SOME PROBLEMS ON MULTI-SENSOR LAYOUT OPTIMIZATION

By

SI WEI FENG

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Department of Computer Science

Written under the direction of

Jingjin Yu

And approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October 2023

ABSTRACT OF THE DISSERTATION

Some Problems on Multi-Sensor Layout Optimization

by SI WEI FENG

Dissertation Director: Jingjin Yu

Robots or robotic applications, like living beings, perceive the world through a great variety of sensors. For range sensors, be it acoustic like sonar, photic like cameras, laser beams or lidars, the sensing range can, to some extent, be ideally abstracted as either 1D lines or 2D circles. Hence, it would be helpful to study the properties of the underlying geometric problems.

The thesis starts in the first chapter with a general review of different sorts of range sensors, especially range and line sensors used in real-world applications. Then, we will give a general overview of the theoretical background and tools used for this thesis. The main body of this thesis is divided into five parts, each of which comes with a specific research problem.

The second chapter talks about perimeter guarding with 1D sensors, where the sensing range is assumed to be a continuous line on top of the perimeter of some regions. Two problems are introduced, perimeter guarding with homogeneous sensors and with heterogeneous sensors. The homogeneous case can be solved using only classical algorithms. While the heterogeneous case is NP-hard, but can still be solved with dynamic programming under a reasonable amount of time.

The third chapter continues with perimeter guarding but uses circles to represent the coverage range of sensors. The study extends to guarding 2D regions beyond the perime-

ters. Our study turns out that even for covering the boundary of a simple polygon, the problem of finding the minimum sensing radius is NP-hard to approximate within a factor of 1.152. However, effective approximation algorithms are developed to solve this problem.

After the two chapters about covering perimeters or regions, which is essentially separating some critical polygonal regions from the outside. The fourth chapter digs deeper into this problem by studying the separation of more than two polygonal sets. To simplify the problem, a line-of-sight sensing model will be adopted. The problem is NP-hard even for the problem of separating two sets of regions with the minimum number of lines. Still, a near-optimal solution using integer programming is provided.

The fifth chapter deals with the dynamic setting, two different but related problems are studied. The first problem is the boundary defense problem in the context of heterogeneous defenders, which is an extension of the perimeter defense problem. The second problem is the coordinated sweeping problem where a group of robots coordinate to sweep a region with obstacles with some certain sweep plan.

The sixth chapter discusses a real-world application of the placement of UV (ultra violet) lights to cover the surface of some regions for sanitization purposes.

Lastly, we summarize our work and bring out potential future work and studies on this subject.

ACKNOWLEDGMENTS

First and foremost, I would like to acknowledge my advisor Prof. Jingjin Yu, who is also the lab director of the Algorithmic Robotics and Control Lab (ARCL). He has been extremely helpful during my Ph.D. journey not only in guiding me in research, funding my study and helping me through academic challenges, but also in providing professional advice during my low time in the last semester. It's really my great pleasure and fortune to work on algorithmic robotics as well as many fun projects with JJ!

I also want to extend my sincere appreciation to my committee members. Prof. Kostas Bekris was the lecturer of the first two robotics classes I fortunately took in the first year: computational robotics and robot manipulation seminar, both of which guided me towards a more mature and experienced roboticist. Prof. Abdeslam Boularias's kindness and amicability as well as his positiveness affect the atmosphere of the whole robotics lab at 1 Spring Street. Although I have not met Dr. Zherong Pan in person before, I have learned a lot by studying Dr. Pan's work, and his career suggestions have benefited me greatly.

Thanks to my labmates at ARCL: Shuai Han, Kai Gao, Teng Guo, Baichuan Huang, Wei Tang, Sijie Ding, Andy Xu, Tanay Punjabi, Justin Yu, and Xinyu Cai.

Ph.D. is not easy, were it not for my friends, my Ph.D. life could not have been as wonderful as it has been so far. So, I would like to thank them here: Rui Wang, Junchi Liang, Chaitanya Mitash, Bowen Wen, Juntao Tan, Isidoros Marougkas, Jonathan Garcia-Mullen, Shiyang Lu, Yinglong Miao, Haonan Chang, Xinyu Zhang, Dhruv Metha Ramesh, Alon Flor, and Hansen Lim.

I would like to thank Nokia Bell Labs for the summer internship experience, and my mentor Dr. Fangzhe Chang for his instructions.

The pandemic happens to be between the 2nd and the 4th year of my Ph.D. journey, and it entirely changed people's lives and working styles. But luckily we are back to normal, I think special thanks should be given to the selfless medical workers and volunteers who

helped us fight the pandemic.

Last but not least, I need to say thanks to my parents for their unconditional love and caring throughout these years of my life.

TABLE OF CONTENTS

LIST OF TABLES

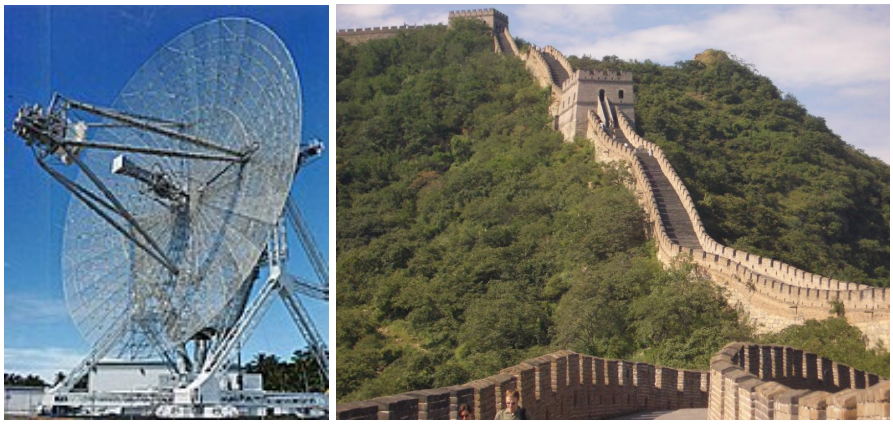
LIST OF FIGURES

CHAPTER 1
INTRODUCTION

In this chapter, we first discuss some regular types of sensing systems used for guarding, tracking or surveillance, with a focus on line and range sensors. These will serve as the motivation for this dissertation study Then, we conduct a literature study of the related work on sensor placement and coverage-related problems. Lastly, some background knowledge of the theories and tools used in this chapter will be given.

1.1 Motivations

Sensor systems are ubiquitous in the modern world. To list a few, systems of radar antennas or other sensor sources are frequently used as base stations for signal transmission, or intruder detection system (IDS) for monitoring hazards. Early intruder defense system can date back to ancient times, where watchtowers of the Great Wall of China are used as signal points. It can be seen as sensors from the broad sense since ancient soldiers lit wood to create smoke and inform others when invaders appear.

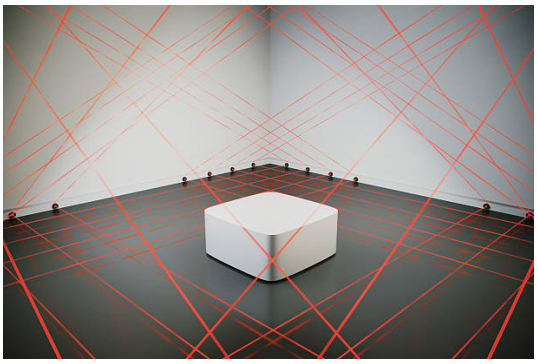


(a) Radar antenna (b) Watch towers on the great wall

Figure 1.1: Two examples of intrusion detection system

In surveillance or tracking systems (Figure 1.2b), sensors like laser beams or cameras are deployed for purposes like thief detection, motion capture, pose tracking and so on.

(a) Optical tracking system



(b) Laser system

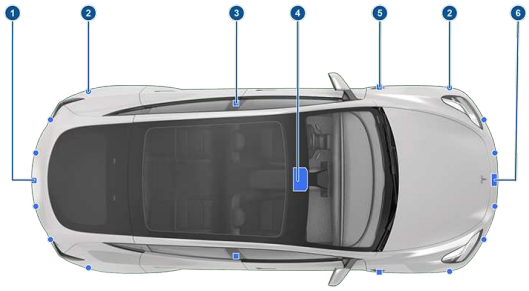
Figure 1.2: Indoor tracking and surveillance systems

The deployment of sensors is not limited only to deploying sensors static to the ground. A notable example is the advent of self-driving cars. On top of those autonomous vehicles, sensor systems are indispensable for obstacle avoidance and interactions between vehicles. Tesla (Figure 1.3a) uses 12 ultrasonic sensors near the front and rear bumper and later changed into a vision system with only cameras [1]. And TuSimple, an autonomous truck company, employs a combination of cameras, radars and lidars for their perception system [2].

The placement of sensors is extremely crucial for many reasons. First, a typical high-frequency tracking camera can cost up to several thousand US dollars in 2023, let alone sensors with specific industrial or military purposes. If a certain sensor layout solution can reduce the number of sensors used in, e.g., in a secure environment, on an autonomous vehicle, or on a patrolling robot, the cost for deployment of such sensor systems can be greatly reduced. Second, real sensors come with many characteristics as mentioned in [1], range sensors can have noise, unexpected objects blocking the view, sensing failures, random measurements and so on. Hence, the objective that a sensor system being more

[1] https://www.tesla.com/en_eu/support/transitioning-tesla-vision

[2] https://www.tusimple.com/blogs/tusimple-1000-meter-perception-system



(a) Tesla Model Y's sensing system equipped with cameras (①, ③, ④, ⑤), ultrasonic sensors ②, and a radar ⑥.

(b) TuSimple autonomous truck equipped with CMOS long-range cameras, LiDARs and radars

Figure 1.3: Sensor systems on autonomous vehicles

balanced in terms of workload is a reasonable assumption when quality assurance and fault tolerance are the main concerns for the system. Lastly, a good sensor deployment in systems like surveillance camera systems means less human labor from the security personnel; and less energy consumption.

1.2 Background

In this section, we provide sufficient background knowledge and terms used in this dissertation, they will be used without explanation in the following chapters.

1.2.1 NP and NP-hardness

Definition 1 (Nondeterministic Polynomial (NP))**.** *A language $L \in NP$ if there is a polynomial p and a polynomial time-bounded Turing machine M, called the verifier, such that for each string $x \in \{0,1\}^*$:*

- *if $x \in L$, then there is a string y (the certificate) of polynomially bounded length, i.e., $|y| \leq p(|x|)$, such that $M(x,y)$ accepts, and*
- *if $x \notin L$, then for any string y, such that $|y| \leq p(|x|)$, $M(x,y)$ rejects.*

A colloquial way in [2] to describe an NP is the class of problems that have "short and quickly verifiable" Yes certificates. And an NP-hard problem is a problem that every

problem in NP can reduce to it. Typically, when we call an optimization problem NP-hard, it means the decision version of it is NP-hard.

1.2.2 Integer programming

Since most natural optimization problems are NP-hard, mathematical programming tools are often used for solving the problem for its generality and efficiency. Essentially, they take in some mathematical models including a set of integral variables $x_1, \ldots, x_n$ and a set of constraints,

$$a_{11}x_{11} + \cdots + a_{1n}x_{1n} \geq b_1,$$

$$\cdots$$

$$a_{m1}x_{m1} + \cdots + a_{mn}x_{mn} \geq b_m,$$

and objective

$$\min a_{11}x_{11} + \cdots + a_{1n}x_{1n}.$$

When the variables $x_1, \ldots, x_n$ contain continuous variables (continuum), the model becomes a mixed integer programming problem (MILP).

Commercial (mixed) integer programming tools include Gurobi [3], IBM CPLEX [4], and so on, while open-source libraries include SCIP [5], CBC [6], GLPK [7], and so on. Modern tools, even in the open-source branch, are pretty mature and developed, and hard to optimize within the framework itself. While work like [8] in bipedal robot footstep planning and [9] in multi-robot path planning seek performance improvement by constructing different instance formulations.

1.2.3 Approximation algorithm

For an optimization problem Π, OPT(Π), or sometimes OPT, denotes the optimal solution of the problem instance. An $(1+\alpha)$-OPT or a $(1+\alpha)$-optimal solution refers to a solution

with an objective value of $(1+\alpha)$OPT for a minimization objective. For the maximization objective, the corresponding term is $(1-\alpha)$OPT.

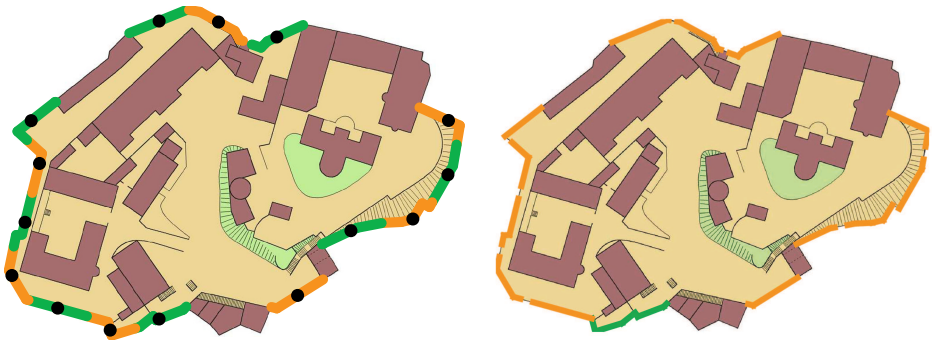
Now we give the definition of PTAS and FPTAS.

Definition 2 (Polynomial Time Approximation Scheme (PTAS))**.** *Given any $\varepsilon > 0$, if an algorithm $\mathcal{A}$ runs in polynomial time to the input length. Then if $\mathcal{A}$ can provide a $(1+\varepsilon)$-OPT solution, the algorithm is called PTAS.*

Definition 3 (Fully Polynomial Time Approximation Scheme (FPTAS))**.** *Given any $\varepsilon > 0$, if an algorithm $\mathcal{A}$ runs in polynomial time to the input length and $1/\varepsilon$. Then if $\mathcal{A}$ gives a $(1+\varepsilon)$-OPT solution, the algorithm is called FPTAS.*

1.3 Problems Studied in the Dissertation

In this section, we provide an overview of the problems studied in the later chapters, where each chapter is independent and self-contained.



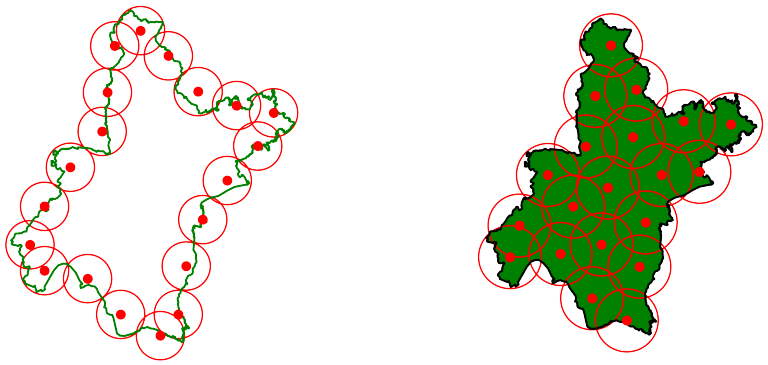
(a) Optimal perimeter guarding with homogeneous sensors

(b) Optimal perimeter guarding with heterogeneous sensors

Figure 1.4: Optimal perimeter guarding

The first set of problems we studied relate to optimally assigning a larger number of sensing robots (or other types of autonomous agents) to guard the perimeters of closed 2D regions, where the perimeter of each region to be guarded may contain multiple disjoint polygonal chains. Each robot is responsible for guarding a subset of the perimeter and any perimeter must be guarded by at some robot. The sensing range of each robot is assumed

to be a continuous line on top of the perimeter of some regions. Specifically, two problems will be introduced, perimeter guarding with homogeneous sensors (Figure 1.4a), and with heterogeneous sensors (Figure 1.4b). The homogeneous case can be solved using only classical algorithms. While the heterogeneous case is NP-hard, but can still be solved with dynamic programming under a reasonable amount of time.



(a) Optimal perimeter guarding with 2D range sensors

(b) Optimal region guarding with 2D range sensors

Figure 1.5: Optimal set guarding with 20 sensors to cover the perimeter and interior of a city

Then, we continue with perimeter guarding but use 2D circles to represent the coverage range of sensors. The study extends to guarding 2D regions beyond the perimeters. When given a bounded $\mathbb{R}^2$ to be guarded, and k mobile sensors with variable sensing ranges of $r_1, \ldots, r_k$, our objective is to minimize $\max_{i=1}^{k} r_i$. Our study shows that even for covering the boundary or interior of a simple polygon, the problem of finding the minimum sensing radius is NP-hard to approximate within a factor of 1.152, i.e., unless P=NP, there is no polynomial solution for finding a 1.152 optimal solution. However, on the side of computational methods, we develop a fully polynomial time approximation algorithm for covering a perimeter with a reasonable assumption of continuous coverage.

After two chapters' discussion on covering perimeters or regions, which is essentially separating some critical polygonal regions from the outside workspace. The fourth chapter digs deeper into this problem by studying the separation of more than two regions. To

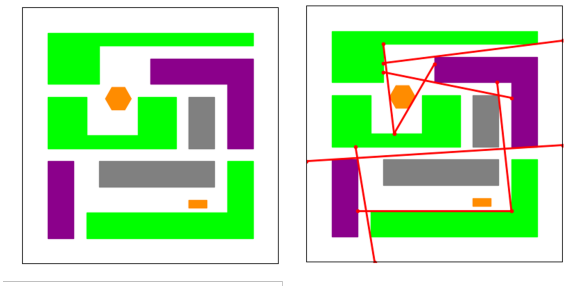


Figure 1.6: Separating three sets of building blocks (orange, purple and green) with the existence of two grey obstacles.

simplify the problem, a line-of-sight sensing model will be adopted, where each sensor can cover an unobstructed line segment like a laser beam. Also, the regions are assumed to be polygonal. The objective in this case is to minimize the number of sensors used to separate these polygonal sets at the existence of obstacles (see Figure 1.6). The problem is NP-hard even for the problem of separating two sets of regions with the minimum number of lines. Still, integer programming can provide a near-optimal solution for around 20 objects in a reasonable amount of time.

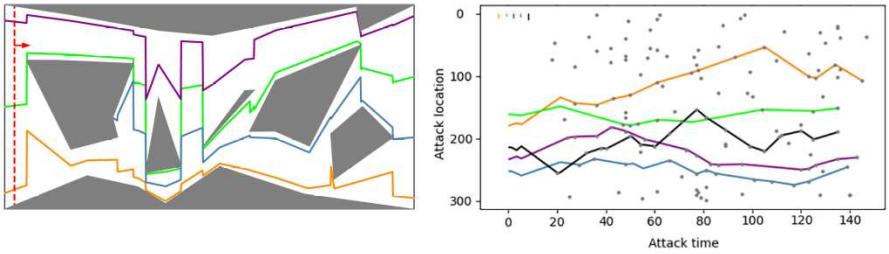


Figure 1.7: [left] the coordinated sweeping problem, [right] the perimeter defense problem.

The fifth chapter deals with the dynamic setting for mobile sensing robots, and two different but related problems are studied. The first problem is the boundary defense problem in the context of heterogeneous defenders first studied in [10]. It can be seen as an extension of the perimeter defense problem [11]. In this problem, there are k sensing robots moving on top of a perimeter with different speeds $v_1, \ldots, v_k$. A sequence of attacks $\langle loc_i, t_i \rangle_{i=1}^{n}$ are given at different time stamps and at different locations. The objective is to intercept as many attacks as possible. The second problem is the coordinated sweeping problem where

a group of robots coordinate to sweep a region with obstacles. Each robot possesses a given sensing capability, and the sweeping trajectory is given beforehand. The objective of it is to minimize the number of robots used such that the sweeping plan can be executed and every point in the workspace is sensed with a certain required quality.

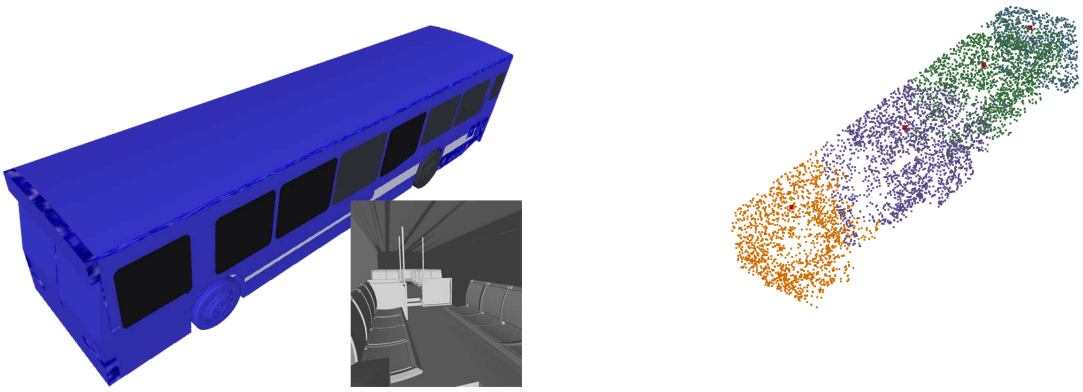


Figure 1.8: UV sanitization lights placement to cover the interior of a bus

The sixth chapter discusses a real-world application in the placement of UV (ultraviolet) lights to cover the surface of an environment for sanitization purposes. The problem can be seen as an extension of the art-gallery problem [12], and thus computationally intractable. Still, a combined integer programming and classical local search approach can be used to solve the problem effectively. The method is evaluated on three real scenarios: the interior of a bus, a subway or an ICU room.

1.4 Literature review

1.4.1 Coverage-related problems in computational geometry

As multi-robot coverage problems, this thesis is intimately connected to Art Gallery problems [12, 13], with origins traceable to half a century ago [14]. Art Gallery problems assume a visibility-based [15] sensing model; in a typical setup [12], the *interior* of a polygon must be visible to at least one of the guards, which may be placed on the boundaries,

corners, or the interior of the polygon. Finding the optimal number of guards are often NP-hard [16]. Alternatively, a disc-based sensing model may be used, which leads to the classical *packing* problem [17, 18], where no overlap is allowed between the sensors' coverage area, the *coverage* problem [19, 20, 21, 22, 23], where all workspace must be covered with overlaps allowed, or the *tiling* problem [24], where the goal is to have the union of sensing ranges span the entire workspace without overlap. For a more complete account on Art Gallery, packing, and covering, see Chapters 2, 3, and 33 of [25]. Despite the existence of a large body of literature performing extensive studies on these intriguing computational geometry problems, these types of research mostly address domains that are 2D and higher

As pointed out in [20, 26], distributed sensor coverage has roots in the study of the facility location optimization problem [27, 19], which examines the selection of facility (e.g., warehouses) locations that minimize the cost of delivery of supplies to spatially distributed customers. In theoretical computer science and operations research, these are known as the k-center, k-means, and k-median clustering problems [28], the differences among which are induced by the cost structure. Our investigations benefit from the vast literature on the study of k-center clustering and related problems, e.g., [29, 30, 31, 32, 33]. These clustering problems are in turn related to packing [18], tiling [17], and the well-studied art gallery problems [12, 13].

1.4.2 Multi-robot coordination

Our work draws inspiration from a long line of multi-robot coverage planning and control research, e.g., [20, 34, 35, 21, 26, 23]. In an influential body of work on coverage control [20, 34], a gradient-based iterative method is shown to drive one or multiple mobile sensors to a locally optimal configuration with convergence guarantees. Whereas [20, 34] assume that the distribution of sensory information is available *a priori*, it is shown that such information can be effectively learned [26]. Subsequently, the control method is further extended to allow the coverage of non-convex and disjoint 2D domains [35] and to

work for mobile robots with varying sensing or actuation capabilities [23]. In contrast to these control-based approaches, which produce iterative locally optimal solutions, our work emphasizes the direct computation of globally optimal deployment solutions and supports arbitrarily shaped bounded (1D) perimeters and (2D) regions.

With an emphasis on robotic swarm deployment, within multi-robot systems research [36, 37, 38, 39], our study is closely related to *formation control*, e.g., [40, 41, 42, 43, 44, 45, 46], where the goal is to achieve certain *distributions* through continuous (often, local sensing based) interactions among the agents or robots. Depending on the particular setting, the distribution in question may be spatial, e.g., rendezvous [40, 46], or maybe an agreement in agent velocity is sought [41, 43]. In these studies, the resulting formation often have some degree-of-freedoms left unspecified. For example, rendezvous results [40, 46] often come with exponential convergence guarantee, but the location of rendezvous is generally unknown *a priori*.

In multi-robot task and motion planning problems (e.g., [47, 48, 49, 50, 51, 52, 53, 54]), especially ones with a *task allocation* element [47, 49, 50, 51, 52, 54], the (permutation-invariant) target configuration is often mostly known. The goal here is to find a one-to-one mapping between individual robots and the target locations (e.g., deciding a *matching*) and then plan (possibly collision-free) trajectories for the robots to reach their respective assigned targets [51, 52, 54].

1.4.3 Search and rescue

The study of the dynamic coverage in this dissertation draws inspiration from the study of several lines of related problems. The Graph-Clear problem, formulated in [55], tasks a group of robots to search and clear an environment with the operations of blocking and clearing. A follow-up work on Line-Clear [56] uses line guards with more focus on computational geometry in that the objective is to minimize the maximum sweep line distance. Both of these problems are NP-hard, establishing the difficulties of finding a sweeping

schedule for a planar environment. The more general pursuit-evasion problem dates back to the research on *search number* on a discrete graph [57], followed by studies on pursuit and evasion in continuous environments with visibility-based model [58, 59, 60, 61].

When working with known patrolling search frontiers, e.g., vertical sweep lines, this sweep line coverage problem is analogous to the perimeter defense problem by placing guards on a static perimeter to defend intruders [62, 63, 64]. Previously, we have also studied a version of static range guard placement problems for securing perimeters and regions [65]. In contrast to the pursuit-evasion algorithms that deal with searching dynamic and unpredictable targets that could escape, coverage planning/control-related algorithms become more suitable for searching or covering predictable or stationary targets, e.g., room sweeping, pesticide and fertilizer spraying, persistent monitoring, and so on [20, 66, 67, 68, 69, 70, 71, 72, 73, 74].

CHAPTER 2
PERIMETER GUARDING

This chapter studies the problem of optimally assigning a larger number of sensing robots (or other types of autonomous agents) to guard the perimeters of closed 2D regions, where the perimeter of each region to be guarded may contain multiple disjoint polygonal chains. The problem is termed "perimeter guarding". We first introduce perimeter guarding with homogeneous sensors, then step into perimeter guarding with heterogeneous sensors (Figure 1.4b).

2.1 Perimeter Guarding with Homogeneous Defenders

2.1.1 Introduction

Consider the scenario from Figure 2.1, which contains a closed region with its boundary (or border) demarcated by the red and dotted blue lines. To secure the region, either from intrusions from the outside or unwanted escapes from within, it is desirable to deploy a number of autonomous agents to monitor or guard either the entire boundary or selected portions of it (e.g., the red line segments), with each agent responsible for a continuous section. Naturally, one might also want to have even coverage by the agents, e.g., minimizing the maximum effort from any agent. In practice, such efforts may correspond to sensing ranges or motion capabilities of robots, which are always limited. As an intuitive example, the figure may represent the top view of a castle, with its entire boundary being a high wall that may be traversed by agents. The portion of the wall marked with the three red line segments must be protected, whereas the art marked by the dotted blue lines may not need active monitoring (e.g., the outside may be a cliff or a body of deep water). The green and orange lines show an optimal distribution of the workload by 8 agents that cover

all red segments but skip two of the three-dotted line segments.

More formally, in this chapter, we study the problem of deploying a large number of robots to guard a set of one-dimensional boundary segments called perimeters. Each perimeter is comprised of one or more 1D segments that are part of a circular boundary (e.g., the red segments in Figure 2.1). Each robot is tasked to guard a continuous 1D segment that covers a portion of a perimeter. As the main objective, we seek an allocation of robots such that *(i)* the union of the robots' coverage encloses all perimeters and *(ii)* the maximum coverage of any robot is minimized. We call this 1D deployment problem the Optimal Perimeter Guarding (OPG) problem.

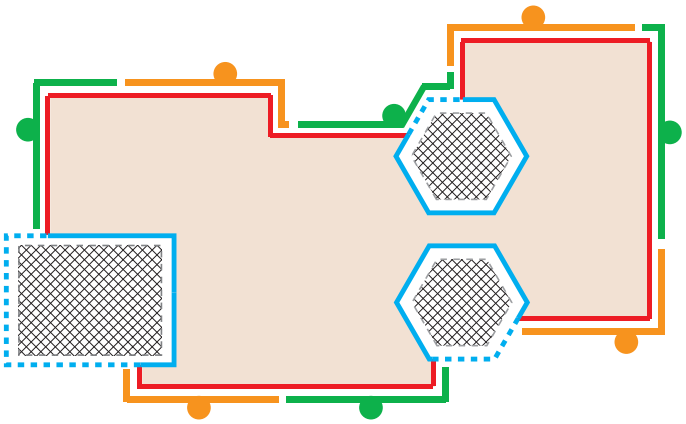


Figure 2.1: An illustrative scenario where a perimeter, in this case represented as the red line segments, must be guarded by $n = 8$ robots, which are constrained to only travel along the perimeter boundary (the red line segments plus the dotted blue lines, which are gaps that do not need to be guarded). An optimal set of locations for the 8 robots and the coverage region for each robot are marked on the perimeter boundary in green and orange, which minimizes the maximum coverage required for any robot.

In this work, three main OPG variants are examined. The settings regarding the perimeter in these three variants are: *(i)* multiple perimeters with each having a single connected component; *(ii)* a single perimeter containing multiple connected components; and *(iii)* multiple perimeters with each containing multiple connected components (the most general case). For all three variants, we have developed exact algorithms for solving OPG that runs in low polynomial time. More specifically, let there be n robots, m perimeters, with perimeter i ($1 \leq i \leq m$) containing q_i connected components. If $m = 1$, then let the only perimeter contains q connected components. For the three variants, our algorithm

computes an optimal solution in time $O(m(\log n + \log m) + n)$, $O(q^2 \log(n + q) + n)$, and $O((\sum_{1 \leq i \leq m} q_i^2) \log(n + \sum_{1 \leq i \leq m} q_i) + n)$, respectively, which are roughly quadratic in the worst case. In addition to computing locations for deploying the robots, we further compute shortest paths for deploying the robots, given some initial configuration of the robots. The modeling of the OPG problem and the development of the efficient algorithms for OPG constitute the main contribution of this chapter.

The rest of the chapter is organized as follows. The OPG problem and some of its most basic properties are described in Section subsection 2.1.2. In Section subsection 2.1.3, a thorough structural analysis of OPG with single and multiple perimeters is performed, paving the way for introducing the full algorithmic solutions in Section subsection 2.1.4. Then, in Section subsection 2.1.8, comprehensive numerical evaluations of the multiple polynomial-time algorithms are carried out. In addition, two realistic application scenarios are demonstrated. In Section subsection 2.1.11, we conclude with additional discussions.

2.1.2 The Optimal Perimeter Guarding Problem

Let $\mathcal{W} \subset \mathbb{R}^2$ be a compact (i.e., closed and bounded) two-dimensional workspace. There are m pairwise disjoint *regions* $\mathcal{R} = \{R_1, \ldots, R_m\}$ where each region $R_i \subset \mathcal{W}$ is homeomorphic to the closed unit disc, i.e., there exists a continuous bijection $f_i : R_i \to \{(x, y) \mid x^2 + y^2 \leq 1\}$ for all $1 \leq i \leq m$. For a given region R_i, let ∂R_i be its (closed) boundary (therefore, f_i maps ∂R_i to the unit circle $\mathbb{S}^1$). With a slight abuse of notation, define $\partial\mathcal{R} = \{\partial R_1, \ldots, \partial R_m\}$. Let $P_i \subset \partial R_i$ be the part of ∂R_i that is accessible, e.g., not blocked by obstacles in $\mathcal{W}$. This means that each P_i is either a single closed curve or formed by a finite number of (possibly curved) line segments. Define $\mathcal{P} = \{P_1, \ldots, P_m\} \subset \mathcal{W}$ as the *perimeter* of $\mathcal{R}$ which must be *guarded*. More formally, each P_i is homeomorphic to a compact subset of the unit circle (i.e., it is assumed that the maximal connected components of P_i are closed line segments). For a given P_i, each one of its maximal connected component is called a *perimeter segment* or simply a *segment*, whereas each maximal connected

components of $\partial R_i \backslash P_i$ is called a *perimeter gap* or simply a *gap*. An example setting is illustrated in Figure 2.2 with two regions.

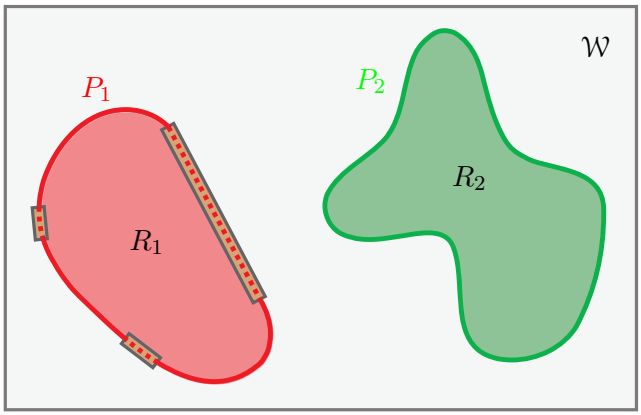


Figure 2.2: An example of a workspace $\mathcal{W}$ with two regions $\{R_1, R_2\}$. Due to three *gaps* on ∂R_1, marked as dotted lines within long rectangles, $P_1 \subset \partial R_1$ has three *segments* (or maximal connected components); $P_2 = \partial R_2$ has a single segment with no gap.

There are n indistinguishable point robots residing in $\mathcal{W}$. These robots are to be deployed to *cover* the perimeter $\mathcal{P}$ such that each robot $1 \leq j \leq n$ is assigned a continuous closed subset C_j of some $\partial R_i, 1 \leq i \leq m$. All of $\mathcal{P}$ must be *covered* by $\mathcal{C}$, i.e., $\bigcup_{P_i \in \mathcal{P}} P_i \subset \bigcup_{C_j \in \mathcal{C}} C_j$, which implies that elements of $\mathcal{C}$ need not intersect on their interiors. Hence, it is assumed that any two elements of $\mathcal{C}$ may share at most their endpoints. Such a $\mathcal{C}$ is called a cover of $\mathcal{P}$.

Given a cover $\mathcal{C}$, for a $C_j \in \mathcal{C}$, $1 \leq j \leq n$, let $len(C_j)$ denote its length (more formally, measure). It is desirable to minimize the maximum $len(C_j)$, i.e., the goal is to find a cover $\mathcal{C}$ such that the value $\max_{C_j \in \mathcal{C}} len(C_j)$ is minimized. This corresponds to minimizing the maximum workload for each robot or agent. The formal definition of the Optimal Perimeter Guarding (OPG) problem is provided as follows.

Problem 1 (Optimal Perimeter Guarding (OPG))**.** *Given the perimeter* $\mathcal{P} = \{P_1, \ldots, P_m\}$ *of a set of 2D regions* $\mathcal{R} = \{R_1, \ldots, R_m\}$*, find a set of* n *continuous line segments* $\mathcal{C}^* =$

$\{C_1^*, \ldots, C_n^*\}$ *such that* $\mathcal{C}^*$ *covers* $\mathcal{P}$*, i.e.,*

$$\bigcup_{P_i \in \mathcal{P}} P_i \subset \bigcup_{C_j^* \in \mathcal{C}^*} C_j^*, \tag{2.1}$$

with the maximum of $len(C_j^*), 1 \leq j \leq n$ *minimized, i.e., among all covers* $\mathcal{C}$ *satisfying* (Equation 2.1)*,*

$$\mathcal{C}^* = \operatorname*{argmin}_{\mathcal{C}} \max_{C_j \in \mathcal{C}} len(C_j). \tag{2.2}$$

Here, we introduce the technical assumption that the ratio between the length of $\partial\mathcal{R}$ and the length of $\partial\mathcal{P}$ is polynomial in the input parameters. That is, the length of $\partial\mathcal{R}$ is not much bigger than the length of $\partial\mathcal{P}$. The assumption makes intuitive sense as any gap should not be much bigger than the perimeter in pratice. We note that the assumption is not strictly necessary but helps simplify the correctness proof of some algorithms.

Henceforth, in general, $\mathcal{C}^*$ is used when an optimal cover is meant whereas $\mathcal{C}$ is used when a cover is meant. We further define the optimal single robot coverage length as

$$\ell^* = \min_{\mathcal{C}} \max_{C_j \in \mathcal{C}} len(C_j). \tag{2.3}$$

Figure 2.1 shows an example of an optimal cover by 8 robots of a perimeter with three components. Note that one of the three gaps (the one on the top area as part of the hexagon) is fully covered by a robot, which leads to a smaller ℓ^* as compared to other feasible solutions. This interesting phenomenon, which is actually a main source of the difficulty in solving OPG, is explored more formally in Section subsection 2.1.3 (Proposition Theorem 7).

Given the OPG formulation, additional details on $\partial\mathcal{R}$ must be specified to allow the precise characterization of the computational complexity (of any algorithm developed for OPG). For this purpose, it is assumed that each $\partial R_i \in \partial\mathcal{R}$, $1 \leq i \leq m$, is a simple (i.e.,

non-intersecting and without holes) polygon with an input complexity $O(M_i)$, i.e., ∂R_i has about M_i vertices or edges. If an OPG has a single region R, then let ∂R have an input complexity of M. Note that the algorithms developed in this work apply to curved boundaries equally well, provided that the curves have similar input complexity and are given in a format that allow the computation of their lengths with the same complexity. Alternatively, curved boundaries may be approximated to arbitrary precision with polygons.

For deploying a robot to guard a C_j, one natural choice is to send the robot to a location $t_j \in C_j$ such that t_j is the centroid of C_j. Since C_j is one dimensional, t_j is the center (or midpoint) of C_j. After solving an OPG, there is the remaining problem of assigning the n robots to the centers of $\mathcal{C}^* = \{C_j^*\}$ and actually moving the robots to these assigned locations. As a secondary objective, it may also be desirable to provide guarantees on the execution time required for deploying the robots to reach target guarding locations. We note that, the task assignment (after determining target locations) and motion planning component for handling robot deployment, essential for applications but not a key part of this work's contribution, is briefly addressed in Section subsection 2.1.8.

With some $\mathcal{C}^*$ satisfying (Equation 2.1) and (Equation 2.2), we may further require that $len(C_j^*)$ is minimized for all $C_j^* \in \mathcal{C}^*$. This means that a gap $G \subset ((\bigcup \partial R_i) \backslash (\bigcup P_i))$ will never be partially covered by some $C_j^* \in \mathcal{C}^*$ because if that is the case, C_j^* needs not cover any part of G at all (and should not, to limit the coverage of the assigned robot). In the example from Figure 2.2, G may be one of the three dotted lines on ∂R_1; clearly, it is not beneficial to have some C_j^* partially cover (i.e., intersect the interior of) one of these. This rather useful condition (note that this is not an assumption but a solution property) yields the following lemma.

Lemma 4. *For a set of perimeters $\mathcal{P} = \{P_1, \ldots, P_m\}$ where $P_i \subset \partial R_i$ for $1 \leq i \leq m$, there exists an optimal cover $\mathcal{C}^* = \{C_1^*, \ldots, C_n^*\}$ such that, for any gap (or maximal connected component) $G \subset ((\bigcup \partial R_i) \backslash (\bigcup P_i))$ and any $C_j^* \in \mathcal{C}^*$, $C_j^* \cap G = G$ or $C_j^* \cap G = \varnothing$.*

Remark 5. *Our definition of coverage is but one of the possible models of coverage. The*

definition restricts a robot deployed to $C_j, 1 \leq j \leq n$, to essential live *on C_j. The definition models scenarios where a guarding robot must travel along C_j, which is one-dimensional. Nevertheless, the algorithms developed for* OPG *have broader applications. For example, subroutines in our algorithms readily solve the problem of finding the minimum number of guards needed if each guard has a predetermined maximum coverage.*

2.1.3 Structural Analysis

In designing efficient algorithms, the solution structure of OPG induced by the problem formulation is first explored, starting from the case where there is a single region.

Guarding a Single Region

Perimeter with a single connected component. For guarding a single region $\mathcal{R} = \{R\}$, i.e., there is a single boundary ∂R to be guarded, all n robots can be directly allocated to ∂R. If the single perimeter $P \subset \partial R$ further has a single connected component that is either homeomorphic to $\mathbb{S}^1$ or $[0, 1]$, then each robot j can be assigned a piece $C_j \subset P$ such that $\bigcup_{C_j \in \mathcal{C}} C_j = P$ and $len(C_j) = len(P)/n$. Clearly, such a cover $\mathcal{C}$ is also an optimal cover.

Perimeter with multiple maximal connected components. When there are multiple maximal connected components (or segments) in a single perimeter P, things become more complex. To facilitate the discussion, assume here P has q segments $S_1, \ldots, S_q$ arranged in the clockwise direction (i.e., $P = S_1 \bigcup \ldots \bigcup S_q$), which leaves q gaps $G_1, \ldots, G_q$ with G_k immediately following S_k. Figure 2.3 shows a perimeter with five segments and five gaps.

Suppose an optimal set of assignments for the n robots guarding P and satisfying (Equation 2.1) and (Equation 2.2) is $\mathcal{C}^* = \{C_j^*\}$. Let G_{max} be a largest gap, i.e., $len(G_{max}) = \max_{1 \leq k \leq q} len(G_k)$. Via small perturbations to the lengths of G_k, we may also assume that G_{max} is unique. If we exclude the gap G_{max} and have the n robots cover the rest of ∂R evenly, then it must hold that $len(C_j^*) \leq (len(\partial R) - len(G_{max}))/n$. On the other hand, $len(C_j^*) \geq$

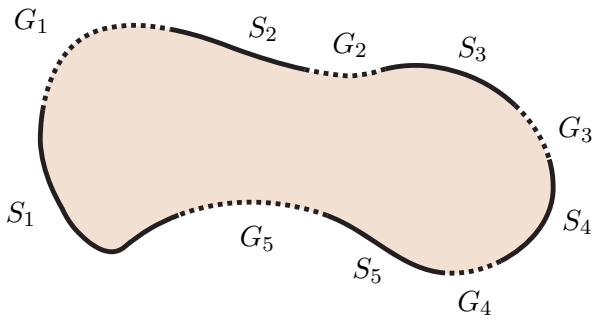


Figure 2.3: A perimeter P with five segments $S_1, \ldots, S_5$, separated by five *gaps* $G_1, \ldots, G_5$.

$(\sum_{1 \leq k \leq q} len(S_k))/n$ always holds. These yield a pair of basic upper and lower bounds for the optimal single robot coverage length ℓ^*, summarized as follows.

Proposition 6. *Define*

$$\ell_{min} = \frac{\sum_{1 \leq k \leq q} len(S_k)}{n} \; and \; \ell_{max} = \frac{len(\partial R) - len(G_{max})}{n},$$

it holds that

$$\ell_{min} \leq \ell^* \leq \ell_{max}. \tag{2.4}$$

On the other hand, it is not always the case that a largest gap G_{max}, even if unique, will be skipped by $\bigcup_{C_j \in \mathcal{C}^*} C_j^*$. That is, an optimal cover C^* may enclose the largest gap.

Proposition 7. *Given a region R and perimeter $P \subset \partial R$, let G_{max} be the unique longest connected component of $\partial R \backslash P$. Let $\mathcal{C}^*$ be an optimal cover of P. Then, it is possible that $G_{max} \subset C_j^*$ for some $C_j^* \in \mathcal{C}^*$.*

Proof. The claim may be proved via contradiction with the example illustrated in Figure 2.4 which readily generalizes. In the figure, there are four gaps $G_1, \ldots, G_4$, in which three gaps (G_1, G_2, and G_4) have the same length (i.e., $len(G_1) = len(G_2) = len(G_4)$) and are evenly spaced (i.e., $len(S_1) = len(S_2) = len(S_3 \cup G_3 \cup S_4)$). Here, $G_{max} = G_3$, which is 1.5 times the length of other gaps, i.e., $len(G_3) = \frac{3}{2} len(G_1)$.

