinite capacity (red crosses), and (2) Multiple robots with a capacity of 1,200 s (blue pluses). The sum of costs for a single robot over all 300 instances is 577,218 s for Bähnemann et al. (2021) and 512,619 s for our method. Our algorithm consistently performs better with an average cost reduction of 10%.

narios: (1) The ground robots of finite size cannot intersect with the obstacles, (2) The aerial robots are not permitted to fly over obstacles, and (3) The aerial robots can fly over obstacles. Using a visibility graph, we can address any combination of the above scenarios by permitting non-required edges only over the obstacles that the robots can traverse. We can also compute the Minkowski sum for the obstacles that a finite-sized robot is not allowed to overlap. Furthermore, non-overlapping disconnected regions of environments can also be addressed by performing cell decomposition and service track generation for each such region individually and computing routes using the MEM algorithm for the service tracks in a unified manner.

4.4.3.1 Area Coverage with Nonholonomic Robots

We now illustrate the use of the algorithms developed for the line coverage problem to generate routes for area coverage with nonholonomic robots. The development of algorithms for multiple depots and nonholonomic constraints for the line coverage

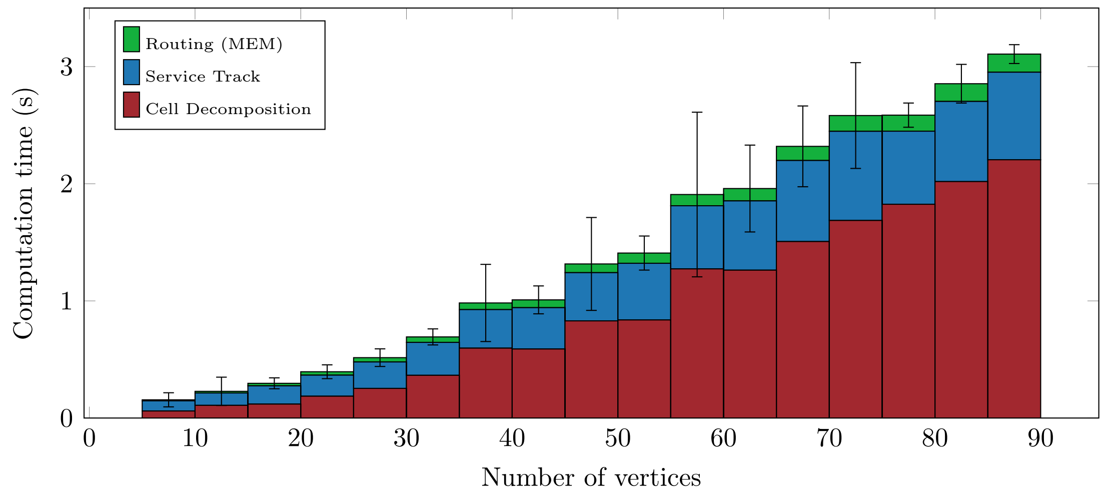


Figure 4.6: Average computation times for the 300 outdoor environment dataset from Bähnemann et al. (2021); the route costs are shown in Figure 4.5. Each environment is placed in bins of size five according to the number of vertices in the environment. The time is averaged over 100 runs for all the environments in a bin. The bars indicate the standard deviation of the computation time. The computation time increases with the number of vertices in the environment. The cell decomposition is the most time-consuming step of the algorithm, while the MEM routing algorithm is very fast.

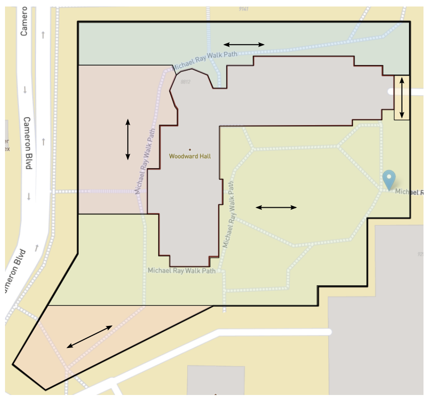
problem extends to area coverage using the transformation developed in this chapter. We illustrate the use of the MEM algorithm with multiple depots and nonholonomic robots in the same outdoor environment setting given in Figure 4.7. As UAVs can fly at high altitudes, we allow non-required edges that cross the building. However, only the region surrounding the building needs to be covered. Figure 4.8(a) shows the generated service tracks, taking the field-of-view of the sensor into account. These service tracks form the linear features, i.e., the required edges, for the line coverage problem. Two depot locations are computed using k-medoids clustering. Dubins curves are used to deadhead between pairs of non-adjacent required edges, and smooth turns are used to deadheadhead between adjacent required edges. Figure 4.8(b) shows the two routes computed using the MEM algorithm with multiple depots and nonholonomic constraints. The example illustrates the use of line coverage algorithms for area coverage, and shows that the enhancements to algorithms for line coverage translate directly to area coverage.

4.5 Summary

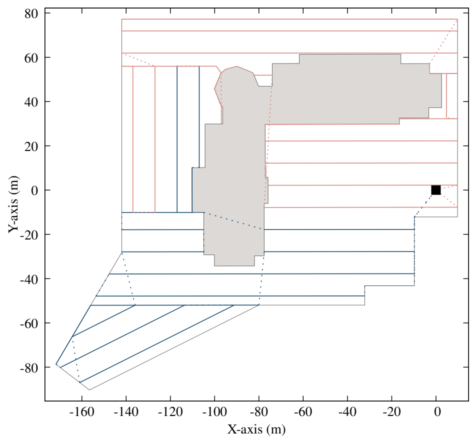
We presented a novel approach for solving the area coverage problem with multiple capacity-constrained robots by transforming it into the line coverage problem. This allowed us to generate routes that minimize the total cost of travel while respecting the capacity constraints. The formulation enables two modes of travel—servicing and deadheading—with distinct and asymmetric costs and demands that can have arbitrary non-negative values. Travel time, travel length, or battery consumption can be used to model costs and demands. A depot, from where the robots start and end their routes, can be specified. These features were demonstrated in an outdoor experiment using a commercial UAV.

The cell decomposition permits non-monotone polygons, thus increasing the feasible solution space for the service directions to further minimize the number of turns. Allowing non-monotone polygons with holes enables further merging of adjacent cells. A new service track generation algorithm generates tracks for non-monotone polygons with or without holes. We benchmarked the approach on a ground robot dataset with 25 indoor environments and an aerial robot dataset with 300 environments, with an average cost improvement of 10%.

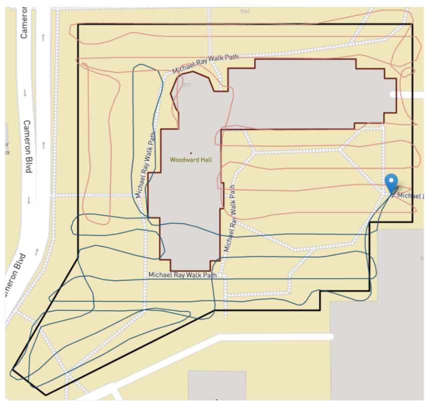
Since we establish that the cells from cell decomposition are no longer required to be monotone, our work raises the following questions: Is there a better strategy for cell decomposition to minimize the number of turns? Is a polynomial-time optimal algorithm possible, or is this cell decomposition problem NP-hard?



