Multi-Robot Path Planning for Persistent Monitoring in Stochastic and Adversarial Environments

by

Ahmad Bilal Asghar

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

Waterloo, Ontario, Canada, 2020

Examining Committee Membership

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

External Examiner:	Dylan A. Shell Associate Professor, Dept. of Computer Science and Engineering, Texas A&M University
Supervisor:	Stephen L. Smith Associate Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member:	Daniel Miller Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member:	John W. Simpson-Porco Assistant Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal-External Member:	Jun Liu Associate Professor, Dept. of Applied Mathematics, University of Waterloo

Abstract

In this thesis, we study multi-robot path planning problems for persistent monitoring tasks. The goal of such persistent monitoring tasks is to deploy a team of cooperating mobile robots in an environment to continually observe locations of interest in the environment. Robots patrol the environment in order to detect events arriving at the locations of the environment. The events stay at those locations for a certain amount of time before leaving and can only be detected if one of the robots visits the location of an event while the event is there.

In order to detect all possible events arriving at a vertex, the maximum time spent by the robots between visits to that vertex should be less than the duration of the events arriving at that vertex. We consider the problem of finding the minimum number of robots to satisfy these revisit time constraints, also called latency constraints. The decision version of this problem is PSPACE-complete. We provide an $O(\log \rho)$ approximation algorithm for this problem where ρ is the ratio of the maximum and minimum latency constraints. We also present heuristic algorithms to solve the problem and show through simulations that a proposed orienteering-based heuristic algorithm gives better solutions than the approximation algorithm. We additionally provide an algorithm for the problem of minimizing the maximum weighted latency given a fixed number of robots.

In case the event stay durations are not fixed but are drawn from a known distribution, we consider the problem of maximizing the expected number of detected events. We motivate randomized patrolling paths for such scenarios and use Markov chains to represent those random patrolling paths. We characterize the expected number of detected events as a function of the Markov chains used for patrolling and show that the objective function is submodular for randomly arriving events. We propose an approximation algorithm for the case where the event durations for all the vertices is a constant. We also propose a centralized and an online distributed algorithm to find the random patrolling policies for the robots. We also consider the case where the events are adversarial and can choose where and when to appear in order to maximize their chances of remaining undetected.

The last problem we study in this thesis considers events triggered by a learning adversary. The adversary has a limited time to observe the patrolling policy before it decides when and where events should appear. We study the single robot version of this problem and model this problem as a multi-stage two player game. The adversary observes the patroller's actions for a finite amount of time to learn the patroller's strategy and then either chooses a location for the event to appear or reneges based on its confidence in the learned strategy. We characterize the expected payoffs for the players and propose a search algorithm to find a patrolling policy in such scenarios. We illustrate the trade off between hard to learn and hard to attack strategies through simulations.

Acknowledgements

I would like to thank my advisor Professor Stephen Smith for his constant guidance and support throughout the PhD program. I would like to express my sincere gratitude for his patience, motivation and highly informative discussions during our meetings.

I would also like to thank the readers of this thesis — Professor Dylan Shell, Professor Jun Liu, Professor Daniel Miller and Professor John Simpson-Porco — for their valuable time.

Finally, I would like to express my gratitude to my family, particularly my mother and my wife for their love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Due to the rapid development in the field of mobile robotics, teams of robots can now perform long term monitoring tasks. Examples of such tasks include infrastructure inspection [20] to detect presence of anomalies or failures; patrolling for surveillance [12, 10] to detect threats in the environment; 3D reconstruction of scenes [84, 13] in changing environments; informative path planning [21] for observing dynamic properties of an area; and forest fire monitoring [66]. The goal of the monitoring tasks is to deploy a team of cooperating mobile agents or robots in the dynamically changing environment to continually observe locations of interest in the environment. With limited amount of resources, stationary agents cannot persistently monitor all the regions of interest in a large environment, and therefore a team of mobile robots must patrol the environment to gather the information. As the environment keeps evolving in persistent monitoring scenarios, the locations in the environment need to be visited repeatedly by the team of robots. The planning problem is to find actions for the team of robots such that the information gained from the environment is maximized, or the staleness of the information is minimized. In this thesis, we focus on finding patrolling paths for a team of robots in order to solve persistent monitoring problems. Since we consider high-level path planning problems, we do not discuss the dynamics and controls of the robots, and assume that the controllers for actions such as collision avoidance are implemented on the robots, so that they can follow a path or strategy returned by the algorithms presented in this work.

We represent the environment to be monitored as a graph where the vertices of the graph represent the locations of importance in the environment and the edge weights give the travel times between the vertices. The robots travel along these edges to monitor the environment and to gather information from the locations of the environment. Different monitoring scenarios and applications result in different variations of the persistent moni-

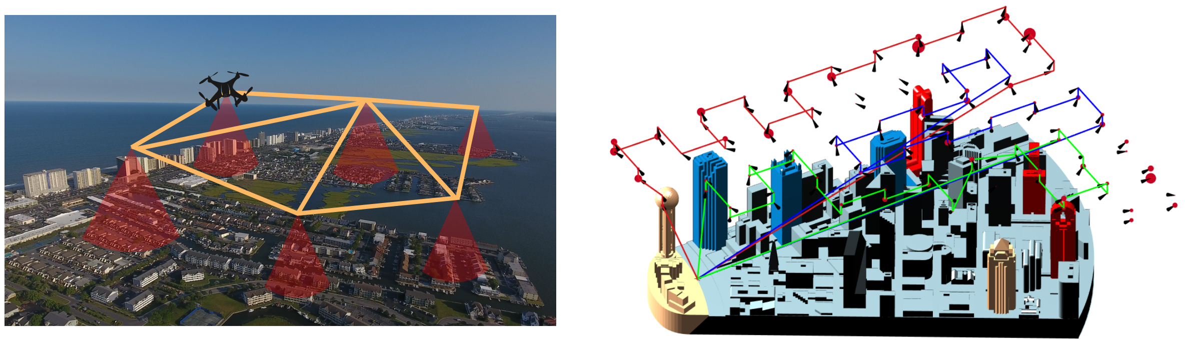


Figure 1.1: Example of persistently monitoring an urban environment. The figure on left shows an example of a UAV visiting locations to monitor regions in an urban environment. The figure on right shows the simulated patrolling paths for multiple aerial robots visiting areas in the environment to maximize the information gathered. The size of a vertex represents the value of information at that vertex.

toring problem. The scenarios and applications that result in the variations studied in this thesis are discussed below.

If the purpose is to minimize the maximum staleness of information, then the problem is to find patrolling paths for the robots to minimize the maximum time spent away from any location. A simple single robot instance of this problem is where all the vertices are equally important. The optimal strategy for this case is for the robot to follow the solution of the Travelling Salesman Problem (TSP) [56] on the given graph as shown in [24]. The single robot problem becomes more challenging when the importance of information gathered from different vertices varies [6]. The multi-robot problem to minimize the maximum staleness of information is one of the problems studied in this thesis.

In security and surveillance oriented scenarios [2, 4, 80, 57], one of the requirements can be that the team of robots should not stay away from any of the locations more than a specified amount of time. We refer to this time constraint as latency constraint for a location. Such latency constraints can also be used in environmental monitoring and sampling scenarios [79, 33, 83], where the frequency of sampling or the rate of change of the properties of environment can dictate the latency constraints. The problem of finding a set of paths for a team of robots that can continually visit the locations of the environment while satisfying the latency constraints is also considered in this thesis.

The problem of finding paths to satisfy latency constraints can also be used in applications where events arrive at locations of the environment and need to be detected within a specified time after their arrival. Such applications may include forest fire monitoring [18] where fires can start randomly at different locations with different probabilities, but need to be detected early for better decision making. Another such application is wildlife monitoring [45] where animals can appear at locations of interest such as ponds and stay there for some time before leaving. To observe all the events arriving in the environment, the time that an event stays at a location before leaving should be known for all the locations, and a minimum number of robots is required so that the path planning problem with latency constraints can be solved. Otherwise, when the exact event durations are not known, or if the number of robots is constrained, we consider the problem of finding patrolling paths for a team of robots to maximize the expected number of observed events. Moreover, we also consider adversarial events that can choose the location and time to appear in order to maximize their chances of remaining undetected. Such events can be used to model intruders in security applications. In order for an adversarial event to decide where and when to appear, it must know the patrolling policy of the robots. We not only consider the case where such events know the patrolling policy beforehand, but also consider events that have a limited time to observe and learn the patrolling policy before devising their strategy.