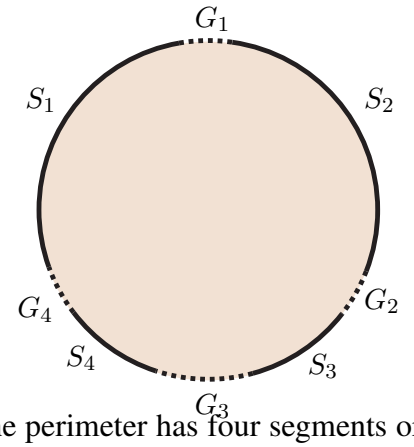


Figure 2.4: A case where the perimeter has four segments or maximal connected components. Three of the gaps, G_1, G_2, and G_4 are of the same length and are evenly spaced, G_3 is 0.5 times longer.

For $n = 3$ robots, the optimal cover $\mathcal{C}^*$ must allocate each robot to guard each of S_1, S_2, and $(S_3 \cup G_3 \cup S_4)$. Without loss of generality, let $C_1^* = S_1$, $C_2^* = S_2$, and $C_3^* = (S_3 \cup G_3 \cup S_4)$. This means that G_3 is covered by C_3^* and not skipped by $\mathcal{C}^*$. In this case, $len(C_1^*) = len(C_2^*) = len(C_3^*) = len(S_1)$.

To see that this must be the case, suppose on the contrary that G_3 is skipped and let $\mathcal{C} = \{C_1, C_2, C_3\}$ be an alternative cover. By Lemma Theorem 4, an optimal cover must skip G_3 entirely. In this case, some C_j, say C_1, must have its left end point[1] coincide with the right end point of of G_3 (the point where G_3 meets S_4). Then C_1 must cover S_4 and G_4; otherwise, C_2 and C_3 must cover $S_1 \cup S_2 \cup S_3$, which makes $len(C_2) + len(C_3) \geq len(S_1 \cup S_2 \cup S_3) > 2len(S_1)$ and $\mathcal{C}$ a worse cover than $\mathcal{C}^*$. By symmetry, similarly, some C_j, say C_3, must have its right end point coincide with the left end point of G_3 and cover S_3 and G_2. However, this means that both G_2 and G_4 are covered by $\mathcal{C}$. Even if G_1 is skipped, this makes $len(C_1 \cup C_2 \cup C_3) = len(S_4 \cup G_4 \cup S_1 \cup S_2 \cup G_2 \cup S_3) > len(S_1 \cup S_2 \cup S_3 \cup G_3 \cup S_4) > 3len(S_1)$, again making $\mathcal{C}$ sub-optimal. By the pigeonhole principle, at least one of the C_1, C_2, or C_3 must be longer than $len(S_1)$. Therefore, skipping $G_{max} = G_3$ in this case leads to a sub-optimal cover. The only other alternative for an optimal cover with $n = 3$ is to have $C^* = \{S_1, S_2, (S_3 \cup G_3 \cup S_4)\}$. □

[1]In this chapter, for a non-circular segment or gap, its left end point is defined as the *limit point* along the counterclockwise direction along the perimeter and its right end point is defined as the limit point in the clockwise direction along the perimeter. So, in Figure 2.4, for S_1, its left end point touches G_4 and its right end point touches G_1.

Proposition Theorem 7 implies that in allocating robots to guard a perimeter $P \subset \partial R$, an algorithm cannot simply start by excluding the longest component from $\partial R \backslash P$ and then the next largest, and so on. This makes solving OPG more challenging. Referring back to Figure 2.1, if the top gap is skipped by the cover, then the three robots on the right side of the perimeter (two orange and one green) need to cover the part of the perimeter between the two hexagons. This will cause ℓ^* to increase.

On the other hand, for an optimal cover $\mathcal{C}^* = \{C_1^*, \ldots, C_n^*\}$ of P, some $C_j^* \in \mathcal{C}^*$ must have at least one of its end point aligned with an endpoint of a component S_k of P (assuming that $P \subsetneq \partial R$).

Proposition 8. *For an optimal cover $\mathcal{C}^* = \{C_1^*, \ldots, C_n^*\}$ of a perimeter $P = S_1 \bigcup \ldots \bigcup S_q \subset \partial R = S_1 \bigcup G_1 \bigcup \ldots \bigcup S_q \bigcup G_q$, for some $S_i \subset P$ and $C_j^* \in C^*$, their right (or left) endpoints must coincide.*

Proof. By Lemma Theorem 4, for any $G_k \subset \partial R \backslash P$, and $C_j^* \in \mathcal{C}^*$, $G_k \cap C_j^* = G_k$ or $G_k \cap C_j^* = \emptyset$. Since at least one G_k, $1 \leq k \leq q$, must be skipped by $C_1^* \cup \ldots C_n^*$, some C_j^*, $1 \leq j \leq n$ must have its right endpoint aligned with the right endpoint of S_k, which is on the left of G_k. Following the same argument, some $C_{j'}^*$ and $S_{k'}$ must have the same left endpoints. □

Proposition Theorem 8 suggests that we may attempt to cover a perimeter P starting from an endpoint of S_1, S_2, and so on. Indeed, as we will show in Section subsection 2.1.4, an efficient algorithm can be designed exploiting this important fact.

Guarding Multiple Regions

In a multiple region setup, there is one additional level of complexity: the number of robots that will be assigned to an individual region is no longer fixed. This introduces another set of variables $n_1, \ldots, n_m$ with $n_1 + \ldots + n_m = n$ and n_i, $1 \leq i \leq m$ being the number of robots allocated to guard ∂R_i. For a fixed n_i, the results derived for a single region, i.e.,

Propositions Theorem 6–Theorem 8 continue to hold.

2.1.4 Efficient Algorithms for Perimeter Guarding

In presenting algorithms for OPG, we begin with the case where each perimeter $P_i \in \mathcal{P}$ has a single connected component (i.e., P_i is homeomorphic to $\mathbb{S}^1$ or $[0, 1]$). Then, we work on the general single region case where the only perimeter is composed of $q > 1$ connected components, before moving to the most general multiple regions case.

2.1.5 Perimeters Containing Single Components

When there is a single perimeter P, the solution is straightforward with $\ell^* = len(P)/n$. With ℓ^* determined, $\mathcal{C}^*$ is also readily computed.

In the case where there are $m > 1$ regions, let the optimal distribution of the n robots among the m regions are given by $n_1^*, \ldots, n_m^*$. For a given region R_i, the n_i^* robots must each guard a length $\ell_i = len(P_i)/n_i^*$. At this point, we observe that for at least one region, say R_i, the corresponding ℓ_i must be maximal, i.e., $\ell_i = \ell^*$. The observation directly leads to a naive strategy for finding ℓ^*: for each R_i, one may simply try all possible $1 \leq n_i \leq n$ and find the maximum $len(P_i)/n_i$ that is feasible, i.e., $n - n_i$ robots can cover all other $R_{i'}$, $i' \neq i$, with each robot covering no more than $len(P_i)/n_i$. Denoting this $len(P_i)/n_i$ as ℓ_i^c and the corresponding n_i as n_i^c, the smallest ℓ_i^c overall $1 \leq i \leq m$ is then ℓ^*.

The basic strategy mentioned above works and runs in polynomial time. However, it is possible to carry out the computation much more efficiently if the longest P_i is examined first. Without loss of generality, assume that P_1 is the longest perimeter, i.e., $len(P_1) \geq len(P_i)$ for all $1 \leq i \leq m$. Recall that n_1^c is the number of robots allocated to P_1 that yields ℓ_1^c, it must hold that

$$\frac{len(P_1)}{n_1^c + 1} < \ell^* \leq \frac{len(P_1)}{n_1^c} = \ell_1^c. \tag{2.5}$$

For an arbitrary P_i, simple manipulating of (Equation 2.5) yields

$$\frac{len(P_i)}{(n_1^c+1)\frac{len(P_i)}{len(P_1)}} < \ell^* \leq \frac{len(P_i)}{n_1^c\frac{len(P_i)}{len(P_1)}}. \tag{2.6}$$

This means that we only need to consider $n_i^c \in \left[\lceil n_1^c\frac{len(P_i)}{len(P_1)}\rceil, \lfloor(n_1^c+1)\frac{len(P_i)}{len(P_1)}\rfloor\right]$. However, since P_1 is the longest perimeter, $\frac{len(P_i)}{len(P_1)} \leq 1$. Therefore, the difference between the two denominators of (Equation 2.6) is no more than 1, i.e.,

$$(n_1^c+1)\frac{len(P_i)}{len(P_1)} - n_1^c\frac{len(P_i)}{len(P_1)} \leq 1.$$

When $len(P_i) \neq len(P_1)$, $(n_1^c+1)\frac{len(P_i)}{len(P_1)} - n_1^c\frac{len(P_i)}{len(P_1)} < 1$ and there are two possibilities. One of these is $\lceil n_1^c\frac{len(P_i)}{len(P_1)}\rceil = \lfloor(n_1^c+1)\frac{len(P_i)}{len(P_1)}\rfloor$, which leaves a single possible candidate for n_i^c. The other possibility is $\lceil n_1^c\frac{len(P_i)}{len(P_1)}\rceil = \lfloor(n_1^c+1)\frac{len(P_i)}{len(P_1)}\rfloor + 1$, in which case there is actually no valid candidate for n_i^c. That is, after computing n_1^c and ℓ_1^c, if $len(P_i) = len(P_i)$ then no computation is needed for P_i. If $len(P_i) < len(P_1)$ then we only need to check at most one candidate for n_i^c.

Additional heuristics can be applied to reduce the required computation. First, in finding n_1^c, we may use bisection (binary search) over $[1, m]$ since if a given n_1 is infeasible, any $n_1' < n_1$ cannot be feasible either because $len(P_1)/n_1 < len(P_1)/n_1'$. Second, let $\ell = (\sum_{1\leq i\leq m} len(P_i))/n$, it holds that $\ell_i^c \geq \ell^* \geq \ell$. This means that for each $1 \leq i \leq m$, it is not necessary to try any $n_i > \lfloor\frac{len(P_i)}{\ell}\rfloor$. Third, if a candidate ℓ_i^c is at any time larger than the current candidate for ℓ^*, that i does not need to be checked further. We only use the first and the third in our implementation since the second does not help much once the bisection is applied. The pseudo code is outlined in Algorithm algorithm 1. Note that we assume the problem instance is feasible ($n \geq m$), which is easy to check.

It is straightforward to verify that Algorithm algorithm 1 runs in time $O(m\log n + m^2)$. The $O(m\log n)$ comes from the **while** loop, which calls the function IsFeasible(ℓ_i^c, n_i^c, i) $\log n$ time. The function checks whether the current ℓ_i^c is feasible for perimeters other

than P_i (note that it is assumed that ISFEASIBLE$(\cdot)$ has access to the input to Algorithm algorithm 1 as well). This is done by computing for $i' \neq i$, $n_{i'} = \lceil len(P_{i'})/\ell_i^c \rceil$ and checking whether $\sum_{i' \neq i} n_{i'} \leq n - n_i^c$. The $O(m^2)$ term comes from the **for** loop. The running time of Algorithm algorithm 1 may be further reduced by noting that the **for** loop examines $(m-1)$ candidate ℓ_i^c. These ℓ_i^c can be first computed and sorted, on which bisection can be applied. This drops the main running time to $O(m(\log n + \log m))$. This second bisection is not reflected in Algorithm algorithm 1 to keep the logic and notation more straightforward. If we also consider input complexity, an additional $O(\sum_{1 \leq i \leq m} M_i)$ is needed to compute $len(P_i)$ from the raw polygonal input and an additional $O(n)$ time is needed for generating the actual locations for the n robots. The total complexity is then $O(m(\log n + \log m) + \sum_{1 \leq i \leq m} M_i + n)$.

Algorithm 1: MULTIREGIONSINGLECOMP

Input : $P_1, \ldots, P_m$: each P_i a continuous polygonal line; assume that P_1 is a longest perimeter
n: the number of robots
Output : ℓ^*, i^*: the optimal coverage and the i realizing it

$\ell^* \leftarrow \infty$; $i^* \leftarrow 1$;
%Compute n_1^c and initial ℓ^*.
$n_1^{min} \leftarrow 1$; $n_1^{max} \leftarrow n$; $n_1^c \leftarrow 1$;
while $n_1^{min} \neq n_1^{max}$ **do**
  $n_1 \leftarrow \lceil \frac{n_1^{min} + n_1^{max}}{2} \rceil$; $\ell_1 \leftarrow \frac{len(P_1)}{n_1}$;
  if ISFEASIBLE$(\ell_1, n_1, 1)$ **then**
    $\ell^* \leftarrow \ell_1$; $n_1^c \leftarrow n_1$; $n_1^{min} \leftarrow n_1$;
  else
    $n_1^{max} \leftarrow n_1 - 1$;
  end
end
%Optimize ℓ^* over all $2 \leq i \leq m$.
for $i \in \{2, \ldots, m\}$ **do**
  $n_i^- = \lceil \frac{n_1^c len(P_i)}{len(P_1)} \rceil$; $n_i^+ = \lfloor \frac{(n_1^c+1) len(P_i)}{len(P_1)} \rfloor$; $\ell_i \leftarrow \frac{len(P_i)}{n_i^+}$;
  if $n_i^- == n_i^+$ *and* ISFEASIBLE(ℓ_i, n_i^+, i) *and* $\ell_i < \ell^*$ **then**
    $\ell^* \leftarrow \ell_i$; $i^* \leftarrow i$;
  end
end
return ℓ^*, i^*

2.1.6 Single Perimeter Containing Multiple Components

In computing ℓ^* for a single perimeter P with multiple connected components, assume that P is composed of q maximal connected components $S_1, \ldots, S_q$ (e.g., Figure 2.3), leaving $G_1, \ldots, G_q$ as the gaps on ∂R. Given an optimal cover $\mathcal{C}^* = \{C_1^*, \ldots, C_n^*\}$, by Proposition Theorem 8, we may assume that the left end point of some C_j^*, $1 \leq j \leq n$ coincides with the left end point of some S_k, $1 \leq k \leq q$. We then look at the right end point of C_j^*. If it does not coincide with the right end point of some $S_{k'}$ (k and k' may or may not be the same), it must coincide with the left end point of C_{j+1}^*. Continuing like this, eventually we will hit some $C_{j'}^*$ where the right end point of $C_{j'}^*$ coincides with the right end point of some $S_{k'}$. Within a partitioned subset $C_j^*, \ldots, C_{j'}^*$, the coverage of each robot is minimized when $len(C_j^*) = \ldots = len(C_{j'}^*)$. Because $\ell^* = len(C_j^*)$ for some $1 \leq j \leq n$, at least one of the subsets must have all robots cover exactly a length of ℓ^*. These two key observations are summarized as follows.

Theorem 9. *Let $\mathcal{C}^* = \{C_1^*, \ldots, C_n^*\}$ be a solution to an* OPG *instance with a single perimeter $P = S_1 \cup \ldots \cup S_q$ and gaps $G_1, \ldots, G_q$. Then, $\mathcal{C}^*$ may be partitioned into disjoint subsets with the following properties*

1. *the union of the individual elements from any subset forms a continuous line segment,*
2. *the left end point of such a union coincides with the left end point of some S_k, $1 \leq k \leq q$,*
3. *the right end point of such a union coincides with the right end point of some $S_{k'}$, $1 \leq k' \leq q$, and*
4. *the respective unions of elements from any two subsets are disjoint, i.e., they are separated by at least one gap.*

Moreover, for at least one such subset, $\{C_j^*, \ldots, C_{j'}^*\}$, it holds that $\ell^* = len(C_j^*) = \ldots = len(C_{j'}^*)$.

A baseline algorithm. In the example from Figure 2.1, $\mathcal{C}^*$ is partitioned into two subsets satisfying the conditions stated in Theorem Theorem 9. The theorem provides a way for computing ℓ^*. For fixed $1 \leq k, k' \leq q$, denote the part of ∂R between S_k and $S_{k'}$ following a clockwise direction (with S_k and $S_{k'}$ included) as $S_{k-k'}$. Theorem Theorem 9 says that for some k, k', $len(S_{k-k'}) = n^*_{k-k'}\ell^*$ for some integer $n^*_{k-k'} \in [1, n]$. We may find k', k', and $n^*_{k-k'}$, ℓ^* by exhaustively going through all possible k, k', and $n^c_{k-k'}$ (as a candidate of $n^*_{k-k'}$). For each combination of k, k' and $n^c_{k-k'}$, we can compute a

$$\ell^c_{k-k'} = \frac{len(S_{k-k'})}{n^c_{k-k'}} \tag{2.7}$$

and check $\ell^c_{k-k'}$'s feasibility. The largest feasible $\ell^c_{k-k'}$ is ℓ^*.

For checking feasibility of a particular $\ell^c_{k-k'}$, i.e., whether $\ell^c_{k-k'}$ is long enough for the rest of the robots to cover the rest of the perimeter, we simply *tile* $(n - n^c_{k-k'})$ copies $\ell^c_{k-k'}$ over the rest of the perimeter, starting from $S_{(k' \mod q)+1}$. As an example, see Figure 2.5 where $n = 6$ robots are to cover the perimeter (in red, with five components $S_1, \ldots, S_5$). Suppose that the algorithm is currently working with S_{1-2} (i.e., $k = 1$ and $k' = 2$). If $n^c_{1-2} = 2$, then $\ell^c_{1-2} = len(S_{1-2})/2$. Each of the five green line segments $C_1, \ldots, C_5$ in the figure has this length. As visualized in the figure, it is possible to cover $P \backslash S_{1-2}$ with three more robots, which is no more than $n - n^c_{1-2} = 4$. Therefore, this ℓ^c_{1-2} is feasible; note that it is not necessary to exhaust all $n = 6$ robots. In the figure, C_3 covers the entire S_3 and G_3, as well as part of S_4. The rest of S_4 is covered by C_4. As C_4 is tiled, it ends in the middle of G_4, so C_5 starts at the beginning of S_5. On the other hand, if $n^c_{1-2} = 3$, the resulting ℓ^c_{1-2} (each of the orange line segments has this length) is infeasible as part of S_5 is now left uncovered.

The tiling-based feasibility check takes $O(q)$ time since there are at most q segments to tile; it takes constant time to tile each segment using a given length. Let us denote this feasibility check IsTilingFeasiblePartial(k, k', $n^c_{k-k'}$), we have obtained an algorithm

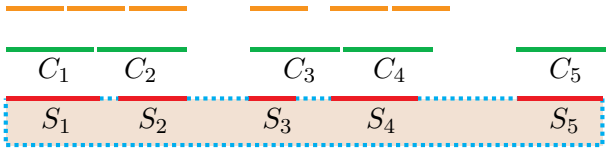


Figure 2.5: An illustration of the feasibility check of ℓ^c_{1-2}. The single rectangular region and the perimeter (five red segments S_1–S_5) are shown at the bottom. The orange and green line segments show two potential covers.

that runs in $O(nq^3)$ times since it needs to go through all $1 \leq k \leq q$, $1 \leq k' \leq q$, and $1 \leq n^c_{k-k'} \leq n$ and for each combination of k, k', and $n^c_{k-k'}$, it makes a call to IsTilingFeasiblePartial($\cdot$). While a $O(nq^3)$ running time is not bad, we can do significantly better.

A much faster algorithm. In the baseline algorithm, for each $k - k'$ combination, up to n candidate $n^c_{k-k'}$ may be attempted. To gain speedups, the first phase of the improved algorithm reduces the range of ℓ^* to limit the choice of $n^c_{k-k'}$. As a necessity, a feasibility check given only a length ℓ is introduced. That is, given an ℓ, a check is done to see whether n robots are sufficient for covering P. This feasibility check is performed in a way similar to the tiling process from IsTilingFeasiblePartial($\cdot$) but now k and k' are not specified. Therefore, we need to try all S_k, $1 \leq k \leq q$ as the possible starting segment for the tiling. Let us denote this procedure IsTilingFeasibleFull(ℓ), which runs in $O(q^2)$.

With IsTilingFeasibleFull(ℓ), starting from the initial bounds for ℓ^* given in Proposition Theorem 6, we can narrow the bound to be arbitrarily small, using bisection, since ℓ^* is the minimum feasible ℓ. To do this, we start with ℓ as the middle point of initial lower bound ℓ_{min} and upper bound ℓ_{max}, and run IsTilingFeasibleFull(ℓ). If ℓ is feasible, the upper bound is lowered to ℓ. Otherwise, the lower bound is raised to ℓ. In doing this, our goal in the first phase of the faster algorithm is to reduce the range for ℓ^* so that there is only a constant number of choices for $n^c_{k-k'}$. We now derive when the bisection process should end.

Assume that when the bisection ends, the lower and upper bounds are ℓ^f_{min} and ℓ^f_{max}.

Then, for some $\ell^c_{k-k'}$ (see (Equation 2.7)), it holds that

$$\ell^f_{min} \leq \ell^c_{k-k'} = \frac{len(S_{k-k'})}{n^c_{k-k'}} \leq \ell^f_{max}.$$

Rearranging the above yields

$$\frac{len(S_{k-k'})}{\ell^f_{max}} \leq n^c_{k-k'} \leq \frac{len(S_{k-k'})}{\ell^f_{min}}.$$

To reduce the number of possible $n^c_{k-k'}$ to at most 1, we want

$$\frac{len(S_{k-k'})}{\ell^f_{min}} - \frac{len(S_{k-k'})}{\ell^f_{max}} < 1,$$

which is the same as requiring

$$\ell^f_{max} - \ell^f_{min} < \frac{\ell^f_{max}\ell^f_{min}}{len(S_{k-k'})}. \tag{2.8}$$

Noting that $\ell^f_{min}, \ell^f_{max} \geq \ell_{min}$ and $len(S_{k-k'}) < len(\partial R)$, Equation (Equation 2.8) is ensured by

$$\ell^f_{max} - \ell^f_{min} < \frac{\ell^2_{min}}{len(\partial R)} = \frac{\left[\sum_{1\leq k\leq q} len(S_k)\right]^2}{n^2 len(\partial R)} \tag{2.9}$$

Equation (Equation 2.9) gives the stopping criteria used for refining the bounds for ℓ^*. After completing the first phase, the faster algorithm moves to the second phase of actually pinning down ℓ^*. In this phase, instead of checking $\ell^c_{k-k'}$ one by one, we collect $\ell^c_{k-k'}$ for all possible combinations of k, k'. Because the first phase already ensures for each k, k' combination there is at most one pair of $n^c_{k-k'}$ and $\ell^c_{k-k'}$, there are at most q^2 total candidates. After all candidates are collected, they are sorted and another bisection is performed over these sorted candidates. Feasibility check is done using IsTilingFeasiblePartial(·). The complete algorithm is given in Algorithm algorithm 2. Note that ℓ^{min} and ℓ^{max}, which change as the algorithm runs, are not the same as the fixed ℓ_{min} and ℓ_{max} from Proposi-

tion Theorem 6.

Algorithm 2: SINGLEREGIONMULTICOMP

Input : $\partial R = S_1 \cup G_1 \cup \ldots \cup S_q \cup G_q$: a single boundary with the perimeter $P = S_1 \cup \ldots \cup S_q$.
n: the number of robots
Output: ℓ^*, k^*, k'^*: the optimal coverage and the pair of k and k' that realize the optimal coverage

```
%Phase one: narrow the range of ℓ*.
```
$\ell^{min} \leftarrow \frac{\sum_{1\le k\le q} len(S_k)}{n}$, $\ell^{max} \leftarrow \frac{len(\partial R)-len(G_{max})}{n}$;
while $\ell^{max} - \ell^{min} > \frac{[\sum_{1\le k\le q} len(S_k)]^2}{n^2 len(\partial R)}$ **do**
  $\ell \leftarrow \frac{\ell^{max}+\ell^{min}}{2}$;
  (ISTILINGFEASIBLEFULL(ℓ)? $\ell^{max} \leftarrow \ell$: $\ell^{min} \leftarrow \ell$);
end
```
%Phase two: pin down ℓ*.
```
$sm \leftarrow []$; `%`sm `is a sorted map.`
for $k, k' \in \{1, \ldots, q\}$ **do**
  $n^{max}_{k-k'} \leftarrow \lfloor \frac{len(S_{k-k'})}{\ell^{min}} \rfloor$; $n^{min}_{k-k'} \leftarrow \lceil \frac{len(S_{k-k'})}{\ell^{max}} \rceil$;
  for $n^c_{k-k'} \in \{n^{min}_{k-k'}, \ldots, n^{max}_{k-k'}\}$ **do**
    sm.put($\frac{len(S_{k-k'})}{n^c_{k-k'}}$, $(n^c_{k-k'}, \frac{len(S_{k-k'})}{n^c_{k-k'}}, k, k')$);
  end
end
$\ell^* \leftarrow \infty$; $k^* \leftarrow 0$; $k'^* \leftarrow 0$;
while sm.size() > 1 **do**
```
  %Extract the element from sm in the middle.
```
  $(n^c, \ell^c, k, k') \leftarrow sm$.middleValue();
  if ISTILINGFEASIBLEPARTIAL(k, k', n^c) **then**
    $\ell^* \leftarrow \ell^c$; $k^* \leftarrow k$; $k'^* \leftarrow k'$;
    $sm \leftarrow sm$.range(sm.minKey(), ℓ^c);
  else
    $sm \leftarrow sm$.removeRange(sm.minKey(), ℓ^c);
  end
end
return ℓ^*, k^*, k'^*

In terms of running time, the first **while** loop starts with $\ell^{max} - \ell^{min} = \frac{len(\partial R)-len(G_{max})}{n} - \frac{\sum_{1\le k\le q} len(S_k)}{n} \le \frac{len(\partial R)}{n}$ and stops when $\ell^{max} - \ell^{min} \le \frac{[\sum_{1\le k\le q} len(S_k)]^2}{n^2 len(\partial R)}$. Therefore, the bisection is executed $\log \frac{n[len(\partial R)]^2}{[\sum_{1\le k\le q} len(S_k)]^2}$ times, which by the assumption that $len(\partial R)$ is a polynomial factor over $\sum_{1\le k\le q} len(S_k)$, is $O(\log(n+q))$. Since each feasibility check takes $O(q^2)$ time, the first **while** loop takes $O(q^2 \log(n+q))$ time. The **for** loops work with a total of $O(q^2)$ candidates and must sort them, taking time $O(q^2 \log q^2) = O(q^2 \log q)$. Then, the second **while** loop bisects $O(q^2)$ candidates and calls ISTILINGFEASIBLEPARTIAL$(\cdot)$ for each check, taking time $O(q \log q^2) = O(q \log q)$. The total running time of Algo-

rithm algorithm 2 is then $O(q^2 \log(n+q) + M + n)$.[2]

2.1.7 Multiple Perimeters Containing Multiple Components

The algorithm for the multiple perimeter case is a direct generalization Algorithm algorithm 2. To facilitate the description, let the perimeter P_i, $1 \leq i \leq m$, contain q_i maximal connected components, i.e., $P_i = S_{i,1} \cup \ldots \cup S_{i,q_i}$ and the boundary $\partial R_i = S_{i,1} \cup G_{i,1} \cup \ldots \cup S_{i,q_i} \cup G_{i,q_i}$. We extend the definition of $S_{k-k'}$ for a single perimeter to $S_{i,k-k'}$ for multiple perimeters. By a straightforward generalization of Theorem Theorem 9 to multiple perimeters, for an OPG instance, the length of some $S_{i,k-k'}$ must be an integer multiple of ℓ^*. Similar to the single perimeter case, we can try all $S_{i,k-k'}$ and for each try all possible $1 \leq n^c_{i,k-k'} \leq n$. This gives us $\ell^c_{i,k-k'} = \frac{len(S_{i,k-k'})}{n^c_{i,k-k'}}$ as candidates for ℓ^*; there are $n(\sum_{1 \leq i \leq m} q_i^2)$ such candidates. For checking the feasibility of $\ell^c_{i,k-k'}$, we may use IsTilingFeasiblePartial$(\cdot)$ for the rest of P_i (taking $O(q_i)$ time) and IsTilingFeasibleFull$(\cdot)$ for all $1 \leq i' \leq m$ and $i' \neq i$ (taking $O(\sum_{1 \leq i' \leq m, i' \neq i} q_{i'}^2)$ time). This yields a baseline algorithm that runs in $O(n(\sum_{1 \leq i \leq m} q_i^2)^2)$ time.

From here, speedups can be obtained as in the single perimeter case using the same reasoning. This yields a two-phase algorithm, which we call MultiRegionMultiComp, that runs in $O((\sum_{1 \leq i \leq m} q_i^2) \log(n + \sum_{1 \leq i \leq m} q_i) + \sum_{1 \leq i \leq m} M_i + n)$.

2.1.8 Performance Evaluation and Applications

Our evaluation first verifies the algorithms' running time matches the claimed bounds. Then, two practical scenarios are illustrated to show how OPG may be adapted to applications.

[2] We note that the assumption that $len(\partial R)$ is a polynomial factor over $\sum_{1 \leq k \leq q} len(S_k)$ is not necessary. However, the corresponding analysis becomes much more involved without it. Since the assumption makes practical sense and also due to space limit, the more general result is omitted from the current chapter. Many additional interesting but non-essential details, including this one, will be included in an extended version.

2.1.9 Algorithm Performance

In the performance results presented here, a data point is the average from 10 randomly generated OPG instances. All algorithms are implemented in Python 2.7, and all experiments are executed on an Intel® Xeon® CPU at 3.0GHz. Some additional performance results can be found in the Appendix.

For the case of m perimeters each containing a single segment, for each $1 \leq i \leq m$, we set $len(\partial R_i) = 1$ and let $len(P_i)$ be uniformly distributed in $(0, 1]$. Figure 2.6 shows the result for an example with $m = 10$ and $n = 30$. For various values of m, n, the running time of MULTIREGIONSINGLECOMP is summarized in Table Table 2.1, which scales very well with m and n (note that the $n \leq m$ case does not make much sense here).

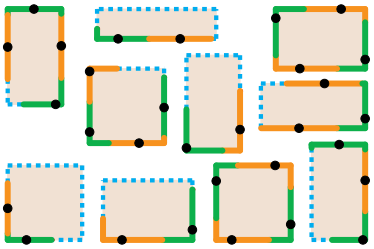


Figure 2.6: An example problem instance when $m = 10$ and $n = 30$. The black dots indicate deployed robot locations; the green and orange lines indicate the coverage.

m \ n	10^8	10^9	10^{10}	10^{11}	10^{12}
10^6	1.152	1.442	1.508	1.652	1.617
10^7	13.963	17.281	18.796	20.354	20.627
10^8	NA	176.115	223.186	227.250	230.000

Table 2.1: MULTIREGIONSINGLECOMP running time (seconds)

For the case of a single perimeter with multiple components, a random polygon is generated on which $2q$ points are randomly sampled that yield q segments (that form the perimeter) and q gaps. An example instance and the optimal solution with $q = 3$ and $n = 10$ is illustrated in Figure 2.7. The computation time for various q and n combinations is given in Table Table 2.2.

For multiple perimeters containing multiple components, m polygons are created with $len(\partial R_i)$ randomly distributed in $[1, 10]$. For setting q_i, we fix a q and let $q_i = q(0.5 +$

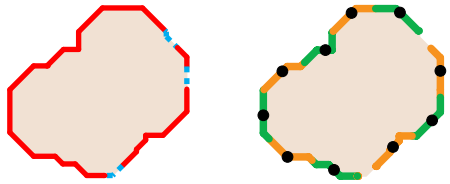


Figure 2.7: An example problem instance when $q = 3$ and $n = 10$. In this case, the optimal cover actually covers one gap.

q \ n	10^1	10^2	10^3	10^4	10^5
10^2	0.013	0.015	0.016	0.016	0.017
10^3	1.363	1.595	1.622	1.634	1.641
10^4	159.404	188.497	210.492	212.473	212.780

Table 2.2: SINGLEREGIONMULTICOMP computation time (seconds)

$random(0, 1)$). Representative computation results of MULTIREGIONMULTICOMP are listed in Table Table 2.3.

q	n	m				
		10	20	30	40	50
10^1	10^3	0.047	0.063	0.076	0.091	0.108
10^2	10^3	2.191	3.771	5.523	7.707	9.369
10^2	10^4	7.105	9.619	11.369	12.760	15.107

Table 2.3: MULTIREGIONMULTICOMP computation time (seconds)

Due to limited space, only selected essential performance data is presented here. More complete performance data and associate analysis can be found in the Appendix.

2.1.10 Two Applications Scenarios

Securing a perimeter. As a first application, consider a situation where a crime has just been committed at the Edinburgh Castle (see Figure 2.8). The culprit remains in the confines of the castle but is mixed within many guests at the scene. As the situation is being investigated and suppose that the brick colored buildings are secured, guards (either personnel or a number of drones) may be deployed to ensure the culprit does not escape by climbing down the castle walls. Using SINGLEREGIONMULTICOMP, a deployment plan can be quickly computed given the amount of resources at hand so that each guard only needs to secure a minimum length along the castle walls. Figure 2.8 shows the optimal

deployment plan for 15 guards.

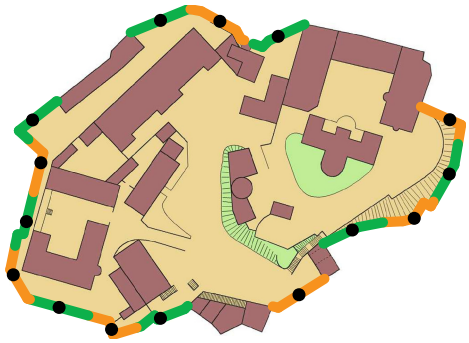


Figure 2.8: Optimal deployment of 15 guards around walls of the Edinburgh Castle. The brick colored structures are buildings that create gaps along the boundary.

Fire monitoring. In a second application, consider Figure 2.9 where a forest fire has just been put out in multiple regions. As there is still some chance that the fire may rekindle and spread, for prevention, a team of firefighters is to be deployed to watch for the possible spreading of the fire. Here, in addition to using MULTIREGIONMULTICOMP to compute optimal locations for deploying the firefighters, we also generate minimum time trajectories for the firefighters to reach their target locations while avoiding going through the dangerous forests. This is done via solving a bottleneck assignment problem [75]. Note that the lake region creates gaps that cannot be traveled by the firefighters; this can be handled by making these gaps infinitely large. Figure 2.9 shows the optimal locations for 34 firefighters. Animations of the deployment process and other test cases can be found in the accompanying video.

2.1.11 Conclusion and Discussion

In this chapter, we propose the OPG problem to model the allocation of large robotic swarms to cover complex 1D topological domains with optimality guarantees. For all variants under the OPG formulation umbrella, we have developed highly efficient algorithms for solving OPG exactly. In addition to rigorous proofs backed by formal analysis, extensive computational experiments further confirm the effectiveness of these algorithms. Moreover, practical relevance of OPG is demonstrated through the integration of OPG

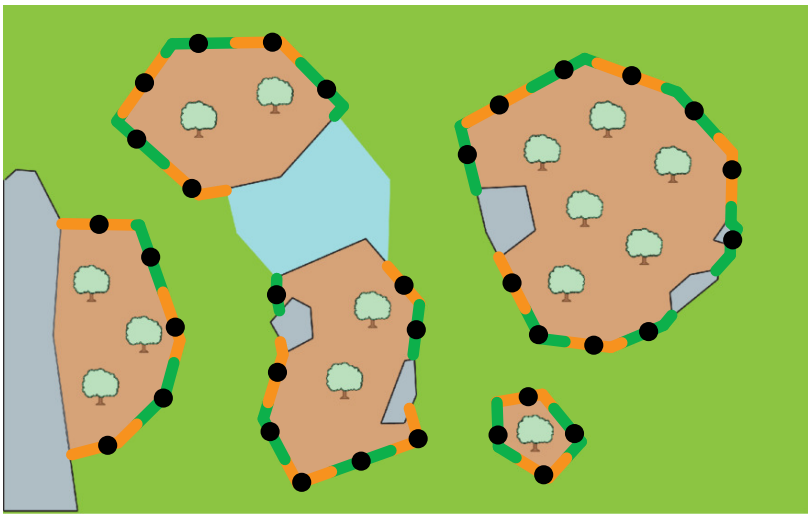


Figure 2.9: Optimal deployment of 34 firefighters for forest fire rekindling prevention.

into realistic task (assignment) and motion planning scenarios.

The study raises many additional interesting open questions; we mention a few here. First, the approach taken in this work is a *centralized* one where decision is made at the global level. It would be highly interesting to explore whether the same can be achieved with *decentralized* methods, which have many advantages. For example, it may be the case that the gaps along the boundaries are not known *a priori* and must be measured by the robots. In such cases, a centralized plan can be hard to come by. Second, as mentioned in Section subsection 2.1.2, the current OPG formulation assumes that the robots are confined to the boundaries $\partial\mathcal{R}$, which is one of many possible choices in terms of the robots' sensing and/or motion capabilities. In future study, we plan to examine additional practical robot sensing and motion models. Third, as exact optimal algorithms are emphasized here, issues including uncertainty and robustness have not been touched in the current treatment, which are important elements when it comes to the deployment of a robotic swarm to tackle real-world challenges.

2.2 Perimeter Guarding with Heterogeneous Defenders

2.2.1 Introduction

Consider the scenario where many mobile guards (or sensors) are to be deployed to patrol the perimeter of some 2D regions (Figure 2.10) against intrusion, where each guard may effectively cover a continuous segment of a region's boundary. When part of a boundary need not be secured, e.g., there may already be some existing barriers (the blue segments in Figure 2.10), optimally distributing the robots so that each robot's coverage is minimized becomes an interesting and non-trivial computational task [76]. It is established [76] that, when the guards have the same capabilities, the problem, called the *optimal perimeter guarding* (OPG), resides in the complexity class P (polynomial time class), even when the robots must be distributed across many different boundaries.

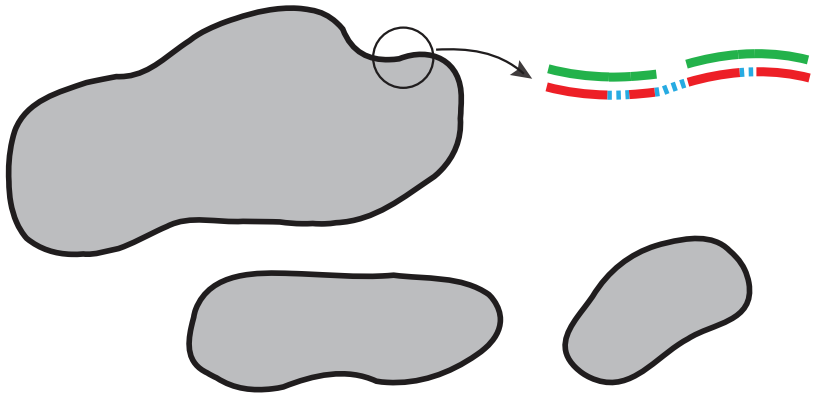


Figure 2.10: A scenario where boundaries of three (gray) regions must be secured. Zooming in on part of the boundary of one of the regions (the part inside the small circle), portions of the boundary (the red segments) must be guarded while the rest (the blue dotted segments) does not need guarding. For example, the zoomed-in part of the boundary may be monitored by two mobile robots, each patrolling along one of the green segments.

In this work, we investigate a significantly more general version of OPG where the mobile guards may be heterogeneous. More specifically, two formulations with different guarding/sensing models are addressed in our study. In the first, the number of available robots is fixed where robots of different types have a fixed ratio of capability (e.g., one

type of robot may be able to run faster or may have better sensor). The guarding task must be evenly divided among the robots so that each robot, regardless of type, will not need to bear a too large coverage/capability ratio. This formulation is denoted as *optimal perimeter guarding with limited resources* or OPG$_{LR}$. In the second, the number of robots is unlimited; instead, for each type, the sensing range is fixed with a fixed associated cost. The goal here is to find a deployment plan so as to fully cover the perimeter while minimizing the total cost. We call this the *optimal perimeter guarding with minimum cost* problem, or OPG$_{MC}$.

Unlike the plain vanilla version of the OPG problem, we establish that both OPG$_{LR}$ and OPG$_{MC}$ are NP-hard when the number of robot types is part of the problem input. They are, however, at different hardness levels. OPG$_{LR}$ is shown to be NP-hard in the strong sense, thus reducing the likelihood of finding a fully polynomial time approximation scheme (FPTAS). Nevertheless, for the more practical case where the number of robot types is a constant, we show that OPG$_{LR}$ can be solved using a pseudo-polynomial time algorithm with reasonable scalability. On the other hand, we show that OPG$_{MC}$ is weakly NP-hard through the establishment of a pseudo-polynomial time algorithm for OPG$_{MC}$ with arbitrary number of robot types. We further show that, when the number of robot types is fixed, OPG$_{MC}$ can be solved in polynomial time through a fixed-parameter tractable (FPT) approach. This paragraph also summarizes the main contributions of this work.

A main motivation behind our study of the OPG formulations is to address a key missing element in executing autonomous, scalable, and optimal robot deployment tasks. Whereas much research has been devoted to multi-robot motion planning [77, 36] with great success, e.g., [78, 47, 48, 52, 53, 54], existing results in the robotics literature appear to generally assume that a target robot distribution is already provided; the problem of how to effectively generate optimal deployment patterns is largely left unaddressed. It should be noted that control-based solutions to the multi-agent deployment problem do exist, e.g.,[40, 41, 20, 43, 35, 46, 79], but the final solutions are obtained through many local iterations and

generally do not come with global optimality guarantees. For example, in [20], Voronoi-based iterative methods compute locally optimal target formations for various useful tasks. In contrast, this work, as well as [76], targets the scalable computation of globally optimal solutions.

As a coverage problem, OPG may be characterized as a 1D version of the well-studied Art Gallery problems [12, 13], which commonly assume a sensing model based on line-of-sight visibility[15]; the goal is to ensure that every point in the interior of a given region is visible to at least one of the deployed guards. Depending on the exact formulation, guards may be placed on boundaries, corners, or the interior of the region. Not surprisingly, Art Gallery problems are typically NP-hard [16]. Other than Art Gallery, 2D coverage problems with other sensing models, e.g., disc-based, have also been considered [17, 18, 19, 20, 21, 23], where some formulations prevent the overlapping of individual sensing ranges [17, 18] while others seek to ensure a full coverage which often requires intersection of sensor ranges. In viewing of these studies, this study helps painting a broader landscape of sensor coverage research.

In terms of structural resemblance, OPG_{LR} and OPG_{MC} share many similarities with *bin packing* [80] and other related problems. In a bin packing problem, objects are to be selected to fit within bins of given sizes. Viewing the segments (the red ones in Figure 2.10) as bins, OPG seeks to place guards so that the segments are fully contained in the union of the guards' joint coverage span. In this regard, OPG is a dual problem to bin packing since the former must overfill the bins and the later cannot fully fill the bins. In the extreme, however, both bin packing and OPG converge to a **SUBSET SUM** [81] like problem where one seeks to partition objects into halves of equal total sizes, i.e., the objects should fit exactly within the bins. With an additional cost term, OPG_{MC} has further similarities with the **KNAPSACK** problem [82], which is weakly NP-hard [83].

The rest of the chapter is organized as follows. In Section subsection 2.2.2, mathematical formulations of the two OPG variants are fully specified. In Section subsection 2.2.3,

both OPG_{LR} and OPG_{MC} are shown to be NP-hard. Despite the hardness hurdles, in Section subsection 2.2.6, multiple algorithms are derived for OPG_{LR} and OPG_{MC}, including effective implementable solutions for both. In Section subsection 2.2.9, we perform numerical evaluation of selected algorithms and demonstrate how they may be applied to address multi-robot deployment problems. We discuss and conclude our study in Section subsection 2.2.13. Please see https://youtu.be/6gYL0_B3YTk for an illustration of the problems and selected instances/solutions.