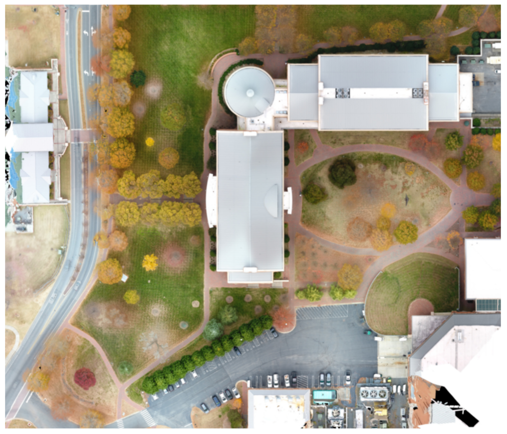
(a) Input environment with cells



(b) Computed routes

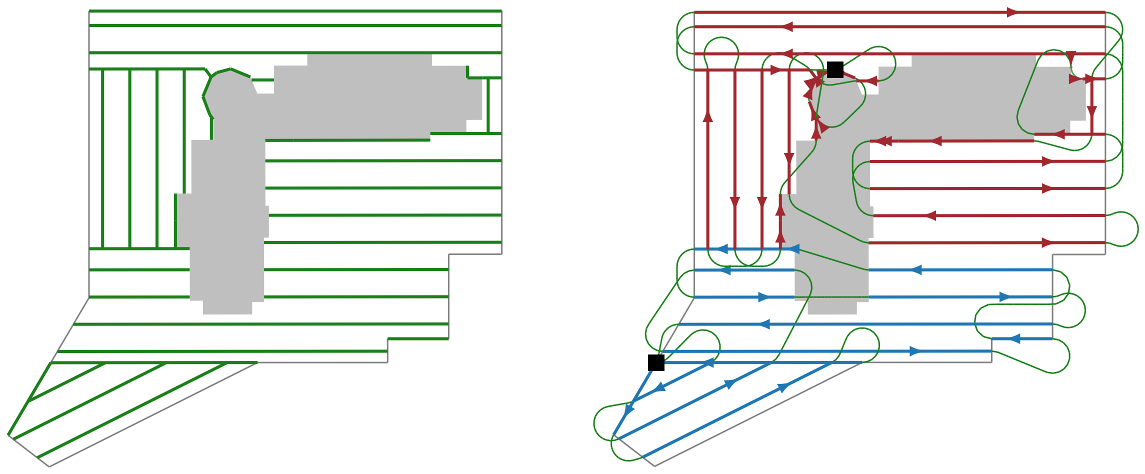


(c) Actual flight paths



(d) Orthomosaic from images

Figure 4.7: Area coverage by autonomous aerial robots: (a) The region surrounding a building is to be covered. The blue marker indicates the depot location for the robots to take off and land. The cell decomposition is shown with double head arrows indicating service directions. Note that the cells are non-monotone. (b) The generated routes for the aerial robots, distinguished by color. The dashed lines correspond to deadheading travel. Here deadheading is permitted over the building for efficiency. (c) The actual paths taken by the aerial robots. (d) Orthomosaic generated from images taken during the flights.



(a) Service tracks (b) Routes with nonholonomic robots

Figure 4.8: Area coverage using nonholonomic aerial robots for the environment given in Figure 4.7: (a) Service tracks are generated for each cell independently. The tracks are parallel to the service directions obtained from cell decomposition. These tracks form the linear features for the line coverage problem. (b) Two routes computed by the MEM algorithm for multiple depots and nonholonomic robots. The green lines indicate deadheading and comprise of Dubins curves between non-adjacent tracks and smooth turns between adjacent tracks.

CHAPTER 5: GENERALIZED COVERAGE

This chapter introduces the generalized coverage problem—the task of finding routes to simultaneously cover point, line, and area features in an environment by a team of resource-constrained robots. Current approaches for coverage problems focus on only one type of feature at a time. However, several applications may require inspection of multiple feature types in the environment. One such application arises in the inspection of traffic, where traffic signal intersections, the road network, and parking lots form the point, line, and area features, respectively. Similarly, the inspection of electricity poles along with power lines and power stations includes all three types of features. We introduce, for the first time, the ability to cover all the three types of environment features in an integrated manner. Since the coverage of any one of these types of features is an NP-hard problem, the generalized coverage problem is also NP-hard. Hence, we develop heuristic algorithms that generate high-quality solutions and have a polynomial-time computational complexity. The central idea of our approach is to transform point and area features into line features. The coverage of line features is formulated as the line coverage problem, which was addressed in Chapter 3. We use the Merge-Embed-Merge algorithm to obtain efficient routes for coverage of the transformed line features. The transformation allows translation of the advances in the line coverage problem to the generalized coverage problem. In particular, the ability to model the following aspects of the problem is achieved: two modes of travel, direction-dependent costs and demands, resource constraints, multiple depots, and nonholonomic robots. We illustrate generalized coverage of five university campuses for traffic analysis.

5.1 Introduction

Generalized coverage is the task of servicing features in an environment using sensors or tools mounted on robots. The features to be serviced can be modeled as zero-dimensional points, one-dimensional lines or curves, and two-dimensional areas. The environment itself may be planar or three-dimensional. Consider an application scenario of traffic analysis during a social event. A team of uncrewed aerial vehicles (UAVs) is deployed to gather information on bottlenecks in the traffic. We may be interested in visiting the traffic signals and roundabouts, the road network, and the parking lots, as shown in Figure 5.1. These three features can be modeled as points, line segments, and areas. The routes for the UAVs must be planned such that the different types of features are considered simultaneously. In contrast, most current approaches consider one type of feature at a time. Similar scenarios arise in the inspection of electricity poles together with power lines and power stations.

The central idea of our approach for solving the novel generalized coverage problem is to transform the point and area features into line features. Together, these transformed features and the natural line features form a new set of global line features. The environment is modeled as a graph with the global line features forming the set of *required edges*. The vertices of the graph are formed by the endpoints of the required edges. Additionally, the graph contains a set of *non-required edges* that model traversal between vertices to connect the required edges. Using the graph, we transform an instance of the generalized coverage to an instance of the line coverage problem. Finally, we can use the algorithms developed for the line coverage problem in Chapter 3, such as the Merge-Embed-Merge (MEM) algorithm, to obtain efficient routes for the robots for the transformed generalized coverage problem.

The transformation allows us to incorporate several practical aspects of robotics addressed by the line coverage problem.

1. **Cost minimization:** When a robot traverses the environment, it incurs a *cost*