1.1 Literature Survey

In this section we review the literature related to multi-robot path planning and persistent monitoring problems. A more detailed review of the literature related to a specific problem is presented in the corresponding chapter.

Persistent monitoring problems have been extensively studied in the literature and there is a substantial amount of work dedicated to finding patrolling paths for the monitoring of environments [46, 87, 70, 35]. For a single robot, cyclic patrolling paths to detect randomly arriving events are studied in [95]. Different ways of randomizing a Travelling Salesman Path are considered in [85] to empirically show that randomizing the paths can help in detecting intrusions. In [6], the problem of minimizing the maximum revisit time is considered and an approximation algorithm is proposed for the problem. Distributed algorithms to minimize refresh times are given in [77]. Two heuristic approaches of minimizing the maximum staleness of information are compared in [71]. In contrast to these works, we consider the multi-robot problem for minimizing the maximum revisit time and provide an algorithm with upper bound on the cost of the solution.

In [73], persistent coverage using multiple robots in a continuous environment is considered. The coverage level of locations decay with time and robots must keep visiting the locations to maintain a desired coverage level. This scenario of coverage quality deteriorating with time is also considered in [75] where equitable partitions of a continuous non-convex environment are found, and then paths covering the partition for each robot are computed. In [74], the coverage of a location increases depending on the time a robot stays at that location, and the task is to find optimal staying times for robots. In [98, 97, 100] a similar problem of determining a visit sequence for a discrete set of locations along with the time spent at each location to gather information is considered from an optimal control perspective. Instead of considering decaying coverage levels at the locations of the environment, the problems studied in this thesis seek to detect events that arrive at the locations and leave after some time.

The robots performing persistent monitoring tasks in an environment stay active for very long times and therefore, they need to be refueled. The authors in [65] find routes for dedicated charging robots that rendezvous with the patrolling UAV's in order to replenish their batteries. In [63] the problem of monitoring a terrain using heterogeneous robots with charging constraints is considered. The problem of scheduling spare drones to take place of the drones that need recharging is considered in [42]. In this thesis, we do not consider the charging constraints of the robots and mention the possible ways to incorporate those constraints along with the associated challenges in the discussion about future work.

Several vehicle routing problems are closely related to persistent monitoring problems. The basic vehicle routing problem is a generalization of the Traveling Salesman Problem and seeks to find optimal set of routes for a group of vehicles to serve a set of customers [90]. In the *Vehicle Routing Problem with Time Windows* [16], customers have to be served within their time windows by several vehicles with limited capacity. The goal is to minimize the number of vehicles required. Since the problem does not require repeated visits, the length of the resulting tour is polynomially bounded, and thus the problem is in NP. In the *Deadline-TSP* [91], the vertices have deadlines for first visits. In the *Period Vehicle Routing Problem* [26], the problem is to design routes for each day of a given period where each customer may require a number of visits (in a given number of allowable combinations) during this period. In the *Orienteering Problem* [37], the robot or vehicle has to visit as many locations as possible within a given time budget. The problem can be generalized to consider vertices with weights and the objective is to maximize the weight of the visited vertices. In the multi-robot version of the problem called the *Team Orienteering Problem* [22], the total weight collected by the team of robots is to be maximized. In the *Team Orienteering Problem With Time Windows*, score can only be collected from the vertices if they are visited within their specified time windows. A general framework for solving such routing and planning problems for multiple robots is given in [47], where the task allocation for robots is encoded as a Boolean Satisfiability problem and the path planning problem is encoded as a Traveling Salesman Problem. The main difference between these problems and the persistent monitoring problems studied in this thesis is that we consider the problems where the locations need to be visited repeatedly and indefinitely.

1.2 Contributions and Organization

The organization and contributions of this thesis are as follows.

Chapter 2 The mathematical preliminaries and background along with some definitions and notation are given in this chapter.

Chapter 3 This chapter considers the persistent monitoring problem of finding the minimum number of robots to satisfy latency constraints (maximum allowed time between two consecutive visits to a location). We present an $O(\log \rho)$ approximation algorithm for the problem where ρ is the ratio of the maximum and minimum latency constraints. We also provide an orienteering-based heuristic algorithm to solve the problem and show through simulations that the heuristic algorithm gives better solutions than the approximation algorithm. We also compare our algorithms with an existing solver on benchmark instances. Moreover, we also provide an algorithm for the problem of minimizing the maximum weighted latency for a given number of robots.

Some of the algorithms and results provided in this chapter are presented in the following publication.

- Ahmad Bilal Asghar, Stephen L. Smith, and Shreyas Sundaram. Multi-Robot Routing for Persistent Monitoring with Latency Constraints. In *American Control Conference*, pages 2620–2625, 2019.

Chapter 4 In this chapter we consider the problem of finding patrolling paths for a given number of robots to maximize the expected value of the events observed. We motivate the use of Markov chains as patrolling policies and characterize the expected value of observed events as a function of the patrolling policy. We show that if the arrival of events is stochastic, the objective function is submodular. We give an approximation algorithm for the case where the event durations are fixed. We also present a centralized local search and an online distributed algorithm to solve the general problem. Finally we study adversarial events that can choose their vertex and time to appear in order to minimize the expected reward of patrolling robots.

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

- Ahmad Bilal Asghar and Stephen L. Smith. Stochastic Patrolling in Adversarial Settings. In *American Control Conference*, pages 6435–6440, 2016.

Chapter 5 This chapter models events as intelligent intruders that can observe the patrolling policy of the robots for a limited amount of time before deciding where and when to appear in the environment to maximize their chances of remaining undetected. We model the scenario as a game and present a strategy for a rational intruder. We characterize the expected payoffs for the players. We set up the problem as a linearly constrained non-convex, non-smooth optimization problem and show that the objective function is locally Lipschitz so that direct search methods guarantee convergence to a Clarke Stationary point.

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

- Ahmad Bilal Asghar and Stephen L. Smith. A Patrolling Game for Adversaries with Limited Observation Time. In *IEEE Conference on Decision and Control*, pages 3305–3310, December 2018.

Chapter 6 The summary of the work presented in this thesis, and potential directions for future work are given in this chapter.

Other Publications I have also collaborated on the following publications during my PhD. However, this work is not presented in this thesis.

- Armin Sadeghi, Ahmad Bilal Asghar, and Stephen L. Smith. On Minimum Time Multi-Robot Planning with Guarantees on the Total Collected Reward. In *IEEE International Symposium on Multi-Robot and Multi-Agent Systems*, pages 16–22, 2019.

- Ahmad Bilal Asghar, Syed Talha Jawaid, and Stephen L. Smith. A Complete Greedy Algorithm for Infinite-Horizon Sensor Scheduling. In *Automatica*, volume 81, pages 335—341, 2017.

Chapter 2

Preliminaries

In this chapter we provide some mathematical background and notation for the thesis. In Section 2.1 we review complexity theory. Some basic definitions from graph theory are given in Section 2.2. In Section 2.3 we present Markov chains. We present some concepts related to probability and random processes in Section 2.4. The discussion on complexity theory is from [28] and [27]. The definitions related to probability and random processes are from [39]. The background on Markov chains is from [51, 39] and the definitions for graph theory are from [68].

2.1 Complexity Theory

Classifying the computational complexity of a problem is an important step in attempting to design algorithms to solve the problem. For example, if problem A is NP-hard (defined below), then no polynomial time algorithm for problem A exists unless $P = NP$, which is a well-known open problem. Therefore, one can either try to devise a super-polynomial time algorithm that solves the problem optimally, or find polynomial time algorithms that give approximate or heuristic solutions to the problem.

2.1.1 Decision and Optimization Problems

The problems discussed in this thesis are optimization problems that look to minimize or maximize an objective which is a function of the problem variables. Decision problems are problems with a *yes* or *no* answer. The decision version of a minimization (or maximization) problem A can be formulated by asking whether a solution exists for A with objective value at most (or at least) c. A search problem is a decision problem that also seeks the solution if the answer to the above mentioned decision problem is yes. For example, the optimization problem for the Travelling Salesman problem is to find the shortest tour visiting all the locations, whereas the decision version of the problem is whether a tour exists visiting all the vertices of distance at most c. A search problem is to find a tour of cost at most c if it exists.

2.1.2 Reductions and Complexity Classes

A *reduction* translates an instance of one problem to an instance of some other problem. A search problem A can be reduced to a search problem B by finding two polynomial time algorithms f and h such that algorithm f transforms an instance I of problem A to instance $f(I)$ of problem B and algorithm h maps the solution S of $f(I)$ into a solution $h(S)$ of instance I. Hence, by reducing problem A to problem B, we can solve problem A using a solver for problem B, which means that problem B is at least as hard as problem A.