2.2.2 Preliminaries

Let $\mathcal{W} \subset \mathbb{R}^2$ be a compact (closed and bounded) two-dimensional workspace. There are m pairwise disjoint *regions* $\mathcal{R} = \{R_1, \ldots, R_m\}$ where each region $R_i \subset \mathcal{W}$ is homeomorphic to the closed unit disc, i.e., there exists a continuous bijection $f_i : R_i \to \{(x, y) \mid x^2 + y^2 \leq 1\}$ for all $1 \leq i \leq m$. For a given region R_i, let ∂R_i be its (closed) boundary (therefore, f_i maps ∂R_i to the unit circle $\mathbb{S}^1$). With a slight abuse of notation, define $\partial\mathcal{R} = \{\partial R_1, \ldots, \partial R_m\}$. Let $P_i \subset \partial R_i$ be the part of ∂R_i that is accessible, specifically, not blocked by obstacles in $\mathcal{W}$. This means that each P_i is either a single closed curve or formed by a finite number of (possibly curved) line segments. Define $\mathcal{P} = \{P_1, \ldots, P_m\} \subset \mathcal{W}$ as the *perimeter* of $\mathcal{R}$ which must be *guarded*. More formally, each P_i is homeomorphic to a compact subset of the unit circle (i.e., it is assumed that the maximal connected components of P_i are closed line segments). For a given P_i, each one of its maximal connected component is called a *perimeter segment* or simply a *segment*, whereas each maximal connected component of $\partial R_i \backslash P_i$ is called a *perimeter gap* or simply a *gap*. An example setting is illustrated in Figure 2.11 with two regions.

After deployment, some number of robots are to *cover* the perimeter $\mathcal{P}$ such that a robot j is assigned a continuous closed subset C_j of some $\partial R_i, 1 \leq i \leq m$. All of $\mathcal{P}$ must be *covered* by $\mathcal{C}$, i.e., $\bigcup_{P_i \in \mathcal{P}} P_i \subset \bigcup_{C_j \in \mathcal{C}} C_j$, which implies that elements of $\mathcal{C}$ need not intersect on their interiors. Hence, it is assumed that any two elements of $\mathcal{C}$ may share at

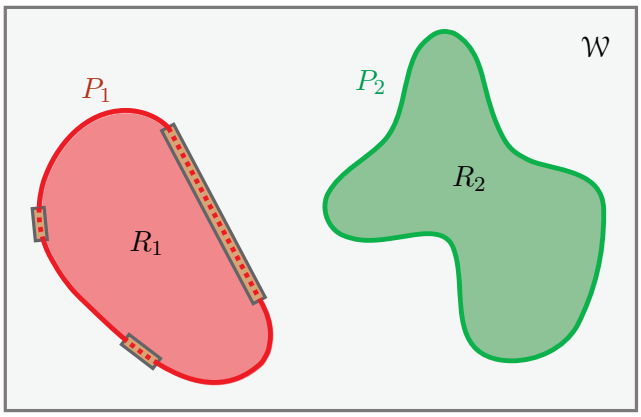


Figure 2.11: An example of a workspace $\mathcal{W}$ with two regions $\{R_1, R_2\}$. Due to three *gaps* on ∂R_1, marked as dotted lines within long rectangles, $P_1 \subset \partial R_1$ has three *segments* (or maximal connected components); $P_2 = \partial R_2$ has a single segment with no gap.

most their endpoints. Such a $\mathcal{C}$ is called a *cover* of $\mathcal{P}$. Given a cover $\mathcal{C}$, for a $C_j \in \mathcal{C}$, let $len(C_j)$ denote its length (more formally, measure).

To model heterogeneity of the robots, two models are explored in this study. In either model, there are t types of robots. In the first model, the number of robots of each type is fixed to be $n_1, \ldots, n_t$ with $n = n_1 + \cdots + n_t$. For a robot $1 \leq j \leq n$, let τ_j denote its type. Each $1 \leq \tau \leq t$ type of robots has some level of *capability* or *ability* $a_\tau \in \mathbb{Z}^+$. We wish to balance the load among all robots based on their capabilities, i.e., the goal is to find cover $\mathcal{C}$ for all robots such that the quantity

$$\max_{C_j \in \mathcal{C}} \frac{len(C_j)}{a_{\tau_j}},$$

which represents the largest coverage-capacity ratio, is minimized. We note that when all capacities are the same, e.g., $a_\tau = 1$ for all robots, this becomes the standard OPG problem studied in [76]. We call this version of the perimeter guarding problem *optimal perimeter guarding with limited resources* or OPG$_{LR}$. The formal definition is as follows.

Problem 2 (Optimal Perimeter Guarding with Limited Resources (OPG$_{LR}$))**.** *Let there be t types of robots. For each type $1 \leq \tau \leq t$, there are n_τ such robots, each having the same capability parameter a_τ. Let $n = n_1 + \cdots + n_t$. Given the perimeter set $\mathcal{P} = \{P_1, \ldots, P_m\}$ of a set of 2D regions $\mathcal{R} = \{R_1, \ldots, R_m\}$, find a set of n continuous line segments $\mathcal{C}^* =$*

$\{C_1^*, \ldots, C_n^*\}$ *such that* $\mathcal{C}^*$ *covers* $\mathcal{P}$*, i.e.,*

$$\bigcup_{P_i \in \mathcal{P}} P_i \subset \bigcup_{C_j^* \in \mathcal{C}^*} C_j^*, \tag{2.10}$$

such that a C_j^* *is covered by robot* j *of type* τ_j*, and such that, among all covers* $\mathcal{C}$ *satisfying* (Equation 2.10)*,*

$$\mathcal{C}^* = \operatorname*{argmin}_{\mathcal{C}} \max_{C_j \in \mathcal{C}} \frac{len(C_j)}{a_{\tau_j}}. \tag{2.11}$$

Whereas the first model caps the number of robots, the second model fixes the maximum coverage of each type of robot. That is, for each robot type $1 \leq \tau \leq t$, n_τ, the number of robots of type τ, is unlimited as long as it is non-negative, but each such robot can only cover a maximum length of ℓ_τ. At the same time, using each such robot incurs a cost of c_τ. The goal here is to guard the perimeters with the minimum total cost. We denote this problem *optimal perimeter guarding with minimum cost* or OPG_{MC}.

Problem 3 (Optimal Perimeter Guarding with Minimum Cost (OPG_{MC}))**.** *Let there be* t *types of robots of unlimited quantities. For each robot of type* $1 \leq \tau \leq t$*, it can guard a length of* $\ell_\tau \in \mathbb{Z}^+$ *with a cost of* $c_\tau \in \mathbb{Z}^+$*. Given the perimeter set* $\mathcal{P} = \{P_1, \ldots, P_m\}$ *of a set of 2D regions* $\mathcal{R} = \{R_1, \ldots, R_m\}$*, find a set of* $n = n_1 + \cdots + n_t$ *continuous line segments* $\mathcal{C}^* = \{C_1^*, \ldots, C_n^*\}$ *where* n_τ *such segments are guarded by type* τ *robots, such that* $\mathcal{C}^*$ *covers* $\mathcal{P}$*, i.e.,*

$$\bigcup_{P_i \in \mathcal{P}} P_i \subset \bigcup_{C_j^* \in \mathcal{C}^*} C_j^*, \tag{2.12}$$

such that a C_j^* *is covered by robot* j *of type* τ_j*, i.e.,* $C_j^* \leq \ell_{\tau_j}$*, and such that, among all covers* $\mathcal{C}$ *satisfying* (Equation 2.12)*,*

$$\mathcal{C}^* = \operatorname*{argmin}_{\mathcal{C}} \sum_{1 \leq \tau \leq t} n_\tau c_\tau. \tag{2.13}$$

2.2.3 Computational Complexity for Variable Number of Robot Types

We explore in this section the computational complexity of OPG_{LR} and OPG_{MC}. Both problems are shown to be NP-hard with OPG_{LR} being strongly NP-hard. We later confirm

that OPG$_{MC}$ is weakly NP-hard (in Section subsection 2.2.6).

2.2.4 Strong NP-hardness of OPG$_{LR}$

When the number of types t is a variable, i.e., t is not a constant and may be arbitrarily large, OPG$_{LR}$ is shown to be NP-hard via the reduction from 3-**PARTITION** [84]:

PROBLEM: 3-**PARTITION**

INSTANCE: A finite set A of $3m$ elements, a bound $B \in \mathbb{Z}^+$, and a "size" $s(a) \in \mathbb{Z}^+$ for each $a \in A$, such that each $s(a)$ satisfies $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a) = mB$.

QUESTION: Is there a partition of S into m disjoint subsets $S_1, \ldots, S_m$ such that for $1 \leq i \leq m$, $\sum_{a \in S_i} s(a) = B$?

3-**PARTITION** is shown to be NP-complete in the strong sense[85], i.e., it is NP-complete even when all numeric inputs are bounded by a polynomial of the input size.

For the reduction, it is more convenient to work with a decision version of the OPG$_{LR}$ problem, denoted as D-OPG$_{LR}$. In the D-OPG$_{LR}$ problem, a_τ is the actual length robot type τ covers. That is, the coverage length of a robot is fixed. The D-OPG$_{LR}$ problem is specified as follows.

PROBLEM: D-OPG$_{LR}$

INSTANCE: t types of robots where there are n_τ robots for each type $1 \leq \tau \leq t$; $n = n_1 + \cdots + n_t$. A robot of type τ has a coverage capacity a_τ. A set of perimeters $\mathcal{P} = \{P_1, \ldots, P_m\}$ of a set of 2D regions $\mathcal{R} = \{R_1, \ldots, R_m\}$.

QUESTION: Is there a deployment of n disjoint subsets $C_1, \ldots, C_n$ of $\{\partial R_1, \ldots, \partial R_m\}$ such that $P_1 \cup \ldots \cup P_m \subset C_1 \cup \ldots \cup C_n$, where C_j is a continuous segment for all $1 \leq j \leq n$, and for each $1 \leq j \leq n$, there is a unique robot whose type τ, $1 \leq \tau \leq t$ satisfies $a_\tau \geq len(C_j)$?

Theorem 10. OPG$_{LR}$ *is strongly NP-hard.*

Proof. A polynomial reduction from 3-**PARTITION** to D-OPG$_{LR}$ is constructed by a re-

striction of D-OPG$_{LR}$. Given a 3-**PARTITION** instance with former notations, we apply several restrictions on D-OPG$_{LR}$: *(i)* there are $3m$ types of robot and there is a single robot for each type, i.e., $n_\tau = 1$ for $1 \leq \tau \leq t$, so $n = t = 3m$ *(ii)* the $3m$ capacities $a_1, \ldots, a_{3m}$ are set to be equal to $s(a)$ for each of the $3m$ elements $a \in A$, and *(iii)* there are $3m$ perimeters and each perimeter P_i is continuous and $len(P_i) = B$ for all $1 \leq i \leq m$.

With the setup, the reduction proof is straightforward. Clearly, the 3-**PARTITION** instance admits a partition of A into $S_1, \ldots, S_m$ such that $\sum_{a \in S_i} s(a) = B$ for all $1 \leq i \leq m$ if and only if a valid depolyment exists in the corresponding D-OPG$_{LR}$ instance. It is clear that the reduction from 3-**PARTITION** to D-OPG$_{LR}$ is polynomial (in fact, linear). Based on the reduction and because 3-**PARTITION** is strongly NP-hard, so is D-OPG$_{LR}$ and OPG$_{LR}$. □

Remark 11. *One may also reduce weakly NP-hard problems, e.g.,* **PARTITION** *[81], to* OPG$_{LR}$ *for variable number of robot types t. Being strongly NP-hard,* OPG$_{LR}$ *is unlikely to admit pseudo-polynomial time solutions for variable t. This contrasts with a later result which provides a pseudo-polynomial time algorithm for* OPG$_{LR}$ *for constant t, as one might expect in practice where robots have limited number of types. We also note that Theorem Theorem 10 continues to hold for a single perimeter with multiple segments, each having a length B in previous notation, separated by "long" gaps. Obviously,* D-OPG$_{LR}$ *is in NP, thus rendering it NP-complete.*

2.2.5 NP-hardness of OPG$_{MC}$

The minimum cost OPG variant, OPG$_{MC}$, is also NP-hard, which may be established through reduction from the **SUBSET SUM** problem [81]:

PROBLEM: **SUBSET SUM**

INSTANCE: A set B with $|B| = n$ and a weight function $w : B \to \mathbb{Z}^+$, and an integer W.

QUESTION: Is there a subset $B' \subseteq B$ such that $\sum_{b \in B'} w(b) = W$?

Theorem 12. OPG_{MC} *is NP-hard.*

Proof. Given a **SUBSET SUM** instance, we construct an OPG_{MC} instance with a single perimeter containing a single segment with length L to be specified shortly. Let there be $t = 2n$ types of robots. For $1 \leq i \leq n$, let robot type $2i-1$ have $\ell_{2i-1} = c_{2i-1} = w(b_i) + (2^{n+1} + 2^i)W'$ and let robot type $2i$ have $\ell_{2i} = c_{2i} = (2^{n+1} + 2^i)W'$. Here, W' can be any integer number no less than $\sum_{b \in B} w(b)$. Set $L = W + (n2^{n+1} + 2^n + \ldots + 2^1)W'$. We ask the "yes" or "no' decision question of whether there are robots that can be allocated to have a total cost no more than L (equivalently, equal to L, as the cost density c_τ / l_τ is always 1).

Suppose the **SUBSET SUM** instance has a yes answer that uses a subset $B' \subseteq B$. Then, the OPG_{MC} instance has a solution with cost L that can be constructed as follows. For each $1 \leq i \leq n$, a single robot of type $2i-1$ is taken if $b_i \in B'$. Otherwise, a single robot of type $2i$ is taken. This allocation of robots yields a total length and cost of L.

For the other direction, we first show that if the OPG_{MC} instance is to be satisfied, it can only use a single robot from type $2i-1$ or $2i$ for all $1 \leq i \leq n$. First, if more than n robots are used, then the total cost exceeds $(n+1)2^{n+1}W' > L$ as $W \leq W'$. Similarly, if less than n robots are used, the total length is at most $(n-1)2^{n+1}W' + (2^{n+1} - 1)W' + W' < L$. Also, to match the $(2^n + \ldots + 2)W'$ part of the cost, exactly one robot from type $2i-1$ or $2i$ for all $1 \leq i \leq n$ must be taken. Now, if the OPG_{MC} decision instance has a yes answer, if a robot of type $2i-1$ is used, let $b_i \in B$ be part of B', which constructs a B' that gives a yes answer to the **SUBSET SUM** instance. □

Remark 13. *It is also clear that the decision version of the* OPG_{MC} *problem is NP-complete. The* **SUBSET SUM** *is a weakly NP-hard problem that admits a pseudo-polynomial time algorithm [83]. As it turns out,* OPG_{MC}*, which shares similarities with* **SUBSET SUM** *and* **KNAPSACK** *(in particular,* **UNBOUNDED KNAPSACK** *[82]), though NP-hard, does admit a pseudo-polynomial time algorithm as well.*

2.2.6 Exact Algorithms for OPG$_{LR}$ and OPG$_{MC}$

In this section, we describe three exact algorithms for solving the two variations of the OPG problem. First, we present a pseudo-polynomial time algorithm for OPG$_{LR}$ when the number of robot types, t, is a fixed constant. Given that OPG$_{LR}$ is strongly NP-hard, this is in a sense a best possible solution. For OPG$_{MC}$, in addition to providing a pseudo-polynomial algorithm for arbitrary t, which confirms that OPG$_{MC}$ is weakly NP-hard, we also provide a polynomial time approximation scheme (PTAS). We then further show the possibility of solving OPG$_{MC}$ in polynomial time when t is a fixed constant. We mention that our development in this section focuses on the single perimeter case, i.e., $m = 1$, as the generalization to arbitrary m is straightforward using techniques described in [76]. With this in mind, we also provide the running times for the general setting with arbitrary m but refer the readers to [76] on how these running times can be derived.

For presenting the analysis and results, for the a perimeter P that we work with, assume that it has q perimeter segments $S_1, \ldots, S_q$ that need to be guarded; these segments are separated by q gaps $G_1, \ldots, G_q$. For $1 \leq i, i' \leq q$, define $S_{i \sim i'} = S_i \cup G_i \cup S_{i+1} \cup \ldots \cup G_{i'-1} \cup S_{i'}$ where i' may be smaller than i (i.e., $S_{i \sim i'}$ may wrap around G_q), For the general case with m perimeters, assume that a perimeter P_i has q_i segments.

2.2.7 Pseudo-Polynomial Time Algorithm for OPG$_{LR}$ with Fixed Number of Robot Types

We set to develop an algorithm for OPG$_{LR}$ for arbitrary t, the number of robot types; the algorithm runs in pseudo-polynomial time when t is a constant. At a higher level, our proposed algorithm works as follows. First, our main effort here goes into deriving a feasibility test for D-OPG$_{LR}$ as defined in Section subsection 2.2.4. With such a feasibility test, we can then find the optimal $\frac{len(C_j)}{a_{\tau_j}}$ in (Equation 2.11) via binary search. Let us denote the optimal value of $\frac{len(C_j)}{a_{\tau_j}}$ as ℓ^*.

Feasibility Test for D-OPG$_{LR}$

The feasibility test for D-OPG$_{LR}$ essentially tries different candidate ℓ to find ℓ^*. Our implementation uses ideas similar to the pseudo-polynomial time algorithm for the **KNAPSACK** problem which is based on dynamic programming (DP). In the test, we work with a fixed starting point on P, which is set to be the counterclockwise end point of a segment S_i, $1 \leq i \leq q$. Essentially, we maintain a t dimensional array M where dimension τ has a size of $n_\tau + 1$. An element of the array, $M[n'_1] \dots [n'_t]$, holds the maximal distance starting from S_i that can be covered by n'_1 type 1 robots, n'_2 type 2 robots, and so on. The DP procedure OPG-LR-FEASIBLE (i, ℓ), outlined in Algorithm algorithm 3, incrementally builds this array M. For convenience, in the pseudo code, $M[\vec{x}]$ denotes an element of M with $\vec{x}$ being a t dimensional integer vector.

Algorithm 3: OPG-LR-FEASIBLE (i, ℓ)

Data: $n_1, \dots, n_t, a_1, \dots, a_t, S_1, \dots, S_q, G_1, \dots, G_q$
Result: **true** or **false**, indicating whether $S_1, \dots, S_q$ can be covered

```
1  Initialize M as a t dimensional array with dimension τ having a size of n_τ + 1;
2  ℓ_τ ← a_τ ℓ for all 1 ≤ τ ≤ t;
3  for x⃗ ∈ [0, n_1] × ··· × [0, n_t] do
4  |   M[x⃗] ← 0;
5  |   for j = 1 to t do
6  |   |   if x⃗_j = 0 then continue;
7  |   |   x⃗' ← x⃗; x⃗'_j ← x⃗'_j − 1;
8  |   |   M[x⃗] ← max(M[x⃗], INC (M[x⃗'], ℓ_j));
9  |   end
10 end
11 return M[n_1] ... [n_t] ≥ len(S_{i∼i−1});
```

In Algorithm algorithm 3, the procedure INC (L, ℓ) checks how much of the perimeter P can be covered when an additional coverage length ℓ is added, assuming that a distance of L (starting from some S_i) is already covered. An illustration of how INC (L, ℓ) works is given in Figure 2.12.

By simple counting, the complexity of the algorithm is $O(q \cdot t \cdot \Pi_{\tau=1}^{t}(n_\tau + 1))$. However, the amortized complexity of INC $(\cdot)$ for each τ is $O(q + n_\tau)$; the algorithm thus runs in

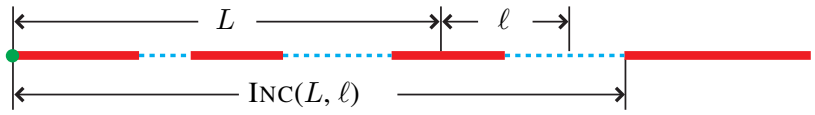


Figure 2.12: Suppose starting from the fixed left point, a length of L on the boundary is successfully guarded by a group of robots. Then, a robot with coverage capacity ℓ is appended to the end of the group of robots to increase the total guarded distance. In the figure, the added additional capacity ℓ can fully cover the third red segment plus part of the third (dashed) gap. Because there is no need to cover the rest of the third gap, INC(L, ℓ) extends to the end of the gap.

$O(t \cdot \Pi_{\tau=1}^{t}(n_\tau + 1) + q \cdot \sum_{\tau=1}^{t} \Pi_{\tau' \neq \tau}(n_{\tau'} + 1))$, which is pseudo-polynomial for fixed t. After trying every possible starting position i with OPG-LR-FEASIBLE (i, ℓ), for a fixed candidate ℓ, D-OPG$_{LR}$ is solved in $O(q \cdot t \cdot \Pi_{\tau=1}^{t}(n_\tau + 1) + q^2 \cdot \sum_{\tau=1}^{t} \Pi_{\tau' \neq \tau}(n_{\tau'} + 1))$.

Solving OPG$_{LR}$ *using Feasibility Test for* D-OPG$_{LR}$

Using OPG-LR-FEASIBLE (i, ℓ) as a subroutine to check feasibility for a given ℓ, bisection can be applied over candidate ℓ to obtain ℓ^*. For completing the algorithm, one needs to establish when the bisection will stop (notice that, even though we assume that $a_\tau \in \mathbb{Z}^+$, for each $1 \leq \tau \leq t$, ℓ^* need not be an integer).

To derive the stop criterion, we note that given the optimal ℓ^*, there must exist some $S_{i \sim i'}$ that is "exactly" spanned by the allocated robots. That is, assume that $S_{i \sim i'}$ is covered by n'_1 of type 1 robots and n'_2 of type 2 robots, and so on, then

$$\ell^* = \frac{len(S_{i \sim i'})}{\sum_{1 \leq \tau \leq t} a_\tau \cdot n'_\tau}. \tag{2.14}$$

(Equation 2.14) must hold for some $S_{i \sim i'}$ because if not, the solution is not tight and can be further improved. Therefore, the bisection process for locating ℓ^* does not need to go on further after reaching a certain granularity[76]. With this established, using similar techniques from [76] (we omit the technical detail as it is quite complex but without additional new ideas beyond beside what is already covered in [76]), we could prove that the full algorithm needs no more than $O(q \log(\sum_\tau n_\tau + q)$ calls to OPG-LR-FEASIBLE (i, ℓ). This directly implies that OPG$_{LR}$ also admits a pseudo-polynomial algorithm for fixed t.

Multiple Perimeters

Also using techniques developed in [76], the single perimeter result can be readily generalized to multiple perimeters. We omit the mechanical details of the derivation and point out that the computational complexity in this case becomes $\tilde{O}((m-1)\cdot((\Pi_{\tau=1}^{t} n_\tau)/\max_\tau n_\tau)^2 + \sum_{k=1}^{m}(t \cdot q_k \cdot \Pi_{\tau=1}^{t}(n_\tau+1) + q_k^2 \sum_{\tau=1}^{t} \Pi_{\tau' \neq \tau}(n_{\tau'}+1)))$.

2.2.8 Polynomial Time Algorithm for OPG$_{MC}$ with Fixed Number of Robot Types

The solution to OPG$_{MC}$ will be discussed here. A method based on DP will be provided first, which leads to a polynomial time algorithm for a fixed number of robot types and a pseudo-polynomial time algorithm when the number of robot types is not fixed. For the latter case, a polynomial time approximation scheme (PTAS) will also be briefly described.

Dynamic Programming Procedure for OPG$_{MC}$

When no gaps exist, the optimization problem becomes a covering problem as follows. Let c_τ, ℓ_τ, n_τ correspond to the cost, coverage length, and quantity of robot type τ, respectively, and let total length to cover be L. We are to solve the optimization problem

$$\min \sum_\tau c_\tau \cdot n_\tau \quad s.t. \quad \sum_\tau \ell_\tau \cdot n_\tau \geq L, n_\tau \geq 0. \tag{2.15}$$

Let the solution to the above integer programming problem be SOL(L). Notice that, for $S_{i\sim i'} := \{S_i,\ G_i, \ldots, G_{i'-1}, S_{i'}\}$, the minimum cost cover is by either: *(i)* covering the total boundary without skipping any gaps, or *(ii)* skipping or partially covering some gap, for example $G_k, i \leq k \leq j-1$. In the first case, the minimum cost is exactly SOL($\lceil len(S_{i\sim(i+k)}\rceil$). In the second case, the optimal structure for the two subsets of perimeter segments $S_{i\sim k}$ and $S_{(k+1)\sim j}$ still holds. This means that the continuous perimeter segments $S_{i\sim j}$ can be divided into two parts, each of which can be treated separately. This leads to a DP approach for OPG$_{MC}$. With $M[i][j]$ denoting the minimum cost to cover

$S_{i\sim j}$, the DP recursion is given by

$$M[i][j] = \min(\textsc{Sol}(\lceil len(S_{i\sim j})\rceil), \min_k(M[i][k] + M[k+1][j]))$$

The DP procedure is outlined in Algorithm algorithm 4. In the pseudo code, it is assumed that indices of M are modulo q, e.g., $M[2][q+1] \equiv M[2][1]$. tmp is a temporary variable.

Algorithm 4: OPG-MC-DP

Data: $\ell_1, \ldots, \ell_t, c_1, \ldots, c_t, S_1, \ldots, S_q, G_1, \ldots, G_q$
Result: c^*, the minimum covering cost

```
M ← a q × q matrix; c* ← ∞;
for k ← 0 to q − 1 do
    for i ← 1 to q do
        tmp ← SOL(⌈len(S_{i∼(i+k)})⌉);
        for j ← i to i + k − 1 do
            tmp ← min(tmp, M[i][j] + M[j + 1][i + k]);
        end
        M[i][i + k] ← c;
        if k = q − 1 then c* ← min(c*, M[i][i + k]);
    end
end
return c*;
```

A Polynomial Time Algorithm for OPG_{MC} *for a Fixed Number of Robot Types*

We mention briefly that, by a result of Lenstra [86], the optimization problem (Equation 2.15) is in P (i.e., polynomial time) when t is a constant. The running time of the algorithm [86] is however exponential in t.

A Pseudo-polynomial Time Algorithm for Arbitrary t

As demonstrated in the hardness proof, similarities exist between OPG and the **KNAPSACK** problem. The connection actually allows the derivation of a pseudo-polynomial time algorithm for arbitrary t. To achieve this, we use a routine to pre-compute $\textsc{Sol}(L)$, called PRESOLVE(), which is itself a DP procedure similar to that for the **KNAPSACK**

problem. The pseudo code of PRESOLVE() is given in Algorithm algorithm 5. PRESOLVE() runs in time $O(t \cdot \lceil len(\partial R) \rceil))$. Overall, Algorithm algorithm 4 then runs in time $O(q^3 + t \cdot \lceil len(\partial R \rceil))$.

Algorithm 5: PRESOLVE

Data: $\ell_1, \ldots, \ell_t, c_1, \ldots, c_t$
Result: A lookup table for retrieving SOL(L)

```
1  I_max = ⌈len(∂R)⌉; %I_max is an integer.
2  M' ← an array of length I_max + 1; M'[0] ← 0;
3  for L ← 1 to I_max do
4  |   M'[L] ← ∞;
5  |   for τ ← 1 to t do
6  |   |   tmp ← (L < ℓ_τ ? 0 : M'[L − ℓ_τ]) + c_τ;
7  |   |   M'[L] ← min(M'[L], tmp);
8  |   end
9  end
10 return M'
```

With the establishment of a pseudo-polynomial time algorithm for OPG$_{MC}$, we have the following corollary.

Corollary 14. OPG$_{MC}$ *is weakly NP-hard.*

FPTAS for Arbitrary t

When the number of robot types is not fixed, Lenstra's algorithm[86] or its variants no longer run in polynomial time. We briefly mention that, by slight modifications of a FPTAS for **UNBOUNDED KNAPSACK** problem from [87], a FPTAS for OPG$_{MC}$ can be obtained that runs in time $O(q^3 + q^2 \cdot \frac{t}{\epsilon^3})$, where $(1+\epsilon)$ is the approximation ratio for both OPG$_{MC}$ and (Equation 2.15).

Multiple Perimeters

For OPG$_{MC}$, when there are multiple perimeters, e.g., $P_1, \ldots, P_m$, a optimal solution can be obtained by optimally solving OPG$_{MC}$ for each perimeter P_i individually and then put together the solutions.

2.2.9 Performance Evaluation and Applications

In this section, we provide examples illustrating the typical optimal solution structures of OPG_{LR} and OPG_{MC} computed by our DP algorithms. Using an application scenario, solutions to OPG_{LR} and OPG_{MC} are also compared. Then, computational results from extensive numerical evaluations are presented, confirming the effectiveness of these algorithms. The implementation is done using the Python and all computations are performed on an Intel(R) Core(TM) i7-7700 CPU@3.6GHz with 16GiB RAM.

2.2.10 Basic Optimal Solution Structure

Figure 2.13 shows the typical outcome of solving an OPG_{LR} instance with two perimeters ($m = 2$) for two types of robots with $n_1 = 3, a_1 = 5$, and $n_2 = 5, a_2 = 8$. In the figure, the red segments are parts of the two perimeters that must be guarded. The three orange (resp., five green) segments across the two perimeters indicate the desired coverage regions of the three (resp., five) type 1 (resp., type 2) robots. These coverage regions correspond to the optimal solution returned by the DP algorithm. As may be observed, the optimal solution is somewhat complex with robots of both types on each of the two perimeters; a gap on the second boundary also gets covered. The coverage lengths for a robot type are generally different; this is due to adjustments that shrink some robots' coverage. For example, the first perimeter has a very short orange cover because the corresponding perimeter segment is short and gaps around it need not be covered (The adjustment procedure is also shown in the video).

Shifting our attention to OPG_{MC}, Figure 2.14 illustrates the structure of an optimal solution to a problem with three types of robots with capacities and costs being $\ell_1 = 11, c_1 = 2$, $\ell_t = 30, c_2 = 4$, and $\ell_3 = 55, c_3 = 7$, respectively. In this case, the majority of the deployed robots are of type 2 with $\ell_2 = 30, c_2 = 4$. Only one type 1 and one type 3 robots are used. The four perimeter segments are covered by three robot groups. The only type 3 robot guards (the purple segment) across two different perimeter segments. Coverage

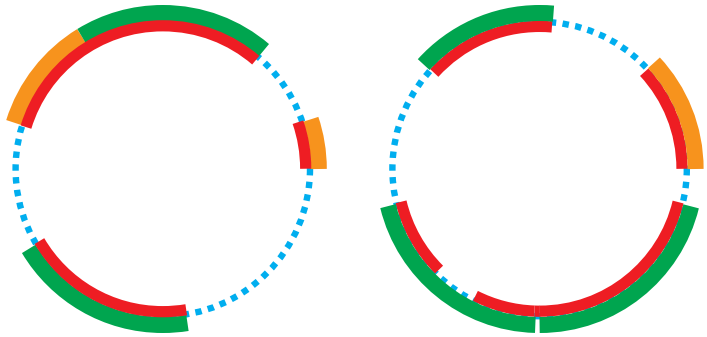


Figure 2.13: An OPG$_{LR}$ problem and an associated optimal solution. The problem has two perimeters and $t = 2$ with n_1=3, n_2=5, a_1=5, a_2=8. The boundaries are shown as circles for ease of illustration.

length adjustment is also performed to avoid the unnecessary coverage of some gaps.

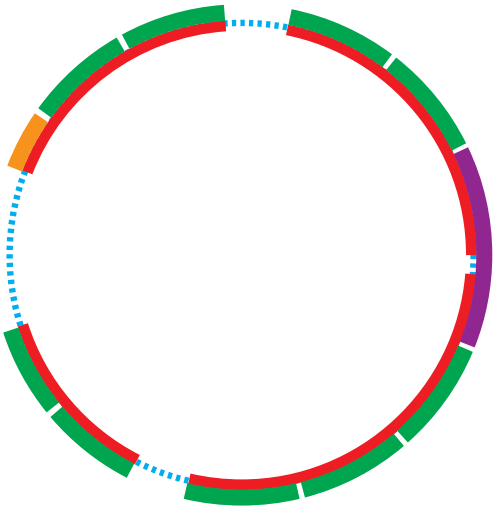


Figure 2.14: An OPG$_{MC}$ problem and an associated optimal solution. The problem has four (red) perimeter segments and three types of robots with $\ell_1 = 11, c_1 = 2$ (orange), $\ell_t = 30, c_2 = 4$ (green), and $\ell_3 = 55, c_3 = 7$ (purple), respectively.

2.2.11 A Robotic Guarding and Patrolling Application

In this subsection, as a potential application, the DP algorithms for OPG$_{LR}$ and OPG$_{MC}$ are employed to solve the problem of securing the perimeter of the Edinburgh castle, an example used in [76]. As shown in Figure 2.15 (minus the orange and green segments showing the solutions), the central region of the Edinburgh castle has tall buildings on its boundary (the blocks in brick red); these parts of the boundary are the gaps that do not

need guarding. In the figure, the top sub-figure shows the optimal solution for an OPG_{LR} instance and an OPG_{MC} instance with a total of 11 robots. The bottom sub-figure is a slightly updated OPG_{MC} instance with slightly higher c_2.

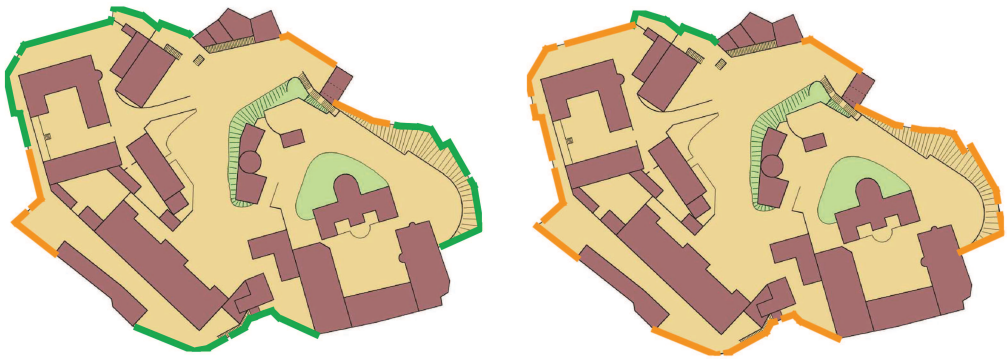


Figure 2.15: [left] OPG_{LR} solution with $n_1 = 4, n_2 = 7, c_1 : c_2 = 2 : 3$ and OPG_{MC} solution with $\ell_1 = 150, c_1 = 100, \ell_2 = 225, c_2 = 145$, and total boundary 3058. Cost of OPG_{MC} solution is 1415. [right] OPG_{MC} solution with $\ell_1 = 150, c_1 = 100, \ell_2 = 225, c_2 = 155$. Cost of solution ($13$ type 1, 1 type 2) is 1455. In both solutions, covers by type 1 (resp., type 2) robots are shown in orange (resp., green).

It can be observed that the results, while having non-trivial structures, make intuitive sense. For the top sub-figure, solutions to both OPG_{LR} and OPG_{MC} (because robot with larger capacity is slightly lower in relative cost) use mainly higher capacity robots to cover longer perimeter segments and use the lower capacity robots mostly fillers. The solution covers a small gap at the bottom. For the bottom sub-figure, while only small changes are made to the cost, because the longer segment is more expensive to use now, the first type of robot is used mainly.

2.2.12 Computational Performance

With Section subsection 2.2.6 fully establishing the correctness and asymptotic complexity of the pseudo-polynomial time algorithms, here, the running time of these algorithms are experimentally evaluated. In doing so, the main goal is demonstrating that, despite the hardness of OPG_{LR} and OPG_{MC}, the proposed algorithms could solve the target problems under reasonably broad settings in a scalable way. For results presented in this subsection,

each data point is an average over 10 randomly generated instances.

The first two numerical evaluations (Table 2.4 and Table 2.5) focus on the running times of the pseudo-polynomial time algorithms for OPG_{LR} over single and multiple perimeters, respectively. In these two tables, t and q are the number of types and the number of segments, respectively. For each type τ, a capacity (a_τ) is randomly sampled as an integer between 1 and 100, inclusive. The number of robots available for each type (n_τ) is sampled uniformly between 5 and 15, inclusive. For the multiple perimeters case, the parameter m represents the number of perimeters for a given instance.

For the single perimeter case (Table 2.4), the results show that the pseudo-polynomial time algorithm is effective for up to five types of robots, for dozens of robots. We expect a more efficient (e.g., C++ based) implementation should be able to effectively handle up to five types of robots with the total number of robots being around a hundred, on a typical PC. This is likely sufficient for many practical applications which have limited types and numbers of robots. Since the algorithm has exponential dependency on t, it becomes less efficient for larger t as expected.

t \ q	5	10	20	30	40	50
2	0.022	0.044	0.131	0.208	0.326	0.516
3	0.281	0.714	1.670	2.577	4.107	4.708
4	5.504	16.07	41.68	71.55	109.9	138.9
5	29.53	75.60	243.6	443.4	528.0	725.0

Table 2.4: Running time in seconds used by the DP algorithm for OPG_{LR} over a single perimeter.

Table 2.5 illustrates the running time of the DP algorithm for OPG_{LR} over multiple perimeters. As can be readily observed, the impact of the number of perimeters m on the running time is relatively small; the number of robot types is still the determining factor for running time. In this case, our proposed solution is effective for t up to 4 and starts to slow down a robot types become larger than 4.

Table Table 2.6 provides performance evaluation of OPG-MC-DP. Since there is no

m \ q	10		20		30	
	t=3	t=4	t=3	t=4	t=3	t=4
2	3.148	133.2	7.077	198.4	10.33	260.0
3	4.828	194.1	10.125	290.6	15.52	376.7
4	6.131	256.8	12.485	381.3	19.75	514.3
5	7.622	321.7	15.355	476.2	24.31	605.8

Table 2.5: Running time in seconds used by the DP algorithm for OPG_{LR} over multiple perimeters.

difference between single and multiple perimeters for OPG_{MC}, only problems with single perimeters are attempted. Here, for each robot type, the cost is an integer randomly sampled between 1 and 20, and the capacity is computed as five times the cost plus a random integer between 1 and 20. In the table, $L = \partial R$, the total length of the entire boundary. Given OPG_{MC}'s lower computational complexity, the DP algorithm, OPG-MC-DP, can effectively deal with over a few hundred types of robots with ease.

t \ L	10^2		10^4		10^6	
	q=20	q=50	q=20	q=50	q=20	q=50
3	0.006	0.064	0.041	0.098	3.040	3.144
10	0.005	0.066	0.094	0.155	9.423	9.409
30	0.009	0.070	0.261	0.320	26.10	28.59
100	0.014	0.077	0.910	0.969	91.28	93.20
300	0.030	0.091	2.652	2.938	275.6	270.7

Table 2.6: Running time in seconds used by OPG-MC-DP algorithm.

2.2.13 Conclusion and Discussions

In this section, we investigate two natural models of optimal perimeter guarding using heterogeneous robots, where one model (OPG_{LR}) limits the number of available robots and the second (OPG_{MC}) seeks to optimize the total cost of coverage.

These formulations have many potential applications. One application scenario we envision is the deployment of multiple agents or robots as "emergency responders" that are constrained to travel on the boundary. An optimal coverage solution will then translate to

minimizing the maximum response time anywhere on the perimeter (the part that needs guarding). The scenario applies to OPG, OPG$_{LR}$, and OPG$_{MC}$.

Another application scenario is the monitoring of the perimeter using robots with different sensing capabilities. A simple heterogeneous sensing model here would be robots equipped with cameras with different resolutions, which may also be approximated as discs of different radii. The model makes sense provided that the region to be covered is much larger than the sensing range of individual robots and assuming that the boundary has relatively small curvature as compared to the inverse of the radius of the smallest sensing disc of the robots. For boundary with relatively small curvature, our solutions would apply well to the sensing model by using the diameter of the sensing disc as the 1D sensing range. As the region to be covered is large, covering the boundary will require much fewer sensors than covering the interior.

On the computational complexity side, we prove that both OPG$_{LR}$ and OPG$_{MC}$ are NP-hard, with OPG$_{LR}$ directly shown to be strongly NP-hard. This is in stark contrast to the homogeneous case, which admits highly efficient low polynomial time solutions [76]. The complexity study also establishes structural similarities between these problems and classical NP-hard problems including 3-**PARTITION**, **KNAPSACK**, and **SUBSET SUM**.

On the algorithmic side, we provide methods for solving both OPG$_{LR}$ and OPG$_{MC}$ exactly. For OPG$_{LR}$, the algorithm runs in pseudo-polynomial time in practical settings with limited types of robots. In this case, the approach is shown to be computationally effective. For OPG$_{MC}$, a pseudo-polynomial time algorithm is derived for the general problem, which implies that OPG$_{MC}$ is weakly NP-hard. In practice, this allows us to solve large instances of OPG$_{MC}$. We further show that a polynomial time algorithm is possible for OPG$_{MC}$ when the types of robots are fixed.

With the study of OPG [76] for homogeneous and heterogeneous cases, some preliminary understanding has been obtained on how to approach complex 1D guarding problems. Nevertheless, the study so far is limited to *one-shot* settings where the perimeters do not

change. In future research, we would like to explore the more challenging case where the perimeters evolve over time, which requires the solution to be dynamic as well. Given the results on the one-shot settings, we expect the dynamic setting to be generally intractable if global optimal solutions are desired, potentially calling for iterative and/or approximate solutions.

We recognize that our work does not readily apply to a visibility-based sensing model, which is also of interest. Currently, we are also exploring covering of the interior using range-based sensing. As with the OPG work, we want to push for optimal or near-optimal solutions when possible.

CHAPTER 3
OPTIMAL SET GUARDING

In this chapter, we consider the problem of using mobile robots equipped with range sensors to guard (1D) perimeters or (2D) regions. Given a bounded polygonal one- or two-dimensional set to be secured, and k mobile robots where robot i's sensor covers a circular region of radius r_i, we seek deployment of the robots so that $\max r_i$ is minimized. That is, we would like to minimize the maximum single-sensor coverage across all sensors. We denote this multi-sensor coverage problem under the umbrella term *optimal set guarding with 2D sensors*, or OSG_{2D}.[1] The specific problem for guarding perimeters (resp., regions) is denoted as *optimal perimeter (resp., region) guarding with 2D sensors*, abbreviated as OPG_{2D} (resp., ORG_{2D}). Beside direct relevance to sensing, surveillance, and monitoring applications using mobile sensors [88, 20, 76], OSG_{2D} applies to other robotics related problem domains, e.g., the deployment of ad-hoc mobile wireless networks [89, 90], in which case an optimal solution to OSG_{2D} provides a lower bound on the guaranteed network strength over the targeted 2D region.

3.1 Introduction

As a summary of the study, on the side of computational complexity, we establish that OPG_{2D} is hard to approximate within a factor of 1.152 even when the perimeter is a simple closed polygonal chain whose length is bounded by the input size, through a reduction from vertex cover on planar bridgeless 3-regular graphs. A unique property of our reduction is that it shows the inapproximability gap remains when each sensor can cover at most two disjoint perimeter segments. The proof also shows that ORG_{2D} is at least

[1] The subscript is placed here to distinguish our setup from the OPG problem studied in [76], which assumes a 1D sensing model.