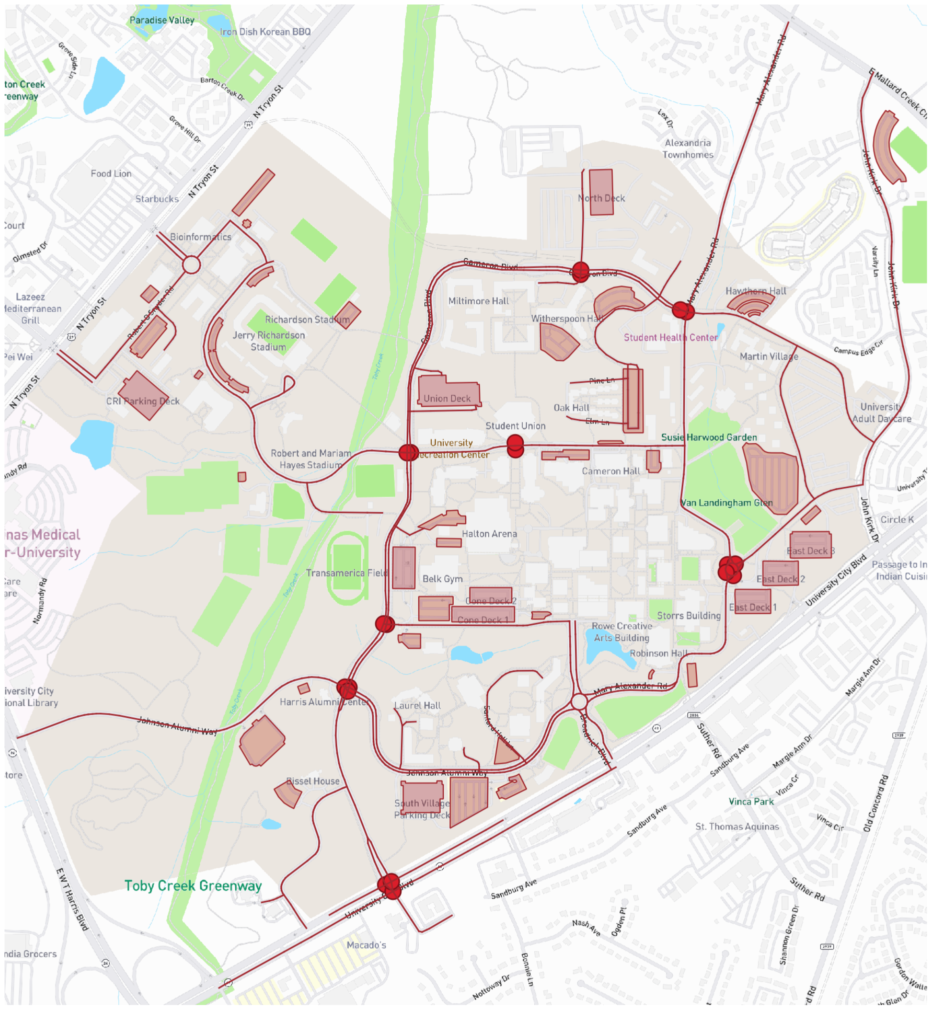


Figure 5.1: Input environment for generalized coverage of the UNC Charlotte campus, obtained from OpenStreetMap: The 20 traffic signals on the campus are shown as red circles and form the point features. The road segments, shown as solid red lines, form the line features. There are 1,160 road segments covering the entire primary road network of the campus. The 44 parking lots in the campus, shown as red shaded regions, form the area features.

such as travel time. The objective of a coverage problem is to minimize the total cost of all the routes of the robots while ensuring all the features are visited.

2. **Resource constraints:** Mobile robots can be severely limited in terms of the resources available to them. Aerial robots, in particular, have a short operation time due to low battery capacity and high consumption of energy. It is, therefore, important that route planning algorithms take resource constraints into account to ensure the safe recovery of the robots. As robots traverse the environment, they incur *demands* on the resources available to them, and the total demand of traversing a planned route for a robot should be less than its available resources. Such a limit on a resource is also referred to as the *capacity* of the robot.

3. **Two travel modes:** We allow two travel modes—servicing and deadheading. A robot is said to be *servicing* an environment feature when it performs task-specific actions such as collecting sensor data; otherwise, it is *deadheading.* Our formulation models different cost functions and resource demands for the two modes of travel for the robots. These modes enable the algorithms to optimize the operation time, conserve energy, and reduce the amount of sensor data.

4. **Asymmetry in costs and demands:** In many robotics applications, the cost of travel and the resource demands are direction-dependent. For example, a ground robot traveling uphill can take longer and consume more energy than traveling downhill. Similarly, the costs and demands of aerial robots may depend on the direction of travel due to wind conditions. Hence, we consider asymmetric cost and demand functions for servicing and deadheading. Such asymmetric functions can also model one-way streets for ground robots.

5. **Turning costs and nonholonomic constraints:** Sharp turns can be very expensive for robots as they need to slow down, take the turn, and accelerate

again. Similarly, nonholonomic robots such as fixed-wing UAVs cannot make point turns. It is imperative to account for the turning costs and nonholonomic constraints to ensure efficient and feasible navigation of the robots.

6. **Multiple depots:** A *depot* is a location in the environment from where the robots start and end their routes. When the environment is vast, it may not be possible to service all the features from a single depot location. In such situations, it is imperative to have a multi-depot formulation where the robots have the flexibility to start and end their routes from one of several depots to optimize the routes.

Another practical aspect of deploying multiple robots simultaneously is collision avoidance, i.e., ensuring that the distance between each pair of robots is greater than a safe distance. Although we do not consider collision avoidance explicitly, existing methods, such as those based on control barrier functions and reciprocal velocities, can be conveniently used for locally changing the robot paths to avoid collisions.

We make the following contributions in this chapter:

1. We introduce a novel generalized coverage problem, which considers three different types of features in the environment. Such a problem has not been addressed before.

2. We develop procedures to transform point and area features into line features, thereby transforming generalized coverage into line coverage.

3. We illustrate the application of generalized coverage to traffic analysis on five university campuses.

We describe the problem formally in Section 5.2 and the solution approach in Section 5.3. The simulation results for five university campuses are given in Section 5.4. The chapter is summarized in Section 5.5.

5.2 Problem Statement

In this section, we formally define the generalized coverage problem. We provide the following environment features as input to the problem:

1. **Point features V_p:** Each element v in the set of point features V_p corresponds to a point in the environment that needs to be serviced by a robot.

2. **Line features E_l:** Each line feature $e \in E_l$ is modeled by two points v_1 and v_2 in the environment, and is denoted by $e = (v_1, v_2)$. The robots must service these line features by traversing along the edge. Curves can also be modeled as line features.

3. **Area features R_a:** The two-dimensional features are given by the set R_a. An area feature $R \in R_a$ is modeled as a polygon-with-holes (PWH), where the outer boundary of R is specified as a sequence of points in the counter-clockwise direction. Subregions within the outer boundary that are not required to be visited are specified as holes with a sequence of points in the clockwise direction.

We have the following cost and demand functions, along with the capacity of robots:

1. **Cost functions:** For each point feature $v \in V_p$, we are given a cost $c_s(v)$ to service the particular vertex. The cost of servicing an edge between two points v_1 and v_2, in the environment in the direction from v_1 to v_2, is given by $c_s(v_1, v_2)$. Similar to service costs, we have deadhead costs, denoted by $c_d(v_1, v_2)$, and $c_d(v_2, v_1)$ for deadheading between two points v_1 and v_2 in the two directions. Functions such as Euclidean distance or travel time can conveniently model these cost functions. Since we have distinct functions for servicing and deadheading, different travel speeds can be selected for the two modes.

2. **Demand functions:** Traversing the environment incurs consumption of resources such as battery life, modeled as demands. For each point feature $v \in V_p$,

we are given a demand on resources $q_s(v)$ for servicing the vertex v. The demand on resources for servicing an edge between two points v_1 and v_2 in the environment is denoted by $q_s(v_1, v_2)$. Similarly, we have demand on resources for deadheading an edge between two points v_1 and v_2, denoted by $q_d(v_1, v_2)$. The service and the deadheading demands are distinct and direction-dependent; hence, $q_s(v_1, v_2)$, $q_s(v_2, v_1)$, $q_d(v_1, v_2)$, and $q_d(v_2, v_1)$ may all differ.

3. **Capacity of the robots Q:** The total demand on resources for each route for a robot must be within the specified capacity Q of the robot. The capacity models the resource constraint on the robots and may be specified in terms of total battery life or maximum operation time.

Definition 5.2.1 (The Generalized Coverage Problem)**.** Given a set of point features V_p, a set of line features E_l, and a set of area features R_a, along with cost functions, demand functions, and a capacity Q for the robots, the generalized coverage problem is the task of finding a set of routes for a team of robots such that the robots service all the three sets of required features. The objective of the problem is to minimize the total cost of the routes under the constraint that the demand for each route should be within the capacity Q of a robot.

5.3 Solution Approach

We now present our approach for transforming the point and area features into line features. After the transformation, we formulate the generalized coverage problem as a line coverage problem. The overall approach is illustrated in Figure 5.2.