Now we define some complexity classes. The complexity class P is the class of search problems that can be solved in time that is polynomial in the size of input to the problem. The class NP consists of problems for which a solution can be verified in polynomial time. Problem A is in class NP-hard if all the problems in NP can be reduced to A in polynomial time. A problem is in class NP-complete if it is in NP and NP-hard. If any NP-complete problem can be solved in polynomial time, then every NP-complete problem can be solved in polynomial time. Another class of problems that we refer to in this thesis is PSPACE-complete. The class PSPACE is the class of problems that can be solved using memory that is polynomial in the input size. Similarly a problem is PSPACE-complete if it in class PSPACE and if every other problem in PSPACE can be reduced to this problem.

2.2 Graphs

A graph G is defined as a pair $G = (V, E)$, where V represents the set of vertices or nodes of the graph and E represents the set of edges connecting the vertices of the graph. The vertices $u, v \in V$ are connected if $\{u, v\} \in E$. The vertices connected to a vertex i by an edge are called the neighbors of vertex i and are denoted as $\mathcal{N}(i)$.

Definition 2.2.1 (Directed Graph). *A graph is called a directed graph if edges are directed from one vertex to the other. In this case, each edge is an ordered pair of the form (u, v) which means that the edge points from u to v. So, $E \subseteq V \times V$.*

Definition 2.2.2 (Weighted Graph). *A graph is weighted if there is a function $\ell : E \to \mathbb{R}$ assigning lengths to edges of the graph.*

In this thesis we may also treat the weights on the edges as a matrix $\mathcal{L} = [\ell_{ij}]$ where the element ℓ_{ij} gives the length on the edge $\{i, j\}$ or (i, j). We will only deal with graphs that have non-negative edge weights.

Definition 2.2.3 (Metric Graph). *A weighted graph $G = (V, E)$ with edge lengths given by $\ell : E \to \mathbb{R}$ is called a metric graph if the edge lengths satisfy triangle inequality, i.e., for edges $(i, j), (j, k)$ and (i, k), $\ell_{ik} \leq \ell_{ij} + \ell_{jk}$.*

Definition 2.2.4 (Subgraph). *A subgraph $H = (V_H, E_H)$ of G is a graph with $V_H \subseteq V$ and $E_H \subseteq E$.*

A subgraph of G induced by vertices $V_H \subseteq V$ is a subgraph $H = (V_H, E_H)$ where E_H consists of all the edges in E that have both endpoints in V_H.

Definition 2.2.5 (Path). *A path in a graph $G = (V, E)$ is a subgraph*

$$P = (\{v_1, v_2, \ldots, v_{k+1}\}, \{e_1, e_2, \ldots, e_k\})$$

such that $v_i \neq v_j, \forall i \neq j$ and $e_i = \{v_i, v_{i+1}\}$.

The distance $\texttt{dist}(i, j)$ between two vertices i and j of a graph is the length of the shortest path from i to j. A graph is called *metric* if the distances satisfy triangle inequality, i.e., $\texttt{dist}(i, k) \leq \texttt{dist}(i, j) + \texttt{dist}(j, k)$, for all $i, j, k \in V$.

Definition 2.2.6 (Simple Walk). *A simple walk in graph $G = (V, E)$ is defined as a sequence of vertices $(v_1, v_2, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E$ for each $1 \leq i < k$.*

In this thesis, we often refer to a path as just a sequence of vertices or a walk in the graph. In this context a *random path* on a graph is just a random sequence of vertices of the graph.

Definition 2.2.7 (Cycle). *A cycle is a simple walk that starts and ends at the same vertex with no other vertex appearing more than once.*

Definition 2.2.8 (Tour). *A tour is a simple walk with no repeating edges.*

Definition 2.2.9 (Infinite Walk). *An infinite walk is a sequence of vertices $(v_1, v_2, \ldots)$ such that $(v_i, v_{i+1}) \in E$ for each $i \in \mathbb{N}$.*

Given walks W_1 and W_2, $[W_1, W_2]$ represents the concatenation of the walks. Given a finite walk W, an infinite periodic walk is constructed by concatenating infinite copies of W together, and is denoted by $\Delta(W)$.

In general, a walk can stay for some time at a vertex before traversing the edge towards the next vertex. Therefore we define a timed walk as follows.

Definition 2.2.10 (Timed Walk). *A* timed walk *W in graph G is a sequence $(o_1, o_2, \ldots, o_k)$, where $o_i = (v_i, t_i)$ is an ordered pair that represents the holding time t_i that the walk W spends at vertex v_i, such that $(v_i, v_{i+1}) \in E$ for each $1 \leq i < k$.*

The definitions of infinite walk and periodic walk can be extended to infinite timed walk and periodic timed walk. A walk with ordered pairs of the form $(v_i, 0)$ becomes a simple walk. The vertices traversed by walk W are denoted by $V(W)$ and the length of walk $W = ((v_1, t_1), (v_2, t_2), \ldots, (v_k, t_k))$ is denoted by $\ell(W) = \sum_{i=1}^{k-1} \ell(v_i, v_{i+1}) + \sum_{i=1}^{k} t_k$.

Since we are considering multi-robot problems, synchronization between the walks is important. Given a set of walks $\mathcal{W} = \{W_1, W_2, \ldots, W_k\}$ on graph G, we assume that at time 0, each robot i is at the first vertex v_1^i of its walk W_i, and will spend the holding time t_1^i at that vertex before moving to v_2^i.

Now we discuss some combinatorial problems related to graph theory that we will be referring to in the thesis.

2.2.1 Hamiltonian Cycle

Given an unweighted undirected graph $G = (V, E)$, the Hamiltonian Cycle Problem is to find a cycle in G that visits each vertex of the graph exactly once.

2.2.2 Orienteering Problem

The Orienteering Problem [37] is a variant of the TRAVELING SALESMAN PROBLEM. The input to the problem is a weighted directed or undirected graph $G = (V, E)$ with edge weights $\ell(e)$ for $e \in E$, two vertices $s, t \in V$, and a time limit T_{max}. Each vertex $i \in V$ has a score ψ_i associated with it. The problem is to find an $s-t$ walk (a walk originating from s and arriving eventually at t) of length not exceeding T_{max} which maximizes the total score obtained by visiting the vertices. If a vertex is visited more than once in the walk, its score is counted only once.

A straightforward reduction from TSP renders the orienteering problem NP-hard. Chekuri *et al.* [23] give a $(2+\epsilon)$ approximation algorithm for the problem. The existing exact and approximation algorithms are discussed and compared in [40].

We will also refer to a slight variation of the Orienteering Problem in Chapter 4 where instead of finding an $s-t$ walk, the objective is to find any cycle in the graph of length at most T_{max} maximizing the collected score. So, the input to the problem does not contain the vertices s and t.

2.2.3 Minimum Cycle Cover Problem

Given a graph G and length λ, the Minimum Cycle Cover Problem (MCCP) is to find minimum number of cycles that cover the whole graph such that the length of the longest cycle is at most λ. This problem is NP-hard and a 14/3-approximation algorithm for MCCP is given in [96].

2.3 Markov Chains

We will be using Markov chains to model random patrolling paths. This section gives the necessary background on Markov chains.

A Markov chain is a discrete stochastic process $X = \{X_0, X_1, \ldots\}$ where X_i takes values in a countable set S called the state space.

Definition 2.3.1 (Markov Chain)**.** *A Markov chain is random process X that satisfies the Markov condition*

$$\mathbb{P}\left[X_n = s | X_0 = x_0, X_1 = x_1, \ldots, X_{n-1} = x_{n-1}\right] = \mathbb{P}\left[X_n = s | X_{n-1} = x_{n-1}\right]$$

for all $n \geq 1$ and for $s, x_0, x_1, x_2, \ldots, x_{n-1} \in S$.

Definition 2.3.2 (Homogeneous Markov Chain)**.** *A Markov chain is called homogeneous if the transition probability from one state to another is independent of time, i.e.*

$$\mathbb{P}\left[X_{n+1} = j | X_n = i\right] = \mathbb{P}\left[X_1 = j | X_0 = i\right]$$

for all $n \geq 1$ and for all $i, j \in S$.

Definition 2.3.3 (Transition Matrix). *The transition matrix of a homogeneous Markov chain P is the $|S| \times |S|$ matrix of the transition probabilities. So,*

$$p_{ij} = \mathbb{P}\left[X_{n+1} = j | X_n = i\right].$$

We will only be considering homogeneous Markov chains and we will use notation P generally for the transition matrix.