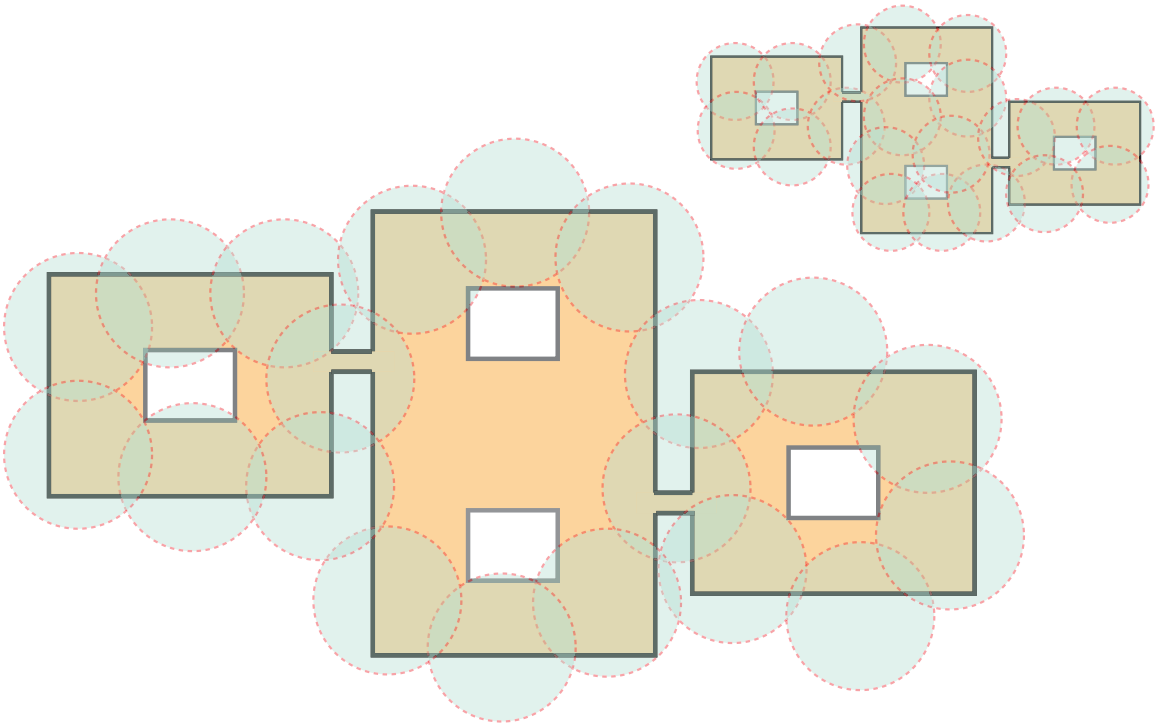


Figure 3.1: An illustration of the OSG_{2D} setup and sample solutions. [center] The background shows the footprint of a building, e.g., an apartment complex. Scenarios may arise that a dangerous criminal might be hiding in the building and we would like to closely monitor the outer boundary of the building. For the setting, the shaded discs provide a near-optimal cover with minimum radii for 20 mobile sensors that fully enclose the outer perimeter, computed using algorithms presented in this work with optimality guarantees. [upper right] A near-optimal solution for guarding the interior of the building footprint minus the four holes.

as hard to approximate. Therefore, no polynomial time algorithm may exist that solves OSG_{2D} to better than the 1.152-optimal lower bound, unless $\text{P} = \text{NP}$. On the algorithmic side, we begin by providing an efficient $(1 + \varepsilon)$ approximation algorithm for a specific class of OPG_{2D} problems in which each mobile sensor must cover a continuous perimeter segment. This implies that the aforementioned inapproximability result on OPG_{2D} under the two-disjoint-segment sensing model is tight. For the general OSG_{2D} problem, we first describe a polynomial time $(2+\varepsilon)$ approximation algorithm as a reasonable approximability upper bound. Then, an integer linear programming (ILP) model is devised that allows the fast computation of highly optimal solutions for fairly large problem instances. Results

described in this paragraph, together with the introduction of $\mathrm{OSG}_{\mathrm{2D}}$ as a practical multi-robot deployment problem focusing on global optimality, constitute the main contributions of this work.

As an intermediate result toward showing the hardness of the simple polygon coverage problem, we also supply a hardness proof of vertex cover on planar bridgeless[2] 3-regular graphs, which may be of independent interest.

Organization. The rest of the chapter is organized as follows. In Section section 3.2, we introduce the $\mathrm{OSG}_{\mathrm{2D}}$ formulation. Section section 3.3 is devoted to establishing that $\mathrm{OSG}_{\mathrm{2D}}$ is hard to approximate to better than 1.152-optimal, providing a theoretical lower bound. In Section section 3.4, focusing on the upper bound, we describe algorithms that for $\mathrm{OSG}_{\mathrm{2D}}$ and the special $\mathrm{OPG}_{\mathrm{2D}}$ variant where a sensor is allowed to cover a continuous perimeter segment. In Section section 3.5, we benchmark the algorithms and illustrate two potential applications. We discuss and conclude the work in Section section 3.6.

3.2 Preliminaries

Let $\mathcal{W} \subset \mathbb{R}^2$ be a polygonal workspace, which may contain one or multiple connected components. A *critical subset* of $\mathcal{W}$ needs to be guarded by k indistinguishable point guards with range sensing capabilities. For example, the workspace may be a forest reserve and the critical subset may be its boundary. Or, the workspace may be a high-security facility, e.g., a prison, and the critical subset the prison yard. The ith guard, $1 \leq i \leq k$, located at $c_i \in \mathbb{R}^2$, can monitor a circular area of radius r_i centered at c_i with r_i being a variable. For example, the guard may be a watchtower equipped with a vision sensor that can detect intruders. As the watchtower's altitude increases, its sensing range also increases; but its monitoring quality will decrease at the same time due to resolution loss. In this study, we seek to compute the optimal strategy to deploy these k guards so that the required sensing range, $\max_i r_i$, could be minimized.

[2]That is, the deletion of any edge does not disconnect the graph.

More formally, we model a connected component of $\mathcal{W}$ as some 2D polygonal region containing zero or more simple polygonal obstacles. For a bounded set $D \subset \mathbb{R}^2$, we define

$$size(k, D) = \min_{c_1,\ldots,c_k \in \mathbb{R}^2} \max_{p \in D} \min_{1 \leq i \leq k} \|c_i - p\|_2$$

and use $B(c, r)$ to denote the disc of radius r centered at a point $c \in \mathbb{R}^2$ (the definition of $size(k, D)$ is used extensively in later sections). Intuitively, $size(k, D)$ represents the minimum radius needed such that there exist k circles with radius $size(k, D)$ that can cover the 2D bounded region D entirely. The main problem studied in this work is:

Problem 4 (Optimal Set Guarding with 2D Sensors)**.** *Given a polygonal workspace $\mathcal{W} \subset \mathbb{R}^2$, let $D \subset \mathcal{W}$ be a critical subset to be guarded by k robots each with a variable coverage radius of r. Find the smallest r and corresponding robot locations $c_1, \ldots, c_k \in \mathbb{R}^2$, such that $D \subset \cup_i B(c_i, r)$.*

For making accurate statements about computational complexity, we make the assumption that the length of $\partial\mathcal{W}$ is bounded by a polynomial with respect to the complexity of $\mathcal{W}$, (i.e. the number of vertices of the polygon).

For convenience, we give specific names to these optimal guarding problems based critical subset types. If the critical subset belongs to $\partial\mathcal{W}$, we denote the problem as *optimal perimeter guarding with 2D sensors* or OPG_{2D}. If the critical subset is $\mathcal{W}$, we denote the problem as *optimal region guarding with 2D sensors* or ORG_{2D}. When there is no need to distinguish, the problem is denoted as *optimal set guarding with 2D sensors* (OSG_{2D}).

As an example, to guard the boundary of a plus-shaped polygon with 5 robots, an optimal solution could be Figure 3.2 where the inner circle covers 4 disconnected boundary segments, such pattern in the optimal solution also renders OPG_{2D} much more difficult than the simplified 1D sensing model studied in [76] (indeed, OPG_{2D} becomes hard to approximate, as will be shown shortly). The solution is also optimal under the ORG_{2D} formulation.

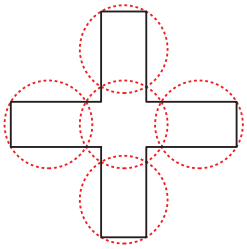


Figure 3.2: An example showing an optimal solution of using five discs to cover the plus-shaped polygon. The solution is optimal for both OPG_{2D} and ORG_{2D} formulations.

3.3 Intractability of Approximate Optimal Guarding of Simple Polygon

In this section, we prove that OSG_{2D} with the set being a simple polygon is strongly NP-hard to approximate within a factor of $\alpha = 1.152$, through a sequence of auxiliary NP-hardness results. First, in Section subsection 3.3.1, we prove an intermediate result that the vertex cover problem is NP-complete on planar bridgeless 3-regular graphs. Next, in Section subsection 3.3.2, starting from a planar bridgeless 3-regular graph, we construct a structure which we call 3-*net* and prove the the problem of finding the minimum coverage radius of the 3-net is NP-hard to approximate within α. Then, in Section subsection 3.3.3, we apply a straightforward reduction to transform the 3-net into a simple polygon to complete the hard-to-approximate proof for OSG_{2D} for simple polygons.

We then further show the inapproximability of the special OPG_{2D} setup when each robot can only guard at most two disjoint perimeter segments (Section subsection 3.3.4), contrasting the FPTAS for the special OPG_{2D} setup when each robot can only guard a continuous perimeter segment in Section subsection 3.4.1.

3.3.1 Vertex Cover on Planar Bridgeless 3-Regular Graph

Our reduction uses the hardness result on the vertex cover problem for planar graphs with maximum degree 3 [91]. Such a vertex cover problem can be fully specified with a 2-tuple (G, k) where $G = (V, E)$ is a planar graph with max degree 3 and k is an integer specifying the allowed number of vertices in a vertex cover. We note that the result has been suggested

implicitly in [92]; we provide an explicit account with a simple proof.

Lemma 15. *Vertex cover on planar bridgeless* 3*-regular graph is NP-complete.*

Proof. For a given planar graph G with max degree 3 and an integer k, we construct a planar bridgeless 3-regular graph G'' and provide an integer k'' such that G has a vertex cover of size k if and only if G'' has a vertex cover of size k''.

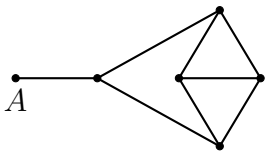


Figure 3.3: A gadget that can be attached to a degree one or two vertices (at the point A) in a max degree 3 graph to make all vertices have degree 3. With each addition of the gadget, we increase the vertex cover by a size of 3, regardless of whether A is part of a vertex cover.

The reduction first makes G 3-regular by attaching (one or two of) the gadget shown in Figure 3.3 to $v \in G$ that are not degree 3. This results in a 3-regular graph G'. For each attached gadget, k is bumped up by 3, i.e., we let k' for G' be $k' = k + 3(3|V(G)| - 2|E(G)|)$. It is straightforward to see that G has a vertex cover size of k if and only if G' has a vertex cover size of k'.

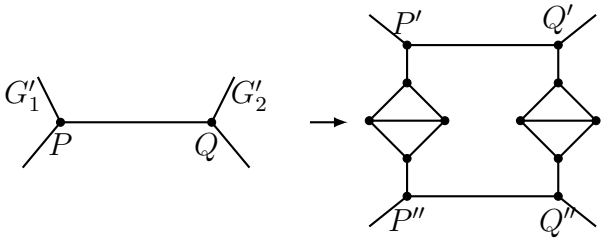


Figure 3.4: Transformation that removes bridge PQ and does not introduce new bridges. The minimum vertex cover number is increased by 6 after each transformation.

In the second and last step, we remove bridges in G'. As in Figure 3.4, for a bridge PQ that divides G' into G'_1 (containing P) and G'_2 (containing Q), we split the bridge edge PQ using the illustrated transformation, which yields a new graph G'' that is planar, bridgeless, and 3-regular, after all bridges are removed this way. For each such augmentation, the size

of the vertex cover is bumped up by six. Let $br(G')$ be the number of bridges in G', G' has a vertex cover of size k' if and only if G'' has a vertex cover of size $k'' = k' + 6br(G')$. This completes the proof. □

3.3.2 Hardness on Optimally Guarding a 3-Net

Starting from a planar cubic graph G, we construct a structure that we call 3-net, T_G, as follows. First, similar to [29], to embed G into the plane, an edge $uw \in E(G)$ is converted to an odd length path $uv_1, v_1v_2, \ldots, v_{2m}w$ where $m > 3$ is an integer. We note that m is different in general for different edges of G. Denote such a path as $u \cdots w$; each edge along $u \cdots w$ is straight and has unit edge length. We also require that each path is nearly straight locally. For a vertex of G with degree 3, e.g., a vertex $u \in V(G)$ neighboring $w, x, y \in V(G)$, we choose proper configurations and lengths for paths, $u \cdots w$, $u \cdots x$, and $u \cdots y$ such that these paths meet at u forming pairwise angles of $2\pi/3$. We denote the resulting graph as G', which becomes the *backbone* of the 3-net T_G.

From here, a second modification is made which completes the construction of T_G. In each previously constructed path $u \cdots w = uv_1 \ldots v_{2m}w$, for each v_iv_{i+1}, $1 \leq i \leq 2m-1$, we add a line segment of length $\sqrt{3}$ that is perpendicular to v_iv_{i+1} such that v_iv_{i+1} and the line segment divide each other in the middle. A graphical illustration is given in Figure 3.5. G' and the bars form the 3-net, which we denote as T_G. An example of transforming K_4 into a *3-net* is given in Figure 3.6.

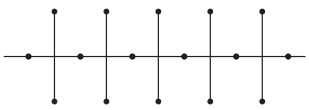


Figure 3.5: Structure within the odd length path and attached perpendicular "bars" with length $\sqrt{3}$. Regarding the representation of such non-integral coordinates in the problem input, we may scale the coordinates to some certain extent and round them to integers so that the relative distance between each other is precise enough for the proof.

Let L be the number of (unit length) edges of G' (i.e., $L = \sum_{uv \in E(G)} len(u \cdots w)$).

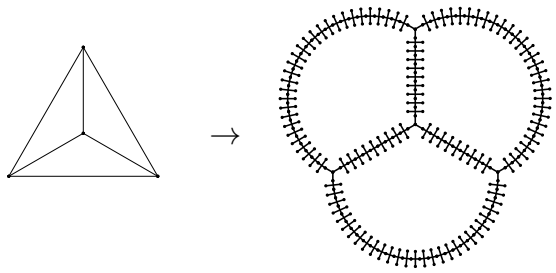


Figure 3.6: Illustration of a 3-net obtained from K_4, the complete graph on 4 vertices.

Lemma 16. *A planar bridgeless* 3*-regular graph* G *has a vertex cover of size* k *if and only if its transformed 3-net* T_G *can be covered by* $K = k + (L - |E(G)|)/2$ *circles of radius approximately* $\alpha = 1.152$*.*

Proof. If G has a vertex cover of size k, then we put k circles of radius 1 at the centers of the corresponding vertices in T_G. For each odd length path $u \cdots w$, since either u or w is already selected as the circle center, applying one coverage pattern shown in Figure 3.7 with $(len(u \cdots w) - 1)/2$ circles will cover the rest of $u \cdots w$ and all bars on it. So, the total number of circles used is K to cover all of T_G.

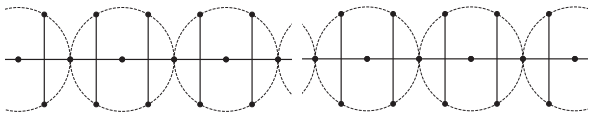


Figure 3.7: Two coverage patterns on an odd length path with robots of range sensing radius 1 which is less than α.

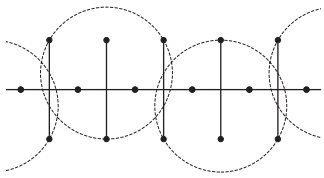


Figure 3.8: Asymmetrical coverage of 4 endpoints requiring a circle of radius at least $2\sqrt{3}/3 \approx 1.155$.

The "if" part requires more analysis. Consider T_G that can be covered by K circles of radius $r \approx \alpha$. For a path $u \cdots w$ on T_G whose length is $2m + 1$, there are $2m - 1$ vertical

bars associated with it. Consider the $4m - 2$ endpoints of these vertical bars. We note that (as shown in Figure 3.8), it requires a radius of $2\sqrt{3}/3 \approx 1.155$ for a circle to cover 4 bar endpoints in an asymmetrical manner (three endpoints on one side of the path, one on the other). Since we set the radius of coverage circle to be about $\alpha = 1.152$ (actually, between 1.152 and 1.153), a circle may only cover up to 4 bar endpoints. When a circle does cover 4 bar endpoints, it must use a symmetrical coverage pattern, i.e. 4 endpoints on two bars, resulting in fully covering two bars. For the rest of the proof, we use "circle" to mean circles with a radius of α, unless otherwise stated explicitly.

Since there are $4m - 2$ bar endpoints, it requires m circles to cover all bar endpoints when $m \geq 3$. Moreover, at least one circle must cover 4 bar endpoints by the pigeonhole principle. Fixate on such a circle S, which must have symmetric coverage, we examine the bars on one side of it, say the left side, assuming the path $u \cdots w$ is horizontal. If there are more than two bars to the left, then it is always beneficial to cover the two bars immediately to the left of S using another circle. To see that this is the case, look at the two bars (DE and FH) and the two associated unit length edges (AB and BC) to the left of S in Figure 3.9. It can be computed that the circle S to the right can cover a maximum length of 0.412 of AB to A'. Circles to the left of S then must cover $A'B$. Let the circle covering A' be S'. We may assume that S' covers at least one of D and E (otherwise, at least one more circle S'' must be added that fall between S' and S, in which case S'' must also cover A').

If S' covers A' and only one of D or E, say D, some other circle S'' must cover E. In this case, the coverage region of S' and S'' are bounded by circles of radius approximately 2.304 with center at D and E, respectively. It is readily observed that S' and S'' can reach at most one more bar to the left of FH (we note that S' and S'' will not be able reach structures on the 3-net beyond $u \cdots w$ path). In this case, we can instead move S' cover $A'C$, DE, and FH, and move S'' cover the bar to the left of FH and potentially one more bar. Therefore, we may assume that S' covers bar endpoints D, E, F, and H, symmetrically

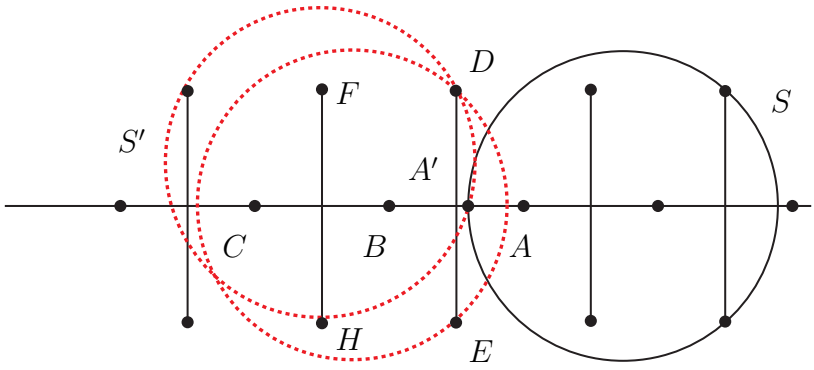


Figure 3.9: When there are enough bars left, it is always better to cover two bars at a time with a circle. Two extremal cases of S' covering A' and D but not E are shown (as dashed circles), which amounts to rotating the circle S' around D.

along the $u \cdots w$ path. Following the reasoning, we may assume that all bars on paths are covered, two at a time by a circle in a symmetrical manner, until there are one or two bars left before a path reaches a junction where it meets other paths.

Because there are odd number of bars on a path, the symmetric coverage pattern extends until one side of a path has two bars remaining while the other side has one bar. $m - 2$ circles have been used so far, which means that at least two more circles are needed to cover the remaining three bars. Without loss of generality, assume two bars out of these three are adjacent and are on the left end of $u \cdots w$ and one is on the right. Denote these bars as b_1, b_2, and b_3, from left to right. We call the end of a path with two bars the *even end* (e.g., the side ending with two bars b_1 and b_2) and the end of the path with one bar the *odd end* (e.g., the side ending with one bar b_3).

We now examine the coverage of b_2 (see Figure 3.10 where b_1 corresponds to CD and b_2 corresponds to AB). Again, if the two endpoints A and B of b_2 are covered by more than one circle, then one of these two circles can be replaced with one that fully covers b_1 and b_2 (the solid circle in Figure 3.10), since a circle covering only one endpoint of b_2 (e.g., B) will not be able to reach structures outside $u \cdots w$. By now, $m - 1$ circles have been used and to cover $u \cdots w$, at least two more circles are needed at the two ends (i.e., $m + 1$ circles are required to cover $u \cdots w$).

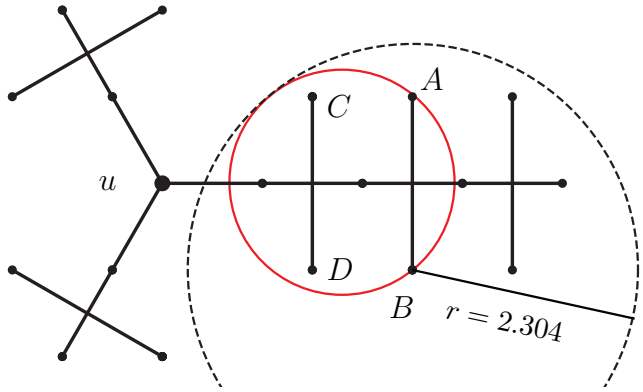


Figure 3.10: When there are two bars at the end of a path, it is preferred to cover them with a single circle (in red). The dotted circle shows that a circle of radius $2 * \alpha$ covering B will not be able to reach structures outside $u \cdots w$.

Next, instead of examining b_3, we examine the possible configurations at junctions where paths meet. There are four possible cases that contains 0-3 odd ends. For the case where only even ends meet, one additional circle is needed to cover the rest of the junction (Figure 3.11(a)). When there is one odd end and two even ends (Figure 3.11(b)), it requires one more circle to cover the junction. This constraint is how the radius $\alpha = 1.152$ is obtained (more precisely, with circles with radius 1.153, no additional circles are needed at the junction). When there are two odd ends and one even end (Figure 3.11(c)), at least one more circle is needed to cover the the junction. For the last case (Figure 3.11(d)), no additional circles are needed. The cases where additional cycles are needed correspond to the junction vertex being selected as a vertex cover. It is straightforward to observe that the constructed vertex cover is a valid one. The cover has size of $K - (L - |E(G)|)/2$.

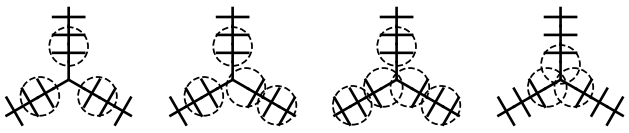


Figure 3.11: The four possible patterns at the junction using circles of radius less than $\alpha = 1.152$. When we increase the radius to $r = 1.152259$, circles shown in (b) and (c) can successfully cover the junction and the odd ends.

□

It is clear that Lemma Theorem 16 holds for discs with radius in $[1, \alpha)$. Thus, approximating $size(T_G, K)$ to less than a factor of α will decide whether G has a vertex cover of k, yielding the hard-to-approximate result. Also, it can be observed that all lengths are polynomial with respect to the problem input size, which implies strongly NP-hardness.

Theorem 17. *The minimum radius for cover a* 3*-net using* k *circular discs is strongly NP-hard to approximate within a factor of* $\alpha = 1.152$.

3.3.3 From 3-Nets to Simple Polygons

We proceed to show that $\text{OSG}_{2\text{D}}$ is hard to approximate for a simply polygon by converting a 3-net into one. Along the backbone G' of a 3-net T_G, we first expand the line segments by δ to get a 2D region (see Figure 3.12(a)). We may describe the interior of the resulting polygon as

$$P = \{p \in \mathbf{R}^2 \mid \min_{q \in T_G}(\|p - q\|_1) \leq \delta/2\}$$

For small enough δ, it's clear that P is a polygon with holes. Let $K = ((L - |E|)2 + k)$, it holds that

$$size(K, T_G) \leq size(K, P) \leq size(K, T_G) + \delta,$$
$$size(K, T_G) \leq size(K, \partial P) \leq size(K, T_G) + \delta.$$

To convert the structure into a simple polygon, we can open "doors" of width δ on the structure to get rid of the holes (see Figure 3.12(b)). Each opening removes one hole from P. This is straightforward to check; we omit the details.

Denoting the resulting simple polygon as P', we have

$$size(K, P) - \delta \leq size(K, P') \leq size(K, P),$$
$$size(K, \partial P) - \delta \leq size(K, \partial P') \leq size(K, \partial P).$$

Figure 3.12: (a) A 3-net T_G maybe readily converted into a simple polygon P with holes by expanding along its backbone. (b) Creating a "door" of width δ will remove one hole from P.

Therefore, both $size(k, P')$ and $size(k, \partial P')$ are between $size(k, T_G) - \delta$ and $size(k, T_G) + \delta$. Suppose the OSG_{2D} for $\partial P'$ or P' has a polynomial approximation algorithm with approximation ratio $1.152 - \varepsilon$ where $\varepsilon > 0$, let $\delta = \varepsilon/2$, then the optimal guarding problem for the T_G can be approximated within 1.152 disobeying the inapproximability gap by Theorem Theorem 17. Therefore,

Theorem 18. OSG_{2D} *is NP-hard and does not admit a polynomial time approximation within a factor of* α *with* $\alpha = 1.152$*, unless P=NP.*

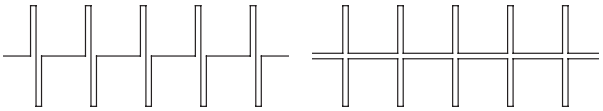
3.3.4 OPG_{2D} with Sensor Guarding Restrictions

The inapproximability gap from Theorem Theorem 18 prompts us to further consider restrictions on the setup with the hope that meaningful yet more tractable problems may arise. On natural restriction is to limit the number of continuous segments a mobile sensor may cover. As will be shown in Section subsection 3.4.1, if a mobile sensor may only guard a single continuous perimeter segment, a $(1 + \varepsilon)$-optimal solution can be computed efficiently. On the other hand, it turns out that if a sensor can guard up to two continuous perimeter segments, OPG_{2D} remains hard to approximate.

Theorem 19. OPG_{2D} *of a simple polygon cannot be approximated within* $\alpha \approx 1.152$ *even when each robot can guard no more than two continuous boundary segments, unless P=NP.*

Proof. Due to [93], every bridgeless 3-regular graph G has a perfect matching. We can obtain such a perfect matching of the 3-regular graph using Edmonds' Blossom algorithm

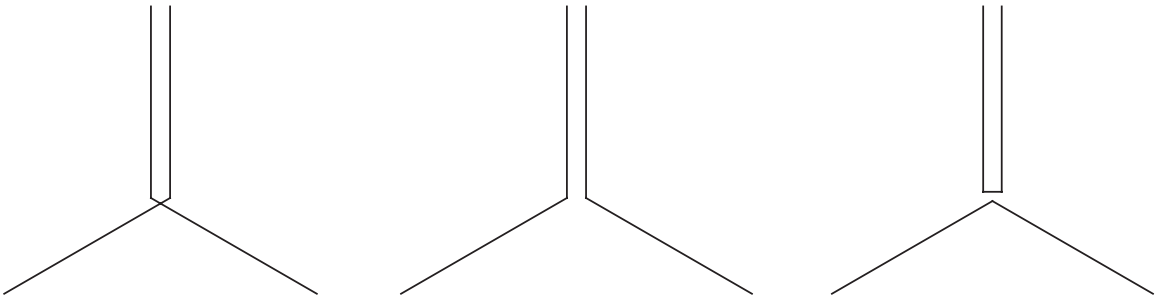
in polynomial time[94]. Doubling the edges in the perfect matching, we can then obtain a 4-regular graph G'.



(a) (b)

Figure 3.13: (a) Part of the augmented Eulerian path for non-doubled paths. (b) Part of the augmented Eulerian path for doubled paths.

The Eulerian tour on T_G may have self-intersections, which will prevent the tour from being a simple polygon. To address this, we may use one of two possible solutions outlined in Figure 3.14 to eliminate the self-intersections.



(a) (b) (c)

Figure 3.14: In order to eliminate possible self-intersections in (a), we may transform it into one of the solutions given in (b) and (c) to make the Eulerian tour remain connected (one of the two solutions will satisfy this).

At this point, we readily observe that Theorem Theorem 18 applies. Furthermore, an optimal solution always allows each mobile sensor to cover only two continuous perimeter segments. This is clear in the middle of any paths of T_G; at junctions, the polygon boundary will be either one of two possibilities shown in Figure 3.15, where a sensor again covers at most two continuous segments of the simple polygon. □

3.4 Effective Algorithmic Solutions for $\mathrm{OSG}_{2\mathrm{D}}$

In this section, we present several algorithmic solutions for $\mathrm{OSG}_{2\mathrm{D}}$. First, a fully polynomial approximation scheme (FPTAS) is presented that solve $\mathrm{OPG}_{2\mathrm{D}}$ with the additional

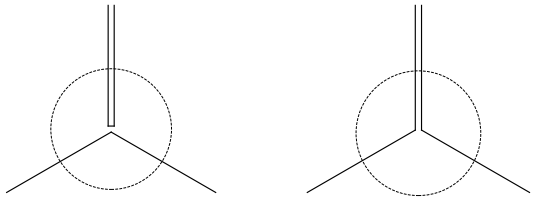


Figure 3.15: The figure shows two possible types of boundaries near a vertex with degree of 4. A robot near the vertex will only be able to cover two disjoint but individually continuous boundary segments with sensing radius less than α if the solution is to be optimal.

requirement that each sensor is responsible for a continuous perimeter segment. It contrasts Theorem Theorem 19. Then, we show that there exist polynomial time algorithms that readily guarantee a $(2+\varepsilon)$-approximation for OSG_{2D}. This is followed by an integer linear programming (ILP) method that delivers high-quality solutions (as compared with the $(2+\varepsilon)$-approximate one) and has good scalability.

In preparation for introducing the result, we first describe a method that is used for discretizing the problem. For a simple polygon P, we can approximately represent its boundary ∂P as a set of balls with radius ε along ∂P, by splitting ∂P into $N = \lceil len(\partial P)/(2\varepsilon) \rceil$ continuous pieces of length at most 2ε and putting the balls' centers at their midpoints. Denote set of ε-balls as S_B, and the set of their centers as $S_O = \{o_1, \ldots, o_N\}$. Since it holds that $size(k, \partial P) \leq size(k, S_B) \leq size(k, S_O) + \varepsilon \leq size(k, \partial P) + \varepsilon$, the minimum coverage radius of the discritized version of covering S_O will differ no more than ε from the original problem of covering ∂P. Similarly, for covering the interior of P, we can put P into a grid with cell side length ε, and set the center of the grid cells intersecting with P as S_O, creating at most $N = O((len(\partial P)/\varepsilon)^2)$ samples. The discretization process converts guarding P or ∂P to guarding S_O.

3.4.1 OPG_{2D} with Single Segment Guarding Limitation

By Theorem. Theorem 19, if a mobile sensor can guard up to two continuous perimeter segments, OSG_{2D} is hard to approximate within 1.152-optimal. Translating this into guarding elements of S_O, this means that a sensor can guard two *chains* of elements from

S_O, where each chain contains some m elements $o_1, \ldots, o_m$ that are neighbors along ∂P. Interestingly, if each sensor may only guard a single chain of elements from S_O, we may compute an optimal cover for S_O using $O(N^2 \log N)$ time. This readily turns into a fully polynomial time approximation scheme (FPTAS) for OPG_{2D}. The algorithm operates by checking multiple times whether a given radius r is sufficient for k discs of the given radius to cover elements of S_O where each disc covers only a single chain of elements.

A single feasibility check is outlined in Algorithm algorithm 6. In the pseudo code, it is assumed that the indices are modulo N, e.g. $M[N+1] = M[1]$, $o_{N+1} = o_1$. Algorithm algorithm 6 is based on an efficient implementation of a subroutine MIN_ENCLOSE_DISC (from e.g., [95, 96]) that computes the disc with minimum radius to enclose a given set of points in expected linear time. With this, a sliding window can be applied to find the rightmost end for each $1 \leq i \leq N$ such that $o_i, \ldots, o_{end}$ can be enclosed in a circle of radius r. The length of this sequence is stored in $M[i]$.

As o_{end} cannot come around and meet o_i, the total call to MIN_ENCLOSE_DISC is no more than $2N$. After this, the algorithm simply tries to put discs from each o_i to cover as many centers as possible to see whether S_O can be enclosed with k discs. An optimization can be made by only examining starting point as $o_1, \ldots, o_{M[1]+1}$, since there is no circle of radius of r that can cover them together by the definition of M. The apparent complexity of Algorithm algorithm 6 is $O(N^2)$. Since there are a total of N points and k robots, in a majority of cases a circle would enclose about N/k points, which effectively lowers the

time complexity to $O(N^2/k)$.

Algorithm 6: OPG_2D_CONT_FEASIBLE

Data: $S_O = \{o_1, \ldots, o_N\}$, sample points in circular order
k, the number of robots
r, the candidate sensing radius
Result: **true** or **false**, indicating whether S_O can be covered with k discs with radius r

if MIN_ENCLOSE_DISC($o_1, \ldots, o_N$)$\leq r$ **then**
  return true
end
%Phase 1: find the maximum number of consecutive points a disc of radius r can enclose from each c_i.
$M \leftarrow$ an array of length N; $end \leftarrow 1$;
for $i = 1$ **to** N **do**
  while MIN_ENCLOSE_DISC($o_i, \ldots, o_{end+1}$) $\leq r$ **do**
    $end \leftarrow end + 1$;
  end
  $M[i] \leftarrow end - i + 1$;
end
%Phase 2: try to tile from each o_i.
for $i = 1$ **to** N **do**
  $j \leftarrow i, \quad cnt \leftarrow k$;
  while $cnt > 0$ **do**
    $j \leftarrow j + M[j]$;
    if $j - i \geq N$ **then**
      return true
    end
    $cnt \leftarrow cnt - 1$
  end
end
return false

Note that for the optimal coverage radius r^*, it holds that $r_{min} = 0 < r^* \leq len(\partial P)/(2k) = r_{max}$. Recall that $N = \lceil len(\partial P)/(2\varepsilon) \rceil$. Hence, after at most

$$\log \frac{r_{max} - r_{min}}{\varepsilon} = \log(\frac{len(\partial P)}{2k\varepsilon}) = O(\log \frac{N}{k})$$

times of binary search on the optimal radius r^* by calling OPG_2D_CONT_FEASIBLE, the search range of r^* or the gap between r_{max} and r_{min} will be reduced to within ε. So, it takes expected $O(N^2 \log(N/k))$ time in total to get an approximate solution with radius at most ε more than $size(k, S_O)$ or $size(k, \partial P)$.

Theorem 20. *Under the rule of continuous coverage,* $\text{OPG}_{2\text{D}}$ *for a simple polygon can be approximated to* $(1+\varepsilon)$*-optimal in expected* $O(N^2 \log N)$ *time, and* $O((N^2/k)\log(N/k))$ *in most cases, where* $N = \lceil len(\partial P)/(2\varepsilon) \rceil$.

Remark 21. *In the running time complexity analysis, we implicitly used the assumption that $len(\partial P)$ is polynomial to problem input size (see Section section 3.2). Also, the algorithm given above computes an $OPT + \varepsilon$ optimal solution. However, it can be naturally assumed that the optimal sensing radius OPT is lower bounded in realistic scenarios. So, an $(OPT + \varepsilon)$ solution directly translates into a $(1 + \varepsilon)$-optimal solution. Lastly, using techniques similar to those from [76, 97], we mention that results in this subsection readily extends to multiple simple polygons with gaps along the boundary. These arguments continue to apply throughout the rest of this section.*

Regarding the choice in implementation, the minimum enclosing disc problem (1-center problem) also has deterministic solution [98] in linear time, but a randomized algorithm is considered to be more efficient [95] and easier to implement.

3.4.2 $(2 + \varepsilon)$ Approximation

In dealing with Euclidean k-clustering problems, two seminal methods are often brought out, both of which compute 2-approximation solutions for k-center problem in polynomial time. This is fairly close to the inapproximability gap of 1.822 for Euclidean k-center problem[29]. The first [30, 99] transforms the clustering problem to a dominating set problem and then applies parametric search on the cluster size (radius), resulting in a 2-approximation in time $O(n^2 \log n)$ with n being the number of points to cover. A second method [31] takes a simpler farthest clustering approach by iteratively choosing the furthest point from the current centers as the new center. The method runs in $O(nk)$ but is subsequently improved to $O(n \log k)$ in [29]. So, by applying either of them on S_O, we have

Proposition 22. $\mathrm{OSG}_{\mathrm{2D}}$ *can be approximated to $(2 + \varepsilon)$-optimal in polynomial time with $N = O(len(\partial P)/\varepsilon)$ samples for perimeter guarding and $N = O((len(\partial P)/\varepsilon)^2)$ samples for region guarding.*

For evaluation, we implemented the farthest clustering approach [31].

3.4.3 Grid and Integer Programming-based Algorithm

Approximation using grids [28] often exhibits good optimality guarantees and bounded time complexity. Seeing that and knowing that OSG_{2D} is hard in general, we attempted grid-based integer linear programming (ILP) methods for solving OSG_{2D} with good success. Our ILP model construction is done as follows.

Consider bounding the polygon P of interest by an $m \times n$ square grid where each cell is $\varepsilon \times \varepsilon$, and denote g_{ij} as the center of the cell at row i and column j. If we limit the possible locations of each robot to the center of some grid cell, the optimal radius with this limitation will only be at most $\sqrt{2}\,\varepsilon/2$ away from $size(k, S_O)$. This could be seen by moving the robot locations in the optimal deployment to their nearest grid centers respectively and applying triangle inequality.

So, given a candidate radius r, to check the feasibility of whether ∂P can be covered by k circles of radius r, we adapt an approach for solving the k-center problem[32] with integer linear programming. Specifically, we create $m \times n$ boolean variables y_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$, indicating whether there is a robot at g_{ij}, then start to check the feasibility of following integer programming model.

$$\sum_{1 \leq i \leq m} \sum_{1 \leq j \leq n} y_{ij} \leq k \tag{3.1}$$

$$\sum_{i,\ j\ s.t.\ \|g_{ij} - o_\ell\|_2 \leq r} y_{ij} \geq 1 \text{ for each } 1 \leq \ell \leq N \tag{3.2}$$

$$y_{ij} \in \{0, 1\} \qquad 1 \leq i \leq m,\ 1 \leq j \leq n \tag{3.3}$$

The first constraint says the number of locations is no more than k, and the second ensures each o_ℓ can be covered by at least one circle with radius r illustrated in Figure 3.16.

When the ILP model has a feasible solution, $r^* = size(k, S_O) \leq r$ and $r \leq r^* = size(k, S_O)$ otherwise. This means that we can do a binary search on r^*, from an initial range of $r^* = size(k, S_O)$: $r^i_{min} = 0 < r^* \leq len(\partial P)/(2k) = r^i_{max}$, until finally $r^f_{max} - r^f_{min}$ is reduced to the selected granularity of ε.

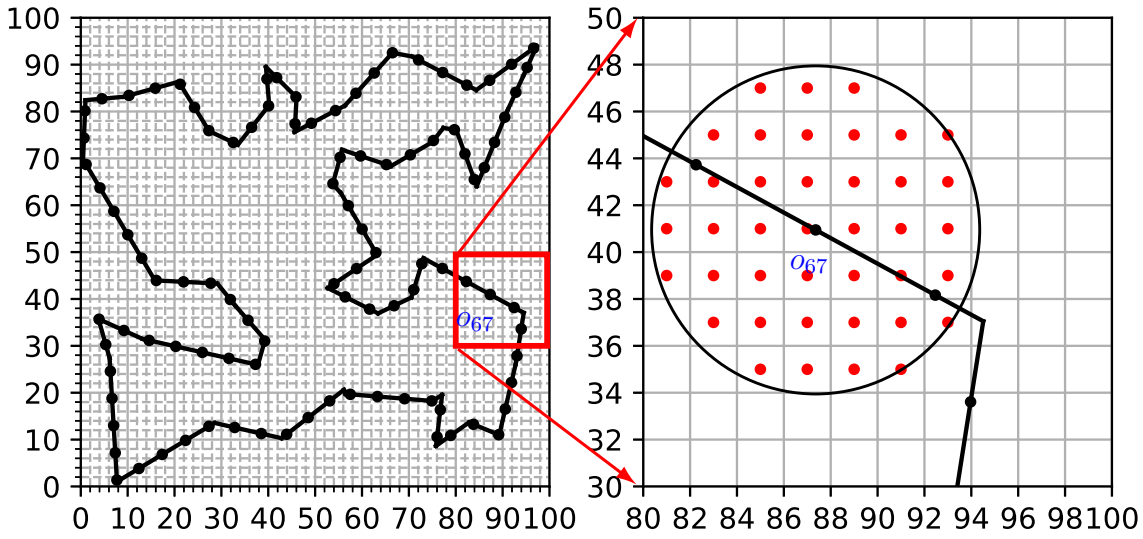


Figure 3.16: This perimeter guarding example illustrates constraint (2) for o_{67} with $r = 7$. The black dots are the sampled $S_O = \{o_1, \ldots, o_{100}\}$. In order to cover o_{67}, at least one among the red color grid cell centers need to be selected as robot location.

Remark 23. *With minor modifications, the ILP model applies to 2D region guarding, where the number of constraint* (2) *will then be* $O(mn)$ *with one for each grid that intersects with the polygon in an* $m \times n$ *grid. The initial upper bound set as* r^* *be* $len(\partial P)$ *and lower bound set as* $\sqrt{area(P)/(k\pi)}$*. It is also possible to apply the* $(2+\varepsilon)$*-approximation algorithm and set the result as the initial upper bound with the half of it as the initial lower bound.*

3.5 Evaluation and Application Scenarios

For the three algorithms described in Section section 3.4, we developed implementations in C++ and evaluated them on an Intel Core i7 PC with a boost clock of 4.2GHz and 16GiB RAM. For solving ILP models, Gurobi solver [3] is used. To evaluate the algorithms, we first generate a set of performance benchmarks obtained by subjecting these algorithms through a large set of benchmark cases. Following the synthetic benchmarks, we applied the algorithms on two potential application scenarios: guarding the outer perimeter of the Warwick Castle and monitoring a building for potential fire eruption points.

3.5.1 Performance Benchmarks

For creating synthetic benchmarks, to generate the test set $\mathcal{W}$, we created simple polygons with the number of vertices ranging between 10 and 200. For each instance of the tested polygon, vertices are picked uniform at random from $[0, 1] \times [0, 1]$ and the TSP tour among these vertices are used for generating a simple polygon of a reasonable shape. An example is given in Figure 3.16.

We first evaluate the computational performance of the special OPG_{2D} algorithm where each sensor may cover a single continuous perimeter segment; denote this algorithm as AL_OPG_2D_CONT. Table 3.1 lists the running time in seconds for various N (number of discretized samples) and k (number of guards). Various values of N suggest the choices of ε according to the setup of N in Section section 3.4, in this case $N = \lceil len(\partial P)/(2\varepsilon) \rceil$. Each data point is an average of 100 examples. As we can observe, the method has very good scalability. It also demonstrates the behavior that running time is inverse proportional to the number of guards, conforming with the statement about time complexity in Section subsection 3.4.1. The normalized average standard deviation is about 0.06, which is pretty small.

N \ k	5	10	20	30	50	100
500	0.097	0.044	0.019	0.013	0.007	0.004
800	0.257	0.118	0.054	0.036	0.019	0.011
1000	0.385	0.183	0.082	0.055	0.029	0.016
1500	0.912	0.436	0.203	0.120	0.073	0.039
2000	1.597	0.743	0.345	0.225	0.123	0.062

Table 3.1: Running time (seconds) for AL_OPG_2D_CONT.

Since the $(2 + \varepsilon)$-optimal algorithm is extremely efficient, we do not report its running time. For the ILP methods, Table 3.2 and Table 3.3 provide the running times for solving OPG_{2D} and ORG_{2D}, respectively (for convenience, denote these two methods as AL_OPG_2D_ILP and AL_ORG_2D_ILP). Each data point is an average over 10 cases.

GS \ k	10	15	20	30	50	100
50×50	0.219	0.127	0.092	0.051	0.023	0.009
100×100	0.686	0.383	0.250	0.141	0.089	0.033
200×200	1.915	1.132	0.792	0.444	0.281	0.115
300×300	7.782	4.201	2.613	1.513	0.814	0.435
400×400	21.23	11.63	7.275	3.827	2.231	1.318

Table 3.2: Running time (seconds) for AL_OPG_2D_ILP.

GS denotes the discrete grid size, suggesting the choice of the grid granularity ε and the single grid cell size $\varepsilon \times \varepsilon$. We observe that the ILP method is highly effective for solving OPG_{2D} and fairly good for solving ORG_{2D}. The normalized average standard deviation is about 0.125 for AL_OPG_2D_ILP (which is reasonable) and 0.545 for AL_ORG_2D_ILP (which is relatively large).

GS \ k	10	15	20	30	50	100
20×20	0.252	0.245	0.200	0.170	0.136	0.094
30×30	1.413	1.064	0.886	0.799	0.858	0.576
40×40	5.048	3.598	3.055	2.252	6.114	1.156
50×50	7.003	5.617	4.984	5.836	10.91	0.925
80×80	87.14	84.18	82.09	423.5	>2e3	>2e3

Table 3.3: Running time (seconds) for AL_ORG_2D_ILP.