5.3.1 Transformation of Point Features to Linear Features

The transformation of point features to line features is straightforward—for each vertex v in the set of point features V_p, we create a new auxiliary line feature represented as an edge $e_v = (v, v)$. The cost and the demand of servicing such a new edge correspond to that of the point feature. Since there is no deadheading for a

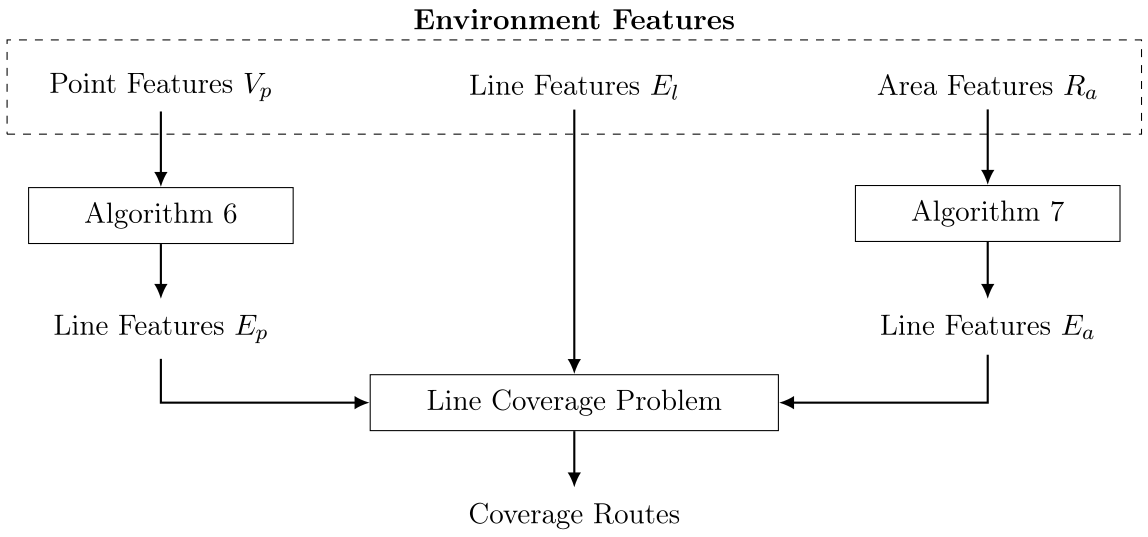


Figure 5.2: Overall approach for computing routes for generalized coverage: The point and area features are transformed to line features, and the coverage routes are computed using the MEM algorithm for the line coverage problem.

point, we set the deadhead costs and demands to zero. The transformation is given in Algorithm 6.

```
Algorithm 6: Transform point features
  Input  : Point features $V_r$
  Output : Transformed line features $E_p$
1 $E_p \leftarrow \emptyset$;
2 for $v \in V_r$ do                                  // Iterate over point features
3     $e_v \leftarrow (v, v)$;                          // Create a new edge
4     $c_s(e_v) \leftarrow c_s(v)$; $c_d(e_v) \leftarrow 0$;   // Assign service and deadhead costs
5     $q_s(e_v) \leftarrow q_s(v)$; $q_d(e_v) \leftarrow 0$;   // Assign service and deadhead demands
6     $E_p \leftarrow E_p \cup e_v$;                    // Add new edge
```

5.3.2 Transformation of Area Features to Linear Features

The area features R_a are modeled as PWHs. For each required area $R \in R_a$, we perform cell decomposition and service track generation, using the computational geometry algorithms described in Chapter 4. The cell decomposition procedure splits the regions into cells and finds the optimal service direction for each cell with the aim of minimizing the total number of turns for the robots. The service track generation procedure uses the sensor field of view to discretize each cell into a set of line segments, using a sweep-line algorithm. These line segments are parallel to the service direction computed for the cells in the cell decomposition procedure. The set of line segments computed for each area feature forms the set of transformed line features E_a. The transformation is given in Algorithm 7.

Algorithm 7: Transform area features

```
Input  : Area features $R_a$
Output : Transformed line features $E_a$
1 $E_a \leftarrow \emptyset$;
2 for $R \in R_a$ do                                   // Iterate over area features
3     $\mathcal{C} \leftarrow$Cell-decomposition($R$);    // Compute cells with service directions
4     for $c \in \mathcal{C}$ do                       // Iterate over cells
5         $E_a \leftarrow E_a \cup$ Service-tracks($c$);   // Generate service tracks
```

5.3.3 Formulation of the Line Coverage Problem

The line coverage problem with multiple resource-constrained robots requires a graph as an input, as described in Chapter 3.

1. Required edges E_r: The set of line features E_l, the transformed point features E_p, and the transformed area features E_a form the set of required edges, i.e., $E_r = E_p \cup E_l \cup E_a$.

2. Vertices V: The endpoints of the required edges form the set of vertices V. We may also have potential depot locations as additional vertices.

3. Non-required edges E_n: The robots may travel from one vertex to another without servicing. Any such travel between a pair of vertices forms a non-required edge. We use a visibility graph to generate the set of non-required edges when there are obstacles or no-fly zones in the environment. These edges are essential to ensure that the graph is connected and the robots can reach each feature in the environment.

4. Graph $G = (V, E, E_r)$: The input graph for the line coverage problem can now

be formed using the set of vertices V, the set of required edges E_r, and the set of non-required edges E_n. The edge set E is the union of the sets of required and non-required edges, i.e., $E = E_r \cup E_n$.

With the graph G as the input, along with the cost and demand functions, we can use the algorithms developed in Chapter 3 to solve the generalized coverage problem. Since the environments are often large, we use the Merge-Embed-Merge algorithm with multiple depot (MEM-MD) locations to solve the generalized coverage problem.

5.4 Simulations on University Campuses

We illustrate generalized coverage for traffic analysis on five university campuses belonging to the University of North Carolina system. The traffic signals form the point features, line segments representing the road network form the line features, and the parking lots are the area features. The data was obtained from OpenStreetMap contributors (2022). These features are transformed into line features, and a graph for the line coverage problem is computed. As the UAVs can fly from one vertex to another, we add a non-required edge between each pair of non-adjacent vertices. The number of different types of features and the number of elements in the line coverage graph for the five campuses are given in Table 5.1.

Table 5.1: Dataset for evaluation of the generalized coverage formulation

UNC Campus	Number of features			Line coverage graph		
	Point	Line	Area	$\lvert V \rvert$	$\lvert E_r \rvert$	$\lvert E_n \rvert$
Greensboro	4	356	9	543	499	146,673
Asheville	0	266	72	780	647	303,183
Wilmington	2	598	35	1,097	891	600,276
Charlotte	20	1,160	44	1,817	1,600	1,648,273
Chapel Hill	14	1,204	70	2,153	1,926	2,314,791

We used k-medoids clustering to obtain depot locations for the environments. Alternatively, these locations can be specified based on operation ease and constraints, e.g., an operator may prefer to launch aerial robots from high vantage points. The service and deadhead speeds were set to $5\,\mathrm{m{\cdot}s^{-1}}$ and $8\,\mathrm{m{\cdot}s^{-1}}$. The flight time for the robots was limited to 1,200 s. The algorithms are implemented in C++ and executed on a single core of a standard laptop with an Intel Core i7-1195G7 processor. The k-medoids clustering, used for computing the depot locations, depends on a random number generator. Hence, we compute the routes 10 times and select the solution with the least number of routes, and ties are broken based on the lower cost.

Figure 5.3 shows the input graph and routes for a smaller campus (UNC Greensboro). Figure 5.4 shows the routes for the input graph in Figure 5.1 of the UNC Charlotte campus. The routes show efficient traversal of service tracks generated for the area features. It can be observed that the individual routes primarily consist of line segments that are close to each other and are connected, indicating that the MEM algorithm can efficiently distribute the line features among routes. Furthermore, the routes are assigned to the closest depot, showing that the multiple depot formulation of the MEM algorithm is well-suited for the generalized coverage problem with large graphs. The details of the routes for all five campuses are shown in Table 5.2. The algorithm generates routes for the largest graph within 6 minutes for 10 executions with random seeds. The last column DH in the table is the percentage of deadhead cost in the total cost of the routes. It indicates the amount of travel time spent in traversal between the required edges of the graph to maintain connectivity and can be viewed as a measure of efficacy of the solutions. Even though the features in the environments span over large regions, the percentage of deadheading is low.