The transition matrix is a row stochastic matrix, i.e.,

- $p_{ij} \geq 0$, for all i, j.
- $\sum_j p_{ij} = 1$ for all i.

Definition 2.3.4 (Recurrent State). *A state $i \in S$ is called recurrent if the probability of eventually visiting state i, starting from state i is 1. If this probability is less than 1, then the state is called transient.*

In the following definition, the notation $p_{ij}^{(n)}$ will represent the element in the i^{th} row and j^{th} column of the matrix P^n.

Definition 2.3.5 (Communication). *We say that state i communicates with state j, if $p_{ij}^{(m)} > 0$ for some $m > 0$. The states i and j intercommunicate with each other if i communicates with j and j communicates with i.*

Definition 2.3.6 (Communicating Class). *A set of states C is called a communicating class if states i and j intercommunicate with each other for $i, j \in C$ and no state in C communicates with any state not in C.*

Notice that if a Markov chain has more than one communicating class, we can decompose the chain into two independent Markov chains. Hence, we will not be considering Markov chains with more than one communicating class in this thesis. However, a Markov chain with one communicating class can have a set of transient states as well. Once, the chain leaves the set of transient states and moves into the communicating class, it will not return to those states.

Definition 2.3.7 (Irreducible Chain). *A Markov chain is called irreducible if states i and j intercommunicate with each other, for all $i, j \in S$.*

Definition 2.3.8 (Regular Markov Chain). *A Markov chain is called regular if all sufficiently high powers of the transition matrix have only positive elements.*

Definition 2.3.9 (Stationary Distribution)**.** *The vector* $\boldsymbol{\pi} \in \mathbb{R}^{1\times|S|}$ *is called the stationary distribution of Markov chain if*

1. $\pi_j \geq 0$*, for all* j*;*
2. $\sum_j \pi_j = 1$*;*
3. $\boldsymbol{\pi} P = \boldsymbol{\pi}$*.*

Definition 2.3.10 (First Passage Time, [78, 51])**.** *Let* T_j *denote the number of transitions after which the chain reaches state* j *for the first time, i.e.* $T_j = \min\{k \geq 1 : X_k = j\}$*. Then the conditional mean of this time* $m_{ij} = \mathbb{E}\left[T_j | X_0 = i\right]$ *is called the first passage time from state* i *to state* j*.*

The mean first passage time can be written as

$$m_{ij} = p_{ij} + \sum_{k\neq j} p_{ik}(m_{ik} + 1).$$

Theorem 2.3.11 (Kemeny Constant, [51])**.** *The quantity* $\sum_j m_{ij}\pi_j$ *is independent of the starting state* i *for a regular Markov chain.*

Definition 2.3.12. *The quantity* $\sum_j m_{ij}\pi_j$ *represents the expected first passage time from state* i *to a random state selected according to the stationary distribution* $\boldsymbol{\pi}$*, and is called the Kemeny constant of the Markov chain.*

For the Markov chains, where all the transitions do not take equal amount of time, the above defined first passage times will refer to the number of transitions instead of the time taken in going from one state to the other. Let w_{ij} be the time spent in the transition from state i to state j. Then the first passage time n_{ij} will be given by [3]

$$n_{ij} = p_{ij}(w_{ij}) + \sum_{k\neq j} p_{ik}(n_{kj} + w_{kj}).$$

This concept of first passage times for weighted graphs is used in [3] to find Markov chains with minimal first passage times.

2.4 Probability and Random Variables

2.4.1 Random Variables

Random variables can be thought of as consequences related to the outcomes of the experiments.

Definition 2.4.1 (Probability Space)**.** *A probability space is a triple* $(\Omega, \mathcal{F}, \mathbb{P})$ *where:*

- Ω *is called the sample space and is the set of all possible outcomes of an experiment.*
- $\mathcal{F} \subseteq 2^{\Omega}$ *such that*
    1. $\Omega \in \mathcal{F}$,
    2. $A \in \mathcal{F} \implies (\Omega \setminus A) \in \mathcal{F}$, *and*
    3. $A_i \in \mathcal{F}$ *for* $i = 1, 2, \ldots \implies (\bigcup_{i=1}^{\infty} A_i) \in \mathcal{F}$.
- $\mathbb{P} : \mathcal{F} \to [0, 1]$ *such that*
    1. $\mathbb{P}[\Omega] = 1$, *and*
    2. *if* $\{A_i\}_{i=1}^{\infty} \in \mathcal{F}$ *is a countable collection of pairwise disjoint sets, then* $\mathbb{P}[\bigcup_{i=1}^{\infty} A_i] = \sum_{i=1}^{\infty} \mathbb{P}[A_i]$.

Definition 2.4.2 (Random Variable)**.** *A random variable is a function* $X : \Omega \to \mathbb{R}$ *such that* $\{\omega \in \Omega | X(\omega) \leq x\} \in \mathcal{F}$ *for each* $x \in \mathbb{R}$.

Definition 2.4.3 (Distribution Function)**.** *The distribution function of a random variable* X *is a function* $F : \mathbb{R} \to [0, 1]$ *given by*

$$F(x) = \mathbb{P}[X \leq x].$$

Definition 2.4.4 (Discrete Random Variable)**.** *The random variable* X *is called discrete if it takes values in some countable set. It has a probability mass function (PMF)* $f : \mathbb{R} \to [0, 1]$ *given by*

$$f(x) = \mathbb{P}[X = x].$$

Definition 2.4.5 (Continuous Random Variable)**.** *The random variable* X *is called continuous if its distribution function* $F(x)$ *can be written as*

$$F(x) = \int_{-\infty}^{x} f(u)du$$

for some integrable function $f : \mathbb{R} \to [0, \infty]$ *called the probability density function (PDF) of* X.

Definition 2.4.6 (Expectation of Discrete Random Variable)**.** *The expected value or expectation of a discrete random variable X with PMF $f(x)$ is*

$$\mathbb{E}[X] = \sum_x x f(x).$$

Definition 2.4.7 (Expectation of Continuous Random Variable)**.** *The expected value or expectation of a continuous random variable X with PDF $f(x)$ is*

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f(x) dx.$$

For a function $g : \mathbb{R} \to \mathbb{R}$, the expectation of the function $g(X)$ of random variable X is given by $\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x) f(x) dx$.

Expectation is a linear operator. So

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y], \quad a, b \in \mathbb{R}.$$

Definition 2.4.8 (Variance)**.** *The variance of a random variable X is*

$$\begin{aligned} \mathtt{Var}[X] =& \mathbb{E}\left[(X - \mathbb{E}(X))^2\right] \\ =& \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2. \end{aligned}$$

2.4.2 Some Distribution Functions

Now we describe some useful distribution functions that we will be using in this thesis.

Multinomial Distribution

Let $X \in \{1, 2, \ldots, K\}$ be a discrete random variable, and $\mathbb{P}[X = j] = \theta_j$, then

$$\mathbb{P}[X] = \prod_{j=1}^{K} \theta_j^{(X=j)}$$

is a multinomial distribution where $(X = j)$ is a truth statement that evaluates to 1 or 0. In Chapter 6, we represent each row of the transition matrix of Markov chain as a multinomial distribution. Then, learning of Markov chains is equivalent to estimating the parameters of the multinomial distributions.

Given n independent trials, let X_i be the discrete random variable representing the number of times the outcome of the trial is $i \in \{1, 2, \ldots, K\}$. Then the probability mass function of multinomial distribution is also defined as

$$f(x_1, \ldots, x_K) = \begin{cases} \frac{n!}{x_1! \ldots x_K!} \theta_1^{x_1} \ldots \theta_K^{x_K}, & \text{when } \sum_{i=1}^{K} x_i = n \\ 0, & \text{otherwise.} \end{cases}$$

Dirichlet Distribution

Dirichlet distribution is a continuous multivariate distribution. The Dirichlet distribution with parameters $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_K\}$ has the probability density function

$$f(x_1, \ldots, x_K) = \frac{1}{\beta(\alpha)} \prod_{i=1}^{K} x_i^{\alpha_i - 1}$$

such that

$$\begin{aligned} x_1, x_2, \ldots, x_{K-1} &> 0 \\ x_1 + x_2 + \ldots + x_{K-1} &< 1 \\ x_K &= 1 - x_1 - \ldots - x_{K-1}. \end{aligned}$$

The function $\beta(\alpha)$ is the normalizing constant.