For solution quality, we compare AL_OPG_2D_CONT, AL_OPG_2D_ILP, and AL_ORG_2D_ILP with the $(2+\varepsilon)$-optimal solution. For example, given a test case, let the resulting radius for AL_OPG_2D_CONT be r_1 and that for the $(2+\varepsilon)$-optimal algorithm be r_2, we compute the optimality gain as the reduce of coverage radius over r_2 in percentage, that is $(r_2 - r_1)/r_2 \cdot 100$. These are then averaged over 10 cases. Selected representative results (only three out of a total of 18 rows) are given in Table 3.4. In the table, m denotes the method where $\mathbf{1}$ = AL_OPG_2D_CONT, $\mathbf{2}$ = AL_OPG_2D_ILP, and $\mathbf{3}$ =AL_ORG_2D_ILP. Number of samples for AL_OPG_2D_CONT is set to 2000. Grid size for AL_OPG_2D_ILP is 200×200. Grid size for AL_ORG_2D_ILP is set to 40×40. For each method, we used polygons with 200 vertices. We observe that these algorithms do significantly better than

2-optimal with AL_OPG_2D_ILP getting very close to being 1-optimal (whose optimality gain is no more than around 50).

m \ k	5	10	20	30	50	100
1	22.34	23.89	27.07	29.14	32.32	34.18
2	36.29	34.82	36.22	36.98	37.69	38.29
3	35.69	32.58	30.06	25.22	21.99	15.46

Table 3.4: AL_OPG_2D_CONT, AL_OPG_2D_ILP, and AL_ORG_2D_ILP over the $(2+\varepsilon)$-optimal method.

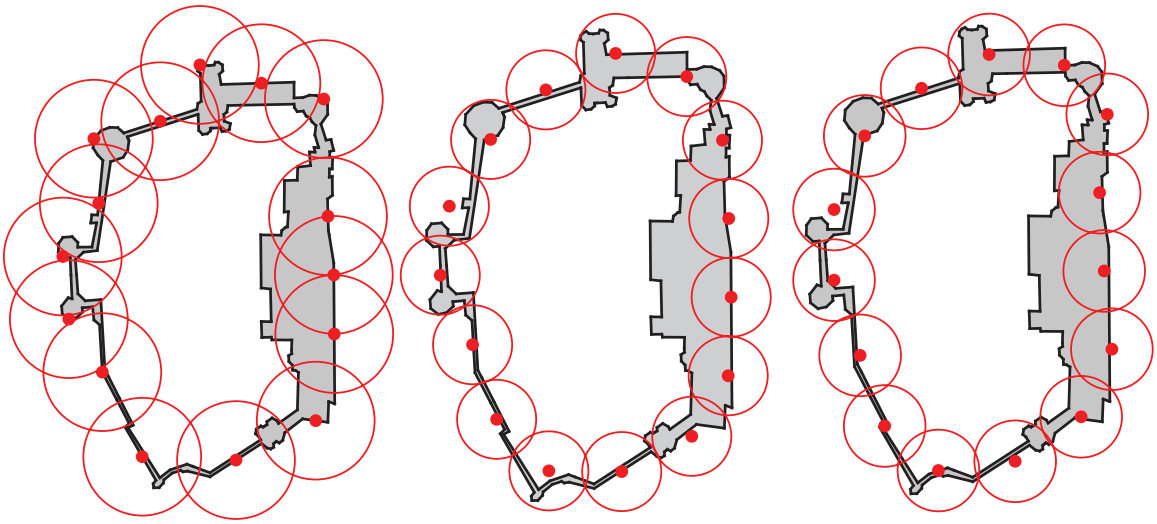
3.5.2 Two Application Scenarios

Next, we demonstrate the solutions computed by our algorithms on two potential application scenarios. For the first one, we apply algorithms for $\text{OPG}_{2\text{D}}$ on the outer boundary of the Warwick Castle in England (data retrieved from openstreetmap.org [100]). Figure 3.17 shows the solution for 15 guards computed by the $(2+\varepsilon)$-optimal algorithm, AL_OPG_2D_CONT, and AL_OPG_2D_ILP, respectively. Both AL_OPG_2D_CONT and AL_OPG_2D_ILP do about 40% better when compared with the $(2+\varepsilon)$-optimal algorithm. AL_OPG_2D_CONT does 3% better than AL_OPG_2D_ILP since the perimeter is suitable for continuous guarding while the ILP method is slightly limited by the chosen resolution.

In a second application, we took the footprint of the Brazil National Museum and use 40 mobile robots to monitor it. The solution, shown in Figure 3.18, is computed using AL_ORG_2D_ILP. This could be useful when a building is on fire and drones equipped with heat sensors can monitoring "hot spots" on top of the building to prioritize fire extinguishing effort. There are also many other similar application scenarios.

3.6 Conclusion and Discussions

In this study, we examine $\text{OSG}_{2\text{D}}$, the problem of directly computing a deployment strategy for covering 1D or 2D critical sets using many mobile sensors while minimizing the



(a) (b) (c)

Figure 3.17: Solutions for deploying 15 mobile sensors to guard the perimeter of the Warwick Castle. Methods: (a) $(2+\varepsilon)$-optimal. (b) AL_OPG_2D_CONT. (c) AL_OPG_2D_ILP.

maximum sensing radius. After showing that OSG_{2D} is computationally intractable to even approximate within 1.152, we describe several algorithmic solutions with optimality and/or computation time guarantees. Subsequent thorough evaluation demonstrates the effectiveness of these algorithmic solutions. Finally, we demonstrate the utility of our algorithmic solutions with two application scenarios. Due to space limit, guarding perimeters with gaps (see, e.g., [76]) is not discussed in this work. However, because our algorithms work with a grid-based discretization, the results directly apply to arbitrary bounded 1D and 2D sets.

Many intriguing questions follow; we mention two here concerning the sensing capabilities. First, OSG_{2D} works with circular regions which is perhaps the simplest one due to symmetry. What if the sensor region is not circular? Whereas such cases appear to be hard [101], effective scalable solutions may still be possible. Secondly, currently we assume that all parts of the critical set to be guarded have equal importance. What if certain subsets are more important?

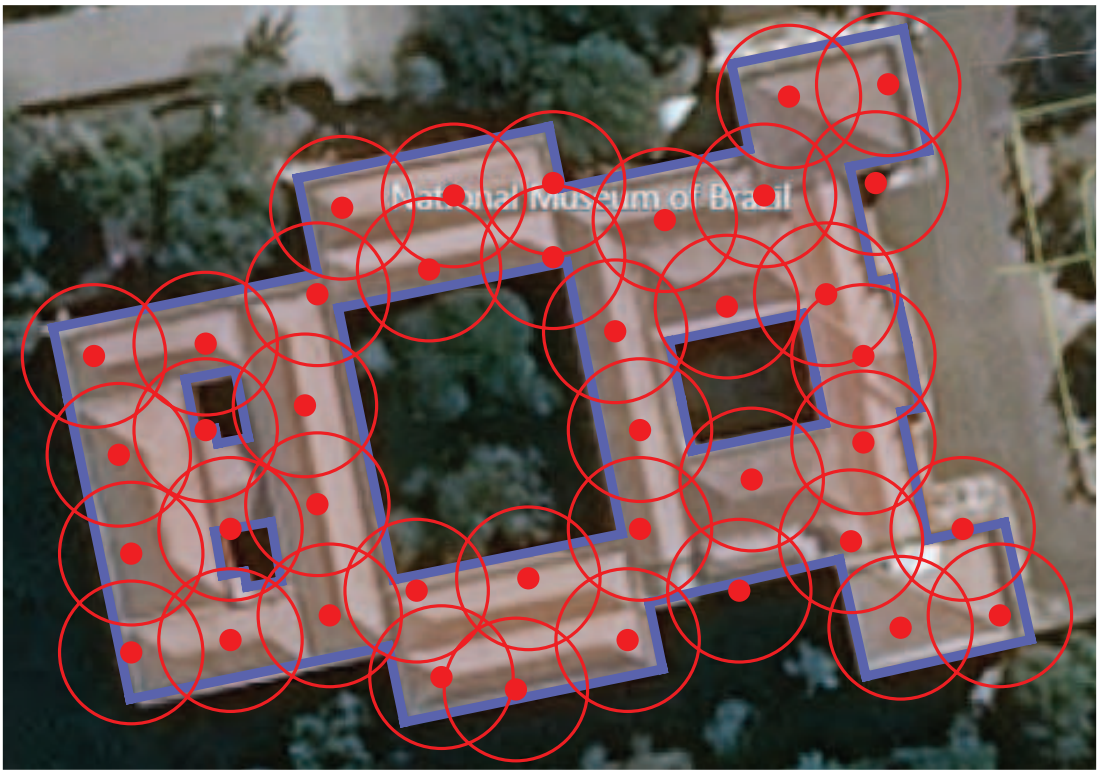


Figure 3.18: A near-optimal solution for deploying 40 mobile robots for monitoring the Brazil National Museum, which caught fire in 2019.

CHAPTER 4
SEPARATING POLYGONAL SETS WITH MINIMUM SETS OF LINES

After two chapters' discussion on covering perimeters or regions, which is essentially separating some critical polygonal regions from the outside workspace. This chapter digs deeper into this problem by studying the separation of more than two regions. To simplify the problem, a line-of-sight sensing model will be adopted, where each sensor can cover an unobstructed line segment like a laser beam. Also, the regions are assumed to be polygonal.

The objective in this case is to minimize the number of sensors used to separate these polygonal sets at the existence of obstacles. The problem is NP-hard even for the problem of separating two sets of regions with the minimum number of lines. Still, integer programming can provide a near-optimal solution for around 20 objects in a reasonable amount of time.

4.1 Introduction

Consider the scenario where one or more rogue agents (e.g., criminals) may be hiding in several isolated regions in a 2D workspace. To prevent them from potentially escaping from these regions to other nearby vulnerable regions, we may wish to set up line-of-sight sensors to detect if rogue agents attempt to escape. For the setup, a natural question one may ask is: what is the minimum number of line segments that are needed to form the desired barrier? The same setting finds many other practical applications, for example, for the identical setting, we may use the deployed sensors to track the movement of agents between different set of regions, e.g., understanding the flow of people between residential areas to commercial areas, which can benefit large-scale decision making, e.g., to help properly allocating resources for improving the city infrastructure. Alternatively, the computed line segments can serve as patrolling routes for autonomous agents (robots or humans) for ac-

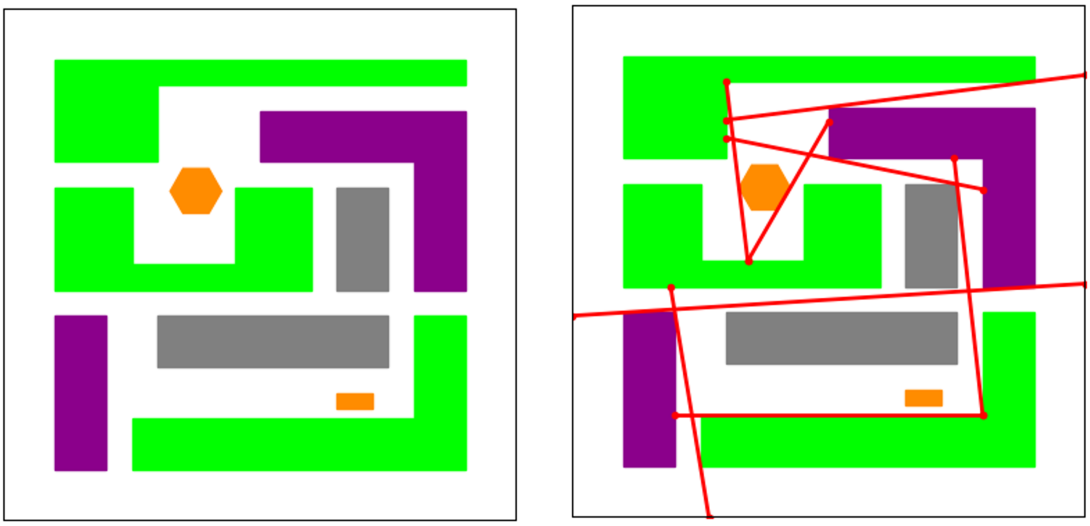


Figure 4.1: Example of barrier forming for separating three sets of complex polygonal shapes. Different colors represent different sets with the grey ones representing obstacles. The red line segments are the barriers computed by our algorithm.

tively monitoring intrusions, where the agents can always keep tracking events along the segments for which they are responsible.

Motivated by the above-stated scenarios and inspired by earlier research in robotics on barrier forming [102, 103], i.e., erecting barriers for separating regions of interest, in this chapter, we examine the variation of finding the minimum number of straight line segments for isolating multiple sets of points of polygons in a two-dimensional workspace. Fig Figure 4.1[left] illustrates an instance where three colored polygonal sets are to be separated from each other and the grey polygons are obstacles. Fig Figure 6.1[right] shows a possible solution which is fairly non-trivial to obtain.

As a summary of this work and its contributions, we study three settings in using line segments to separate sets of disjoint geometric shapes in a 2D workspace: (1) separating point sets, (2) separating point sets among polygonal obstacles, and (3) separating polygonal sets among polygonal obstacles. Whereas all three settings are NP-hard to optimally solve, we derive an effective method for computing optimal solutions for the first two settings, capable of handling tens of objects (points and/or polygons) partitioned into multiple

sets. The method first systematically computes candidate barrier set containing a minimum separating barrier, and then builds a novel integer programming model for finding the minimum barrier. Following a similar approach, we also develop a method that computes solutions for the third setting that is proven to be at least 2-optimal. We provide theoretical analysis that shed some light on why the setting involving polygonal sets is more difficult to solve. Extensive simulation study corroborates the effectiveness of our algorithms.

Related Work. Our investigation of barrier forming has its origins from several research areas. In the study of pursuit evasion or more generally differential games [104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115], such scenarios often happen where several agents are tasked to search an environment for hidden rogue agents or to protect critical regions from outsider invaders, which amount to creating and maintaining static or dynamic barriers of some form. Among the approaches taken in finding solutions for these problems, some resort to discretizing the environment into graph representations and conducting search over them [111, 112], while others adopt probabilistic reasoning [113, 114]. In particular, Tovar et al. [107] studied a problem that examines how to untangle sensor beam crossings to reason about the state of a robot, and use the insight to build algorithms for driving a robot to a desired state. In a sense, their study can be viewed as studying a problem that is a dual of the problem that we study here.

Barrier forming problems can also be seen as a type of sensor coverage problem. A series of study of perimeter defense/guarding problems aim at covering the boundaries of some regions to protect them from the outside [62, 63, 76, 65], which share similar motivations. The barrier forming problem has more flexible solutions by not fixing the specific boundaries to secure for the critical regions, which can be more general and closer to reality. In [102], the authors solved the barrier coverage problem optimally under non-trivial environment to minimize the total length of the barriers between two groups of polygonal objects. To tackle the proposed problem, a novel and efficient network-flow based method is applied. In [116], the work is extended to multiple groups of objects. The following work

[117] includes a similar problem formation to the problem studied in this chapter, where the objective is minimizing the number of fixed-length line segments used. However, the proposed method in [117] uses Tarski sentence [118] which, to our knowledge, does not have very effective method to deal with.

This work is closely related to several problems in computational geometry, especially point set shattering which seeks a complete separation among a single set of points [119, 120]. Bichromatic point sets or polyhedral separation problem uses wedges, axis-aligned lines, chords, parallel lines, or circles to separate two sets of points or polyhedra [121, 122, 123, 124]. Work on these problems often introduces specific constraints such as working with convex objects or simple polygons without holes, adopting non-crossing or parallel lines as the separator, and so on.

Chapter Structure. The rest of the chapter starts with formulating the barrier formation problem and introducing the three variants studied in section 4.2. Then, we describe the structure analysis of the problem in section 4.3, which will base the algorithm proposed in section 4.4. Lastly, in section 4.5 we evaluate the algorithm on four different scenarios.

4.2 Preliminaries

4.2.1 Barrier Forming with Minimum Number of Line Segments

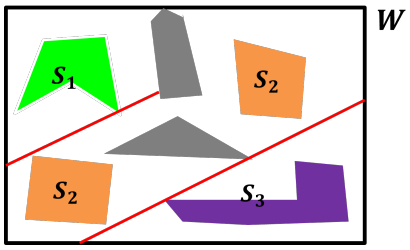


Figure 4.2: Illustration of a barrier forming problem for the three sets of (colored) polygonal objects $S_{1\sim3}$ among (gray) obstacles. For the setting, 2 straight lines are sufficient to cut off path connections between any pair of polygon regions from two different sets. In this work, our general goal is to build such barriers with the least number of line segments.

Let $\mathcal{W}$ be a simply connected polygonal workspace in $\mathbb{R}^2$. Consider k sets of objects

$S_1, \ldots, S_k$ in $\mathcal{W}$, as well as a set of polygonal obstacles $\mathcal{O}$. We seek to use a set of straight line segments L to separate $S_1, \ldots, S_k$ from each other, such that for any two objects $o_1 \in S_i$ and $o_2 \in S_j, i \neq j$, any path between o_1 and o_2 in the free space of $\mathcal{W}$ must be "cut off" by one or more line segments from L. The line segments do not have length limit and can cross each other, but they cannot cross objects or obstacles in the workspace. We want to find the minimum number of line segments to complete the separation task.

Under the general formulation of Barrier Forming, we examine three different variants with increasing difficulty:

1. *Barriers for point sets*, in which $S_1, \ldots, S_k$ as well as O are sets of points.

2. *Barriers for point sets among polygonal obstacles*, in which $S_1, \ldots, S_k$ are sets of points but O is a set of polygonal objects.

3. *Barriers for polygonal objects*, in which $S_1, \ldots, S_k$ as well as O contain polygonal objects.

4.2.2 Computational Intractability

Because the separation of even two sets of points in the plane, a special case of the first variant of our barrier forming problem, is computationally intractable [124], our first formulation is also NP-hard. From here, we can reduce from the first variant to other variants involving polygonal shapes by converting each point object to a sufficiently small polygon. Therefore, all three versions of the barrier forming problems studied in this chapter are NP-hard. We omit the straightforward details.

4.3 Structural Analysis

Given that barrier forming problems studied in this work are NP-hard, a natural algorithmic choice for addressing the challenge is through exploring mathematical programming. To that end, a model must be built that selects from candidate barriers, which in turn requires

the construction of a representative set of barrier candidates, a rather non-trivial task. The set of candidate barriers should satisfy two conflicting constraints: (1) it should contain a minimum set line segments that achieves the desired separation and (2) its size should not be too big that it will cripple the barrier selection process. Through careful structural analysis, we notice that the barriers to be considered can be limited to *tangent* or *bitangent* line segments. A tangent line segment, with respect to an object or an obstacle, is a line that passes through a vertex or an edge of the object/obstacle but does not intersect its interior. A bitangent is a line segment that is tangent to two objects and/or obstacles. This allows us to significantly reduce the number of candidates to be examined at the later selection stage.

Theorem 24. *For any k sets of polygonal or point objects $S_1, \ldots, S_k$ in the workspace $\mathcal{W}$, the set of line segments that are tangential to the objects and obstacles contains a set of minimum cardinality that separates $S_1, \ldots, S_k$.*

Proof. Consider a set of line segments L^* with minimum cardinality that separates $S_1, \ldots, S_k$. Without loss of generality, we assume all line segments in L^* do not end in the free space, i.e., each line segment in L^* ends at either object boundaries or workspace boundaries. If some line segment in a minimum barrier is not tangent to any object vertex, denoted it as $\ell = OA$ (shown in Figure 4.3), we show that it can be replaced by a line segment that is tangent to some object vertex. Fix one end of ℓ, O in this case, and rotate ℓ around O in both clockwise and counterclockwise directions until it hits some object vertex and becomes tangential to the object. Denote the two line segments resulting from clockwise rotation and counterclockwise rotation as $\ell'_1 = OB$ and $\ell'_2 = OC$, respectively.

We show ℓ can be replaced with ℓ'_1 or ℓ'_2. If this is not the case, since replacing ℓ with ℓ'_1 cannot make the separation work, there must be some point P_1 between AB that is path connected to some point in the other class without crossing any line segments in L^* when ℓ is replaced with ℓ'_1. Denote the point as D_1 and the path as $path_1$. The same analysis goes for ℓ'_2, that if ℓ cannot be replaced by ℓ_2 then there is some point P_2 in AC and path $path_2$ that connects P_2 to some other point D_2 in a different class and crosses segment ℓ but not

ℓ'_2. Since there are no objects or obstacles inside triangle OCB, we can assume the parts of $path_1$ and $path_2$ inside triangle OCB are straight lines. So, $path_1$ and $path_2$ must cross each other at some point. Denote the cross point as $Q \in path_1 \cap path_2$. Then, $path_1 = path_{11}(from\ P_1\ to\ Q) + path_{12}(from\ Q\ to\ D_1)$ and $path_2 = path_{12}(from\ P_2\ to\ Q) + path_{22}(from\ Q\ to\ D_2)$. Path $p_{11} + p_{22}$ connects P_1 to D_2, and $p_{21} + p_{12}$ connects P_2 to D_1, one of which must not cross ℓ. This leads to a contradiction to the fact that the original line set L^* separates the k classes of objects.

Therefore, each non-tangent line segment in L^* can be replaced with a tangent line segment. It will eventually result in a set of tangent barriers with minimum cardinality that separates $S_1, \ldots, S_k$. □

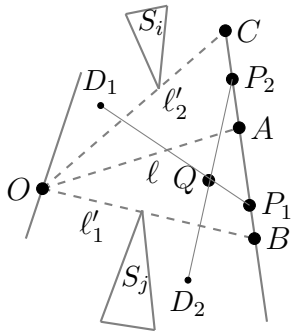


Figure 4.3: Rotating non-tangent barrier line segment ℓ in clockwise and counterclockwise directions around its endpoint O until it becomes tangential to some objects.

Although we can limit the candidate barriers to line segments tangent to object vertices, there can still be infinite number of candidates. One may consider using line segments that are bitangent to object vertices, i.e. line segments crossing two object or obstacle vertices. If there are n object/obstacle vertices, there can be at most n^2, i.e., a quadratic number of bitangents. Unfortunately, bitangent lines are insufficient to act as candidate barriers by themselves for polygonal objects. A counterexample in Figure 4.4 shows that there is an instance where an optimal solution must contain line segments that are not bitangents. In this counterexample, we need separate the orange objects from the lime object. A minimum of three line segments are used, and it is not possible that all of them are bitangent, i.e.

Proposition 25. *Bitangent line segments do not always contain optimal solution for the barrier forming problem for polygonal objects.*

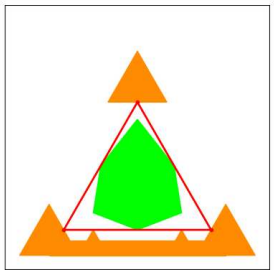


Figure 4.4: Counterexample that shows using only bitangent line segments cannot create the optimal solution

Despite the caveat, for the first two formulations that deal with barrier forming for point sets, even with polygonal obstacles, bitangent line segments always contain an optimal solution. More precisely,

Theorem 26. *For any k sets of point objects $S_1, \ldots, S_k$ in a workspace $\mathcal{W}$, there exists a set of line segments with minimum cardinality that separates $S_1, \ldots, S_k$, and only consists of bitangent line segments.*

Proof. From Theorem Theorem 24, we can see using single tangent line segments is always enough for an optimal solution. Now we turn an optimal solution, L^*, with only tangent line segments, into a solution with only bitangent line segments while still maintaining the same number of barriers.

For a tangent line segment $\ell = AB \in L^*$ with tangent point O (shown in Figure 4.5), and if O is a point object, assume it is beneath ℓ, rotate ℓ clockwise around O until it hits a point object or an obstacle vertex. Denote the resulting line segment as ℓ', and replace ℓ with ℓ'. Since the objects are point objects, so BB' and AA' must belong to obstacles or workspace boundary, and thus there is no object point inside OAA' or OBB'. Therefore, the replacement won't result in any path connecting objects in different classes. If this is not the case, then there will be some path connecting two object points in different classes that crosses ℓ but does not cross ℓ' or other barriers. Since the triangle areas OAA' and OBB'

are empty, that path must enter region OAA' or OBB' and leave them from ℓ. Then, that part of the path could be replaced with a straight line segment parallel to ℓ which prevents it from crossing ℓ. This contradicts the assumption that L^* prevents all connections between objects in different classes.

Continuing the replacement until all line segments are bitangent will result in an optimal solution with only bitangent line segments.

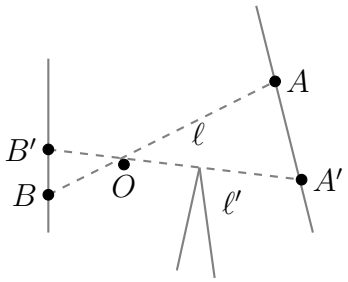


Figure 4.5: Rotating a single tangent barrier line segment ℓ around its tangent point O clockwise until it becomes bitangent.

□

For separating polygonal objects, although using bitangent line segments cannot guarantee an optimal solution that uses minimum number of line segments, they can still ensure that solutions limited to bitangents are at least 2-optimal.

Proposition 27. *For any k sets of polygonal objects $S_1, \ldots, S_k$ in the workspace $\mathcal{W}$, there exists a set of line segments with cardinality at most twice the minimum cardinality, that separates $S_1, \ldots, S_k$, and only consists of line segments that are bitangent to object or obstacle vertices.*

Proof. Starting from an optimal solution L^* with only tangent line segments, we will replace each tangent line segment with two bitangent line segments.

Rotate each non-bitangent line segment $\ell \in L^*$ around its tangent point O in clockwise or counterclockwise directions until the line segment become bitangent, as illustrated in Figure 4.6. Since any path connecting two objects in different classes and is cut by barrier

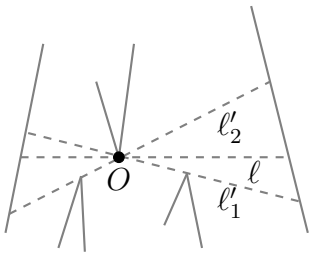


Figure 4.6: Rotating tangent barrier line segment ℓ both clockwise and counterclockwise around its tangent point O until it becomes bitangent.

ℓ will still be cut by ℓ'_1 and ℓ'_2. The replacement can still guarantee the separation among the object groups. After replacing all barriers, we can obtain a 2-OPT solution with the number of line segments twice the minimum. □

4.4 Fast Computation of High-Quality Solutions

In this section, we will apply the structural results of the barrier forming problem and provide effective method to tackle it. First, we start with a general method for obtaining the optimal solution among a set of candidate barriers. Then, based on different ways to generate the candidate barriers, two approaches are given while one is based on using bitangent line segments and the other is based on sampling.

4.4.1 Optimal Solution for Given Line Separator Candidates

In the barrier forming problem, if the candidate barriers are available as a finite set, we can tackle the problem with integer programming (IP). To solve it, we first perform a decomposition of the workspace using the candidate barriers, which partitions the workspace into cells whose edges are part of some candidate barriers. Denote N as the number of candidate barrier line segments, and M as the number of cells dissected using the candidate barriers. Figure 4.7 shows an example of dissecting the workspace into six cells with three candidate barriers. Then, we can start to construct an IP model to solve the problem of minimizing the number of selected barriers. First, we use $\lceil \log k \rceil$ binary variables for each cell, resulting in $M \cdot \lceil \log k \rceil$ such variables $c_{11}, \ldots, c_{M\lceil \log k \rceil}$. The value of $\overline{c_{i1}c_{i2}\ldots c_{i\lceil \log k \rceil}}$

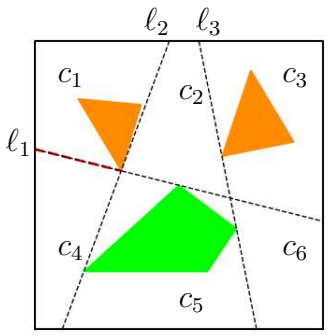


Figure 4.7: In this example, we aim to separate two groups of objects with the given candidate barriers. The workspace is decomposed into 6 cells by the 3 candidate barriers. As an example of constraint setup, the pair of adjacent cells c_1 and c_4 create a constraint of $\ell_1 \geq c_1 \oplus c_4$, which is equivalent to $\ell_1 \geq c_1 - c_4 \wedge \ell_1 \geq c_4 - c_1$. (Since there is only $k = 2$ classes of objects and $\log k = 1$ here, the second index of c_{i*} is eliminated.)

will represent the class of cell i. Thus, if there is an object in cell i, $\overline{c_{i1}c_{i2}\ldots c_{i\lceil \log k \rceil}}$ should have a fixed value according to the class of the object. A binary variable for each candidate line segment is used to indicate whether that line candidate is selected, resulting in N such variables: $\ell_1, \ldots, \ell_N$.

Between each pair of adjacent cells i and j (let the candidate line segment between them correspond to ℓ_σ), we have ($\oplus$ denotes "exclusive or")

$$\begin{gathered} \ell_\sigma \geq c_{i\tau} \oplus c_{j\tau} \Leftrightarrow \ell_\sigma \geq c_{i\tau} - c_{j\tau} \wedge \ell_\sigma \geq c_{j\tau} - c_{i\tau}, \\ \text{for each adjacent cell } i, j, \text{ and } 1 \leq \tau \leq \lceil \log k \rceil, \end{gathered} \tag{4.1}$$

which means if two adjacent cells belong to different classes, then the line segment candidate between them must be selected. An example of the constraint setup is illustrated in Figure 4.7. Naturally, we have fixed $c_{i1}, \ldots, c_{i\lceil \log k \rceil}$ if ceil i contains an object to be separated, and the value of $\overline{c_{i1}c_{i2}\ldots c_{i\lceil \log k \rceil}}$ is set to be the same as its class index: $0 \sim k-1$. The objective is set to minimize the total number of line candidates selected, i.e., $\min \sum_{i=1}^{M} l_i$.

4.4.2 Near-optimal Solution Using Bitangent Lines

As the results in Section section 4.3 show, using bitangent line segments can always provide optimal solution for the problem of barrier forming for point objects, and at least 2-OPT solution for separating polygonal objects. Since the number of bitangent line segments is

at most quadratic to the number of object or obstacle vertices, we can enumerate them, and apply IP method in Section subsection 4.4.1 to find a solution. Figure 4.8 illustrates the candidate barriers constructed for point objects and polygonal objects. It is worth noting that for enumerating bitangent barriers for point objects, the side of the point to the barrier also matters. For example, a pair of point objects will create 4 bitangent barrier candidates as there are 4 different possible cases depending on the how the corresponding objects are placed with respect to the line.

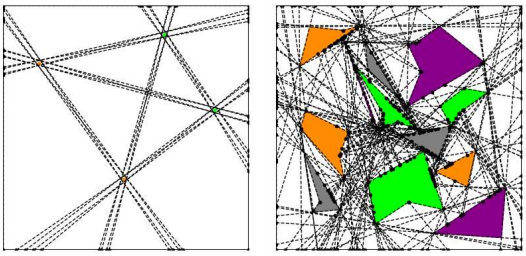


Figure 4.8: Illustration of bitangent barrier candidates. The left picture shows the bitangent candidates for 2 point sets, each with 2 points. In this case, a pair of points will create 4 candidates. We note that we made the points non-zero-dimensional for visibility purposes. The right picture shows the bitangent candidates for 12 polygonal objects in four sets.

4.4.3 Sampling-based Resolution Complete Algorithm

Although using bitangent line segments works well in most of the cases, it unfortunately cannot provide an optimal guarantee for the barriers formed when dealing with polygonal objects. However, theorem Theorem 24 provides a good starting point for sampling line segments: we may limit candidate barrier sets to single tangents, i.e., we sample line segments passing through each object vertex in a radial manner. Hence, if we gradually increase the sampling resolution around each object and obstacle vertex and use the sampled line segments as candidate barriers, we can guarantee the asymptotic optimality of the resulting solution.

4.5 Experimental Evaluation

In this section, we describe our experimental study of the method proposed in the paper. Four settings were used to account for the three variants of the barrier forming, the instances and solution examples of which are shown in Figure 4.9. The experiments were carried out on a Hexa-Core processor with 16 GiB memory, and Gurobi [3] was used as the Integer Programming solver. In the polygonal object instances generation, random polygons were generated by computing traveling salesperson tour (TSP) tours of random point sets each consisting of 3 to 6 vertices, which we found to be effective in generating sensible looking polygons that are not necessarily convex. When a problem setup contains obstacles, the number of obstacles in the environment is set to be the same as the number of objects for each object set.

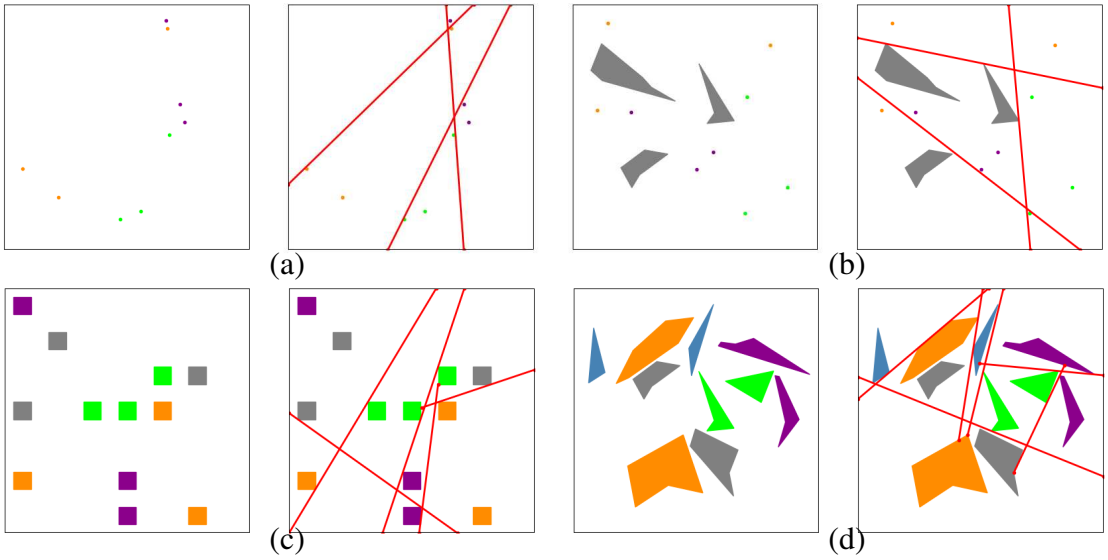


Figure 4.9: Illustration of the four types of instances used in our experimental evaluation. (a) Barrier forming for point sets. (b) Barrier forming to separate point sets from polygonal obstacles. (c) Barrier forming to separate uniform square-shaped objects among uniform square-shaped obstacles. (d) Barrier forming to separate random polygonal objects among random polygonal obstacles.

4.5.1 Separating Sets of Points

The first type of instances aims at forming barriers among randomly generated point sets, shown in Figure 4.9(a). The number of object sets to separate from each other range from 2 to 4, and the number of objects in each set range from 1 to 6. Each entry in the Table 4.1 is the result of average computation times over 10 instances. From the result, we observe that the IP based method is fairly effective in separating two sets of objects with the presence of obstacles. The method scales to about 10 point objects plus obstacles. The number of line segments in the optimal barrier are generally small, e.g., 3-10.

#Sets	1	2	3	4	5	6
2	0.005	0.011	0.083	0.419	2.010	15.887
3	0.013	0.316	12.773	962.883	-	-
4	0.051	14.641	-	-	-	-

Table 4.1: Running time in seconds for Expr. 1 where all objects and obstacles are points (Figure 4.9(a)). "-" denotes the result cannot be computed in 1h on average (the same is true for other tables). The column index means the number of objects in each set, and the row index means the number of object sets. The number of obstacles is set to be the same as the number of objects for each set. These also apply to the following tables.

The second type of instances generates barriers for randomly generated point sets with the existence of polygonal obstacles, shown in Figure 4.9(b). The other specification is the same as Expr. 1, and the number of obstacles for each experiment is set to be the same as the number of objects for each set. The resulting time cost, shown in Table 4.2, is similar to Expr. 1 despite the existence of obstacles. Similarly, we observe decent performance when it comes to separating two sets of objects among obstacles. The introduction of polygonal obstacles does not cause performance degradation.

#Set	1	2	3	4	5	6
2	0.009	0.061	0.480	5.899	3.433	21.121
3	0.044	3.287	71.346	320.955	-	-
4	0.249	13.801	-	-	-	-

Table 4.2: Running time in seconds for Expr. 2 where objects to be separated are points and obstacles are randomly generated polygons (Figure 4.9(b)).

4.5.2 Separating Sets of Polygonal Shapes

The third set of experiments uses randomly placed squares as obstacles and objects, shown in Figure 4.9(c). The squares are sampled from a 7×7 grid. Each square is half the scale of a grid cell and is positioned at the center of a grid cell. The running time for this case turns out to be the greatest among all 4 experiments. This is due to the rectlinear nature of the instance, which creates many small cells that are difficult to process.

#Set	1	2	3	4	5	6
2	0.010	0.065	0.652	31.536	575.933	1259.653
3	0.065	10.608	337.050	-	-	-
4	0.235	124.963	-	-	-	-

Table 4.3: Running time in seconds for Expr. 3 with square-shaped objects and obstacles (Figure 4.9(c)). "-" denotes the result cannot be computed in 1h on average.

The last type of instances uses random polygons with $3 \sim 6$ vertices as objects and obstacles, shown in Figure 4.9(d). Counter intuitively, these experiments turn out to have the least time cost among the 4 experiments despite the most complex environment; we see that even for four different sets of objects where each set contains six objects, the problem can be solved very quickly.

In the end, the running time of the algorithm provided is more dependent on the number of cells and candidate barrier line segments. When objects and obstacles are more densely packed in the experiment, there will be less barrier candidates and cells due to collisions between objects and the candidate line segments. While in a sparse environment or even with just point objects, there will be more barrier candidates and cells. This explains the reduced time cost in a more complex environment from the sparse settings.

#Set	1	2	3	4	5	6
2	0.006	0.031	0.080	0.143	0.157	0.156
3	0.031	0.234	0.994	1.271	1.980	1.187
4	0.103	0.518	3.000	6.050	9.692	17.996

Table 4.4: Running time in seconds for Expr. 4 where both the objects to be separated and the obstacles are randomly generated polygons (Figure 4.9(d)).

CHAPTER 5
SWEEP LINE COVERAGE AND BOUNDARY DEFENSE

This chapter deals with the dynamic setting for mobile sensing robots, and two different but related problems are studied. The first problem is the boundary defense problem in the context of heterogeneous defenders first studied in [10]. It can be seen as an extension of the perimeter defense problem [11]. In this problem, there are k sensing robots moving on top of a perimeter with different speeds $v_1, \ldots, v_k$. A sequence of attacks $\langle loc_i, t_i \rangle_{i=1}^{n}$ are given at different time stamps and at different locations. The objective is to intercept as many attacks as possible. The second problem is the coordinated sweeping problem where a group of robots coordinate to sweep a region with obstacles. Each robot possesses a given sensing capability, and the sweeping trajectory is given beforehand. The objective of it is to minimize the number of robots used such that the sweeping plan can be executed and every point in the workspace is sensed with a certain required quality.

5.1 Boundary Defense

5.1.1 Introduction

Perimeter-defense problems model scenarios where the interior of a region must be defended against a set of incoming attacks. With a team of defenders (i.e., mobile robots and/or agents) located on the perimeter of a given region, an attack is considered successfully intercepted if at least one defender is present at the attack location when the attack happens. This problem finds broad applications, including the protection of endangered wildlife [68], aerial defense [125, 126], and border security [127, 65], to list a few.

The study of perimeter defense problems finds some of its origins in target guarding problems [128], a classical pursuit-evasion game studying the strategies of multiple de-

fenders to guard a static region. In [129], the target-guarding problem was first specialized to produce perimeter defense in which the perimeter is the target to be guarded, and defenders can only move on the perimeter. Various methods have been developed, including region decomposition [129], assignment or matching-based algorithm [62], network flow formulation [64], and coverage control-based method [63].

Perimeter-defense problems may also be considered a variant of the reach-avoid game between two parties, where one party tries to send as many attackers to a region the other party must defend [128]. Studies of the reach-avoid game usually apply specific case analysis, which suffers from the exponential increase in time complexity and size of state space as the number of attackers increase [130, 131, 132]. Recent work leans toward adopting maximum matching or assignment frameworks from graph theory to address this problem for multiple defenders [133, 134, 135].

In this work, we study perimeter-defense problems under a perfect information assumption that the attackers' attack time and locations on the perimeter are known as soon as attackers appear. The assumption is adopted in several recent related research including [10, 63]. The particular emphasis of our work is on the more challenging case of heterogeneous defenders, where the defenders have different speeds in responding to incoming attacks. We denote the problem as *boundary-defense with heterogeneous defenders* or `BDHD`.[1] The heterogeneous setting for perimeters is first carefully investigated in [10], which carried out the detailed structural study and introduced an algorithm based on dynamic programming (DP) [136] for handling infinite horizon attacking sequences, focusing mostly on two defenders.

The main contribution of our work lies in the development of efficient computational tools for and an empirical structural study of `BDHD`. More specifically,

- We developed a network-flow formulation of `BDHD`, leading to an exact method, based on integer linear programming (ILP), for solving the problem. The IP-based method is

[1] We use *boundary* instead of *perimeter* since perimeter generally refers to the 1D boundary of a 2D geometry shape. 2D hemisphere has been examined in [126] for constraining defender.

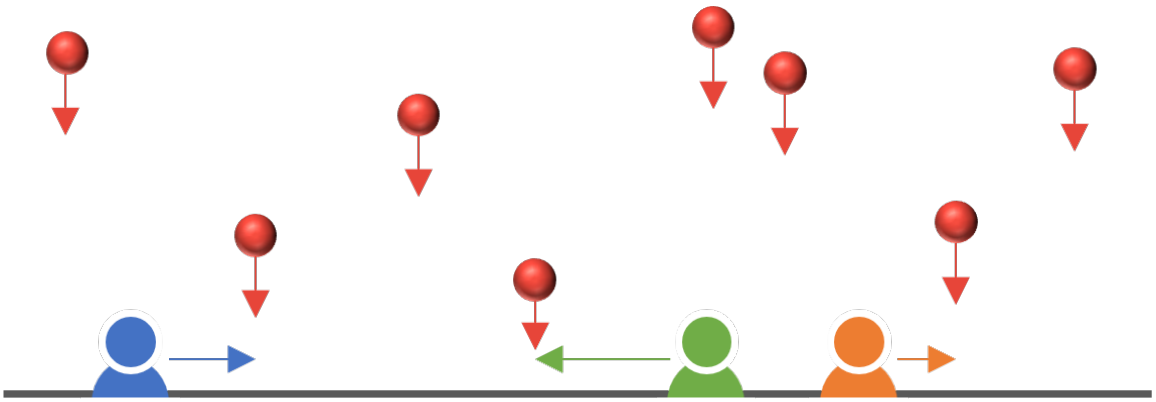


Figure 5.1: Illustration of the problem of boundary defense with heterogeneous defenders, where multiple defenders with different capabilities must do their best to intercept incoming attacks (signified as red balls moving downwards). The defenders are constrained to move on the boundary, which is a 1D perimeter in this case.

much more scalable than the DP-based method (which is only effective for no more than three defenders).

- Using the DP solution as a sub-routine, we developed a highly scalable heuristics method, called *exhaustive defender pairing* (EDP), that runs in low polynomial time. EDP is demonstrated to compute high-quality (in fact, often optimal or near-optimal) solutions. The design philosophy of EDP is of independent algorithmic interest.

- We further generalize our algorithms to apply to finite-horizon `BDHD` (more challenging than infinite-horizon `BDHD`), where only attacks whose attack time is within some T time of hitting the boundary are revealed.

- Utilizing the scalable algorithms developed in this work, we performed a thorough empirical study of the solution characteristics of `BDHD` under various possible problem settings, including (1) different attack densities, (2) different defender speed distributions, and (3) different domain topology, i.e., S^1 (circle), $I = [0, 1]$ (unit interval), S^2 (sphere), and $I \times I$ (unit square).