5.5 Summary

In this chapter, we proposed a novel generalized coverage problem—the coverage of point, line, and area features in an environment using a team of resource-constrained robots. This problem arises in applications in coverage problems where multiple types of features require inspection. The central idea of our approach is to transform the

Table 5.2: Results for generalized coverage of the five campus dataset

UNC Campus	Length (m)	Area (km^2)	Depots	Cost (s)	Routes	Compute time (s)	DH (%)
Greensboro	12,501	1.12	3	2,953	4	5	15
Asheville	15,082	1.10	4	3,684	4	13	18
Wilmington	39,092	5.03	8	9,367	9	39	17
Charlotte	39,048	4.82	9	9,495	10	208	18
Chapel Hill	42,744	5.33	10	10,819	11	341	21

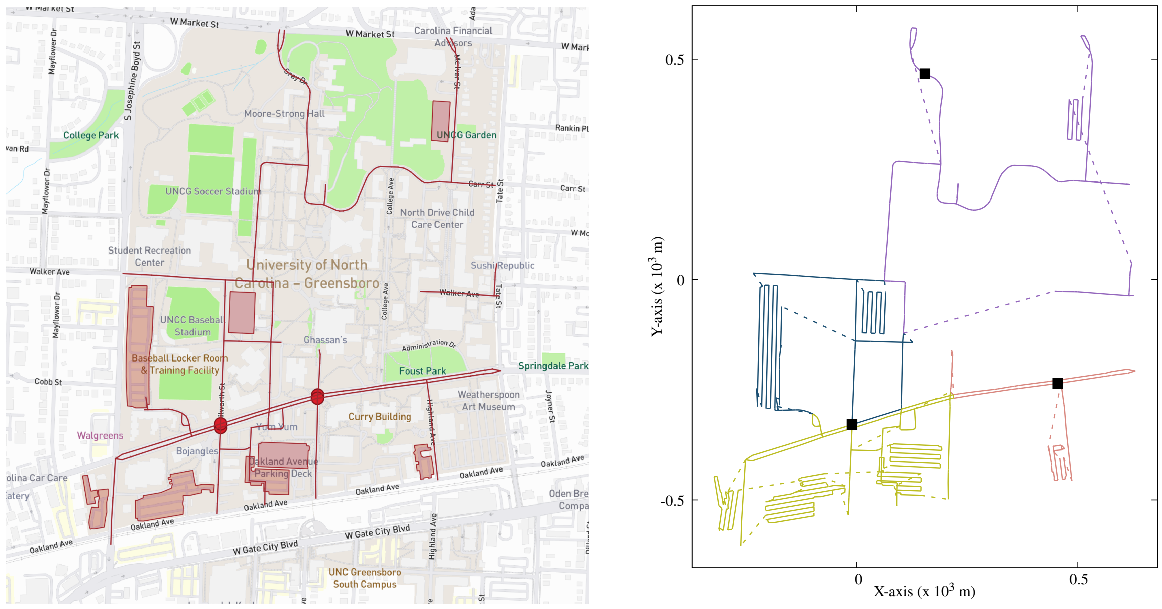


Figure 5.3: Generalized coverage of the UNC Greensboro main campus. The campus includes 4 traffic signals, 356 line segments for the road network, and 9 parking lots. Three depot locations, shown by black squares, were computed using clustering. The routes were computed using the MEM algorithm with multiple depots. Four routes were obtained and are shown in different colors. The solid lines represent servicing and the dashed lines represent deadheading.

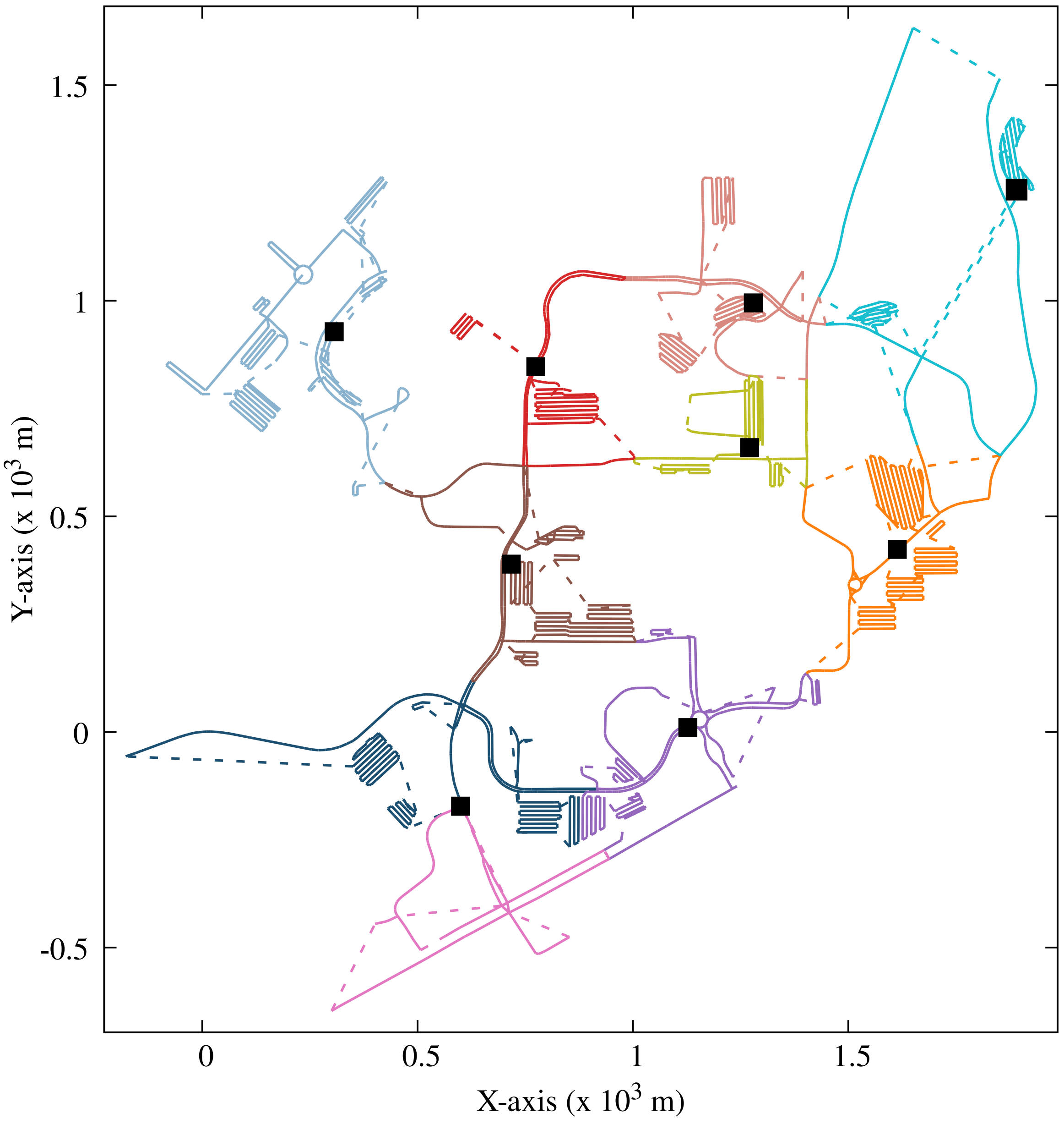


Figure 5.4: Routes for generalized coverage of the UNC Charlotte campus: The final line coverage graph has 1,817 vertices, 1,600 line features, and more than 1.6 million non-required edges. Since the environment is very large, we compute 9 depot locations as shown by black squares. The MEM algorithm with multi-depot formulation computed 10 routes to cover all the features. The routes are shown in different colors; the solid lines represent servicing, and the dashed lines represent deadheading.

point and area features into line features. The point features were transformed by creating auxiliary edges, and area features were transformed using cell decomposition and service track generation. The transformation enables formulating the generalized coverage problem as a line coverage problem. We used the Merge-Embed-Merge (MEM) algorithm with multiple depots to generate efficient routes for generalized coverage. The formulation allowed the translation of the advances in the line coverage problem (described in Chapter 3) to the generalized coverage problem. The MEM algorithm supports two modes of travel, direction-dependent costs and demands, resource constraints, multiple depots, and nonholonomic robots. Hence, these algorithm features translate to the generalized coverage problem seamlessly.