The mean of the random variable X_i from the Dirichlet distribution is given by

$$\mathbb{E}\left[X_i\right] = \frac{\alpha_i}{\alpha_0} \tag{2.1}$$

and the variance of X_i is given by

$$\texttt{Var}[X_i] = \frac{\alpha_i(\alpha_0 - \alpha_i)}{\alpha_0^2(\alpha_0 + 1)} \tag{2.2}$$

where $\alpha_0 = \sum_i \alpha_i$.

Chapter 3

Persistent Monitoring with Latency Constraints

Persistent monitoring tasks such as surveillance and environmental sampling require that locations in an environment be visited continually by a team of robots. In such tasks, locations of more importance should be visited more often as compared to locations with relatively low importance. This requirement can be modeled using a latency constraint for each location which specifies the maximum time the robots are allowed to stay away from a location. The locations that are at higher risk in a surveillance application will have a lower latency constraint and hence will be visited more often by the robots to satisfy that latency constraint. In this chapter we consider the problem of finding the patrolling paths for the robots to satisfy the latency constraints of the locations in the environment. In an application, where the number of robots is constrained, finding a feasible solution to the problem may not be possible. In such cases, we consider the problem of minimizing the maximum weighted latency of the locations in the environment.

The contributions of this chapter are as follows. The literature related to the problems discussed in this chapter is reviewed in Section 3.1. We formally define the problem of minimizing the number of robots to satisfy the latency constraints in Section 3.2. In Section 3.3 we present an $O(\log \rho)$ approximation algorithm for the problem where ρ is the ratio of the maximum and the minimum latency constraints. We provide some heuristic algorithms to solve the problem in Section 3.4 and show through simulations that the proposed orienteering-based heuristic algorithm gives better solutions than the approximation algorithm. In Section 3.5 we study the problem of minimizing the maximum weighted latency using multiple robots and provide an algorithm for the problem. We also give some results that relate this problem to the problem of satisfying latency constraints. Finally, in Section 3.6 we evaluate the performance of the algorithms on large problem instances in a patrolling scenario and in an image collection application for 3D scene reconstruction. We also compare our algorithms against an existing solver on benchmark instances.

3.1 Related Work

The problem of finding paths to satisfy latency constraints has been studied in [54, 44, 31]. The authors in [54] use incomplete greedy heuristics to find if a single robot can satisfy the constraints on all vertices of a graph. They show that if a solution exists, then a periodic solution also exists. In this chapter, we consider the multi-robot problem and our objective is to minimize the number of robots that can satisfy the latency constraints on the given graph. The multi-robot version of the problem has been considered in [31, 44], where it is called *Cyclic Routing of Unmanned Aerial Vehicles*. The decision version of the problem for a single robot is shown to be PSPACE-complete in [44]. The authors also show that the length of even one period of a feasible walk can be exponential in the size of the problem instance. In [31], the authors propose a solver based on Satisfiability Modulo Theories (SMT). To apply an SMT solver, they impose an upper bound on the length of the period of the solution. Since an upper bound is not known *a priori*, the solver will not return the optimal solution if the true optimal period exceeds the bound. The authors generate a library of test instances, but since their algorithm scales exponentially with the problem size, they solve instances up to only 7 vertices. We compare our algorithms with their solver and show that our algorithms run over 500 times faster on average and return solutions with the same number of robots on 98% of the benchmark instances provided by [31].

A related single robot problem is studied in [6] where each vertex has a weight associated with it and the objective is to minimize the maximum weighted latency (time between consecutive visits) for an infinite walk. The authors provide an approximation algorithm for this problem. In this chapter, we consider the multi-robot version of this problem and provide an algorithm to solve the problem. The authors in [12] study the single robot problem for a security application where the length of attack on each vertex of the graph is given. To intercept all possible attacks, they design an algorithm to repeatedly patrol all vertices with the maximum revisit time to each vertex less than its length of attack. The algorithm is not guaranteed to find a solution if it exists.

Timed automata have been used to model general multi-robot path planning problems [82] as the clock states can capture the concurrent time dependent motion. In [92], temporal logic constraints are used to specify high-level mission objectives to be achieved by a set of robots. The routing problem with latency constraints can also be modeled as a timed-automaton since multiple robots may require synchronization to satisfy the latency constraints. A timed automaton based solution to the problem is presented in [30], however it is shown to perform more poorly than the SMT-based approach in [31], which we use as a comparison for our proposed algorithms.

3.2 Problem Statement

Consider a directed weighted graph $G = (V, E)$, where the set of vertices V represents the locations to be monitored in the environment. The edge lengths are given by $\ell(e)$ for each $e \in E$. These edge lengths are metric and represent the time taken by the robots to travel between the vertices. The latency constraint for each vertex v is denoted by $r(v)$ and represents the maximum time allowed between consecutive visits to v. In a practical application, the robots may need to stay at a vertex for some time to inspect the vertex or to gather information from that vertex. This time taken by the robots to stay at a vertex can be added to the length of the incoming edges of that vertex to get an equivalent metric graph with zero stay times and modified latency constraints [31]. Hence, we assume without loss of generality, that the time required by the robots to inspect a vertex is zero. We formally define the problem statement after the following definition of latency.

Definition 3.2.1 (Latency)**.** *Given a set of infinite walks $\mathcal{W} = \{W_1, W_2, \ldots, W_k\}$ on a graph G, let a_i^v represent the i^{th} arrival time for the walks to vertex v. Similarly, let d_i^v represent the i^{th} departure time from vertex v. Then the latency $L(\mathcal{W}, v)$ of vertex v on walks $\mathcal{W}$ is defined as the maximum time spent away from vertex v by the walks, i.e., $L(\mathcal{W}, v) = \sup_i (a_{i+1}^v - d_i^v)$.*

Problem 3.2.2 (Minimizing Robots with Latency Constraints)**.** *Given latency constraints $r : V \to \mathbb{R} \cup \{\infty\}$, the optimization problem is to find a set of walks $\mathcal{W}$ with minimum cardinality such that $L(\mathcal{W}, v) \leq r(v), \forall v \in V$.*

The decision version of the problem is to determine whether there exists a set of R walks $\mathcal{W} = \{W_1, W_2, \ldots, W_R\}$ such that $L(\mathcal{W}, v) \leq r(v)$ for all $v \in V$. Note that although we have assumed that the robots do not need to stay at a vertex to inspect that vertex, in a general solution to the problem the robots might still need to stay at vertices in order to satisfy the latency constraints. Therefore a general solution to Problem 3.2.2 will be a set of timed walks with possibly non-zero holding times.

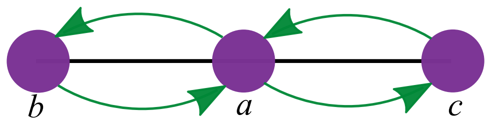


Figure 3.1: A graph with three vertices and the walk (a, b, a, c, a). The length of shown edges is one. Equally spacing two robots on this walk does not halve the latencies.

In this problem definition, the graph and its edge lengths capture the robot motion in the environment. This graph can be generated from a probabilistic roadmap (PRM), or any other environment decomposition method [50]. The latency constraints provide the maximum allowable time between visits to a vertex. For example, in dynamic scene reconstruction, each vertex corresponds to a viewpoint [84]. The latency constraints may encode the maximum staleness of information that can be tolerated for the voxels captured at that viewpoint.

3.2.1 Multiple Robots on the Same Walk

In multi-robot problems that involve finding cycles or tours in a graph, the cost of the tour can be reduced by a factor of n by placing n robots on the tour such that each robot follows the one ahead of it at a distance of $1/n$ times the length of the tour [24]. We will refer to this placement of multiple robots on a tour as equally spacing robots on a tour. Equally spacing multiple robots on the solution for a single robot does not work in a similar manner for Problem 3.2.2: if a periodic walk W gives latency $L(W, v)$ on vertex v, equally spacing more robots on one period of that walk does not necessarily reduce the latency for that vertex. Figure 3.1 gives an example of such a walk. The latency of vertices a, b and c on the walk (a, b, a, c, a) are $2, 4$ and 4 respectively. The length of one period of the walk is 4. If we place another robot following the first robot with a lag of 2 units, the latency of vertex a remains the same. If we place the second robot at a lag of 1 unit, the latency will reduce to 1 for vertex a and 3 for vertices b and c. Hence cycles are an exception, for general walks we need more sophisticated algorithms than finding a walk for a single robot and adding more robots on that walk until the constraints are satisfied.