The rest of the chapter is organized as follows. In Sec. subsection 5.1.2, we formulate the problems studied in this chapter and introduce the notations used. In Sec. sub-

section 5.1.3, we describe the previous dynamic programming method in [10], and our ILP-based algorithms and EDP. We discuss the performance of our algorithms and solution characteristics of BDHD under different settings in Sec. subsection 5.1.8, and conclude with Sec. subsection 5.1.15.

5.1.2 Preliminaries

In boundary defense with heterogeneous defenders (BDHD), there are k defenders (which may be robots and/or other types of agents) with speeds $v_1, \ldots, v_k$, where each defender is modeled as a point in some domain $\mathcal{E} = \mathbb{R}^2$ or $\mathbb{R}^3$. The defenders live on a lower dimensional subspace of $\mathcal{E}$ (i.e., some boundary of a subset of $\mathcal{E}$). There are also n attack events $\left\langle loc_i, t_i \right\rangle_{i=1}^{n}$, where each attack event is a pair $\left\langle loc, t \right\rangle$ in which loc is the location of the attack and t is the time it happens. The i^{th} attack is intercepted by a defender only if the defender is located at loc_i at time t_i. For initialization, we denote the initial locations of the k defenders at $t = 0$ as $loc_1, \ldots, loc_k$.

Following the definition in [10], we define the horizon of the defenders as follows,

Definition 28 (Horizon)**.** *The (look ahead) horizon T of the defenders is defined as the amount of time defenders can peek into the attack sequence in the future. That is, given the current time t, and a horizon T, defenders have access to complete information on attacks happening on or before $t + T$.*

Now, we provide formulations of the two versions of the BDHD problem studied in this subsection. In the infinite horizon setting, $T = \infty$, all attack events are given in a single batch.

Problem 5 (Infinite-horizon BDHD)**.** *Given k defenders with speed $v_1, \ldots, v_k$ and initial locations $loc_1, \ldots, loc_k$, and n attack events $\left\langle loc_i, t_i \right\rangle_{i=1}^{n}$, intercept as many attacks as possible.*

In a finite-horizon setting, the attack events are not all revealed at $t = 0$ but are given

as a stream of attacks $\langle loc_i, t_i \rangle_{i=1}^{\infty}$. The defenders can only know the attack events within a horizon or time window of $T < \infty$ in the future.

Problem 6 (Finite-horizon BDHD)**.** *There are k defenders with speed $v_1, \ldots, v_k$ and initial locations $loc_1, \ldots, loc_k$, and a stream of attack events. At each time instance t, the defenders only know attack events happening no later than time T in the future (i.e. $t_i \leq t + T$). Intercept as many attacks as possible.*

We introduce here two useful notations: $next(a, d)$ $(a \in [0, n],\ d \in [1, k])$ and $prev(a, d)$ $(a \in [1, n],\ d \in [1, k])$. They are defined as

$$next(a, d) = \{a' | dist(loc_a, loc_{a'}) \leq v_d \cdot (t_{a'} - t_a)\} \tag{5.1}$$

$$prev(a, d) = \{a' | dist(loc_a, loc_{a'}) \leq v_d \cdot (t_a - t_{a'})\} \tag{5.2}$$

where $dist(x, y)$ denotes the distance between location x and y. In other words, $next(a, d)$ is the set of attack events that can be reached from the location of the a^{th} attack event by defender d. And $prev(a, d)$ is the set of attack events from whose location defender d can reach the a^{th} attack event. Additionally, $next(0, d)$ denotes the set of attack events that can be reached by defender d from its initial location. Similarly, $prev(a, d)$ contains 0 if defender d can reach the location of attack a from its initial location. Figure 5.2 gives an example for defender 1.

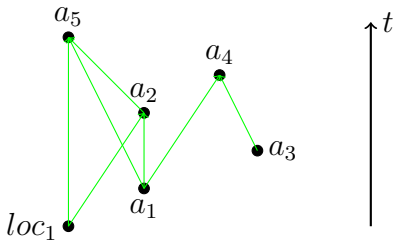


Figure 5.2: Illustration of an example of reachability between different attack events for defender 1. For example, $next(1, 1) = \{2, 4, 5\}$, $next(0, 1) = \{2, 5\}$, $prev(2, 1) = \{0, 1\}$, $prev(1, 1) = \varnothing$. loc_1 cannot reach a_1 since v_1 is not sufficiently large. For the same reason, defender 1 cannot reach a_3 from a_1.

5.1.3 Algorithmic Solutions for BDHD

In this section, we describe three methods for solving infinite-horizon BDHD (Problem 5). Sec. subsection 5.1.4 provide a dynamic programming (DP) algorithm based on one developed in [10], followed by Sec. subsection 5.1.5 formulating an integer linear programming model, and Sec. subsection 5.1.6 discusses a heuristic search algorithm based on the DP algorithm. The extension to an infinite attack stream with a finite look-ahead horizon (Problem 6) will be discussed in Sec. subsection 5.1.7.

5.1.4 Exact Dynamic Programming Based Method

The DP algorithm builds a recursion formula on the last attacks intercepted by the defenders. Without loss of generality, we assume that each attack is only intercepted by one defender. For a DP state $a_1, \ldots, a_k$ where a_i is the last attack defender i intercepts, let $T[a_1] \ldots [a_k]$ store the maximum number of attacks that can be intercepted when the i^{th} defender's last intercepted attack events is a_i ($i \in [1, k]$), and denote a_{ma} as the maximum of $a_1, \ldots, a_k$. Base on which attack the ma^{th} defender intercepts before intercepting attack a_{ma}, we can write the DP recursion formula as follows.

$$T[a_1] \ldots [a_{ma}] \ldots [a_k] = \max_{p \in prev(a_{ma}, ma) \wedge p \neq a_1 \ldots a_k} T[a_1] \ldots [p] \ldots [a_k] + 1 \tag{5.3}$$

Pseudo-code in algorithm 7 provides a sketch of a possible implementation of the dynamic programming algorithm. Effectively implemented, the time complexity of running algorithm 7 is $O((n+1)^{k+1})$, which is polynomial when k, the number of defenders, is fixed.

Algorithm 7: Dynamic Programming for BDHD

Data: $E = \langle loc_i, t_i \rangle_{i=1}^{n}$: n attack events
$loc_1, \ldots, loc_k$: initial locations of the k defenders
$v_1, \ldots, v_k$: speeds of the k defenders
Result: Maximum number of attacks intercepted

$T \leftarrow$ an $(n+1)^k$-length array initialized to $-\infty$
$result \leftarrow 0$
$T[0] \leftarrow 0$
for $mask \leftarrow 0$ **to** $(n+1)^k - 1$ **do**
  $\overline{a_1 a_2 \ldots a_k} \leftarrow mask$
  %$\overline{a_1 a_2 \ldots a_k}$ represents a base-$(n+1)$ number, i.e., $a_1 \cdot (n+1)^{k-1} + a_2 \cdot (n+1)^{k-2} + \cdots + a_k$
  if $\exists\ a_i = a_j$ **then**
    continue
  end
  $ma \leftarrow argmax_i a_i$
  for $p \in prev(a_{ma}, ma)$ **do**
    if $\forall i\ pos_i \neq p$ **then**
      $pm \leftarrow mask - (a_{ma} - p) \cdot (n+1)^{k-ma}$
      %pm is the result of replacing a_{ma} with p
      $T[mask] \leftarrow \max(T[mask], T[pm] + 1)$
    end
  end
  $result \leftarrow \max(result, T[mask])$
end
return $result$

The algorithm presented here is a slight modification of the DP algorithm in [10] with two subtle differences. First, we enforce that an attack event can only be handled by one defender. Second, we explicitly use the initial locations of the defenders in the algorithm, which is essential for handling the finite horizon extension.

5.1.5 Solving BDHD with Integer Linear Programming Model based on a Flow Formulation

It is not difficult to see that BDHD can also be seen as a network flow problem by treating each attack event as a node and the reachability between each pair of attack events for each defender as the edges in the graph.

Specifically, there are n nodes in the graph, representing the n attack events. There are also $O(n^2k)$ connections between nodes inside the graph. If defender j can reach attack event i' from i, there is an connection $edge[i][i'][j]$ between node i and i'. Also, we use

a binary variable $intercept[i]$ to denote whether attack event i is successfully intercepted. These give rise to the following integer linear programming (ILP) formulation of the problem.

Eq. (Equation 5.4) sets the criteria of an attack i being intercepted as at least one defender to come into the node i, which means intercept attack i. Eq. (Equation 5.5) sets the defender flow conservation rule that the number of type j defender exiting node i must be larger than or equal to the number of type j defender coming to node i. Eq. (Equation 5.6) sets the initial constraints on the number of each type of defender used (coming out from node 0).

$$\sum_{j\in[1,k],\ i'\in prev(i,j)} edge[i'][i][j] \geq intercept[i] \tag{5.4}$$

$$\sum_{nxt_i\in next(i,j)} edge[i][nxt_i][j] \leq \sum_{prv_i\in prev(i,j)} edge[prv_i][i][j] \tag{5.5}$$

$$\sum_{nxt_0\in next(0,j)} edge[0[nxt_0][j] \leq 1 \tag{5.6}$$

$$Objective \quad \max \sum_i intercept[i] \tag{5.7}$$

Denote M as the number of connections in the graph, clearly $M < n^2k$. This integer linear programming formulation uses $M + n$ variables, and $nk + n$ constraints.

Remark 29. *The flow formulation of the problem is an NP-hard problem. The proof is similar to the NP-completeness proof of the two-commodity flow in [137]. This seems to suggest that* `BDHD` *is NP-hard as well.*

5.1.6 Exhaustive Defenders Pairing Heuristic Search Method

We now develop a heuristic search method using the dynamic programming algorithm discussed in algorithm 7 applied on two defenders. The DP algorithm that computes the optimal solution for two defenders is used as a local improvement primitive for the local search heuristic algorithm.

We call the resulting algorithm *exhaustive defender pairing* (EDP). In each iteration, EDP pick two defenders, and the attack events the two defenders have intercepted and the attack events that have not been intercepted by any defender. Then, EDP uses the DP algorithm in algorithm 7 to increase the number of attack events intercepted by the two defenders selected. The complete algorithm is sketched in algorithm 8.

Algorithm 8: Exhaustive Defender Pairing

Data: $E = \langle loc_i, t_i \rangle_{i=1}^{n}$: n attack events
$loc^1, \ldots, loc^k$: initial locations of the k defenders
$v_1, \ldots, v_k$: speeds of the k defenders
Result: Number of attacks intercepted

1 $Intercept \leftarrow$ a length-n array initialized to -1
`%`$Intercept$ `array stores for each event the defender that intercepts it`
2 $result \leftarrow 0$
3 **for** $u, v \in \{1, \ldots, k\} \times \{1, \ldots, k\}, u \neq v$ **do**
4 $\quad E' \leftarrow \{w \mid Intercept[w] \in \{u, v, -1\}\}$
$\quad$ `%`E' `stores the set of attack events intercepted by defender` u, v `and the attacks not intercepted by any defender`
5 $\quad \tilde{n} \leftarrow E'.size$
6 $\quad$ T $\leftarrow$ an $(\tilde{n}+1)^2$-length array initialized to $-\infty$
7 $\quad T[0] \leftarrow 0$
$\quad$ `%Apply the DP algorithm for defender` u `and` v
8 $\quad$ **for** $mask \leftarrow 0$ **to** $(\tilde{n}+1)^2 - 1$ **do**
9 $\quad\quad \overline{a_1 a_2} \leftarrow mask$
10 $\quad\quad$ **if** $a_1 = a_2$ **then**
11 $\quad\quad\quad$ **continue**
12 $\quad\quad$ **end**
13 $\quad\quad ma \leftarrow argmax_i a_i$
14 $\quad\quad$ **for** $p \in prev(a_{ma}, ma)$ **do**
15 $\quad\quad\quad$ **if** $\forall i\ a_i \neq p$ **then**
16 $\quad\quad\quad\quad pm \leftarrow mask - (a_{ma} - p) \cdot (n+1)^{2-ma}$
17 $\quad\quad\quad\quad T[mask] \leftarrow \max(T[mask], T[pm] + 1)$
18 $\quad\quad\quad$ **end**
19 $\quad\quad$ **end**
20 $\quad$ **end**
21 $\quad$ **if** *Solution is improved* **then**
22 $\quad\quad$ Update $Intercept, result$
23 $\quad$ **end**
24 **end**
25 **return** $result$

We can try different defender pairing orders and choose the best one. In our EDP implementation, we choose to run line 3 of algorithm 8 in 3 different iteration ordering of u, v. algorithm 8's running time is $O(k^2 n^3)$ as we try $O(k^2)$ pairs of defenders, and each

run of the DP algorithm takes $O(n^3)$.

5.1.7 Handling Infinite Attack Streams with a Finite Look-Ahead Horizon

For Problem 6 where the attacks $\langle loc_i, t_i \rangle_{i=1}^{\infty}$ may be infinite, and the look-ahead horizon is finite, not all attacks are revealed at once. As such, previous methods cannot be directly applied. As the future attack sequence cannot be foreseen, defenders need to *react* to information (e.g., the attack events in the next T time interval) obtained so far.

Towards addressing the problem, a greedy *replanning* approach can be applied. Whenever an attack event is observed, the defender team replans the capture sequence given the new attacks added to the attack queue. This gives rise to an online algorithm sketched in algorithm 9.

Algorithm 9: Online Exhaustive Defender Pairing

Data: $E = \langle loc_i, t_i \rangle_{i=1}^{\infty}$: a stream of attack events
$loc_1, \ldots, loc_k$: initial locations of the k defenders
$v_1, \ldots, v_k$: speeds of the k defenders
$E' \leftarrow$ an empty queue
`%`E' `stores attack events seen so far`
while *new attack events added to* E' **do**
  Apply algorithm 8 to compute a plan for the defenders and attack events E'
  Execute the plan, pop out from E' attack events passed, and update $loc_1, \ldots, loc_k$ to the defenders' current locations until new attacks are foreseen
end

5.1.8 Evaluation and Empirical Study of `BDHD`

In this section, we describe our experimental study of the methods mentioned or proposed in the chapter, which serves two purposes: (1) demonstrating the performance of our fast computational methods, and (2) characterizing the solution structure of `BDHD` under different problem settings.

The algorithms are implemented in C++ and run on a 3.6GHz quad-core CPU with 16G memory. For integer linear programming, Gurobi [3] is used as the solver with a time limit of 60 seconds applied. In the evaluation, two main factors are considered: computation

time and solution quality measured by interception rate.

The instances were generated for up to dozens of defenders, and the defender speeds are sampled uniformly at random from 1 to v_{max}. The attack sequence was generated according to a Poisson process with parameter λ (i.e., the time gap between two consecutive attacks conforms with an exponential distribution $P(t) = \lambda e^{-\lambda t}$), and the locations of the attack are sampled uniformly at random from the boundary. We use 4 types of boundary for testing a) unit circle S^1, b) unit interval $I = [0, 1]$ c) unit square $I^2 = [0, 1] \times [0, 1]$, and d) unit sphere S^2.

5.1.9 Infinite-Horizon BDHD: Basic Performance Evaluation

This section provides experimental study and comparisons between the three methods described in the section: dynamic programming (DP), integer linear programming (ILP), and exhaustive defender pairing (EDP) algorithms.

To test and compare the general performance of these methods, we set the defending task on S^1 (i.e., a circular boundary with a distance of 2π) with the attack sequence's coming rate λ (used in the Poisson process) proportional to k, the number of defenders. The results on computational time and interception rate are given in Figs. Figure 5.3 and Figure 5.4.

For 2 to 7 defenders, in Figure 5.3, we show the computation time of running the algorithms. Each data point is an average of 20 runs. We remind the readers that both DP and ILP guarantee to produce an optimal solution (if they can finish). For the two-defender scenario, which is the main focus of [10], DP is the most efficient. However, DP only fully scales for $k = 2, 3$ defenders, and start to peter out for $k = 4$ defenders due to memory limit (so its running time is not shown for $k > 4$). As can be observed, ILP demonstrates much better scalability in comparison to DP, but fails to find a solution for $k > 5$ defenders.

As expected, exhaustive defender pairing (EDP) has the least time cost for $k \geq 3$. Figure 5.4 shows the corresponding solution qualities of these methods. We observe that,

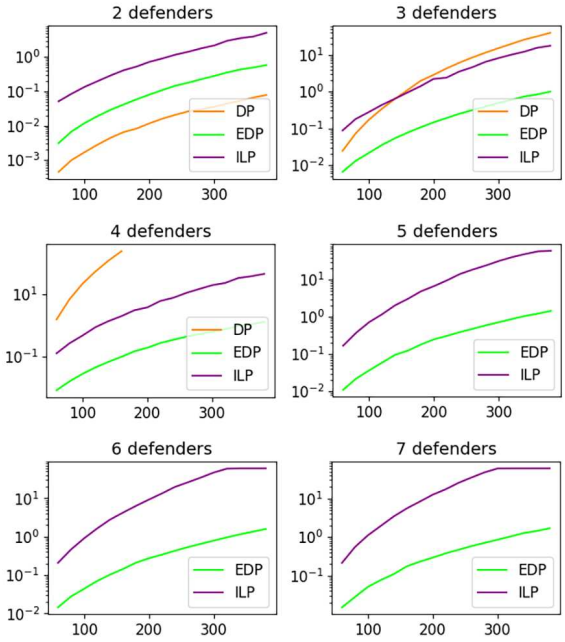


Figure 5.3: Computation time in seconds (y-axis) for dynamic programming (DP), integer linear programming (ILP) and EDP for 2 to 7 defenders and up to 400 attack events (x-axis). DP quickly becomes intractable as the number of defenders goes beyond 3; ILP is about 2-3 magnitudes slower than EDP.

while EDP does not guarantee solution optimality, it generated solutions that are virtually the same as the exact methods (DP and ILP). Through these empirical evaluations, EDP demonstrates superior scalability with negligible solution optimality loss.

5.1.10 Scaling up the Number of Defenders

The number of defenders appears in the index term of the time complexity of the DP algorithm, which limits it from scaling up the number of defenders. Hence, we compare the performance of ILP and the local search algorithm when pushing the number of defenders up to dozens. With the main goal being the evaluation of optimality, we limit the attack events to 200 so that the ILP method can return the optimal solution in 60 seconds. At the same time, to make the problem more challenging, λ is increased to $2 \cdot k$ from k so that not all attacks can be intercepted. Figure 5.5 and Figure 5.6 show the corresponding computation time and interception rate, each data point is the average of 20 runs.

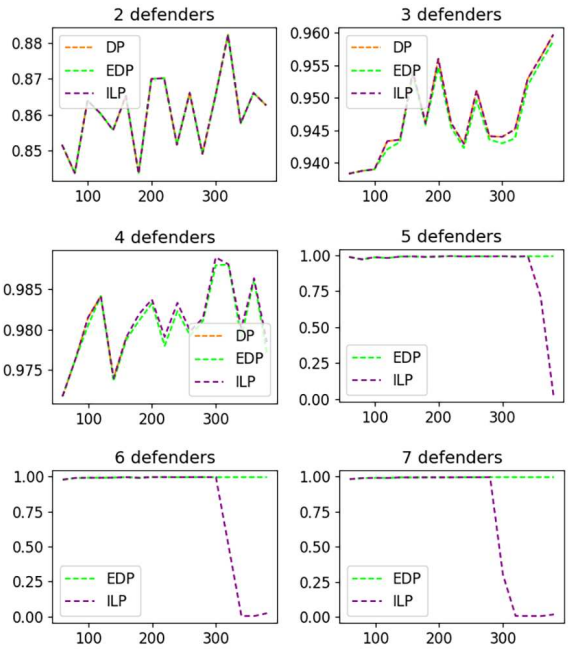


Figure 5.4: Solution quality (interception rate) for dynamic programming (DP), integer linear programming (ILP) and EDP for 2 to 7 defenders. For all settings, EDP achieves optimality nearly identical to the optimal DP and ILP (when DP and ILP can complete the computation). ILP starts to fail as the number of defenders reaches 5 and the number of events exceeds 300.

As can be observed, EDP, due to its quadratic dependence on the number of defenders, consistently and significantly outperforms the ILP method in terms of computation time. At the same time, it yields virtually the same level of optimality as ILP, which guarantees that the result is optimal.

5.1.11 Impact of Defender Heterogeneity

To explore the impact of the diversity of defender speeds on interception rate, we focus on a 5-defender setting and to make the difference more visible, λ is set to be $5 \cdot k = 25$ so that the interception rate no more than 80%. Here, the defender speed diversity is tested by setting the v_{min} as 1, and v_{max} from 1 (uniform speed) to 10, and then normalizing the defender speed to make them sum up to 15 (i.e., with an average of 3.0). Each data point is based on 100 runs. The result is summarized in Figure 5.7, from which we can observe that uniform speed $v_{max} = 1$ has the least interception rate, while the interception rate gets into

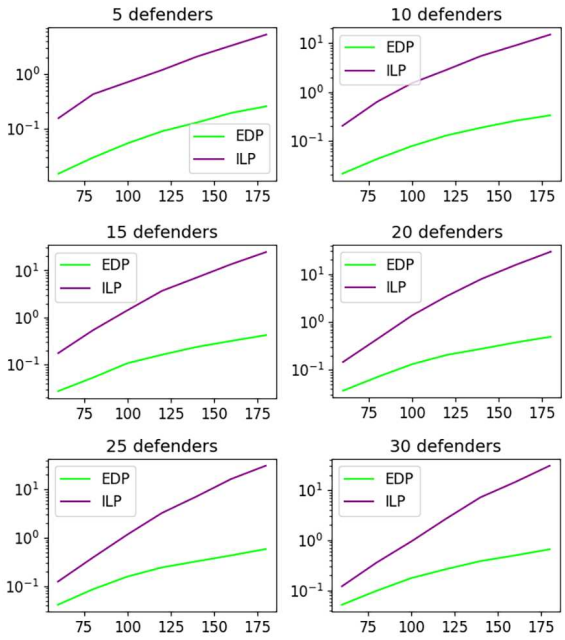


Figure 5.5: Computation time in seconds for ILP and EDP algorithms for 5 to 30 defenders and up to 200 attack events (Limit the number of events because as shown in Figure 5.3, more events will effectively cripple ILP). EDP consistently demonstrates 1-2 magnitudes faster computation time compared with ILP.

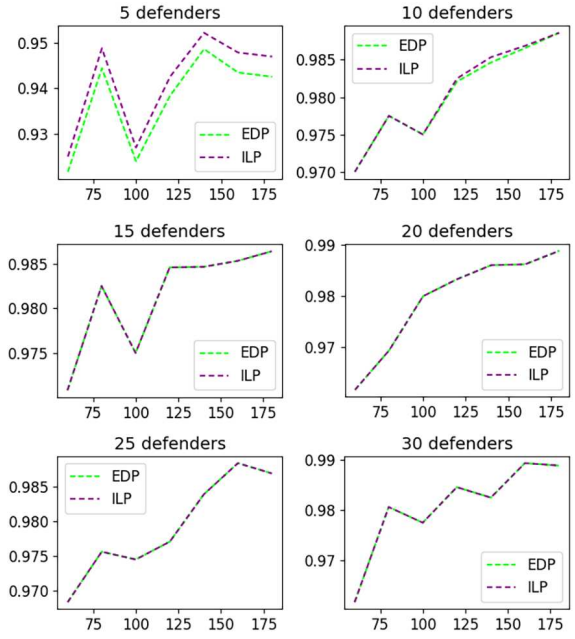


Figure 5.6: Solution quality (interception rate) for the ILP and EDP for 5 to 30 defenders. It is straightforward to observe that EDP achieves defending quality nearly identical to the optimal ILP solution (while ILP is still scalable).

its maximum after $v_{max} : v_{min} = 3$ or 4. This suggests that heterogeneity can significantly enhance the interception rate, warrant the effort going into the current study (and previous study of heterogeneous defender setups). At the same time, as heterogeneity continues to increase, the effect becomes negligible. This also follows intuition: when the capability of the defenders varies too much, they can no longer effectively collaboratively intercept attacks.

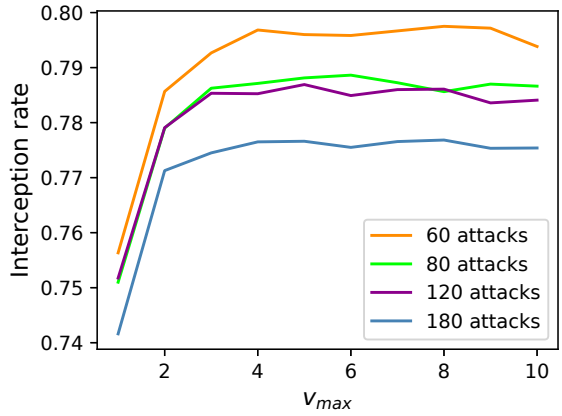


Figure 5.7: Solution quality (interception rate) for a different level of defender speeds diversity. It can be observed that the major difference happens as $v_{max} : v_{min}$ increases from 1 to up to 4. After that, the increase of interception rate no longer shows improvements.

5.1.12 Impact of Attack Rate λ

With the increase of λ in the Poisson process for the attack sequence, there will be fewer attacks intercepted, and more defenders will be required to reach the same interception rate. In this regard, we test the interception rate computed using EDP for 1 to 20 defenders and different λ from 1.0 to 60.0. The number of attacks is set to be 400, and each data point represents an average of 20 runs. Figure 5.8 shows the resulting capture rate with respect to the attack rate λ as a heat map for a different number of defenders k, which, unsurprisingly, suggests a gradual change as one might expect. Nevertheless, such a graph can serve as a quick reference for selecting an appropriate number defenders given an expected attack rate.

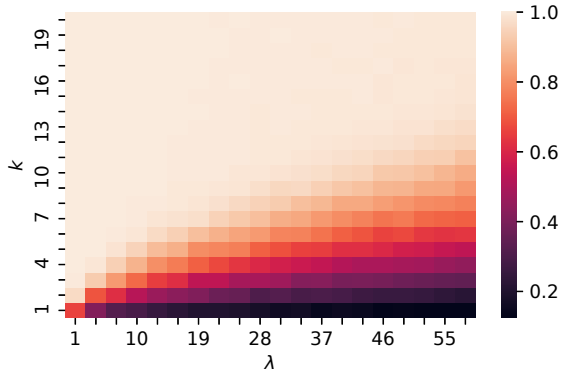


Figure 5.8: Interception rate with different λ (x-axis) and k (y-axis, number of defenders). As one might expect, as attacks happens more frequently, holding other factors unchanged, the interception rate decreases. Similarly, holding other factors unchanged, increasing the number of defenders leads to increased interception rates.

5.1.13 Impact of Boundary Topology

EDP applies to BDHD with arbitrary boundary topology; we evaluated EDP on $I = [0, 1]$ (unit interval), S^1 (circle with a total circumference shrinked to 1), I^2 (unit square), and S^2 (sphere with surface area of 1) (see Figure 5.9) with $k = 5$, $v_{max} = 5$ and λ from 1 to 200. The results are shown in Figure 5.10, where each data point is the average of 20 runs. We observe that I and S^1 have similar difficulty to defend, with I being a bit more challenging due to having boundaries. The same is observed for I^2 and S^2.

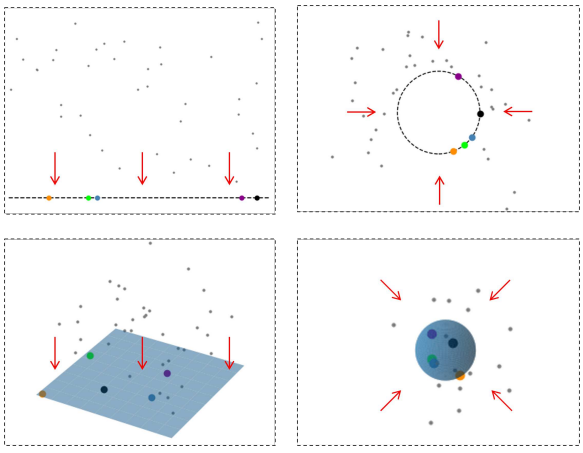


Figure 5.9: BDHD with boundary topologies I (line segment), S^1 (circle), I^2 (unit square) and S^2 (sphere). The red arrows illustrate attack directions. The pictures are captured from animations of actual simulations, available at https://youtu.be/x0mQD_7RhKI.

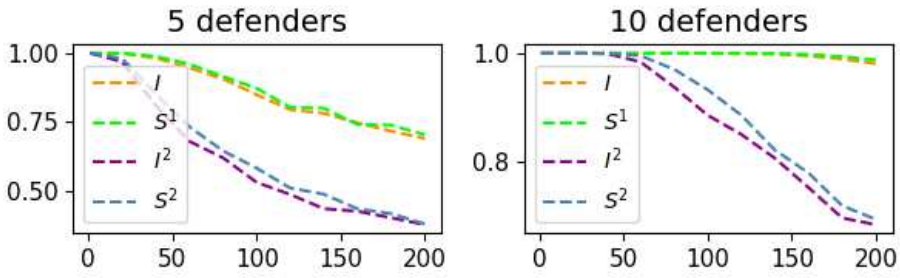


Figure 5.10: Interception rate for I, S^1(with length 1), I^2, S^2(with surface area 1) with different λ from 1 to 200. Degradation as the number of attacks increases is as expected.

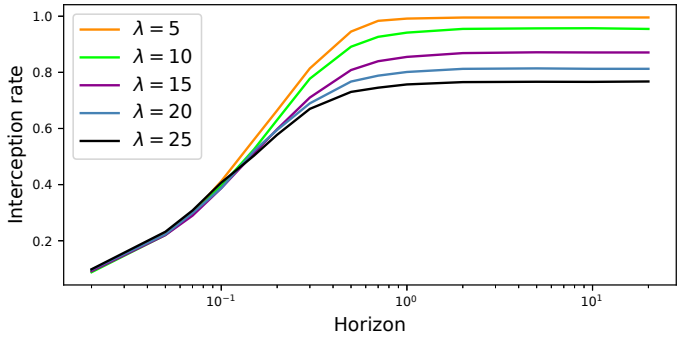


Figure 5.11: Interception rate with respect to different horizons (logarithmic scale). It can be observed that longer horizon allows better planning.

5.1.14 Infinite Attack Stream with Finite Look-Ahead Horizon

algorithm 9 describes an online EDP algorithm for handling the finite horizon version of the problem where the defenders only have the information about attacks in the next T time frame. Here, we present some simulation studies using the 5-defender and 400 attacks setup on an S^1 boundary with $v_{max} = 5$. Figure 5.11 gives the interception rate with respect to varying defender horizons. Each data point is an average of 20 runs. The interception rate converges to around 1 with a fairly short horizon, indicating that a long look-ahead horizon may not be necessary for effectively intercepting attacks.

Lastly, in Figure 5.12, we provide the interception trajectories of a typical run for ILP, EDP, and online EDP.

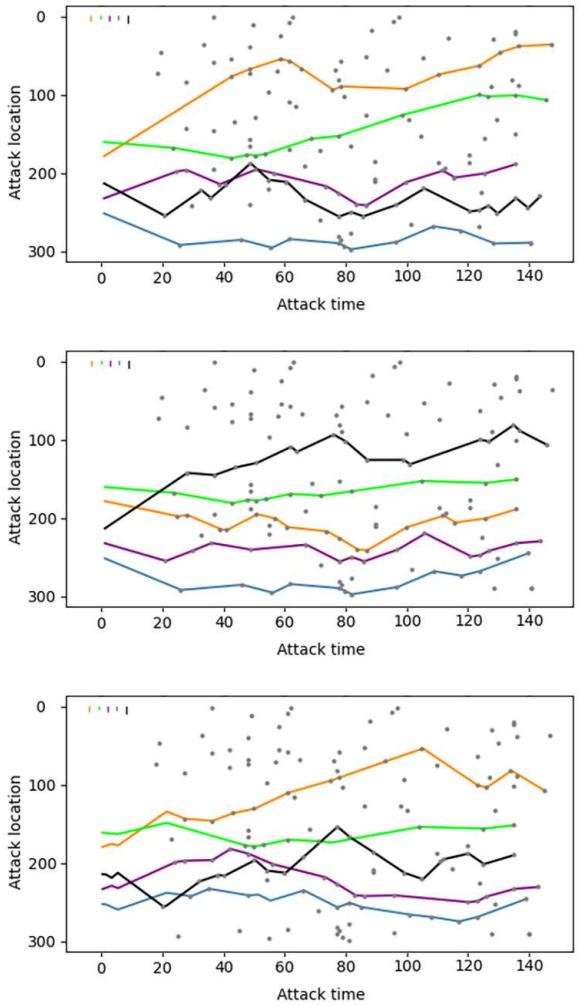


Figure 5.12: The results computed by ILP, EDP, and online EDP with a horizon of 60, for a sequence of 140 attacks (x-axis is the location, y-axis is the time of attacks). The number of captured attack events is 71, 70, and 67, respectively.

5.1.15 Conclusion

This chapter studied the heterogeneous perimeter defense problem, as formulated in [10], with the aim of significantly boosting the scalability to work on many defenders while achieving high levels of solution optimality. Toward achieving this goal, we developed an exact integer programming-based algorithm with better scalability than the original dynamic programming (DP) based algorithm from [10]. We then further developed a fast and highly-effective heuristic based on the pairwise application of DP for computing near-optimal solutions, scaling to dozens of defenders. This pairwise application of DP is of

independent algorithmic interest. In addition, we extend our solution to a more realistic online version in which only attackers within a finite time window are visible. The algorithms are thoroughly evaluated on a diverse set of boundary topologies including circles, intervals, spheres, and squares. The evaluation not only confirms that EDP is highly effective but clearly demonstrates the benefit of employing heterogeneous defenders.

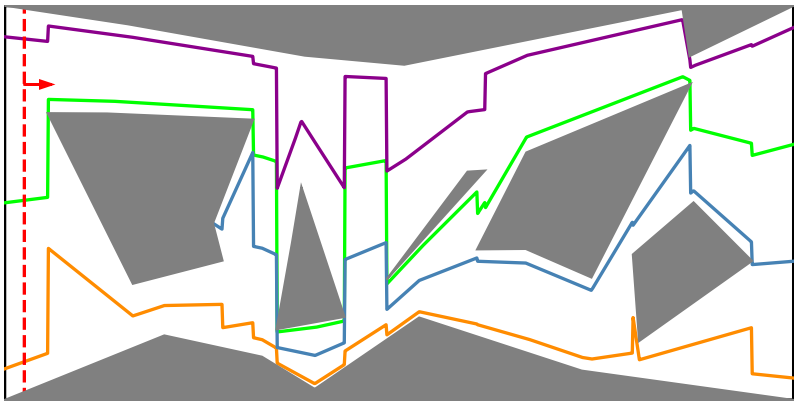
5.2 Sweep Line Coverage

5.2.1 Introduction

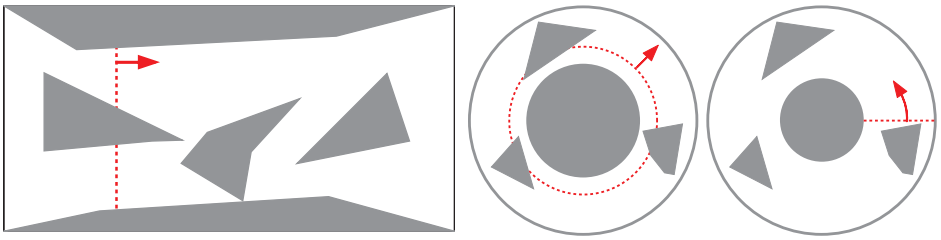
Searching for a static or moving target in a planar environment is a classic problem in robotics [58, 59, 60, 61, 55]. The setting applies to many real-world applications, including searching for lost person/object, checking for potential hazards, and generally, search and rescue tasks conducted in a known environment. Research tackling this problem mostly focuses on devising a search plan with different types of objectives, such as minimizing the total length of the frontier of the search schedule from the start to the end [56], minimizing the number of robots used for the search plan in a visibility-based robot sensing model [57], and so on.

In certain cases, the high-level plan for searching a given region may already be pre-determined and fixed. For example, in search and rescue efforts, a frequently carried out plan is to perform a single sweep of an environment with a marching frontier, which is easy to execute when many participating robots/agents are involved. The high-level search plan may also be determined by existing algorithms that compute only the search frontier. However, even in the case where the search plan consists of pre-determined sweep (frontier) lines; it remains non-trivial to find an optimal organization of the mobile robots to execute the search plan, for either minimizing the number of robots needed for a given sensing probability requirement or utilizing a fixed number of robots to maximize the minimum sensing probability of locating something in the environment.

In this work, we address the challenge of how to best allocate many robots to execute



(a) Optimal robot allocations in a vertical sweep



(b) Vertical (c) Circular (d) Radial

Figure 5.13: (a) An illustration of robots' locations along a left-to-right vertical sweep schedule. Four robots are allocated to execute the vertical sweep, and their trajectories are illustrated in different colors. (b)(c)(d) Illustrations of three use cases: vertical, circular, and radial sweeps.

a pre-determined search schedule for a known environment. More specifically, for a two-dimensional closed and bounded workspace, and a known search schedule, which gives a search frontier for any given time, the robot guards are required to stay on the search frontier to carry out the sensing task. Because each robot's coverage quality deteriorates with distance, their relative placement on the frontier must be carefully decided to maximize the coverage quality. For the setup, we focus on the problem of finding the minimum number of robots required to execute a pre-determined sweep schedule such that a minimum sensing quality is guaranteed for each point in the workspace. Solutions to this minimization problem readily translate to solutions for maximizing the coverage quality for a fixed number of robots, which is a dual problem.

In summary, the main contributions of this work are twofold. First, we generalize

the notion of boustrophedon decomposition [138], which partitions the plane via vertical sweep, into a decomposition of the plane with a *continuous monotone sweep schedule*, which we prove can always be represented as a directed acyclic graph (DAG). Then, we show the problem of minimizing the number of line guards required to execute a sweep schedule can be transformed into a network flow problem. Since the generalized boustrophedon decomposition and the network flow problem can be solved in low polynomial time; our method achieves high levels of scalability. The strengths of our method are further corroborated in extensive simulation experiments on three realistic use cases: *vertical sweep*, *circular sweep*, and *radial sweep* (see, e.g., Figure 5.13 (b)(c)(d)).

5.2.2 Preliminaries

In this section, we first describe a general probabilistic robot sensing model used in this section and introduce the notion of a *continuous monotone sweep schedule*, along which robots must be deployed to carry out the search task. Then, the problem of finding the minimum number of robots for a certain sweep schedule is formally defined.

Robot Sensing Model

In this work, a robot is assumed to be a *line guard* capable of sensing relevant events happening on a continuous line segment passing through the robot. On the line segment guarded by a robot, for a (point) target at a distance of r to the robot, the robot has probability $\rho(r)$ of detecting or capturing the target, where $\rho : \mathbb{R}^+ \to [0, 1]$ is a decreasing sensing probability function.

Individual robots' 1D sensing range aligns to form a *sweep line* or *sweep frontier*. It is assumed that robots carry out their sensing functions independently without interfering with each other. For a point p in the sweep line, if it falls between two robots r_1, r_2, the coverage of that point is provided by r_1 and r_2, and the probability of target detection at that point is computed as $1 - (1 - \rho(\ell_1)) \cdot (1 - \rho(\ell_2)) = \rho(\ell_1) + \rho(\ell_2) - \rho(\ell_1) \cdot \rho(\ell_2)$, where ℓ_1

(resp., ℓ_2) is the distance between p and r_1 (resp., r_2). If it lies at the ends of the segment, the coverage of that point is only provided by the closest robot, and the coverage probability is simply $\rho(\ell)$, where ℓ is the distance between p and the robot. These are illustrated in Figure 5.14. We note that *sweep lines* are not necessarily straight. For example, a sweep line could be part of a circle in the case of circular sweeps (Figure 5.13(c)).

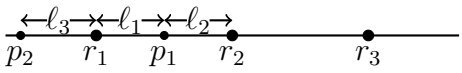


Figure 5.14: Illustration of the robot sensing model. The segment is being covered by three robot guards $r_1 \sim r_3$. The coverage probability of p_1, falling between robots r_1 and r_2, is given as $\rho(\ell_1) + \rho(\ell_2) - \rho(\ell_1) \cdot \rho(\ell_2)$, and the coverage probability of p_2, covered only by r_1 is given as $\rho(\ell_3)$.

For covering a line segment with a length of ℓ and sensing function of ρ, we denote as $\zeta : \mathbb{R}^+ \to \mathbb{N}^+$ a primitive for computing the minimum number of robots needed to achieve the required minimum covering probability of ρ_0.

Take the exponential decaying sensing probability function as an example, where $\rho(r) = e^{-c \cdot r}$, $c > 0$ is some constant. In this case, the maximum distance at the two ends to guarantee the sensing probability of ρ_0 is $d_1 = -\ln(\rho_0)/c$. If a point is between two robots with distance ℓ_1 and ℓ_2 to the two robots, its sensing probability is bounded by

$$\rho(\ell_1) + \rho(\ell_2) - \rho(\ell_1) \cdot \rho(\ell_2) \geq 2e^{-\frac{c \cdot d}{2}} - e^{-c \cdot d}$$
$$where\ d = \ell_1 + \ell_2$$

So, the required maximum distance between two neighboring robots is $d_2 = -2\ln(1 - \sqrt{1 - \rho_0})/c$. Therefore, the minimum number of robots required to cover a line segment with length ℓ is $\zeta(\ell) = \max(1, 1 + \lceil(\ell - 2 \cdot d_1)/d_2\rceil)$.

5.2.3 Problem Formulation

In this chapter, we work with a 2D compact (i.e., closed and bounded) workspace $\mathcal{W} \subset \mathbb{R}^2$, which can contain a set of obstacles. An existing *sweep schedule* is an input to the problem,

which we define the *continuous monotone sweep schedule* for $\mathcal{W}$ as

Definition 30 (Continuous Monotone Sweep Schedule)**.** *A function $P(t)$ that maps a positive timestamp t to a continuous curve is a continuous monotone sweep schedule for $\mathcal{W}$ if $P(t)$ changes continuously, and for each point $p \in \mathcal{W}$, there exists a single t' such that $p \in P(t')$.*

Intuitively, a continuous monotone sweep schedule defines a function that maps the time step to a 1-D curve in the workspace that sweeps through each point in $\mathcal{W}$ only once. Depending on the continuous curve, $P(t)$ could take various forms. For example, the sweep line may be straight in a search-and-rescue scenario. Or the sweep line may be circular in a search-and-capture scenario. In the rest of this chapter, we simply refer to a continuous monotone sweep schedule as a *sweep schedule*. Next, we introduce the notions of *arrival time* and *monotone chain*.

Definition 31 (Arrival Time)**.** *For a search schedule $P(t)$, and a point in the workspace $o \in \mathcal{W}$, the arrival time at o, $arrival(o)$, is defined as the time step t when $o \in P(t)$.*

Definition 32 (Monotone Chain)**.** *For a bounded 2D chain: $\tau(s) : [0, 1] \to \mathcal{W}$, where τ is a continuous function, it is considered as a monotone chain to the search schedule $P(t)$ if and only if*

$$s_1 < s_2 \Leftrightarrow arrival(\tau(s_1)) < arrival(\tau(s_2)).$$

In a search schedule or plan, $P(t)$ may be intersected by obstacles. In this case, $P(t)$ is separated into multiple continuous segments; each of these segments requires a dedicated group of robots. That is, robots on one continuous segment cannot provide coverage of other segments.

With the probabilistic robot sensing model, we formulate the robot team scheduling problem for a given sweep schedule as the following,

Problem 7. *Given a sweep schedule $P(t)$ for a 2D region $\mathcal{W}$, and a group of robots with coverage capability function ρ, what is the minimum number of robots required to execute*

the sweep schedule on the sweep line, such that for every point o in $\mathcal{W}$, the probability that o is covered is at least some fixed $0 < \rho_0 \leq 1$?

5.2.4 Optimal Robot Allocation

Our proposed algorithm can be divided into two steps at the high level. In the first step, the algorithm conducts a *generalized boustrophedon decomposition* for the given environment along the sweep line, which generates a *directed acyclic graph* (DAG) representation of the workspace $\mathcal{W}$ for a given sweep schedule. Using the DAG, a max-flow-based algorithm is then applied to compute the minimum number of robots required for executing the sweep schedule, as well as the corresponding arrangement of robots.