We illustrated the coverage of road networks with traffic signals and parking lots on five university campuses using UAVs. The MEM algorithm computed efficient routes for the largest graph with more than 1,900 required edges within a few minutes. The results indicate that the generalized coverage problem is well-suited for modeling and solving coverage problems with different types of features in the environment. Furthermore, our approach of transforming an instance of the generalized coverage problem to an instance of the line coverage problem enables computation of efficient routes while addressing practical aspects of robot deployment.

CHAPTER 6: CONCLUSION

This dissertation develops theory and algorithms for efficient coverage of an environment using a team of resource-constrained robots and validates them through simulations and experiments. The research unifies coverage of point, line, and area features into a novel generalized coverage framework, formalized as optimization problems on graphs. Most current approaches consider only one type of feature at a time. However, this can be limiting when applications have two or more types of features. One such application is traffic analysis, where the traffic signals, road network, and parking lots form the point, line, and area features, respectively. A similar scenario arises in the inspection of electric poles along with power lines and power stations. In the generalized coverage problem, the task is to compute efficient routes for the resource-constrained robots to cover all the features in the environment.

The research incorporates several practical aspects of robot deployment in the generalized coverage framework. These aspects are essential for ensuring efficient and safe routes for robots.

- **Cost models:** Functions, such as travel times, are used to define the cost model and characterize the efficiency of the routes computed using the algorithm. We formulated the objective of the coverage problems as minimization of the total cost over all the routes.

- **Resource constraints:** Mobile robots can be severely limited in terms of resources available to them. Aerial robots, in particular, have a short operation time due to low battery capacity and high consumption of energy. Therefore, we incorporated resource constraints in the route planning algorithms to ensure the safe recovery of robots. The consumption of resources as robots traverse the environment is modeled as demands. The algorithms ensure that the total demand incurred in a route is within the specified resource limit of robots.

- **Two modes of travel:** A unique characteristic of our formulation is that we allow two modes of travel for the robots—servicing and deadheading. A robot is said to be *servicing* an environment feature when it performs task-specific actions such as collecting sensor data; otherwise, it is *deadheading*. In our formulation, we modeled different cost functions and resource demands for the two modes of travel for the robots. These modes enabled the algorithms to optimize the operation time, conserve energy, and reduce the amount of sensor data.

- **Asymmetry in costs and demands:** The cost of travel and the demand on resources can be direction dependent. For example, a ground robot traveling uphill can take longer and consume more energy than when traveling downhill. Similarly, the cost and the demand functions of aerial robots may depend on the direction of travel due to wind conditions. We incorporated such asymmetries in the formulation and modeled wind conditions in the cost and demand functions. Asymmetric functions can also model one-way streets for ground robots.

- **Turning costs and nonholonomic constraints:** Sharp turns can be very expensive for robots as they need to slow down, take the turn, and accelerate again. Similarly, nonholonomic robots such as fixed-wing UAVs cannot make point turns. Thus, the formulations and the algorithms account for the turning costs and nonholonomic constraints to ensure efficient and feasible navigation of the robots.

- **Multiple depots:** When the environment is vast, it may not be possible to service all the features from a single depot location. Therefore, we developed the multi-depot formulation, which can handle large-scale environments so that robots have the flexibility to start and end their routes at one of several depots.

6.1 Contributions

The coverage problems are classified into three types based on the feature type. The dissertation first comprehensively addressed the single robot line coverage problem, i.e., coverage of one-dimensional features using a single robot, in Chapter 2. This problem has not been sufficiently studied in robotics. We modeled the environment as a graph and posed line coverage as an optimization problem using an integer linear program (ILP). The formalization gave insights into the inherent structure of the problem, using which we developed approximation algorithms with provable guarantees on the quality of the solutions. The algorithms provide the best approximation factor for the single robot line coverage problem and related arc routing problems (ARPs). We evaluated the algorithms on a dataset of 50 road networks from the most populous cities in the world. The results show that our algorithm computes high-quality solutions that are within 10% of the optimum in less than 3 s. The algorithms are sufficiently fast for robotics applications. Experiments with a commercial aerial robot, DJI Phantom 4, were performed on a portion of the UNC Charlotte road network and on lanes of a set of parking lots. As the images were collected only during servicing and not while deadheading, a smaller number of images of only the features of interest were collected. Orthomosaic maps were generated using these images.

Next, we addressed the line coverage problem with multiple resource-constrained robots in Chapter 3. The problem was formulated as an optimization problem on graphs using ILP while incorporating the resource constraints on the robots. We designed a heuristic algorithm, Merge-Embed-Merge (MEM), which has a polynomial-time complexity of $\mathcal{O}(m^2 \log m)$, where m is the number of line features. The algorithm maintains a set of feasible routes and iteratively merges pairs of routes to form new larger ones. Eight possible ways of merging two routes dictate the savings in merging routes. The algorithm is constructive in the sense that new routes are constructed as the algorithm progresses, in contrast to other graph procedures where a digraph is used as a proxy for the routes. The constructive nature of the algorithm allows us to extend the algorithm to several variants of the line coverage problem. We formulated the line coverage problem with multiple depots and extended the MEM algorithm to solve the problem. MEM is the first fast and efficient algorithm for the line coverage problem with multiple depots and applies to related arc routing problems commonly used for human-driven vehicles. We further incorporated smooth turns and nonholonomic constraints into the algorithm by including costs and demands on resources due to the turns. We evaluated the MEM algorithm on the 50 road network dataset, which indicates that the MEM algorithm computes high-quality solutions with costs within 7% of the costs of ILP solutions. The MEM algorithm also performs similarly when the capacity of the robots is varied. The evaluation of the computation times shows that the algorithm is very fast as it solves the largest graph within 0.5 s, and the entire procedure, including creating graphs, takes around 2 s. For graphs with less than 200 required edges, the solutions are generated within 0.1 s. We demonstrated the algorithm in experiments with a UAV on the UNC Charlotte road network. For the first experiment, two routes from a single depot location were autonomously executed by a commercial UAV to cover a portion of the road network. In the second experiment, we used the formulation for the line coverage problem with multiple depots to obtain eight depots and routes for the road network. These routes were autonomously executed by a commercial UAV to collect images of the road network. Next, we demonstrated the line coverage problem with multiple depots and nonholonomic robots on lanes of a set of parking lots. These experiments show that the algorithm models coverage of real-world linear infrastructure well and can be conveniently used in commercial applications.

We presented a novel approach for coverage of area features with multiple capacity-constrained robots in Chapter 4. The approach consists of three components: cell decomposition, generation of service tracks, and routing. The central aspect of our approach is transforming the area coverage problem into a line coverage problem using computational geometry techniques, and then generating routes that minimize the total cost of travel while respecting the resource constraints. Existing methods have a strong bias towards using monotone polygons for cell decomposition and service track generation, which leads to significant inefficiencies. Using the line coverage transformation allows us to handle non-monotone polygons with obstacles while minimizing the number of turns for the robots. Furthermore, several practical aspects of coverage using robots that were addressed for the line coverage problem are translated to the area coverage problem with the help of the transformation. We established the efficacy of our algorithm on a ground robot dataset with 25 indoor environments and an aerial robot dataset with 300 outdoor environments. The algorithm generates solutions whose costs are 10% lower on average than state-of-the-art methods. We additionally demonstrated our algorithm in experiments with aerial robots.