3.3 Approximation Algorithm

Since Problem 3.2.2 is PSPACE-complete, we resort to finding approximate and heuristic solutions to the problem. In this section, we present an approximation algorithm for the problem.

3.3.1 $O(\log \rho)$ Approximation

We first mention a simple approach to the problem and then improve it incrementally to get the approximation algorithm. One naive solution is to find a TSP tour of the graph and equally space robots on that tour to satisfy all the latency constraints. However, a single vertex with a very small $r(v)$ can result in a solution with the number of robots proportional to $1/r(v)$. To solve this issue, we can partition the vertices of the graph such that the latencies in one partition are close to each other, and then place robots on the TSP tour of each partition. If more than one robot is required for a partition V', then another approach is to solve the MCCP for that partition. The benefit of using the MCCP over placing multiple robots on a TSP is that if all the vertices in V' had the same latency requirement, then we have a guarantee on the number of cycles required for that partition. However, a general solution to the problem might not consist of simple cycles. Lemma 3.3.2 relates solutions consisting of cycles to general solutions and shows that a solution consisting of cycles will have latencies no more than twice that of any general solution with same number of robots. Therefore, if we solve the MCCP on a partition with its latency constraints multiplied by two, we have a guarantee on the number of cycles. We can then use the naive idea from TSP based solutions and place multiple robots on each cycle to satisfy the latency constraints.

The approximation algorithm is given in Algorithm 1. The first five lines of the algorithm partition the vertices according to their latency constraints. For each of those partitions, the function MCCP(V, λ) called in Line 7 uses an approximation algorithm for the minimum cycle cover problem from [96]. Then, the appropriate number of robots are placed on each cycle returned by the MCCP function to satisfy the latency constraints. We will need the following definition to establish the approximation ratio of Algorithm 1. A similar relaxation technique was used in [6].

Definition 3.3.1. *Let $r_{min} = \min_v r(v)$. The latency constraints of the problem are said to be relaxed if for any vertex v, its latency constraint is updated from $r(v)$ to $\bar{r}(v) = r_{min}2^x$ such that x is the smallest integer for which $r(v) < r_{min}2^x$.*

We will also need the following lemma that follows from Lemma 2 in [24] and we omit the proof for brevity.

Lemma 3.3.2. *For any set of walks $\mathcal{W}$ on an undirected metric graph with vertices V, there exists a set of walks $\mathcal{W}'$ on V with $|\mathcal{W}| = |\mathcal{W}'|$, such that each walk $W_i \in \mathcal{W}'$ is a cycle of vertices $V_j \subseteq V$, and the sets V_j partition V, and $\max_v L(\mathcal{W}', v) \leq 2\max_v L(\mathcal{W}, v)$.*

Algorithm 1 ApproximationAlgorithm

```
Input: Graph $G = (V, E)$, latency constraints $r(v), \forall v$
Output: A set of walks $\mathcal{W}$, such that $L(\mathcal{W}, v) \leq r(v)$
1: Let $r_{\texttt{max}} = \max_v r(v)$, $r_{\texttt{min}} = \min_v r(v)$, $\rho = \frac{r_{\texttt{max}}}{r_{\texttt{min}}}$
2: if $r_{\texttt{max}}/r_{\texttt{min}}$ is an exact power of 2 then $\rho = \frac{r_{\texttt{max}}}{r_{\texttt{min}}} + 1$
3: end if
4: $\mathcal{W} = \{\}$
5: Let $V_i$ be the set of vertices $v$ such that $r_{\texttt{min}} 2^{i-1} \leq r(v) < r_{\texttt{min}} 2^i$ for $1 \leq i \leq \lceil \log_2 \rho \rceil$
6: for $i = 1$ to $\lceil \log_2 \rho \rceil$ do
7:     $\mathcal{C} = \text{MCCP}(V_i, r_{\texttt{min}} 2^{i+1})$
8:     for $C$ in $\mathcal{C}$ do
9:         Equally space $\lceil \ell(C)/\min_{v \in V(C)} r(v) \rceil$ robots on cycle $C$ to get walks $\mathcal{W}'$
10:        $\mathcal{W} = \{\mathcal{W}, \mathcal{W}'\}$
11:    end for
12: end for
```

The following proposition gives the approximation factor of Algorithm 1.

Proposition 3.3.3. *Given an undirected metric graph $G = (V, E)$ with latency constraints $r(v)$ for $v \in V$, Algorithm 1 constructs R walks $\mathcal{W} = \{W_1, W_2, \ldots, W_R\}$ such that $L(\mathcal{W}, v) \leq r(v)$ for all $v \in V$ and $R \leq 4\alpha \lceil \log(\rho) \rceil R_{\texttt{OPT}}$, where $R_{\texttt{OPT}}$ is the minimum number of robots required to satisfy the latency constraints and α is the approximation factor of MCCP.*

Proof. Given that $R_{\texttt{OPT}}$ robots will satisfy the latency constraints $r(v)$, they will also satisfy the relaxed constraints $\bar{r}(v)$ since $\bar{r}(v) > r(v)$. Therefore, there exists a set of at most $R_{\texttt{OPT}}$ walks $\mathcal{W}^*$ such that for $v \in V_i$, $L(\mathcal{W}^*, v) \leq r_{\texttt{min}} 2^i$.

Using Lemma 3.3.2, given the set $\mathcal{W}^*$, $R_{\texttt{OPT}}$ cycles can be constructed in V_i such that the latency of each vertex in V_i is at most $r_{\texttt{min}} 2^{i+1}$. Hence, running an α approximation algorithm for Minimum Cycle Cover Problem (MCCP) on the subgraph with vertices V_i and with maximum cycle length $r_{\texttt{min}} 2^{i+1}$ will not return more than $\alpha R_{\texttt{OPT}}$ cycles.

Since MCCP returns cycles, equally spacing k robots on each cycle will reduce the latency of each vertex on that cycle by a factor of k. As $r(v) \geq r_{\texttt{min}} 2^{i+1}/4$ for each $v \in V_i$, we will need to place at most 4 robots on each cycle.

Finally, since there are at most $\lceil \log \rho \rceil$ partitions V_i, the algorithm will return $R \leq 4\alpha \lceil \log(\rho) \rceil R_{\texttt{OPT}}$ walks. □

Runtime: Since we run the approximation algorithm for MCCP on partitions of the graph, Algorithm 1 has the same time complexity as that of the approximation algorithm of MCCP. That is because the runtime of MCCP is superlinear, so if $\sum |V_i| = |V|$, then $\sum |V_i|^p \leq |V|^p$ for $p \geq 1$.

Remark 3.3.4 (Heuristic Improvements)**.** *Instead of finding cycles using MCCP for each partition V_i in Line 7 of the algorithm, we can also equally space robots on the Traveling Salesman Tour of V_i to get a feasible solution. In practice, we use both of these methods and pick the solution that gives the lower number of robots for each V_i. This modification can return better solutions to the problem but does not improve the approximation guarantee established in Proposition 3.3.3.*

3.4 Heuristic Algorithms

The approximation algorithm for Problem 3.2.2 presented in Section 3.3 is guaranteed to provide a solution within a fixed factor of the optimal solution. In this section, we propose a heuristic algorithm based on the Orienteering Problem, which in practice provides high-quality solutions.

3.4.1 Partitioned Solutions

In general, walks in a solution of the problem may share some of the vertices. However, sharing the vertices by multiple robots requires coordination and communication among the robots. Such strategies may also require the robots to hold at certain vertices for some time before traversing the next edge, in order to maintain synchronization. This is not possible for vehicles that must maintain forward motion, such as fixed-wing aircraft. The following example illustrates that if vertices are shared by the robots, lack of coordination or perturbation in edge weights can lead to large errors in latencies.

Example 3.4.1. *Consider the problem instance shown in Figure 3.2. An optimal set of walks for this problem is given by $\{W_1, W_2, W_3\}$ where $W_1 = ((a, 1), (b, 1))$, $W_2 = ((b, 0), (c, 0))$ and $W_3 = ((c, 0), (d, 1), (c, 1))$. Note that walk W_1 starts by staying on vertex a, while W_2 leaves vertex b and W_3 leaves vertex c. Also note that any partitioned solution will need 4 robots. Moreover, if the length of edge $\{b, c\}$ changes from 3 to $3 - \epsilon$, (e.g., if the robot's speed increases slightly) the latencies of vertices b and c will keep changing with time and will go up to 5. Hence, a small deviation in robot speed can result in a large impact on the monitoring objective.*

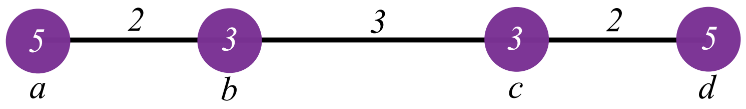


Figure 3.2: A problem instance with an optimal set of walks that share vertices. The latency constraints for each vertex are written inside that vertex. The edge lengths are labeled with the edges. The optimal walks are $\{W_1, W_2, W_3\}$ where $W_1 = ((a, 1), (b, 1))$, $W_2 = ((b, 0), (c, 0))$ and $W_3 = ((c, 0), (d, 1), (c, 1))$.