5.2.5 Generalized Boustrophedon Decomposition

In solving search and coverage problems, various decomposition techniques have been proposed, including trapezoidal, Voronoi, boustrophedon, Morse decompositions, and so on [139, 138, 140, 141]. For our robot allocation task, it is also natural to start with a decomposition of the environment. However, we need a decomposition supporting non-straight boundaries between the decomposed cells, created by the sweep schedule. For this, we propose a generalization of boustrophedon decomposition.

Before describing the generalization of boustrophedon decomposition, we briefly introduce boustrophedon decomposition (readers are referred to [138] for further details), which in turn is based on trapezoidal decomposition. The difference is that it removes the sweeping events that cross inner vertices. As illustrated in Figure 5.15, compared with trapezoidal decomposition, boustrophedon decomposition has fewer cells, which leads to fewer (back-and-forth) boustrophedon motions.

We extend the boustrophedon decomposition from using only vertical sweep lines to allow the use of any *continuous monotone sweep schedule*. We are given a sweep schedule $P(t)$ for a workspace $\mathcal{W}$, which could take any curved form (see Figure 5.16). By fol-

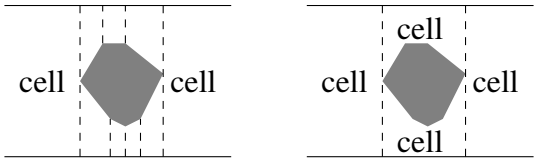


Figure 5.15: [left] A trapezoidal decomposition, creating a total of 9 cells. [right] Boustrophedon decomposition of the same environment, creating only 4 cells.

lowing the sweep schedule, it is possible to construct a decomposition of $\mathcal{W}$, based on the events of cell splitting and merging.

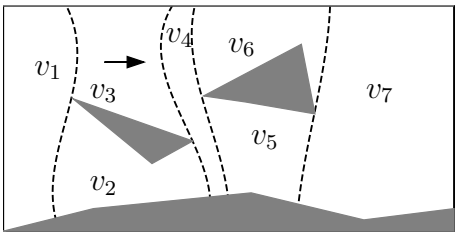


Figure 5.16: Suppose we have a sweep schedule that sweeps the environment from left to right, with curved sweep fronts. The curves, as they cross critical vertices of objects, are shown as the dashed lines. The generalized boustrophedon decomposition decomposes and $\mathcal{W}$ into 7 cells, $v_1 \sim v_7$. It is important to note here that, for any continuous monotone sweep schedule, a decomposition can be obtained.

Theorem 33. *A sweep schedule can be organized in a DAG by the generalized boustrophedon decomposition.*

Proof. Following the sweep schedule, we can conduct a generalized boustrophedon decomposition of the workspace $\mathcal{W}$. A node in the DAG will represent a cell after the decomposition, whose parents and children are the predecessor and successor cells along the sweep schedule.

Additionally, we add a source node that links to all nodes without a parent and add a terminal node that links from all nodes without a child. Since obstacles in the environment can contain concave vertices, we must consider two special cases involving concave vertices for the construction of DAG, as illustrated in Figure 5.17. When a line segment of the sweep schedule ends at a concave vertex, the corresponding node in the DAG is linked to the terminal node t. When a line segment of the sweep schedule starts at a concave

vertex, the corresponding node in the DAG is linked from the source node s. In mapping these scenarios to detailed plans, it means that some robots will start later or end earlier compared with others.

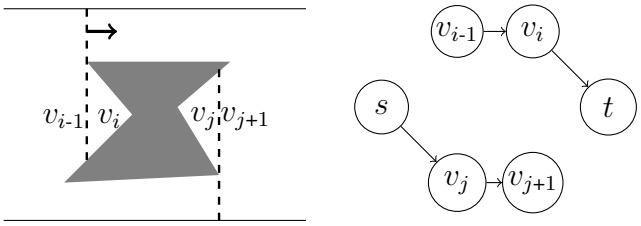


Figure 5.17: An example that contains two concave scenarios. As a result of the DAG construction, some robots will start their work at v_j (by having s as its parent) and some robots will end their work at v_i (by having t as its child).

□

The implementation of the generalized boustrophedon decomposition is similar to vertical decomposition, we provide here an implementation algorithm 10 adapted from [142] under the *generation position* assumption (i.e., there are no degenerative settings). Basically, the algorithm works by maintaining a binary search tree for *boundary chains* representing cell boundaries. The algorithm considers two types of events during the sweeping process: splitting a cell into two cells and merging two cells into one cell. Since the time cost mainly comes from maintaining the binary search tree of chains, with an efficient implementation, algorithm 10 requires $O(n \log n)$ time, where n is the complexity of the environment.

5.2.6 Reduction to Circulation with Demand

Since we are working with *monotone sweep schedules*, for any point p in $\mathcal{W}$, it can only be contained in $P(t')$ for a single t', i.e., each point $p \in \mathcal{W}$ is only swept once. Denote the length of the line segment where it is contained as L, which requires at least $\zeta(L)$ robots inside that segment at time t' (recall that ζ is the primitive defined in Sec. subsubsection 5.2.2). For each decomposed cell, the minimum number of robots required is $\zeta(\ell_{max})$,

where ℓ_{max} is the maximum length of the sweep line inside that cell. Given a DAG $G(V, E)$ that represents the sweep schedule, the arrangement problem of the robots along the sweep line can be transformed into a network flow problem on the DAG. For each node $v \in G$, there is a requirement of coverage for the node, $demand(v)$, which can be computed as $\zeta(v.\ell_{max})$. Given the DAG and the demands, we are then to "flow" the robots through the schedule, allocating a certain number of robots to each decomposed cell along the way to satisfy these demands.

Specifically, our problem may be further cast as a *circulation with demand* problem [143], with the following augmentation. We replace each node v in G with two vertices v_1 and v_2 and replace every edge uv in the previous graph with edge u_2v_1, as illustrated in Figure 5.19. The edge between v_1 and v_2 has flow demand of $demand(v)$. The following minimum circulation with demand problem is then obtained.

Algorithm 10: `GenBouDecomp(`P, $\mathcal{W}$`)`: Generalized Boustrophedon Decomposition

---

Data: $P(t)$: a sweep schedule. $\mathcal{W}$: the workspace.
Result: a DAG from the decomposition.

1. 1. Separate the boundaries of obstacles into a set of *monotone chains*, where each chain has the points on it *arrival time* arranged in an increasing order.;
2. 2. $T \leftarrow$ A binary search tree of the chains
3. 3. Construct an *event array* that contains the events of the start of a chain and the end of a chain, sorted by their occurrence time during the sweep.
4. 4. Iterate over the event array, which inserts and deletes chains from the binary search tree T, while making sure that T represents the current order of the chains.
5. When two chains are added to T, a cell is split into two new cells in the case of a convex vertex, or a new cell is created for a concave vertex, where an edge directed from s is added.
6. When two chains are removed from T, two cells are merged into a new cell in the case of a convex vertex or a cell disappears for a concave vertex, where an edge is added directed to t.
7. 5. Return the DAG constructed based on $P(t)$.

Problem 8 (Minimum Circulation with Demand)**.** *There is a graph $G(V, E)$, where s, t are the source and terminal nodes. Every edge uv in G has a demand of $d(uv)$ and a capacity*

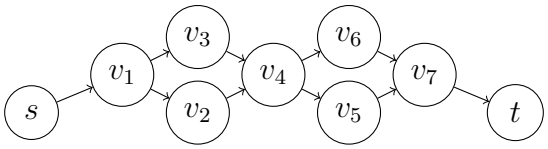


Figure 5.18: The constructed DAG from the example in Figure 5.16

Algorithm 11: `MinSweepDAG(`dag, ζ`)`

Data: $dag(V, E)$: the DAG obtained from `GenBouDecomp`. ζ: the sensing requirement primitive function.

Result: dag: the updated dag with the robot allocation information

```
G' ← a new empty graph;
for v ∈ dag do
    G'.add_vertex (v_1), G'.add_vertex (v_2);
    G'.add_edge (v_1, v2, capa=∞, demand=ζ(v.ℓ_max));
    for u ∈ dag.neighbor[v] do
        G'.add_edge (v_2, u, capa=∞, demand=0);
    end
end
MinCirculationWithDemand (G');
for v ∈ dag do
    dag.v.guards_num ← G'.flow[v_1][v_2];
    for u ∈ dag.neighbor[v] do
        dag.flow[v][u] = G'.flow[v_2][u_1];
    end
end
return dag
```

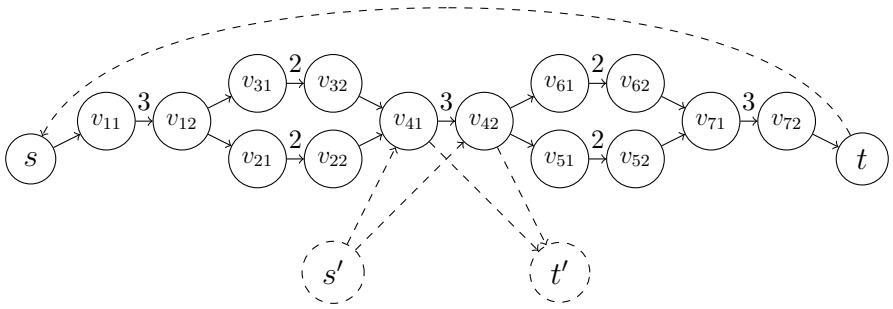


Figure 5.19: Transform the DAG in Figure 5.18 into a circulation with demand problem. The value above each edge represents the demand of the edge, which is eliminated when it is zero. The source node is s, and the target node is t. s' and t' are the auxiliary source and target for solving the "circulation with demand" problem. Also, auxiliary edges are added between s', t' and every other vertex.

of $c(uv)$. *Compute minimal network flow from* s *to* t *that saturates all the edge demands.*

Readers are referred to [143] for the details of the classical "circulation with demand" problem solution with max-flow.

algorithm 11 outlines the operations used in this section. We note that the graph data structure needs to have the functionality of adding vertex and adding edge with edge demand, and $G'.flow[v_1][v_2]$ denotes the flow from v_1 to v_2 on G'.

5.2.7 The Complete Allocation Algorithm

The overall algorithm for robot allocation is described as in algorithm 12. To analyze the running time of algorithm 12 for a polygonal environment, we denote the environment complexity (number of vertices) as n. As mentioned in the previous section, the generalized boustrophedon decomposition takes $O(n \log n)$ time. As the generation of a node in the DAG comes from some chain insertion or deletion events, and an event would require one vertex to happen, there is at most $O(n)$ decomposed cells, i.e., nodes in the DAG. Similarly, adding edges between nodes comes from some chain insertion or deletion events, and the number of events is at most $O(n)$. So, both the number of edges and the number of nodes in the DAG is $O(n)$. Moreover, it is easy to see that the DAG is a planar graph since a node corresponds to a decomposed cell, and there is an edge between nodes only if the two corresponding cells are adjacent. The circulation with demand problem requires solving two max-flow problems based on the DAG. If we use the push-relabel algorithm [144] that runs in $O(V^2\sqrt{E})$, solving the max flow for this problem will cost $O(n^{2.5})$, since $|V|, |E| = O(n)$.

Remark 34. *In the case of having a fixed number of robots, and the objective is to maximize the minimum coverage probability of a point in* $\mathcal{W}$. *We can apply binary search on the minimum coverage probability we can guarantee, where algorithm 12 can be used to decide whether some coverage probability can be guaranteed by the fixed number of robots.*

Algorithm 12: `MinSweep` $(P, \mathcal{W}, \rho_0)$: Computing Minimum Number of Robots for a Sweep Schedule

Data: $P(t)$: a sweep schedule. $\mathcal{W}$: a compact workspace. ρ_0: required sensing probability guarantee.

Result: n: the minimum number of robot guards required. $plan$: the corresponding allocation plan

Construct ζ based the sensing model and ρ_0;

`%Primitive function` ζ `is to compute the minimum number of robots required for a continuous line segment`

$dag \leftarrow$ `GenBouDecomp`$(P, \mathcal{W})$;

$dag \leftarrow$ `MinSweepDAG`(dag, ζ);

$plan \leftarrow dag$;

`%The plan of the robots can be constructed based on the flow on the` dag`.`

return $dag.s.guards\_num$, ***plan***;

5.2.8 Simulation Evaluation

In this section, we perform a numerical evaluation of our proposed method with two goals: (1) confirms the scalability and (2) observe the behavior of the method as input parameters change. We implemented our proposed algorithms in C++. Dinic's algorithm is used to solve the max-flow problem [145] for simplicity. The methods are evaluated at an Intel® Core™ i5-10600K CPU at 4.1HZ. For the three use cases (Figure 5.13 (b)(c)(d)), we programmatically create a large number of test cases, with up to $6,000$ randomly generated polygonal obstacles. The number of vertices of each polygon ranges from 3 to 50, making the total vertices up to $100,000+$. Figure 5.20 shows the largest problem instances for the experiments with around $100,000$ vertices.

Algorithm performance. In a first set of evaluations, under an exponentially decaying sensing model, $\rho(r) = e^{-c \cdot r}$, we test the performance of our algorithm over the set of instances. Example allocation of robots along the sweep frontiers for the three cases are shown in Figure 5.13(a) and Figure 5.21. We limited the number of robots to be small so that the trajectories are more easily observed. It can be seen that the trajectories can vary

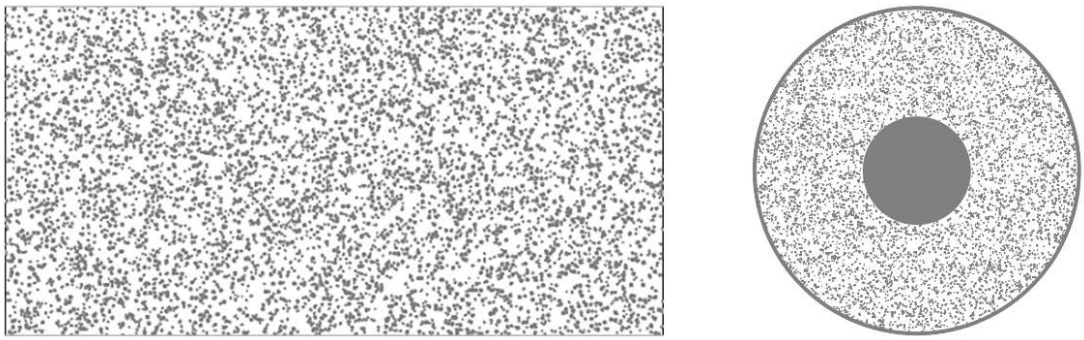


Figure 5.20: Examples of programmatically generated test environments with total vertices around $100,000$.

significantly along the sweep frontiers for each case.

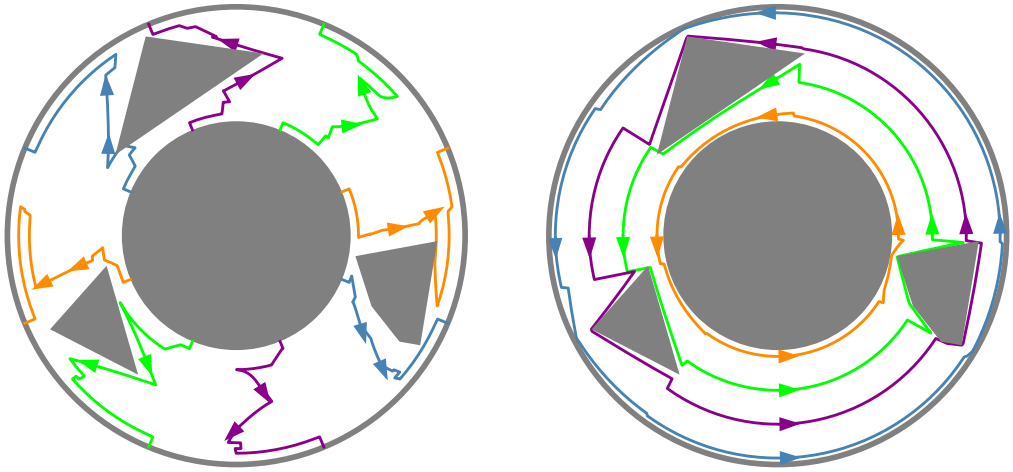


Figure 5.21: Example robot trajectories computed by our method for circular and radial sweep use cases, respectively.

In Figure 5.22, the computational performance is thoroughly evaluated. As can be observed, our method scales fairly well, taking less than two seconds to handle all cases, even those involving over $100,000$ vertices. Interestingly, the experimental running time is almost linear, even using the less efficient Dinic's algorithm. We suspect the near linear running time is due to the fact that the DAG is a planar graph; studies show that max-flow for planar graphs can be computed in $O(n \log n)$ [146]. Although the graph constructed when solving the "circulation with demand" problem is not planar, large portions are planar. This could reduce the actual time complexity of running the max-flow algorithm.

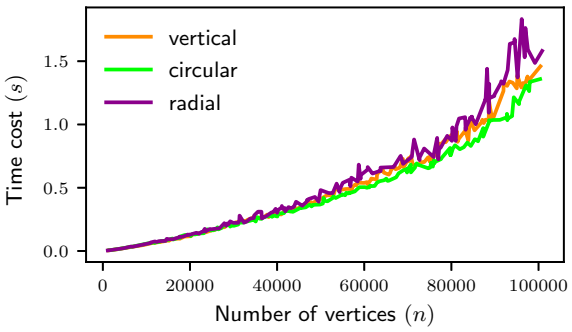


Figure 5.22: Running time in seconds with respect to environment complexity (number of vertices).

Different environment settings. Lastly, we evaluate the impact of environmental changes, considering factors including the spatial distribution of polygonal obstacles as well as the size distribution of the obstacles. All in all, three settings are considered: (1) The polygons are regularly distributed and are of similar size, (2) the polygons are randomly distributed and are of similar size, and (3) the polygons are regularly distributed, and their sizes can vary dramatically. The first and the third settings are illustrated in the top row of Figure 5.23. For these settings, we compare the time it takes to compute solutions for many polygonal obstacles and also the number of robots required to achieve 80% probabilistic guarantee under the exponential decay sensing model. As we can observe, there is little difference as the settings change.

5.2.9 Conclusion

In this section, we studied the problem of allocating a minimum number of robots for a sweep schedule with a probabilistic line sensing model, where a desired level of scanning quality can be guaranteed. Towards this, a novel decomposition technique is proposed that generalizes the well-known boustrophedon decomposition. The decomposition leads naturally to a transformation of the problem into a network-flow problem. Due to the decomposition and the transformation, our proposed algorithm runs in low polynomial time and even near $\tilde{O}(n)$ in simulation experiments for polygonal environments, where n is the

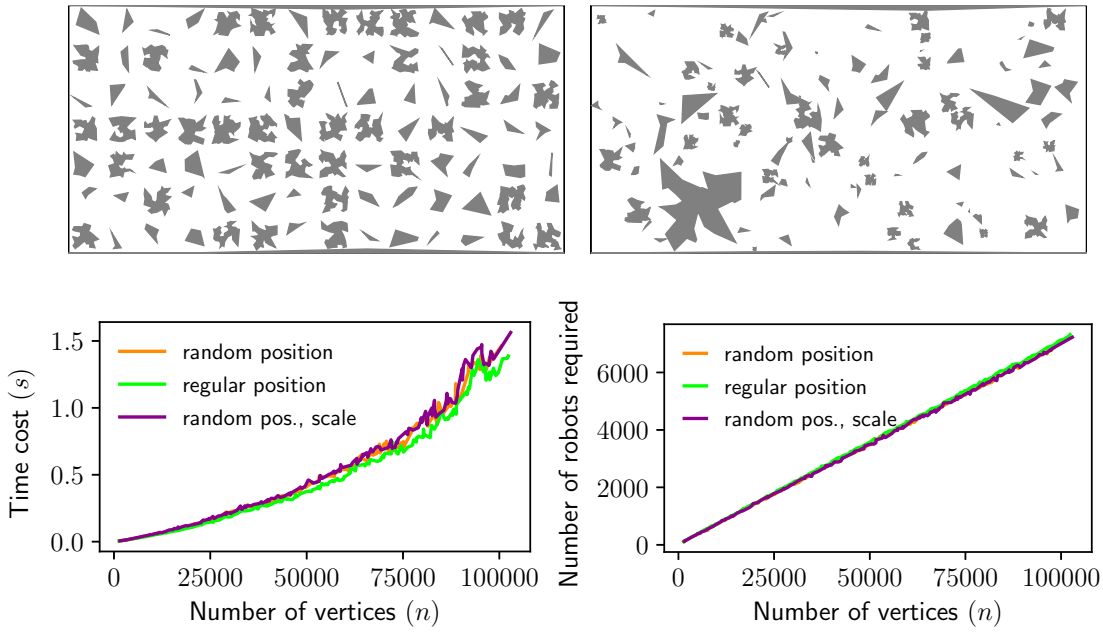


Figure 5.23: [top] Random instance with regularly distributed obstacles and instance with random obstacle scales and positions. [bottom] Running time for different randomly generated environments and the number of robots required for different randomly generated environments.

complexity of the environment, measured as the number of vertices of polygons. Extensive simulation-based evaluation corroborates the effectiveness of our algorithm, which is applicable to multiple types of environments.

CHAPTER 6
REALWORLD APPLICATION

This chapter discusses a relevant real-world application in the placement of UV (ultraviolet) lights to cover the surface of an environment for sanitization purposes. The problem can be seen as an extension of the art-gallery problem [12], and thus computationally intractable. Still, a combined integer programming and classical local search approach can be used to solve the problem effectively. The method is evaluated on three real scenarios: the interior of a bus, a subway or an ICU room.

6.1 Covering 3D surfaces

6.1.1 Introduction

We perform a systematic study of a class of mobile sensor[1] deployment optimization problems targeting the coverage of 2D surfaces embedded in 3D domains, applicable to a broad set of practical settings, for example: (1) the selection of surveillance camera locations for maximizing the joint coverage of an art museum, (2) the deployment of mobile robots with range-based sensing apparatus for the monitoring of complex 3D terrains with quality assurances, and (3) the optimization of UVC light source locations for disinfecting interior surfaces of public indoor spaces, e.g., airplanes, buses, hospital rooms, and schools (Figure 6.1), which is of key relevance to the ongoing COVID-19 pandemic. Despite the problems' apparent differences, the above-mentioned tasks fall under the general problem of placing mobile sensors for optimizing some form of coverage. This work is devoted to providing such a unifying problem formulation, understanding its rich structural properties, and delivering effective computational methods for readily solving the multiple variations,

[1] As will be explained, "mobile sensor" is used here in a broad sense.

which all have high application potentials.

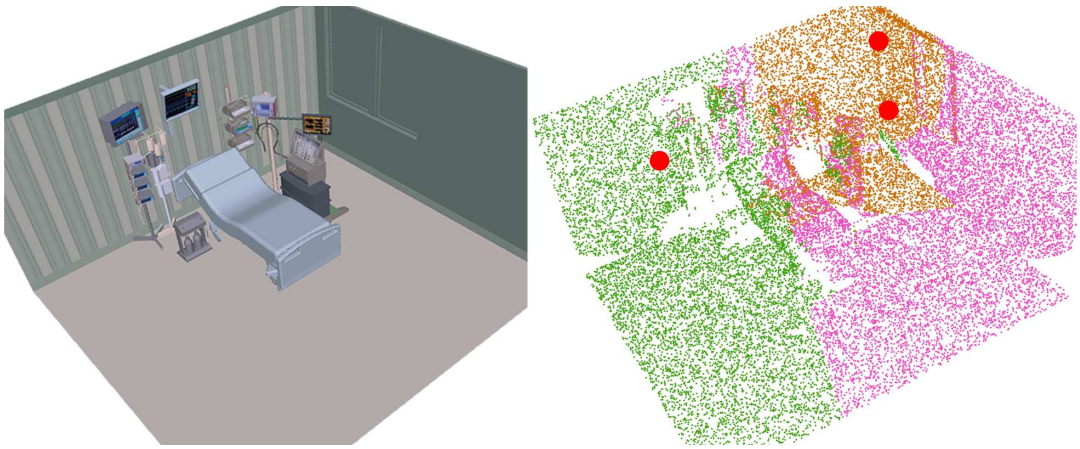


Figure 6.1: [left] The 3D model of an intensive care unit (ICU) in a hospital. [right] A near-optimal coverage of the ICU using three UVC light sources deployed on the ceiling of the room (shown as red discs) where a minimum level of exposure dose can be guaranteed. Each color (orange, green, pink) marks the covered surface locations of a given UVC source. Areas with insufficient exposure are also readily shown as scattered white regions, which can be eliminated through adding more UVC sources.

As a summary of the research and its contributions, under a general formulation, three coverage optimization problems are examined that are based on *visibility*, *best quality*, and *cumulative quality*, respectively. The visibility model only takes into account line-of-sight sensing. In the best quality model, the coverage quality of a point in the environment is determined by the closest visible sensor, i.e., the quality is determined by distance. A max-min optimization over this quantity is performed. In the cumulative quality model, the coverage of a point is the sum of coverage by all visible "sensors". For this model, the area over which a minimum quality can be guaranteed is maximized. Even in simpler 2D settings, these formulations, are known to induce significant computational challenges. They are frequently NP-hard and sometimes hard to approximate, which we briefly discuss. On the algorithmic side, we show how some of these problems can be approximately solved in polynomial time and then develop general integer programming methods, assisted with local improvements, for quickly computing high quality (i.e., $(1+\varepsilon)$-optimal) solutions. Extensive evaluations are performed over multiple realistic application scenarios, confirm-

ing the effectiveness of our algorithmic solutions.

The general formulation and specific models investigated in this work have their origins in two main lines of research: Art Gallery [12, 13, 147] and studies on mobile sensor networks, e.g., [148, 20, 34, 149, 26, 150]. Our visibility-based model has deep roots in Art Gallery problems [12, 13, 147], which commonly assume a sensor model based on line-of-sight visibility[15]; a main task is to guard every point in the interior of a bounded 2D regions (a point is guarded when it is visible to at least one of the guards). Depending on the exact formulation, guards may be placed on boundaries, corners, or the interior of the region. Not surprisingly, Art Gallery problems are typically NP-hard [16]. Other than Art Gallery, 2D coverage problems with other sensing models, e.g., disc-based, have also been considered [17, 18, 19, 20, 21, 23]. Some formulations prevent the overlapping of individual sensing ranges [17, 18] while others seek to ensure a full coverage which often require overlapping sensor coverage.

In an influential body of work [20, 34], a gradient-based iterative method was devised that drives multiple mobile sensors to a locally optimal coverage configuration, with convergence guarantees. Whereas [20, 34] assume availability of gradients *a priori*, such information can also be learned [26]. Subsequently, the method is further extended to allow the coverage of non-convex and disjoint 2D domains [35] and to work for mobile robots with heterogeneous capabilities [23]. In contrast to these iterative local interaction-based methods, this work emphasizes the direct computation of globally optimal solutions under challenging 3D settings.

Distributed sensor coverage [20, 26] builds on the study of facility location problems [27, 19] examining the selection of facility (e.g., warehouses) locations that minimize the cost of delivery of goods to spatially distributed customers. These are known (e.g., in operations research and computer science) as clustering problems [28], with many variations depending on the cost structure. Our investigation benefits from the vast literature on clustering and related problems, e.g., [29, 30, 31, 32, 33]. Clustering problems are in turn

related to packing [18], tiling [17], and Art Gallery problems [12, 13].

This work is a continuation of our systematic effort [76, 97, 65] at tackling sensor coverage problems. Our earlier studies focus on 1D/2D sensor models covering 1D/2D domains, which are significantly less complicated than the 3D settings examined in the current study.

The rest of the section is organized as follows. In Section subsection 6.1.2, we provide an umbrella problem statement and detail the three coverage models. Computational complexity of the formulations is briefly discussed. In Section subsection 6.1.5, we describe effective algorithmic solutions for solving these sensor coverage problems. Extensive evaluation results are provided in Section subsection 6.1.9 containing multiple realistic application settings. We conclude the work in Section subsection 6.1.10.

6.1.2 Preliminaries

6.1.3 Sensor Placement for Optimal Coverage: Formulations

Let $\mathcal{E} \subset \mathbb{R}^3$ be a bounded three-dimensional workspace that is path-connected, e.g., a hilly terrain or an intensive care unit (ICU) in a hospital. We consider the problem of deploying k "mobile sensors", $c_1, \ldots, c_k$, to guard a *critical subset* S embedded in the surface of $\mathcal{E}$, i.e., $S \subset \partial\mathcal{E}$ where ∂ is the boundary operator. For example, if $\mathcal{E}$ is a hospital ICU, S may be a part of its interior surface. The sensors are to be deployed to achieve a globally optimal coverage of S satisfying some prescribed objective, to be made more precise as the problems are further grounded for specific sensor models. We denote this broad class of problems as *Sensor Placement for Optimal Coverage* (`SPOC`).

The terms *mobile sensor* and *coverage* are used in a broad sense. Beyond traditional sensors that only collect information, we are interested in mobile robots with means to effect the environment as well. For example, a mobile robot may be equipped with a disinfecting light source (e.g., UVC) for eradicating harmful microbes (e.g., viruses and bacteria). Nevertheless, such settings can be nicely captured under a general sensor coverage

formulation.

In this study, we explore two common types of coverage models: *visibility*-based and *quality*-based. In a *visibility*-based sensor coverage model, as the term suggests, a point $p \in S$ is considered covered by a sensor c if p is visible from c. When there is more than one sensor, a point p is covered if it is visible from any sensor $c_i \in \{c_1, \ldots, c_k\}$. In a *quality*-based model, the coverage quality of a point $p \in S$ is captured by some function that potentially depends on the sensors, the point p, and its neighborhood in S. For example, one type of quality measurement can be based on the inverse of the distance between a point p and its closest sensor c.

Formally, we capture the different sensor models under a unified function $f(c_1, \ldots, c_k, p, \mathcal{E})$ whose co-domain is non-negative reals, i.e., $f : \mathbb{R}^{3k+3} \times \mathcal{E} \to \mathbb{R}_+ \cup \{0\}$. In what follows, $\mathcal{E}$ is omitted but understood to be part of the input to f. In this chapter, the following instantiations are considered:

- Let $vis(p, c) = 1$ if the point p is visible to the sensor c. Otherwise, $vis(p, c) = 0$. For example, an omni-directional visibility model would have $vis(p, c) = 1$ if the open line segment between p and c does not intersect $\mathcal{E}$. In a model based purely on **visibility**,

$$f(c_1, \ldots, c_k, p) := \max_{1 \le i \le k} vis(p, c_i). \tag{6.1}$$

- Let the *coverage quality* of a point p by a sensor c_i be represented as a function $\phi(p, c_i) \in \mathbb{R}_+ \cup \{0\}$. In a **quality maximization** sensor model, the coverage quality for a point p is determined by a single best sensor:

$$f(c_1, \ldots, c_k, p) := \max_{1 \le i \le k} \phi(p, c_i) vis(p, c_i). \tag{6.2}$$

- In a **cumulative quality** sensor model, the overall quality of coverage at a point p is the sum of the effects of all visible sensors:

$$f(c_1, \ldots, c_k, p) := \sum_{1 \le i \le k} \phi(p, c_i) vis(p, c_i). \tag{6.3}$$

All above models have direct and practical applications. A visibility model is applicable to the deployment of a network of 360° cameras for monitoring. Letting $\phi(p, c_i) = \|pc_i\|^{-1}$

($\|pc_i\|$ denotes the distance between p and c_i), the quality maximization model becomes the k-center problem [27] if we optimize $\min_p f(c_1, \ldots, c_k, p)$, a broadly applicable problem. For the cumulative quality model, when the "sensors" are UVC lights, one may ask the question of how to optimally place these lights to ensure the highest percentage of S can be exposed to sufficient UVC light for eradicating SARS-CoV-2 and other microbes. Here, a cumulative quality model clearly makes sense.

To fully ground the discussion that follows, we formulate three concrete optimization problems, one for each of the above-mentioned sensor models.

Problem 9 (Visibility Maximization)**.** *Given $S \subset \partial\mathcal{E}$ with $\mathcal{E} \subset \mathbb{R}^3$, $c_i \in \mathbb{R}^3$, $1 \leq i \leq k$, and $f(c_1, \ldots, c_k, p)$ from* (Equation 6.1)*, determine a placement of $c_1, \ldots, c_k$ that maximizes the support of f, i.e., $supp(f) = \{p \in S \mid \max_{1 \leq i \leq k} vis(p, c_i) > 0\}$.*

Problem 10 (Quality Maximization)**.** *Given $S \subset \partial\mathcal{E}$ with $\mathcal{E} \subset \mathbb{R}^3$, $c_i \in \mathbb{R}^3$, $1 \leq i \leq k$, and $f(c_1, \ldots, c_k, p)$ from* (Equation 6.2) *with $\phi(p, c_i) = \|pc_i\|^{-1}$, determine a placement of $c_1, \ldots, c_k$ that maximizes the minimum coverage quality, i.e., $\min_{p \in S} \max_{1 \leq i \leq k} \frac{vis(p,c_i)}{\|pc_i\|}$.*

Problem 11 (Cumulative Quality)**.** *Given $S \subset \partial\mathcal{E}$ with $\mathcal{E} \subset \mathbb{R}^3$, $c_i \in \mathbb{R}^3$, $1 \leq i \leq k$, and $f(c_1, \ldots, c_k, p)$ from* (Equation 6.3) *with $\phi(p, c_i) = |lVertpc_i\|^{-2} \langle \hat{n}_p, \hat{n}_{pc_i} \rangle$ where $\hat{n}_p$ is the unit normal of S at p and $\hat{n}_{pc_i}$ is the unit vector in the direction from p to c_i, determine a placement of $c_1, \ldots, c_k$ that maximizes the coverage area where f is above a given threshold $\Phi > 0$, $supp(f - \Phi) = \{p \in S \mid \sum_{1 \leq i \leq k} \frac{vis(p,c_i) \langle \hat{n}_p, \hat{n}_{pc_i} \rangle}{\|pc_i| rVert^2} > \Phi\}$.*

An implicit assumption for Problem 10 to be meaningful is that an arbitrary $p \in S$ is visible to the closest sensor, which limits the choice of S. Problem 10 is a suitable model for, e.g., surveillance applications that cannot tolerate any blind spots. A generalization with less limitation can require a certain percentage of S, e.g., 80%, to have optimized coverage. Problem 11, which computes coverage quality using the formula $\langle \hat{n}_p, \hat{n}_{pc_i} \rangle \|pc_i\|^{-2}$, takes after a standard

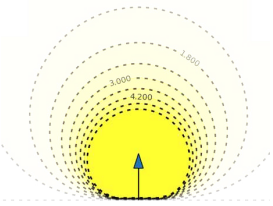


Figure 6.2: For a point with the normal shown as the arrow, light sources on the same dotted curve provide same level of exposure.

light exposure model that depends on the inverse of the squared distance and incoming light angle with respect to a local surface region (see Figure 6.2).

A 2D illustration of the three models is given in Figure 6.3.

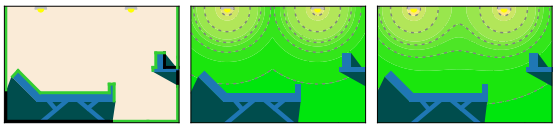


Figure 6.3: Illustration of the effects of two sensors (or light sources) under three different models. Only a 2D slice of a simple ICU with a bed and a counter is shown for clarity. [left] A visibility-based sensing model where the green segments show the visible surfaces. [middle] A quality maximization model where the dotted lines show the equal-quality level sets. $\phi(p, c_i) = \|pc_i\|^{-1}$ [right] A cumulative quality model with $\phi(p, c_i) = \|pc_i\|^{-2}\langle \hat{n}_p, \hat{n}_{pc_i} \rangle$. Again, the dotted lines show the additive quality. The impact of the surface normal is not displayed in the figure.

Problems 9-11 allow sensor locations to be anywhere in $\mathcal{E}$. In practice, sensor locations are often limited to a 2D surface. For example, in a museum or a bus, cameras are mounted on walls and ceilings. For drones surveying an area, there is often a preferred height to fly at. With this in mind, whereas algorithms we develop are general, the evaluation is mainly focused on practical settings where sensors locations are confined to some 2D surface.

6.1.4 Computational Complexity

As the computation of 3D visibility is a well-known hard problem [151] Problems 9-11 are all computationally intractable because they all involve, as part of the solution, computation of 3D visibility sets. The involvement of multiple robots/sensors introduces additional sources of computational complexity, which we briefly discuss.

Problem 9 may be viewed as an Art Gallery [12] problem in 3D. The basic 2D Art Gallery problem, which asks the question that how many guards with omnidirectional visibility are needed to ensure that every point in a simply-connected (2D) polygon is visible to at least one guard, is shown to be NP-hard [16]. Problem 9 is then also NP-hard through

reducing the 2D Art Gallery problem to a 3D one by creating a third dimension that is very "thin".

Our recent work [65] shows that a 2D version of Problem 10, called Optimal Set Guarding (OSG), is NP-hard to approximate within a factor of 1.152. Similarly, we may reduce OSG to Problem 10 by adding a thin third dimension. Therefore, through this route, we know that optimal solutions to Problem 10 is hard to approximate within a factor of 1.152, even when the surface S is a simple polygon.

From an instance of Problem 10, reduced from an instance of OSG, we can add further 3D structures to obtain an instance of Problem 11 such that cumulative effect from multiple sensors are limited. That is, when sensors are forced to have no interactions, a version of Problem 11 that is similar to Problem 10 is obtained, which is again NP-hard.

We summarize the discussion in Theorem Theorem 35. Full proofs of these complexity results, which are too lengthy to be included here and are not as essential in comparison to the problem formulations and the algorithmic results, will be detailed in an extended version of this work.

Theorem 35. *Problems 9-11 are NP-hard.*

6.1.5 Fast Computation of High-Quality Solutions

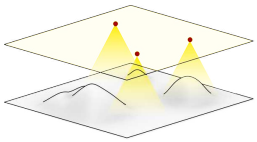
In this section, we first describe a polynomial time approximation algorithm for a restricted version of Problem 10. Then, we describe a general integer linear programming framework for solving Problems 9-11, and local improvement techniques for enhancing solution quality.

6.1.6 Polynomial Time Approximation Algorithm

In their general forms, Problems 9-11 require the computation of 3D visibility, a hard task on its own. Due to this reason, a polynomial time algorithm with guaranteed good approximation ratio for these problems appear difficult to come by. It is an interesting

question to ask whether some form of approximation scheme can be derived. Here, we show that for Problem 10, if one relax the visibility requirement, i.e. letting $vis(\cdot, \cdot) \equiv 1$, then a polynomial time $(2 + \varepsilon)$-approximation algorithm can be obtained.

Taking Problem 10, we examine a setup assuming that each point $p \in S$ has good visibility, i.e., p is always visible to the nearest sensor. Such scenarios happen when the 3D domain does not have large curvatures that would easily block sensors' view, e.g., covering the earth with GPS satellites or using drones to survey a vineyard. To drive a specific approximation bound, we further assume that sensors have spherical range sensing and are in a plane of some fixed height h from the ground, which may be relaxed. Denote this surface as H_C. The figure on the right provides an illustration of the target environment setting.



The main idea is to first obtain a dense sample of S and then adapt 2-approximation algorithms for the corresponding 2D setting, which requires some non-trivial reasoning. Two well-known approximation algorithms for k-center like problem in 2D are based on *farthest point clustering* [31] and *dominating set* [30, 99]. Both of these approaches work for our purpose; we show how to work with the former.

Let a uniformly sampled set of point of S be $S_N = \{o_1, \ldots, o_N\}$. We apply farthest point clustering [31] on S_N as follows. As the name suggest, it picks farthest sensor locations until the number of sensors are exhausted. In the original approach, the points to be clustered are also sensor locations, which is not true here. Instead, we perform clustering in the set S_N and project the selected samples to H_C gradually. The relatively straightforward process is given in Algorithm algorithm 13 (($d(\cdot, \cdot)$ denotes the distance between the inputs, one or both of which may be sets).

To prove the claimed $(2 + \varepsilon)$-approximation bound, denote the optimal sensor location set and the sensor location set derived by Algorithm algorithm 13 as $\mathcal{C}_{OPT}$ and $\mathcal{C}$, respec-

Algorithm 13: Farthest Point Clustering

Input : S_N=$\{o_1, \ldots, o_N\}$: N sampled points on the surface $S \subset \partial\mathcal{E}$; k: number of sensors;
H_C: a plane with a fixed height

Output : $\mathcal{C}$: sensor location set

```
C ← {o_1's vertical projection onto H_C}
for i ← 1 to k do
    for o ∈ S_N do
        compute the distance d(o, C) between o and C
    end
    o ← point v ∈ S_N with the largest d(v, C)
    C ← C ∪ {o's projection onto H_C}
end
return C
```

tively. Since these are centers of spherical sensing ranges, we call them center set for short. Denote the minimum coverage radius in the spherical sensing model as r_{OPT}. Let h be the minimum distance between surface S and sensor space H_C, i.e. $h := d(S, H_C)$. r_{OPT} and $r_{\mathcal{C}}$ are defined as follows:

$$r_{OPT} := \max_{o \in S_N} d(o, \mathcal{C}_{OPT}) \tag{6.4}$$

$$r_{\mathcal{C}} := \max_{o \in S_N} d(o, \mathcal{C}) \tag{6.5}$$

Proposition 36. *The center set obtained by Algorithm algorithm 13 achieves coverage radius of at most* $\sqrt{4r_{OPT}^2 - 3h^2}$.

Proof. Denote the center set generated at the ith round as $\mathcal{C}_i$, and r_i as the cluster radius $r_i := \max_{o_\tau} \min_{c_j \in \mathcal{C}_i} d(o_\tau, c_j)$. It is straightforward to observe that $r_k \leq r_{k-1} \leq \cdots \leq r_1$. Consider the relationship between the optimal center set $\mathcal{C}_{OPT}$ and the center set obtained by Algorithm algorithm 13, we have the following 2 cases.

Case 1: For each sphere $\mathcal{B}_c$ centered at a point $c \in \mathcal{C}_{OPT}$ with radius of r_{OPT}, the projection of $\mathcal{B}_c \bigcap S$ onto the sensor space H_C contains exactly one point of $\mathcal{C}_k$.

In this case, let v be an arbitrary point in S. Let c_α be the nearest center to v in $\mathcal{C}_{OPT}$

and c_β be the point in $\mathcal{C}_k$ whose projection on S is inside $\mathcal{B}_{c_\alpha}$. Therefore, we have:

$$d(v, c_\beta) \leq \sqrt{4r_{OPT}^2 - 3h^2} \tag{6.6}$$

Case 2: There exists a sphere $\mathcal{B}_c$ centered at a point $c \in \mathcal{C}_{OPT}$ with radius of r_{OPT}, the projection of $\mathcal{B}_c \bigcap S$ onto the sensor space H_C contains at least 2 points of $\mathcal{C}_k$. In this case, denote the two centers by c_i and c_j $(i < j)$, and their projections on S are in the same sphere $\mathcal{B}_c$. As c_j is added after c_i, then,

$$\begin{aligned} r_{\mathcal{C}} = r_k \leq r_j &\leq d(c_i, c_j\text{'s projection on } S) \\ &\leq \sqrt{4r_{OPT}^2 - 3h^2} \end{aligned} \tag{6.7}$$

Summarizing the two cases proves Proposition Theorem 36. □

6.1.7 Integer Programming-Based Algorithmic Framework

With Problems 9-11 being computationally intractable, a natural algorithmic alternative is mathematical programming. In [65], an integer linear programming (ILP) model was shown to be effective for a 2D setting. For our 3D problems, visibility constraints must be effectively handled. We pre-compute pairwise visibility at a given sample granularity. The information is then fed to an ILP model. As the discretization granularity gets smaller, we can then realize *globally optimal* $(1 \pm \varepsilon)$-approximations (depending whether it is a maximization or a minimization).

As a first step to building the ILP model, visibility information must be computed. We work with two discretizations, the surface S to be covered and the space where the sensors may be deployed (as discussed in Section subsection 6.1.2, this is a 3D space though in practice it is frequently a 2D subset). For each pair of samples, we use a collision checker [152] to determine whether the line segments between the two samples intersects $\mathcal{E}$. During the process, we also compute for each sample $p \in S$ its normal $\hat{n}_p$.