We proposed a novel problem, the generalized coverage problem, which considers the coverage of point, line, and area features in an environment using a team of resource-constrained robots in Chapter 5. The central idea of our approach is to transform the point and area features into line features. The point features were transformed by creating auxiliary edges, and area features were transformed using cell decomposition and service track generation. The transformation enables formulating the generalized coverage problem as a line coverage problem. We used the MEM algorithm with multiple depots to generate efficient routes for generalized coverage. The MEM algorithm supports two modes of travel, direction-dependent costs and demands, resource constraints, multiple depots, and nonholonomic robots. The formulation allowed the seamless translation of the advances in the line coverage problem to the generalized coverage problem. We illustrated the coverage of road networks with traffic signals and parking lots on five university campuses using UAVs. The MEM algorithm computed efficient routes for the largest graph with more than 1,900 required edges within six minutes.

6.2 Future Work

Several avenues for future work arise naturally from the advances in the line coverage problem and the generalized coverage problem described in this dissertation.

Informative path planning (IPP): In the dissertation, we required the robots to gather data from all the features in the environment while minimizing the total cost of the routes. In contrast to coverage problems, the IPP problem does not require visiting all the features in the environment, and the objective is to maximize the amount of information gathered by the robots. Recently, we introduced the correlated arc orienteering problem (CAOP), where the task is to maximize the information collected by resource-constrained robots while exploiting the spatial correlations—visiting a feature may provide data related to another correlated feature (Agarwal and Akella, 2022a). The information is associated with environmental features that can be one-dimensional or points. We formulated a mixed integer quadratic program that formalizes the problem and gives optimal solutions. However, the problem is NP-hard, and therefore we developed an efficient greedy constructive algorithm. One of the future tasks is to develop algorithms that can efficiently handle stochastic models of the information and the costs within the CAOP framework.

Heterogeneous team of robots: In our research, we assumed all the robots to be identical. However, one may have a fleet of robots with a mixture of ground and aerial robots. The cost and demand functions will be different for each type of robot. Furthermore, in certain applications, features may require inspection using a diverse set of sensors. Formulating coverage problems and developing heuristic algorithms for heterogeneous teams of robots are important future directions.

Online decision-making: Recent advances in machine learning algorithms have made it possible to analyze and interpret sensor data on the robot in real time for several applications. An interesting future direction is to use such learning algorithms to provide onboard decision-making capabilities to the robots. Consider the agricultural application of detecting infestations occurring in plants using UAVs. If the UAVs fly at high altitudes, they will have a large sensor field of view and gather more data. However, as the altitude increases, the accuracy and the resolution of the data decrease. If we augment the UAVs with algorithms that can identify potential locations with a high probability of infestation, then the UAVs can alter their planned paths and altitudes to get additional high-resolution data. The fast algorithms presented in the dissertation can potentially be used for computing routes online on the robots. Such autonomous capabilities can significantly enhance the efficiency of the robots.

REFERENCES

Agarwal, S. and Akella, S. (2020). Line coverage with multiple robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3248–3254, Paris, France.

Agarwal, S. and Akella, S. (2021). Approximation algorithms for the single robot line coverage problem. In LaValle, S. M., Lin, M., Ojala, T., Shell, D., and Yu, J., editors, *Algorithmic Foundations of Robotics XIV*, pages 534–550, Cham, Germany. Springer International Publishing.

Agarwal, S. and Akella, S. (2022a). The correlated arc orienteering problem. In *15th International Workshop on Algorithmic Foundations of Robotics*, College Park, MD, USA.

Agarwal, S. and Akella, S. (2022b). Area Coverage With Multiple Capacity-Constrained Robots. *IEEE Robotics and Automation Letters*, 7(2):3734–3741.

Apuroop, K. G. S., Le, A. V., Elara, M. R., and Sheu, B. J. (2021). Reinforcement learning-based complete area coverage path planning for a modified hTrihex robot. *Sensors*, 21(4).

Arkin, E. M., Fekete, S. P., and Mitchell, J. S. (2000). Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1):25–50.

Bähnemann, R., Lawrance, N., Chung, J. J., Pantic, M., Siegwart, R., and Nieto, J. (2021). Revisiting boustrophedon coverage path planning as a generalized traveling salesman problem. In Ishigami, G. and Yoshida, K., editors, *Field and Service Robotics*, pages 277–290, Singapore. Springer Singapore.

Bellman, R. (1962). Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63.

Berg, M. d., Cheong, O., Kreveld, M. v., and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 3rd edition.

Bevern, R. v., Komusiewicz, C., and Sorge, M. (2017). A parameterized approximation algorithm for the mixed and windy capacitated arc routing problem: Theory and experiments. *Networks*, 70(3):262–278.

Bochkarev, S. and Smith, S. L. (2016). On minimizing turns in robot coverage path planning. In *IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1237–1242, Fort Worth, TX, USA.

Cabreira, T. M., Brisolara, L. B., and Ferreira Jr., P. R. (2019). Survey on coverage path planning with unmanned aerial vehicles. *Drones*, 3(1).

Cabreira, T. M., Franco, C. D., Ferreira, P. R., and Buttazzo, G. C. (2018). Energy-aware spiral coverage path planning for UAV photogrammetric applications. *IEEE Robotics and Automation Letters*, 3(4):3662–3668.

Campbell, J. F., Corberán, A., Plana, I., and Sanchis, J. M. (2018). Drone arc routing problems. *Networks*, 72(4):543–559.

Choset, H. (2000). Coverage of known spaces: The boustrophedon cellular decomposition. *Autonomous Robots*, 9(3):247–253.

Choset, H. (2001). Coverage for robotics – A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1):113–126.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.

Chung, S., Paranjape, A. A., Dames, P., Shen, S., and Kumar, V. (2018). A survey on aerial swarm robotics. *IEEE Transactions on Robotics*, 34(4):837–855.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581.

Corberán, A., Eglese, R., Hasle, G., Plana, I., and Sanchis, J. M. (2021). Arc routing problems: A review of the past, present, and future. *Networks*, 77(1):88–115.

Corberán, A. and Laporte, G., editors (2014). *Arc routing: problems, methods, and applications.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Corberán, A. and Prins, C. (2010). Recent results on arc routing problems: An annotated bibliography. *Networks*, 56(1):50–69.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition.

Dasgupta, S., Papadimitriou, C., and Vazirani, U. (2006). *Algorithms.* McGraw-Hill Higher Education New York.

Dille, M. and Singh, S. (2013). Efficient aerial coverage search in road networks. In *AIAA Guidance, Navigation, and Control Conference*, pages 5048–5067, Boston, MA, USA.

Easton, K. and Burdick, J. (2005). A coverage algorithm for multi-robot boundary inspection. In *IEEE International Conference on Robotics and Automation*, pages 727–734, Barcelona, Spain.

Edmonds, J. and Johnson, E. L. (1973). Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5(1):88–124.

Floreano, D. and Wood, R. J. (2015). Science, technology and the future of small autonomous drones. *Nature*, 521:460–466.

Franco, C. D. and Buttazzo, G. C. (2015). Energy-aware coverage path planning of UAVs. In *IEEE International Conference on Autonomous Robot Systems and Competitions*, pages 111–117, Vila Real, Portugal.

Frederickson, G. N. (1979). Approximation algorithms for some postman problems. *Journal of the ACM*, 26(3):538–554.