Since the above mentioned issues will not occur if the robots do not share the vertices of the graph, and the problem is PSPACE-complete even for a single robot, we focus on finding partitioned walks in this section. The general greedy approach used in this section is to find a single walk that satisfies latency constraints on a subset of vertices $V' \subseteq V$. Note that we do not know V' beforehand, but a feasible walk on a subset of vertices will determine V'. We then repeat this process of finding feasible walks on the remaining vertices of the graph until the whole graph is covered.

3.4.2 Greedy Algorithm

We now consider the problem of finding a single walk on the graph $G = (V, E)$ that satisfies the latency constraints on the vertices in $V' \subseteq V$. Given a robot walking on a graph, let $p(k)$ represent the vertex occupied by the robot after traversing k edges (after k steps) of the walk. Also, at step k, let the maximum time left until a vertex i has to be visited by the robot for its latency to be satisfied be represented by $s_i(k)$. If that vertex is not visited by the robot within that time, we say that the vertex expired. Hence, the vector $s(k) = [s_1(k), \dots, s_{|V'|}(k)]^T$ represents the time to expiry for each vertex. At the start of the walk, $s_i(0) = r(i)$, and $s_i(k)$ evolves according to the following equation:

$$s_i(k) = \begin{cases} r(i) & \text{if } p(k) = i \\ s_i(k-1) - \ell(p(k-1), p(k)) & \text{otherwise.} \end{cases} \tag{3.1}$$

We will use the notation s_i without the step k if it is clear that we are talking about the current time to expiry. An incomplete greedy heuristic for the decision version of the problem with $R = 1$ is presented in [54]. The heuristic is to pick the vertex with minimum value of $s_i(k)$ as the next vertex to be visited by the robot. This heuristic does not ensure that all the vertices on the walk will have their latency constraints satisfied since the distance to a vertex i to be visited might get larger than $s_i(k)$. To overcome this,

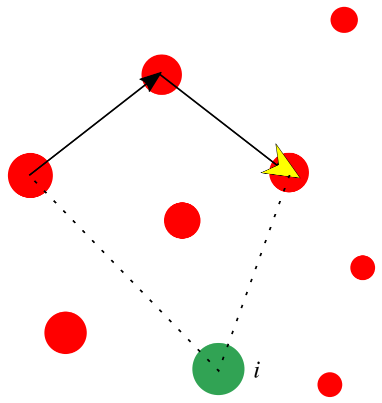


Figure 3.3: An example depicting a step of the greedy algorithm. The solid black lines show the existing walk W, and the yellow arrow shows the current position of the robot. The green vertex i is the vertex with least time to expiry. That vertex is appended to the existing walk and the closed walk is checked for feasibility before appending i to W.

we propose a modification to the heuristic to apply it to our problem. Given a walk W on graph G, the function $\textsc{PeriodicFeasibility}(W, G)$ determines whether the periodic walk $\Delta(W)$ is feasible on the vertices that are visited by W. This can be done simply in $O(|W|)$ time by traversing the walk $[W, W]$ and checking if the time to expiry for any of the visited vertices becomes negative. Given this function, the simple greedy algorithm is to pick the vertex $i = \arg\min\{s_j\}$ subject to the constraint that $\textsc{PeriodicFeasibility}([W, i], G)$ returns true, where W is the walk traversed so far. The algorithm terminates when all the vertices are either expired, or covered by the walk. An example of a step of the greedy algorithm is depicted in Figure 3.3.

3.4.3 Recursive Greedy Algorithm

The greedy algorithm presented above selects the vertex with the minimum time to expiry and adds it to the current walk if the resulting walk is feasible. In this section we extend the greedy algorithm and instead of going directly to vertex $i = \arg\min\{s_j\}$, we check if other vertices can be visited on the way to vertex i. This is done greedily as well, where we pick the vertex $j \neq i$ with the minimum time to expiry such that the walk $[W, j, i]$ remains feasible on the vertices visited by the walk. This is done recursively until no more vertices can be added to the walk. We refer to this algorithm as RecursiveGreedy.

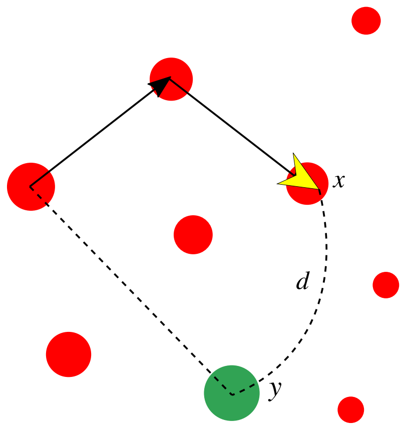
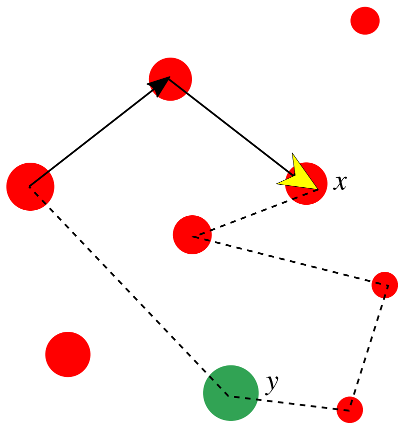



Figure 3.4: A step of the OrienteeringGreedy algorithm. The solid lines represent the walk traversed so far. Yellow arrow is the current position of the robot. Vertex y is picked greedily. The figure on left shows the time d that can be spent before going to y. The figure on right shows the orienteering path from x to y.

3.4.4 Orienteering Based Greedy Algorithm

In this section we take the idea of the RecursiveGreedy algorithm one step further and try to visit the 'best' combination of vertices on the way to vertex $i = \arg\min\{s_j\}$. This algorithm also finds partitioned walks like the previous two heuristic algorithms by finding a feasible walk on a subset of vertices and then considering the remaining subgraph. The algorithm is presented in Algorithm 2. From the current vertex x, the target vertex y is picked greedily as described in Section 3.4.2. Then the time d is calculated in Line 10 which is the maximum time to go from x to y for which the periodic walk remains feasible. In Line 17, Orienteering$(V - V_{\texttt{exp}}, x, y, d, \psi)$ finds a path in the vertices $V - V_{\texttt{exp}}$ from x to y of length at most d maximizing the sum of the weights ψ on the vertices of the path. The set $V_{\texttt{exp}}$ represents the expired vertices whose latencies cannot be satisfied by the current walk, and they will be considered by the next robot. The vertices with less time to expiry are given more importance in the path by setting weight $\psi_i = 1/s_i$ for vertex i. The vertices that are already in the walk will remain feasible, and so their weight is discounted by a small number m to encourage the path to explore unvisited vertices. One step of the algorithm is depicted in Figure 3.4. The following result shows that the algorithm will always find a feasible solution.

Proposition 3.4.2. *Algorithm 2 returns a feasible solution, i.e., for the set of walks $\mathcal{W}$ returned by Algorithm 2, $L(\mathcal{W}, v) \leq r(v)$, for all $v \in V$.*

Algorithm 2 OrienteeringGreedy

```
Input: Graph $G = (V, E)$, latency constraints $r(v), \forall v$
Output: A set of walks $\mathcal{W}$, such that $L(\mathcal{W}, v) \leq r(v)$
1: $j = 1$, $\mathcal{W} = \{\}$
2: while $V$ is not empty do
3:     $V_{\texttt{exp}} = \{\}$
4:     $s_i = r(i)$ for all $i \in V$
5:     $W_j$ = pick vertex $a$ randomly from $V$
6:     while $V - V(W_j) - V_{\texttt{exp}}$ is not empty do
7:         $x$ = last vertex in $W_j$
8:         for $y \in V - V_{\texttt{exp}}$ in increasing order of $s$ do
9:             if PeriodicFeasibility($[W_j, y], G$) then
10:                Use binary search between $\ell(x, y)$ and $s_y$ to get $d$ (time to go from $x$ to
                   $y$) such that $[W_j, y]$ remains feasible
11:                for $z$ in $V - (V_{\texttt{exp}} \cup V(W_j))$ do
12:                    if $s_z < d + \ell(y, a)$ then $V_{\texttt{exp}} = V_{\texttt{exp}} \cup z$
13:                    end if
14:                end for
15:                $\psi_i = 1/s_i$ for all non expired vertices $i$
16:                $\psi_i = m\psi_i$ for $i$ in $V(W_j)$
17:                $W_j = [W_j$,Orienteering$(V - V_{\texttt{exp}}, x, y, d, \psi)]$
18:                Update $s$ using Equation (3.1)
19:            else
20:                $V_{\texttt{exp}} = V_{\texttt{exp}} \cup y$
21:            end if
22:        end for
23:    end while
24:    $\mathcal{W} = \{\mathcal{W}, W_j\}$, $j = j + 1$
25:    $V = V - V(W_j)$
26: end while
```

Proof. The vertices covered by the walk W_j added to the solution in Line 24 are removed from the set of vertices before finding the rest of the walks. Hence the latencies of the vertices $V(W_j)$ will be satisfied by only W_j. We will show that every time W_j is appended in Line 17, it remains feasible on $V(W_j)$.