With the visibility pre-computation performed, we are ready to construct the fully ILP models. For all three problems, recall that we have $S_N = \{o_1, \ldots, o_N\}$ for discretizing the surface S through grid-based sampling. We use boolean variable y_i to indicate whether

sample o_i is covered. Candidate sensor locations are also discretized to obtain a sample set $\{c_1, \ldots, c_M\}$, from which k locations would be selected with z_i indicating whether c_i is selected. The ILP model for the Problem 9 is

$$y_i \leq \sum_{j \ s.t. \ vis(o_i,c_j)=1} z_j \quad \text{for each } o_i \tag{6.8}$$

$$\sum_j z_j \leq k \tag{6.9}$$

$$\max \ \ y_1 + \cdots + y_N \tag{6.10}$$

The cumulative quality case (Problem 11) is similar. Denoting the sensing quality between sensor location c_j and surface point p as $\phi(p, c_j) = vis(p, c_j) \cdot (\hat{n}_p, \hat{n}_{pc_j})/||pc_j||^2$, the ILP model may be constructed as

$$y_i \cdot \Phi \leq \sum_j \phi(o_i, c_j) \cdot z_j \quad \text{for each } o_i \tag{6.11}$$

$$\sum_j z_j \leq k \tag{6.12}$$

$$\max \ \ y_1 + \cdots + y_N \tag{6.13}$$

For quality maximization (Problem 10), the objective is to maximize the minimum distance of a sampled point on the surface to its nearest sensor location. For a required coverage ratio ρ and radius r, we can verify whether it is possible to put k sensors and cover $N\rho$ discretized points by checking the feasibility of the following model:

$$y_i \leq \sum_{j \ s.t. \ ||c_j - o_i|| \leq r} z_j \quad \text{for each } o_i \tag{6.14}$$

$$\sum_j z_j \leq k \tag{6.15}$$

$$N\rho \leq \sum_i y_i \tag{6.16}$$

A subsequent binary search can be applied to find the smallest feasible r.

6.1.8 Local Enhancement of Coverage Quality

Whereas the ILP models for Problems 9-11 support arbitrary precision, given that these problems are all computationally intractable, it can be expected that a pure ILP-based solution will only be scalable up to a certain point before an exorbitant amount computation time is needed. Inspired by the iterative update approach form [20], we propose a two-phase optimization pipeline of using ILP (or the approximation algorithm for Problem 10) as the first phase with a good level of *global* optimality guarantee and follow that with *local* improvements that can be quickly computed to enhance the initial solution. We note that, as the local improvement is enhancing a solution with a level of global optimality guarantee, the enhancement is also global in effect. For example, in Problem 10, if we start with a 2-approximation solution and obtain an initial coverage quality r and subsequent local improvement reduce that to $0.75r$, then the final solution is a globally 1.5-optimal solution.

We develop two such routines. The first is generally applicable and straightforward to implement: as the discretization level increases, we move the set of initial sensor locations (computed by Algorithm algorithm 13 or ILP) locally, one at a time. More formally, given an initial solution $\mathcal{C} = \{c_1, \ldots, c_k\}$, denote $S_j \subset S$ as the region covered (possibly partially when working with Problem 11) by the sensor deployed at c_j. For each S_j, we try improving the quality of the solution by finding a better location for c_j to cover S_j at a finer resolution. Subsequently, S_j can be updated based on the new c_j. The process may be repeated until convergence.

The second local improvement routine is via solving a "1-cener" like problem and is applicable to Problem 10 and Problem 11. Due to limited space, we omit the lengthy algorithmic details and give a high-level description. For Problem 10, a sensor located at c_j is "responsible" for visible points of S that falls within a ball $B(c_j, r)$. Our improvement routine examines $S \cap B(c_j, r)$ and attempts to compute a new ball with a smaller radius that covers all of $S \cap B(c_j, r)$. The routine uses the ideas from Welzl's algorithm for computing minimum enclosing discs [95, 96] and take time that is expected linear with respect to

the number of samples that falls within $B(c_j, r)$, which is fairly fast. This method can be readily extended to Problem 11 where the spheres become "distorted" (Figure 6.2).

6.1.9 Experimental Evaluation

For each of Problems 9-11, extensive experimental evaluations were carried out to evaluate our proposed algorithmic solutions. Here, we present representative evaluation demonstrating the effectiveness of our methods, with a focus on three realistic settings (the ICU model from Figure 6.1, bus and subway car models shown in Figure 6.4). For all environments, the surface S is selected to be all visible surfaces not facing downward. Due to limited space, result on the (2+ε)-approximation algorithm (Algorithm algorithm 13) is omitted (as shown in [65], such methods are fast but are quite sub-optimal). The experiments were carried out on a median-end quad-core Intel i7 processor with 16GiB RAM. Algorithms were implemented in C++. Gurobi [3] was used as the Integer Programming solver. Source code is available at https://github.com/rutgers-arc-lab/3d_coverage.

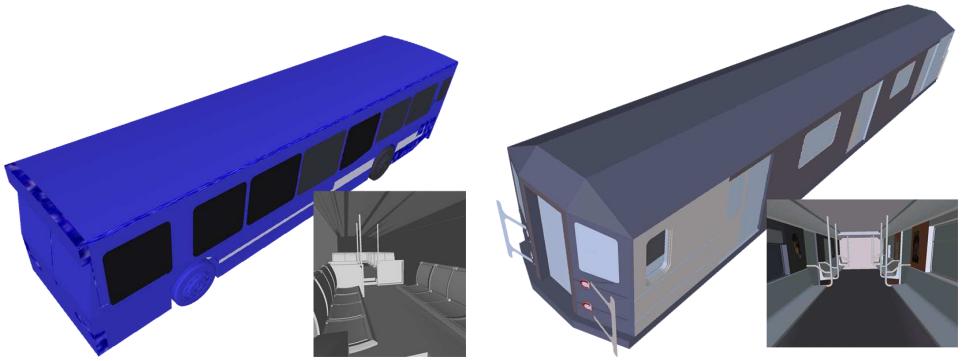


Figure 6.4: Realistic 3D environments used in our evaluation in addition to the ICU model from Figure 6.1. [left] A 40-foot large bus model and its interior. [right] A subway car and its interior.

Results on Problem 9, using ILP, is given in Figure 6.5. For each model, 600 candidate sensor locations and $20,000$ coverage surface points are sampled using grids. As expected, the surface coverage ratios increase as the number of sensors increase, approaching full coverage. We note that certain surface region is not visible, e.g., ground underneath seats in subway cars, leading to plateaus below 100% coverage. The computation time is very

reasonable for offline computations. The spikes in the middle of the computation time plot correspond to hard cases when the visibility coverage is about to plateau. We also observe that subway $<$ ICU $<$ bus in terms of computation time, which may be explained by the interior complexity of these environments. This aspect is different across the three problems.

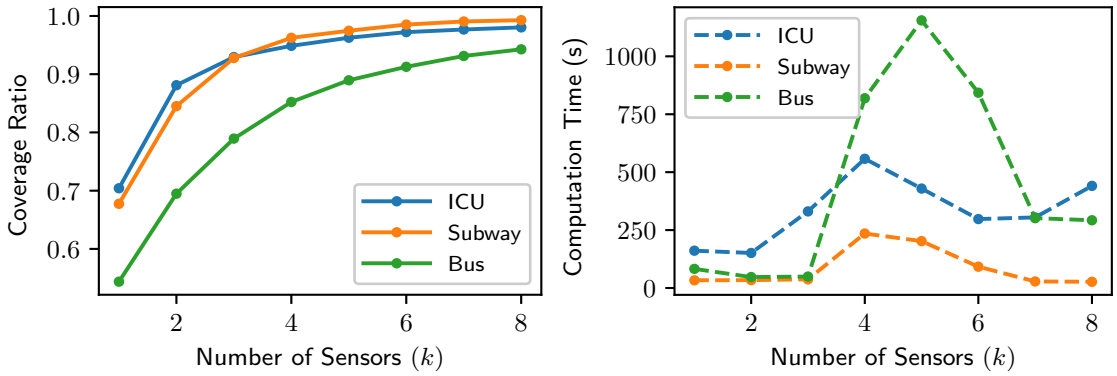
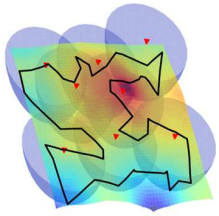


Figure 6.5: Coverage quality and computation time for Problem 9 for the three environments as the number of sensors change.

In evaluating Problem 10, we first examine a case where mobile sensors (e.g., camera drones) are deployed to cover a synthetic terrain with relatively small curvature, i.e., $vis(\cdot,\cdot) \equiv 1$. An illustration of the setting is given in the figure on the right, where the color indicates the height of the terrain. The sensors (8 red triangles in the figure) are placed at a fixed height above the terrain and must guard the region enclosed in the black curve. Spherical range sensing is assumed. For the setup (600 sensor locations and $20,000$ surface points), computation time and solution quality as the number of sensors changes are listed in the table. Computation time decreases as the number of sensors increases, indicating the problem is harder when sensors are too few to provide a good coverage. It also shows that the ILP running time does not depend positively on sensor quantity. The quality (smaller is better) increase becomes minimal as sensor quantity reaches 10.



#Sensors	2	4	6	8	10	12	14	16
Time (s)	42.9	25.4	18.5	12.7	13.1	11.3	7.64	6.70
Radius	5.10	3.31	2.76	2.43	2.27	2.16	2.07	1.99

In a second evaluation of Problem 10, visibility is considered with the optimization coverage ratio set to 80%. That is, at least 80% of the maximum visible target surface (for a given k) will be guaranteed the achieved coverage quality. The result, summarized in Figure 6.6, again demonstrates a negative correlation between the computational time and the number of sensors. Here, however, the computational time is $20+$ times more than when having full visibility.

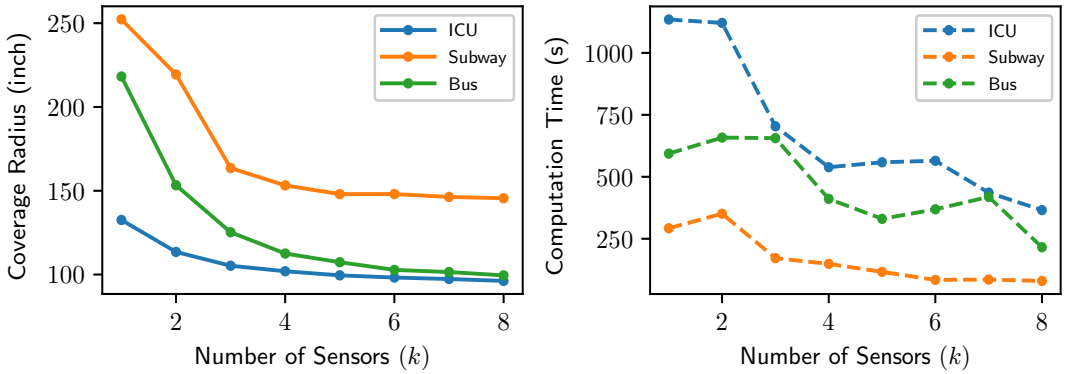


Figure 6.6: Coverage quality (lower is better) and computation time for Problem 10 for the three environments, as sensors increase.

Our last benchmark on Problem 10 tests the effectiveness of the local improvement following the resolution of a coarsely generated ILP, at 60 candidate sensor locations and 1000 surface samples (Figure 6.7). The right figure shows much faster computation time. The left figure shows that the faster method does a decent job for the bus environment (other environments have similar outcomes). The result suggests which method to use would depend on whether computational time or solution optimality is more important to the task at hand. We note that the local improvement method does not help improve the ILP result at the higher resolution.

For Problem 11, computation becomes more demanding. At the specified discretization level, most ILP models did not complete the optimization process in 10 minutes. The

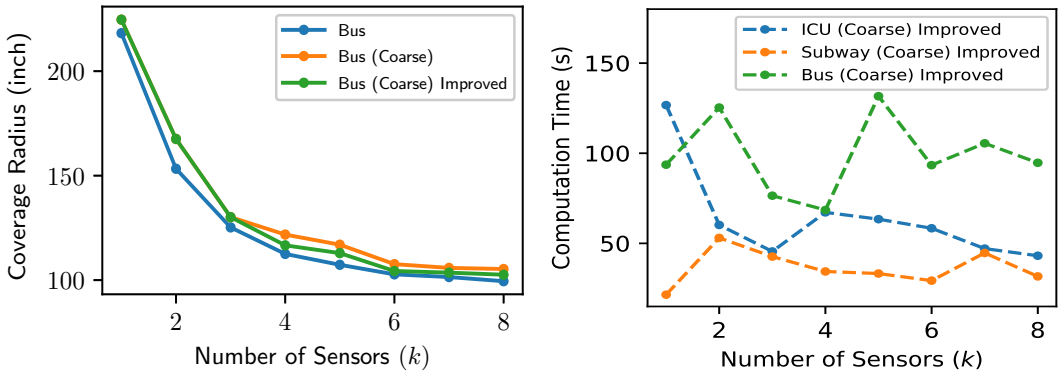


Figure 6.7: [left] Solution quality (lower is better) for the bus environment. The first curve (Bus) is the same as that from Figure 6.6. [right] Computation time using coarse ILP + local improvement.

intermediate quality result is given in Figure 6.8, on the left (the same threshold, selected to make the computation challenging, is used for all three environments), where the lines corresponds to the coverage ratio returned by the ILP model and the attached vertical bars show the reported optimality gap. The crosses show the updated ratio after local improvement is carried out (the triangles will be explained shortly). The subway data was shifted to the left to improve readability. For the bus, we see that the local improvement does help improve solution optimality, suggesting it is the most difficult problem. For the other two, it appears that the solution by the ILP model is already quite optimal, but the ILP solver still needs time to close the gap from the above. The subway case has worse coverage by the same number of sensors because it is larger. If we run a coarse ILP model (60 sensor candidates, 1000 surface sample) for one minute and then do local improvement, we get coverage ratios shown as the triangles in Figure 6.8, left. Figure 6.8, right shows the total computation time used. We observe that except for the challenging bus model, the faster method achieves essentially identical optimality as running ILP at higher resolutions. Subway costs most time here because it is the largest.

Lastly, we provide some additional visualization to help further demonstrate the structure of the problems. Figure 6.9 shows that Problems 10 and 11 induce different optimal distribution of sensors. Generally, Problems 10 tends to cause the sensors to be evenly

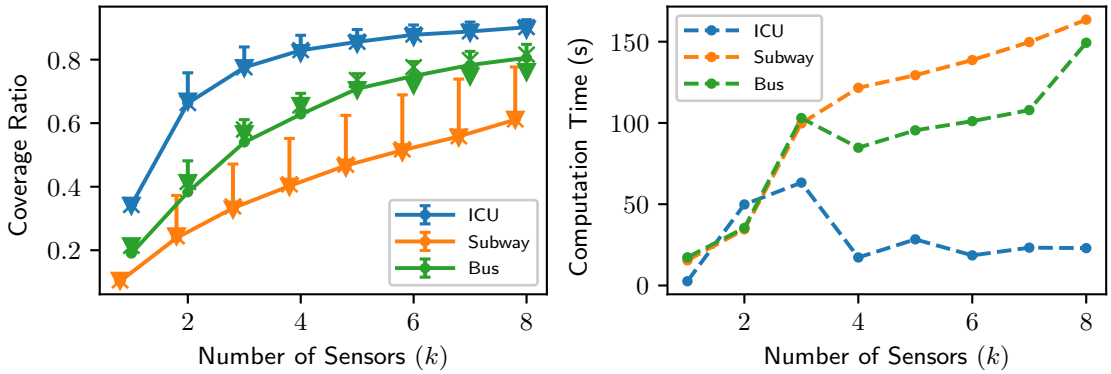


Figure 6.8: [left] Coverage ratio (lines) for Problem 11 returned by multiple methods. [right] Computation time used by running a coarse ILP plus local improvements.

spaced out. On the other hand, Problem 11 tends to balance between spacing out sensors and provide good cumulative coverage, which may require sensors to aggregate, which can be observed in Figure 6.10.

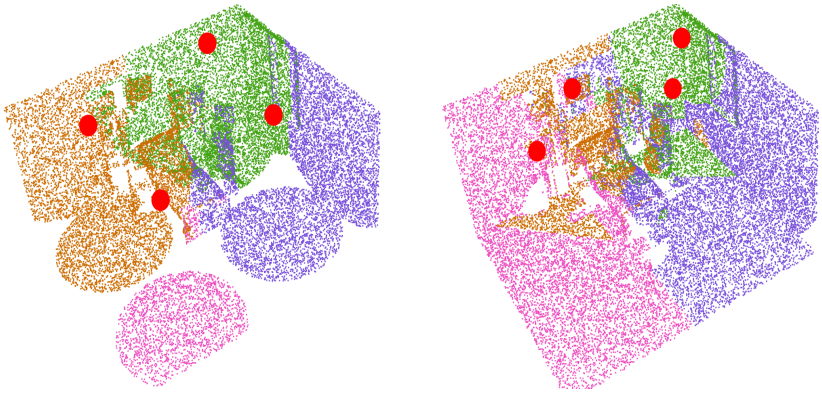


Figure 6.9: 4 sensor ICU result for Problems 10 (left) and 11(right).

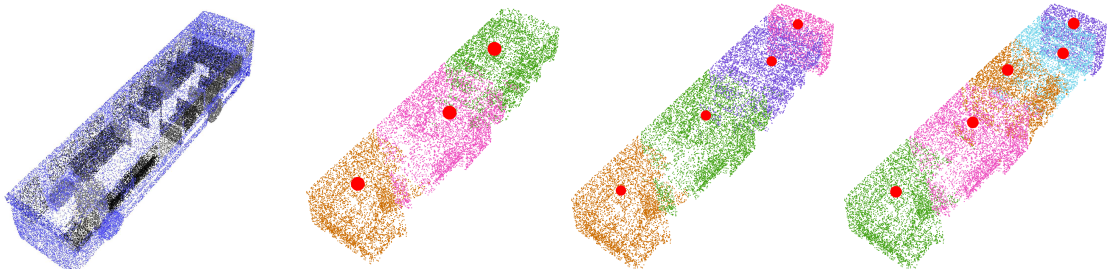


Figure 6.10: The coverage of bus (see through model on the left) using 3 to 5 sensors under Problem 11. Aggregation of sensors at the front of the bus, which is structurally more complex, can be observed.

6.1.10 Conclusion

We have formulated a general Sensor Placement for Optimal Coverage (SPOC) problem with three concrete instantiations, each with distinct practical applications. We provide near-optimal methods for solving these challenging optimization problems and demonstrated their effectiveness with extensive evaluations. We are currently exploring real-world applications of our methods.

CHAPTER 7
CONCLUSION & FUTURE WORK

In this thesis, we studied the optimal multi-sensor placement problems with different sensing models: distance-based line sensor, line-of-sight line sensor, range sensor, mobile line sensor, and mobile point sensing robot. Some of the problems are solvable in low polynomial time with traditional methods, while most of them are NP-hard. Our way to deal with these intractable problems includes methods like dynamic programming, approximation algorithms, local search-type method, integer programming and their combinations.

In the future, the work in this thesis can be extended in multiple directions.

First, the thesis focus on sensing range as some basic shapes like lines or circles, and they are considered individually. A heterogeneous combination of sensors like line sensors used together with range sensors can be very interesting to explore. Another possibility is to model the uncertainty of sensors like sensor blackout, which is considered in works like [153]. All of these sensing models are used to simplify the sophisticatedness of real-world sensing models of laser beams, radars, lidars and so on. So, any model that can drag the problem closer to reality should be valuable to explore.

Second, more realistic constraints related to mobile robots can be added to the problem of sweep line coverage or boundary defense: e.g., for the mobile sensing robot, dynamic constraints like acceleration and turning radius of the robot are not considered in our work, as well as the shape of the robot. Considering these will increase the effectiveness of applying the algorithms developed on real-world mobile sensing robot applications. Also, for a mobile sensing robot, other constraints or objectives can be modeled such as energy consumption (where acceleration and deceleration matter), and blockage of view from each other.

REFERENCES

[1] S. Thrun, W. Burgard, and D. Fox, "Probabilistic robotics," in. MIT Press, 2002, ch. 6.3.

[2] V. V. Vazirani, *Approximation algorithms*. Springer, vol. 1.

[3] G. Optimization, "Gurobi optimizer 9.0," *Gurobi: http://www.gurobi.com*, 2019.

[4] I. I. Cplex, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[5] T. Achterberg, "Scip: Solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, pp. 1–41, 2009.

[6] J. Forrest and R. Lougee-Heimer, "Cbc user guide," in *Emerging theory, methods, and applications*, INFORMS, 2005, pp. 257–277.

[7] A. Makhorin, "Glpk (gnu linear programming kit)," *http://www. gnu. org/s/glpk/glpk. html*, 2008.

[8] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *2014 IEEE-RAS international conference on humanoid robots*, IEEE, 2014, pp. 279–286.

[9] T. Guo, S. D. Han, and J. Yu, "Spatial and temporal splitting heuristics for multi-robot motion planning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 8009–8015.

[10] A. Adler, O. Mickelin, R. K. Ramachandran, G. S. Sukhatme, and S. Karaman, "The role of heterogeneity in autonomous perimeter defense problems," in *Algorithmic Foundations of Robotics XV: Proceedings of the Fifteenth Workshop on the Algorithmic Foundations of Robotics*, Springer, 2022, pp. 115–131.

[11] D. Shishika and V. Kumar, "A review of multi agent perimeter defense games," in *International Conference on Decision and Game Theory for Security*, Springer, 2020, pp. 472–485.

[12] J. O'rourke *et al.*, *Art gallery theorems and algorithms*. Oxford University Press Oxford, 1987, vol. 57.

[13] T. C. Shermer, "Recent results in art galleries (geometry)," *Proceedings of the IEEE*, vol. 80, no. 9, pp. 1384–1399, 1992.

[14] V. Klee, "Is every polygonal region illuminable from some point?" *Amer. Math. Monthly*, vol. 76, p. 180, 1969.

[15] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[16] D. Lee and A. Lin, "Computational complexity of art gallery problems," *IEEE Transactions on Information Theory*, vol. 32, no. 2, pp. 276–282, 1986.

[17] A. Thue, *Über die dichteste Zusammenstellung von kongruenten Kreisen in einer Ebene.* Christiania : Dybwad in Komm., 1910.

[18] T. C. Hales, "A proof of the kepler conjecture," *Annals of Mathematics*, vol. 162, no. 3, pp. 1065–1185, 2005.

[19] Z. Drezner, *Facility Location: A Survey of Applications and Methods*. Springer Verlag, 1995.

[20] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[21] M. Pavone, A. Arsie, E. Frazzoli, and F. Bullo, "Equitable partitioning policies for robotic networks," in *2009 IEEE International Conference on Robotics and Automation*, IEEE, 2009, pp. 2356–2361.

[22] S. Martinez, "Distributed interpolation schemes for field estimation by mobile sensor networks," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 491–500, 2010.

[23] A. Pierson, L. C. Figueiredo, L. C. Pimenta, and M. Schwager, "Adapting to sensing and actuation variations in multi-robot coverage," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 337–354, 2017.

[24] R. B. Kershner, "On paving the plane," *The American Mathematical Monthly*, vol. 75, no. 8, pp. 839–844, 1968.

[25] C. D. Toth, J. O'Rourke, and J. E. Goodman, *Handbook of discrete and computational geometry*. CRC press, 2017.

[26] M. Schwager, D. Rus, and J.-J. Slotine, "Decentralized, adaptive coverage control for networked robots," *The International Journal of Robotics Research*, vol. 28, no. 3, pp. 357–375, 2009.

[27] A. Weber, *Theory of the Location of Industries*. University of Chicago Press, 1929.

[28] S. Har-Peled, *Geometric Approximation Algorithms*, 173. American Mathematical Soc., 2011.

[29] T. Feder and D. Greene, "Optimal algorithms for approximate clustering," in *Symposium on Theory of Computing*, ACM, 1988, pp. 434–444.

[30] D. S. Hochbaum and D. B. Shmoys, "A best possible heuristic for the k-center problem," *Mathematics of Operations Research*, vol. 10, no. 2, pp. 180–184, 1985.

[31] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical computer science*, vol. 38, pp. 293–306, 1985.

[32] M. S. Daskin, "A new approach to solving the vertex p-center problem to optimality: Algorithm and computational results," *Communications of the Operations Research Society of Japan*, vol. 45, no. 9, pp. 428–436, 2000.

[33] M. I. Shamos and D. Hoey, "Closest-point problems," in *16th Annual Symposium on Foundations of Computer Science (FOCS 1975)*, IEEE, 1975, pp. 151–162.

[34] S. Martınez, J. Cortés, and F. Bullo, "Motion coordination with distributed information," *IEEE Control Systems Magazine*, vol. 27, no. 4, pp. 75–88, 2007.

[35] M. Schwager, B. J. Julian, and D. Rus, "Optimal coverage for multiple hovering robots with downward facing cameras," in *2009 IEEE international conference on robotics and automation*, IEEE, 2009, pp. 3515–3522.

[36] T. Arai, E. Pagello, L. E. Parker, *et al.*, "Advances in multi-robot systems," *IEEE Transactions on robotics and automation*, vol. 18, no. 5, pp. 655–661, 2002.

[37] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International journal of robotics research*, vol. 23, no. 9, pp. 939–954, 2004.

[38] W. Ren and R. W. Beard, *Distributed consensus in multi-vehicle cooperative control*, 2. Springer, 2008, vol. 27.

[39] F. Bullo, J. Cortés, and S. Martinez, *Distributed control of robotic networks: a mathematical approach to motion coordination algorithms*. Princeton University Press, 2009, vol. 27.

[40] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita, "Distributed memoryless point convergence algorithm for mobile robots with limited visibility," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 5, pp. 818–828, 1999.

[41] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on automatic control*, vol. 48, no. 6, pp. 988–1001, 2003.

[42] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on automatic control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[43] W. Ren and R. W. Beard, "Consensus seeking in multiagent systems under dynamically changing interaction topologies," *IEEE Transactions on automatic control*, vol. 50, no. 5, pp. 655–661, 2005.

[44] P. Cheng and V. Kumar, "An almost communication-less approach to task allocation for multiple unmanned aerial vehicles," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, 2008, pp. 1384–1389.

[45] M. Mesbahi and M. Egerstedt, "Graph theoretic methods in multiagent networks," in *Graph Theoretic Methods in Multiagent Networks*, Princeton University Press, 2010.

[46] J. Yu, S. M. LaValle, and D. Liberzon, "Rendezvous without coordinates," *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, vol. 57, no. 2, p. 421, 2012.

[47] S. L. Smith and F. Bullo, "Monotonic target assignment for robotic networks," *IEEE Transactions on Automatic Control*, vol. 54, no. 9, pp. 2042–2057, 2009.

[48] N. Ayanian and V. Kumar, "Decentralized feedback controllers for multiagent teams in environments with obstacles," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 878–887, 2010.

[49] L. Liu and D. A. Shell, "Multi-level partitioning and distribution of the assignment problem for large-scale multi-robot task allocation," *Robotics: Science and Systems VII; MIT Press: Cambridge, MA, USA*, pp. 26–33, 2011.

[50] ——, "Optimal market-based multi-robot task allocation via strategic pricing.," in *Robotics: Science and Systems*, vol. 9, 2013, pp. 33–40.

[51] M. Turpin, K. Mohta, N. Michael, and V. Kumar, "Goal assignment and trajectory planning for large teams of interchangeable robots," *Autonomous Robots*, vol. 37, no. 4, pp. 401–415, 2014.

[52] M. Turpin, N. Michael, and V. Kumar, "Capt: Concurrent assignment and planning of trajectories for multiple robots," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 98–112, 2014.

[53] J. Alonso-Mora, S. Baker, and D. Rus, "Multi-robot navigation in formation via sequential convex programming," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 4634–4641.

[54] K. Solovey, J. Yu, O. Zamir, and D. Halperin, "Motion planning for unlabeled discs with optimality guarantees," in *Robotics: Science and Systems*, 2015.

[55] A. Kolling and S. Carpin, "The graph-clear problem: Definition, theoretical properties and its connections to multirobot aided surveillance," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2007, pp. 1003–1008.

[56] A. Kolling, A. Kleiner, and S. Carpin, "Coordinated search with multiple robots arranged in line formations," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 459–473, 2017.

[57] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, "The complexity of searching a graph," *Journal of the ACM (JACM)*, vol. 35, no. 1, pp. 18–44, 1988.

[58] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani, "A visibility-based pursuit-evasion problem," *International Journal of Computational Geometry & Applications*, vol. 9, no. 04n05, pp. 471–493, 1999.

[59] I. Suzuki and M. Yamashita, "Searching for a mobile intruder in a polygonal region," *SIAM Journal on computing*, vol. 21, no. 5, pp. 863–888, 1992.

[60] S. M. LaValle, B. H. Simov, and G. Slutzki, "An algorithm for searching a polygonal region with a flashlight," in *Proceedings of the sixteenth annual symposium on Computational geometry*, 2000, pp. 260–269.

[61] N. M. Stiffler, A. Kolling, and J. M. O'Kane, "Persistent pursuit-evasion: The case of the preoccupied pursuer," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 5027–5034.

[62] D. Shishika, J. Paulos, and V. Kumar, "Cooperative team strategies for multi-player perimeter-defense games," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2738–2745, 2020.

[63] D. G. Macharet, A. K. Chen, D. Shishika, G. J. Pappas, and V. Kumar, "Adaptive partitioning for coordinated multi-agent perimeter defense," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 7971–7977.

[64] A. K. Chen, D. G. Macharet, D. Shishika, G. J. Pappas, and V. Kumar, "Optimal multi-robot perimeter defense using flow networks," in *International Symposium Distributed Autonomous Robotic Systems*, Springer, 2021, pp. 282–293.

[65] S. W. Feng and J. Yu, "Optimally guarding perimeters and regions with mobile range sensors," in *Robotics: Science and Systems*, 2020.

[66] J. M. Correia Marques, R. Ramalingam, Z. Pan, and K. Hauser, "Optimized coverage planning for uv surface disinfection," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 9731–9737.

[67] T. Oksanen and A. Visala, "Coverage path planning algorithms for agricultural field machines," *Journal of field robotics*, vol. 26, no. 8, pp. 651–668, 2009.

[68] R. N. Haksar, S. Trimpe, and M. Schwager, "Spatial scheduling of informative meetings for multi-agent persistent coverage," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3027–3034, 2020.

[69] M. Wei and V. Isler, "Coverage path planning under the energy constraint," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 368–373.

[70] D. Deng, W. Jing, Y. Fu, Z. Huang, J. Liu, and K. Shimada, "Constrained heterogeneous vehicle path planning for large-area coverage," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 4113–4120.

[71] X. Lan and M. Schwager, "Planning periodic persistent monitoring trajectories for sensing robots in gaussian random fields," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 2415–2420.

[72] C. G. Cassandras, X. Lin, and X. Ding, "An optimal control approach to the multi-agent persistent monitoring problem," *IEEE Transactions on Automatic Control*, vol. 58, no. 4, pp. 947–961, 2012.

[73] J. Yu, S. Karaman, and D. Rus, "Persistent monitoring of events with stochastic arrivals at multiple stations," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 521–535, 2015.

[74] J. M. Palacios-Gasós, Z. Talebpour, E. Montijano, C. Sagüés, and A. Martinoli, "Optimal path planning and coverage control for multi-robot persistent coverage in environments with obstacles," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 1321–1327.

[75] R. E. Burkard and E. Cela, *Linear assignment problems and extensions*. Springer, 1999.

[76] S. W. Feng, S. D. Han, K. Gao, and J. Yu, "Efficient algorithms for optimal perimeter guarding," in *Robotics: Science and Systems*, 2019.

[77] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, pp. 477–521, 1987.

[78] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE international conference on robotics and automation*, Ieee, 2008, pp. 1928–1935.

[79] D. Morgan, G. P. Subramanian, S.-J. Chung, and F. Y. Hadaegh, "Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1261–1285, 2016.

[80] D. S. Johnson, "Near-optimal bin packing algorithms," PhD thesis, Massachusetts Institute of Technology, 1973.

[81] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 1972.

[82] G. S. Lueker, *Two NP-complete problems in nonnegative integer programming*. Princeton University. Department of Electrical Engineering, 1975.

[83] G. B. Dantzig, "Discrete-variable extremum problems," *Operations research*, vol. 5, no. 2, pp. 266–288, 1957.

[84] M. R. Garey and D. S. Johnson, "Complexity results for multiprocessor scheduling under resource constraints," *SIAM journal on Computing*, vol. 4, no. 4, pp. 397–411, 1975.

[85] ——, *Computers and intractability*. W. H. Freeman, 1979, vol. 174.

[86] H. W. Lenstra Jr, "Integer programming with a fixed number of variables," *Mathematics of operations research*, vol. 8, no. 4, pp. 538–548, 1983.

[87] O. H. Ibarra and C. E. Kim, "Fast approximation algorithms for the knapsack and sum of subset problems," *Journal of the ACM (JACM)*, vol. 22, no. 4, pp. 463–468, 1975.

[88] M. A. Batalin and G. S. Sukhatme, "Spreading out: A local approach to multi-robot coverage," in *Distributed Autonomous Robotic Systems 5*, Springer, 2002, pp. 373–382.

[89] N. Correll, J. Bachrach, D. Vickery, and D. Rus, "Ad-hoc wireless network coverage with networked robots that cannot localize," in *IEEE International Conference on Robotics and Automation*, IEEE, 2009, pp. 3878–3885.

[90] S. Gil, D. Feldman, and D. Rus, "Communication coverage for independently moving robots," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 4865–4872.

[91] M. R. Garey and D. S. Johnson, "The rectilinear steiner tree problem is np-complete," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[92] B. Mohar, "Face covers and the genus problem for apex graphs," *Journal of Combinatorial Theory, Series B*, vol. 82, no. 1, pp. 102–117, 2001.

[93] J. Petersen, "Die theorie der regulären graphs," *Acta Math.*, vol. 15, pp. 193–220, 1891.

[94] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of Mathematics*, vol. 17, pp. 449–467, 1965.

[95] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in *New Results and New Trends in Computer Science*, Springer, 1991, pp. 359–370.

[96] M. Berg de, O. Cheong, M. Kreveld van, and M. Overmars, "Computational geometry: Algorithms and applications," in. Springer, 2008, ch. 4.7, ISBN: 978-3-540-77973-5.

[97] S. W. Feng and J. Yu, "Optimal perimeter guarding with heterogeneous robot teams: Complexity analysis and effective algorithms," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 430–437, 2020, Note: presented at ICRA 2020.

[98] N. Megiddo, "Linear-time algorithms for linear programming in $r\hat{3}$ and related problems," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 759–776, 1983.

[99] V. V. Vazirani, "Approximation algorithms," in. Springer, 2013, ch. 5.

[100] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[101] J. C. Culberson and R. A. Reckhow, "Covering polygons is hard," in *29th Annual Symposium on Foundations of Computer Science*, IEEE, 1988, pp. 601–611.

[102] S. Kloder and S. Hutchinson, "Barrier coverage for variable bounded-range line-of-sight guards," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, IEEE, 2007, pp. 391–396.

[103] ——, "Partial barrier coverage: Using game theory to optimize probability of undetected intrusion in polygonal environments," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, 2008, pp. 2671–2676.

[104] Y. Ho, A. Bryson, and S. Baron, "Differential games and optimal pursuit-evasion strategies," *IEEE Transactions on Automatic Control*, vol. 10, no. 4, pp. 385–389, 1965.

[105] R. Isaacs, *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*. Courier Corporation, 1999.

[106] O. Hájek, *Pursuit games: an introduction to the theory and applications of differential games of pursuit and evasion*. Courier Corporation, 2008.

[107] B. Tovar, F. Cohen, and S. M. LaValle, "Sensor beams, obstacles, and possible paths," in *Algorithmic Foundation of Robotics VIII*, Springer, 2009, pp. 317–332.

[108] B. H. Simov, G. Slutzki, and S. M. LaValle, "Pursuit-evasion using beam detection," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, IEEE, vol. 2, 2000, pp. 1657–1662.

[109] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani, "Visibility-based pursuit-evasion in a polygonal environment," in *Workshop on Algorithms and Data Structures*, Springer, 1997, pp. 17–30.

[110] T. Kameda, M. Yamashita, and I. Suzuki, "Online polygon search by a seven-state boundary 1-searcher," *IEEE Transactions on Robotics*, vol. 22, no. 3, pp. 446–460, 2006.

[111] L. M. Kirousis and C. H. Papadimitriou, "Searching and pebbling," *Theoretical Computer Science*, vol. 47, pp. 205–218, 1986.

[112] S. Sachs, S. M. LaValle, and S. Rajko, "Visibility-based pursuit-evasion in an unknown planar environment," *The International Journal of Robotics Research*, vol. 23, no. 1, pp. 3–26, 2004.

[113] H. Lau, S. Huang, and G. Dissanayake, "Optimal search for multiple targets in a built environment," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2005, pp. 3740–3745.

[114] J. Yu and S. M. LaValle, "Shadow information spaces: Combinatorial filters for tracking targets," *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 440–456, 2011.

[115] T. Olsen, N. M. Stiffler, and J. M. O'Kane, "Robust-by-design plans for multi-robot pursuit-evasion," in *IEEE International Conference on Robotics and Automation*, 2022.

[116] M. Abrahamsen, P. Giannopoulos, M. Löffler, and G. Rote, "Geometric multicut: Shortest fences for separating groups of objects in the plane," *Discrete & Computational Geometry*, vol. 64, no. 3, pp. 575–607, 2020.

[117] S. Kloder, "Barrier coverage: Deploying robot guards to prevent intrusion," PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2008, ch. 3.

[118] A. Tarski, "A decision method for elementary algebra and geometry," *Journal of Symbolic Logic*, vol. 14, no. 3, 1949.

[119] R. Freimer, J. S. Mitchell, and C. Piatko, "On the complexity of shattering using arrangements," Cornell University, Tech. Rep., 1991.

[120] S. Har-Peled and M. Jones, "On separating points by lines," *Discrete & Computational Geometry*, vol. 63, no. 3, pp. 705–730, 2020.

[121] O. Devillers, F. Hurtado, M. Mora, and C. Seara, "Separating several point sets in the plane," in *In Proc. 13th Canadian Conference on Computational Geometry*, Citeseer, 2001.

[122] B. A. Armaselu, "On the geometric separability of bichromatic point sets," PhD thesis, Computer Science Department, The University of Texas at Dallas, 2017.

[123] J.-D. Boissonnat, J. Czyzowicz, O. Devillers, and M. Yvinec, "Circular separability of polygons," *Algorithmica*, vol. 30, no. 1, pp. 67–82, 2001.

[124] E. D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, H. Meijer, M. Overmars, and S. Whitesides, "Separating point sets in polygonal environments," *International Journal of Computational Geometry & Applications*, vol. 15, no. 04, pp. 403–419, 2005.

[125] G. Lykou, D. Moustakas, and D. Gritzalis, "Defending airports from uas: A survey on cyber-attacks and counter-drone sensing technologies," *Sensors*, vol. 20, no. 12, p. 3537, 2020.

[126] E. S. Lee, D. Shishika, and V. Kumar, "Perimeter-defense game between aerial defender and ground intruder," in *2020 59th IEEE Conference on Decision and Control (CDC)*, IEEE, 2020, pp. 1530–1536.

[127] N. Agmon, S. Kraus, and G. A. Kaminka, "Multi-robot perimeter patrol in adversarial settings," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, 2008, pp. 2339–2345.

[128] R. Isaacs, *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*. Courier Corporation, 1965.

[129] D. Shishika and V. Kumar, "Local-game decomposition for multiplayer perimeter-defense problem," in *2018 IEEE conference on decision and control (CDC)*, IEEE, 2018, pp. 2093–2100.

[130] K. Margellos and J. Lygeros, "Hamilton–jacobi formulation for reach–avoid differential games," *IEEE Transactions on automatic control*, vol. 56, no. 8, pp. 1849–1861, 2011.

[131] Z. Zhou, R. Takei, H. Huang, and C. J. Tomlin, "A general, open-loop formulation for reach-avoid games," in *2012 IEEE 51st IEEE conference on decision and control (CDC)*, IEEE, 2012, pp. 6501–6506.

[132] R. Yan, Z. Shi, and Y. Zhong, "Reach-avoid games with two defenders and one attacker: An analytical approach," *IEEE transactions on cybernetics*, vol. 49, no. 3, pp. 1035–1046, 2018.

[133] M. Chen, Z. Zhou, and C. J. Tomlin, "A path defense approach to the multiplayer reach-avoid game," in *53rd IEEE conference on decision and control*, IEEE, 2014, pp. 2420–2426.

[134] ——, "Multiplayer reach-avoid games via low dimensional solutions and maximum matching," in *2014 American control conference*, IEEE, 2014, pp. 1444–1449.

[135] R. Yan, X. Duan, Z. Shi, Y. Zhong, and F. Bullo, "Matching-based capture strategies for 3d heterogeneous multiplayer reach-avoid differential games," *arXiv preprint arXiv:1909.11881*, 2019.

[136] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to algorithms," in. MIT press, 2022, ch. 14.

[137] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *16th annual symposium on foundations of computer science (sfcs 1975)*, IEEE, 1975, pp. 184–193.

[138] H. Choset, "Coverage of known spaces: The boustrophedon cellular decomposition," *Autonomous Robots*, vol. 9, no. 3, pp. 247–253, 2000.

[139] W. H. Huang, "Optimal line-sweep-based decompositions for coverage algorithms," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, IEEE, vol. 1, 2001, pp. 27–32.

[140] A. Breitenmoser, M. Schwager, J.-C. Metzger, R. Siegwart, and D. Rus, "Voronoi coverage of non-convex environments with a group of networked robots," in *2010 IEEE international conference on robotics and automation*, IEEE, 2010, pp. 4982–4989.

[141] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull, "Morse decompositions for coverage tasks," *The international journal of robotics research*, vol. 21, no. 4, pp. 331–344, 2002.

[142] S. M. LaValle, "Planning algorithms," in. Cambridge university press, 2006, ch. 6.2.

[143] J. Kleinberg and E. Tardos, "Algorithm design," in. Pearson Education, 2006, ch. 7.7.

[144] J. Cheriyan and S. Maheshwari, "Analysis of preflow push algorithms for maximum network flow," *SIAM Journal on Computing*, vol. 18, no. 6, pp. 1057–1086, 1989.

[145] Y. A. Dinitz, "An algorithm for the solution of the problem of maximal flow in a network with power estimation," in *Doklady Akademii Nauk*, Russian Academy of Sciences, vol. 194, 1970, pp. 754–757.

[146] G. Borradaile and P. Klein, "An o (n log n) algorithm for maximum st-flow in a directed planar graph," *Journal of the ACM (JACM)*, vol. 56, no. 2, pp. 1–30, 2009.

[147] J. O'Rourke, "Visibility," in *Handbook of discrete and computational geometry, 3rd Ed.* 2017, pp. 875–896.

[148] A. Howard, M. J. Matarić, and G. S. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," in *Distributed Autonomous Robotic Systems 5*, Springer, 2002, pp. 299–308.

[149] A. Krause, A. Singh, and C. Guestrin, "Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies," *Journal of Machine Learning Research*, vol. 9, no. Feb, pp. 235–284, 2008.

[150] G. A. Hollinger and G. S. Sukhatme, "Sampling-based motion planning for robotic information gathering.," in *Robotics: Science and Systems*, Citeseer, vol. 3, 2013.

[151] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *28th Annual Symposium on Foundations of Computer Science (FOCS 1987)*, IEEE, 1987, pp. 49–60.

[152] P. Alliez, S. Tayeb, and C. Wormser, "3d fast intersection and distance computation (aabb tree)," in *CGAL User and Reference Manual*, 5.1, CGAL Editorial Board, 2020.

[153] T. Olsen, N. M. Stiffler, and J. M. O'Kane, "Robust-by-design plans for multi-robot pursuit-evasion," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 10 716–10 722.