Frederickson, G. N., Hecht, M. S., and Kim, C. E. (1976). Approximation algorithms for some routing problems. In *17th Annual Symposium on Foundations of Computer Science*, pages 216–227, Houston, USA.

Gabriely, Y. and Rimon, E. (2001). Spanning-tree based coverage of continuous areas by a mobile robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1927–1933, Seoul, South Korea.

Galceran, E. and Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276.

Geisberger, R. and Vetter, C. (2011). Efficient Routing in Road Networks with Turn Costs. In Pardalos, P. M. and Rebennack, S., editors, *Experimental Algorithms*, pages 100–111, Berlin, Heidelberg. Springer Berlin Heidelberg.

Golden, B. L., Dearmon, J. S., and Baker, E. K. (1983). Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10(1):47–59.

Golden, B. L. and Wong, R. T. (1981). Capacitated arc routing problems. *Networks*, 11(3):305–315.

Gouveia, L., Mourão, M. C., and Pinto, L. S. (2010). Lower bounds for the mixed capacitated arc routing problem. *Computers & Operations Research*, 37(4):692–699.

Guan, M. (1984). On the windy postman problem. *Discrete Applied Mathematics*, 9(1):41–46.

Gurobi Optimization, L. (2021). Gurobi optimizer reference manual.

Held, M. and Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210.

Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130.

Helsgaun, K. (2015). Solving the equality generalized traveling salesman problem using the Lin-Kernighan-Helsgaun algorithm. *Mathematical Programming Computation*, 7(3):269–287.

Huang, W. H. (2001). Optimal line-sweep-based decompositions for coverage algorithms. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 27–32, Seoul, South Korea.

Karapetyan, N., Benson, K., McKinney, C., Taslakian, P., and Rekleitis, I. (2017). Efficient multi-robot coverage of a known environment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1846–1852, Vancouver, Canada.

Kirlik, G. and Sipahioglu, A. (2012). Capacitated arc routing problem with deadheading demands. *Computers & Operations Research*, 39(10):2380–2394.

Knuth, D. E. (2011). *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1.* Addison-Wesley.

LaValle, S. (2006). *Planning Algorithms.* Cambridge University Press.

Lenstra, J. K. and Kan, A. H. G. R. (1976). On general routing problems. *Networks*, 6(3):273–280.

Lewis, J. S., Edwards, W., Benson, K., Rekleitis, I., and O'Kane, J. M. (2017). Semi-boustrophedon coverage with a Dubins vehicle. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5630–5637.

Lynch, K. M. and Park, F. C. (2017). *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, USA.

Macharet, D. G. and Campos, M. F. M. (2018). A survey on routing problems and robotic systems. *Robotica*, 36(12):1781–1803.

Mannadiar, R. and Rekleitis, I. (2010). Optimal coverage of a known arbitrary environment. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5525–5530, Anchorage, USA.

Mourão, M. C., Nunes, A. C., and Prins, C. (2009). Heuristic methods for the sectoring arc routing problem. *European Journal of Operational Research*, 196(3):856–868.

Muyldermans, L., Cattrysse, D., and Oudheusden, D. V. (2003). District design for arc-routing applications. *Journal of the Operational Research Society*, 54(11):1209–1221.

Nielsen, L. D., Sung, I., and Nielsen, P. (2019). Convex decomposition for a coverage path planning for autonomous vehicles: Interior extension of edges. *Sensors*, 19(19):4165.

Noon, C. E. and Bean, J. C. (1993). An efficient transformation of the generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, 31(1):39–44.

Oh, H., Kim, S., Tsourdos, A., and White, B. (2014). Coordinated road-network search route planning by a team of UAVs. *International Journal of Systems Science*, 45(5):825–840.

Oksanen, T. and Visala, A. (2009). Coverage path planning algorithms for agricultural field machines. *Journal of Field Robotics*, 26(8):651–668.

OpenStreetMap contributors (2022). Planet dump retrieved from `https://planet.osm.org`. Online: `https://www.openstreetmap.org`.

Orlin, J. B. (1993). A faster stronger polynomial minimum cost flow algorithm. *Operations Research*, 41:338–350.

Orloff, C. S. (1974). A fundamental problem in vehicle routing. *Networks*, 4(1):35–64.

Papadimitriou, C. H. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., USA.

Raghavachari, B. and Veerasamy, J. (1999a). A 3/2-approximation algorithm for the mixed postman problem. *SIAM J. Discrete Math.*, 12:425–433.

Raghavachari, B. and Veerasamy, J. (1999b). Approximation algorithms for the asymmetric postman problem. In *Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 1999, pages 734–741, Baltimore, Maryland, USA.

Ravankar, A., Ravankar, A. A., Kobayashi, Y., Hoshino, Y., and Peng, C.-C. (2018). Path Smoothing Techniques in Robot Navigation: State-of-the-Art, Current and Future Challenges. *Sensors*, 18(9).

Roughgarden, T. (2020). *Algorithms Illuminated, Part 4: Algorithms for NP-Hard Problems.* Soundlikeyourself Publishing, LLC, New York, USA.

Svensson, O., Tarnawski, J., and Végh, L. A. (2018). A constant-factor approximation algorithm for the asymmetric traveling salesman problem. In *50th Annual ACM*

SIGACT Symposium on Theory of Computing, STOC 2018, pages 204–213, Los Angeles, CA, USA.

The CGAL Project (2021). *CGAL User and Reference Manual*. CGAL Editorial Board, 5.2.1 edition.

Theile, M., Bayerlein, H., Nai, R., Gesbert, D., and Caccamo, M. (2020). UAV coverage path planning under varying power constraints using deep reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1444–1449.

Toth, P. and Vigo, D. (2014). *Vehicle Routing*. Society for Industrial and Applied Mathematics, Philadelphia, 2nd edition.

Traub, V. and Vygen, J. (2020). An improved approximation algorithm for ATSP. In *52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 1–13, Chicago, IL, USA.

Vandermeulen, I., Groß, R., and Kolling, A. (2019). Turn-minimizing multirobot coverage. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1014–1020, Montreal, QC, Canada.

Wagner, G. and Choset, H. (2015). Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24.

Wei, M. and Isler, V. (2018). Coverage path planning under the energy constraint. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 368–373.

Williams, K. and Burdick, J. (2006). Multi-robot boundary coverage with plan revision. In *IEEE International Conference on Robotics and Automation*, pages 1716–1723, Orlando, USA.

Williamson, D. P. and Shmoys, D. B. (2011). *The Design of Approximation Algorithms*. Cambridge University Press, USA, 1st edition.

Win, Z. (1989). On the windy postman problem on Eulerian graphs. *Mathematical Programming*, 44(1):97–112.

Winter, S. (2002). Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361.

Wøhlk, S. (2008). An Approximation Algorithm for the Capacitated Arc Routing Problem. *The Open Operational Research Journal*, 2(1):8–12.

Xu, A., Viriyasuthee, C., and Rekleitis, I. (2014). Efficient complete coverage of a known arbitrary environment with applications to aerial operations. *Autonomous Robots*, 36(4):365–381.

Xu, L. and Stentz, A. (2011). An efficient algorithm for environmental coverage with multiple robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4950–4955, Shanghai, China.

Xu, L. and Stentz, T. (2010). A fast traversal heuristic and optimal algorithm for effective environmental coverage. In *Proceedings of Robotics: Science and Systems*, pages 161–168, Zaragoza, Spain.

Yongguo Mei, Yung-Hsiang Lu, Hu, Y. C., and Lee, C. S. G. (2006). Deployment of mobile robots with energy and timing constraints. *IEEE Transactions on Robotics*, 22(3):507–522.

Yu, X. and Hung, J. Y. (2015). Coverage path planning based on a multiple sweep line decomposition. In *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, pages 4052–4058.