W_j starts from a single vertex a, and hence is feasible at the start. Let us denote W_j^- as the walk before Line 17 and W_j^+ as the walk after Line 17. Due to Line 10, if W_j^- is feasible in a particular iteration, then $[W_j^-, y]$ will remain feasible. Hence the only vertices than can possibly have their latency constraints violated in W_j^+ are in the orienteering path from x to y. Consider any vertex z in the path from x to y returned by the ORIENTEERING function in Line 17. If $z \in V(W_j^-)$, then $L(W_j^+, z) \leq r(z)$ because of Line 10. If $z \notin V(W_j^-)$, then $r(z) = \ell(W_j^-) + s_z$ and by Line 12, $r(z) \geq \ell(W_j^-) + d + \ell(y, a)$. As z is only visited once in W_j^+, $L(W_j^+, z) = \ell(W_j^+) \leq \ell(W_j^-) + d + \ell(y, a) \leq r(z)$. □

An approximation algorithm for ORIENTEERING can be used in Line 17 of Algorithm 2. In our implementation, we used an ILP formulation to solve ORIENTEERING. To improve the runtime in practice, we pre-process the graph before calling the ORIENTEERING solver to consider only the vertices z such that $\ell(x, z) + \ell(z, y) \leq d$. We show in the next section that although the runtime of Algorithm 2 is more than that of Algorithm 1, it can still solve instances with up to 90 vertices in a reasonable amount of time, and it finds better solutions.

3.5 Min Max Weighted Latency Problem

The approximation algorithm and analysis presented above helps in formulating an algorithm for the multi-robot version of a related problem. In [6], the authors study in detail the problem of minimizing the maximum weighted latency of a graph given a single robot. Here we define the multiple robot version of the problem.

Problem 3.5.1 (Minimizing Maximum Weighted Latency)**.** *Given a graph $G = (V, E)$ with weights $\phi(v)$ for $v \in V$, and a set of walks $\mathcal{W}$, the weighted latency of v is defined as $C(\mathcal{W}, v) = \phi(v)L(\mathcal{W}, v)$. Given the number of robots R, the problem of minimizing maximum weighted latency is to find a set of R infinite walks $\mathcal{W} = \{W_1, W_2, \ldots, W_R\}$ such that the cost $\max_v C(\mathcal{W}, v)$ is minimized.*

Without loss of generality, $\phi(v)$ is assumed to be normalized such that $\max_v \phi(v) = 1$. In this section we present an algorithm for this problem and relate this problem to Problem 3.2.2.

In [6], an approximation algorithm for the single robot version of Problem 3.5.1 is given. We will refer to that approximation algorithm as MinMaxLatencyOneRobot(G_j), which returns a walk in graph G_j such that the maximum weighted latency of that walk is not more than $(8 \log \rho_j + 10)\texttt{OPT}_1^j$, where ρ_j is the ratio of maximum to minimum vertex weights in G_j and $\texttt{OPT}_1^j$ is the optimal maximum weighted latency for one robot in G_j. We use this approximation algorithm as a subroutine in Algorithm 3 for minimizing the maximum weighted latency with multiple robots.

Algorithm 3 LatencyWalks

```
    Input: Graph $G = (V, E)$, vertex weights $\phi(v), \forall v \in V$, and number of robots $R$.
    Output: A set of $R$ walks $\{W_1, \dots, W_R\}$ in $G$.
 1: $\rho = \max_{i,j} \phi_i / \phi_j$
 2: if $\max_{i,j} \phi_i / \phi_j$ is an exact power of 2 then $\rho = \max_{i,j} \phi_i / \phi_j + 1$
 3: end if
 4: Let $V_i$ be the set of vertices of weight $\frac{1}{2^i} < \phi(u) \leq \frac{1}{2^{i-1}}$ for $1 \leq i \leq \lceil \log_2 \rho \rceil$
 5: if $R < \log \rho$ then
 6:     for $j = 1$ to $R$ do
 7:         Let $G_j$ be a subgraph of $G$ with vertices $V_i$ for $\lceil \frac{j-1}{R} \log \rho \rceil + 1 \leq i \leq \lceil \frac{j}{R} \log \rho \rceil$
 8:         $W_j$ = MinMaxLatencyOneRobot($G_j$)
 9:     end for
10: end if
11: if $R \geq \log \rho$ then
12:     Equally space $\lfloor R / \lceil \log \rho \rceil \rfloor$ robots on TSP tour of $V_i$ for all $i$ to get
        $\{W_1, \dots, W_{\lceil \log \rho \rceil \lfloor \frac{R}{\lceil \log \rho \rceil} \rfloor}\}$
13:     for $k = R - \lceil \log \rho \rceil \lfloor \frac{R}{\lceil \log \rho \rceil} \rfloor + 1$ to $R$ do
14:         Find subset $V_i$ that has the maximum cost with currently assigned robots
15:         Equally space all the robots on $V_i$ along with robot $k$ to get $W_k$
16:     end for
17: end if
```

Proposition 3.5.2. *Given an instance of Problem 3.5.1, Algorithm 3 constructs R walks such that the maximum weighted latency of the graph is not more than $(\frac{8 \log \rho}{R} + 10)OPT_1$ if $R \leq \log \rho$ and $3OPT_1 / \lfloor R / \lceil \log \rho \rceil \rfloor$ otherwise, where $\rho = \max \frac{\phi(v_i)}{\phi(v_j)}$ and OPT_1 is the maximum weighted latency of the single optimal walk.*

Proof. The maximum vertex weight in the subgraph G_j constructed at Line 7 of the algorithm will be at most $1/(2^{\frac{j-1}{R} \log \rho})$, whereas the minimum vertex weight in G_j will be at least $1/(2^{\frac{j}{R}\log\rho})$. Hence the ratio of the maximum to minimum vertex weights in G_j will be at most $\rho_j = 2^{\frac{\log\rho}{R}}$. Therefore, the approximation algorithm for one robot will return a walk W_j such that the maximum weighted latency of W_j will not be more than $(8\log\rho_j + 10)\mathtt{OPT}_1^j$. Moreover, $\mathtt{OPT}_1^j \leq \mathtt{OPT}_1$ and hence if $R < \log\rho$, the maximum weighted latency will be at most $(\frac{8\log\rho}{R} + 10)\mathtt{OPT}_1$.

The TSP tour of V_i is an optimal solution when all the vertex weights in V_i are equal. Since the vertex weights within V_i differ by a factor of 2 at most, and the approximation factor of TSP tour in metric graphs is 3/2, the maximum weighted latency of the TSP tour will be at most $3\mathtt{OPT}_1$. Equally spacing $\lfloor R/\lceil\log\rho\rceil\rfloor$ will decrease the latency of all the vertices by a factor of $\lfloor R/\lceil\log\rho\rceil\rfloor$. □

Note that Algorithm 3 bounds the cost of the solution by a function of the optimal cost of a single robot. This algorithm shows that R robots can asymptotically decrease the weighted latency given by a single walk by a factor of R, which is not straightforward for this problem as discussed in Section 3.2.1. A relation between $\mathtt{OPT}_1$ and the optimal weighted latency could result in an approximation ratio for Algorithm 3, however, we were not able to establish such a relation.

Now we show that if there is an approximation algorithm for Problem 3.5.1, it can be used to solve Problem 3.2.2 using the optimal number of robots but with the latency constraints relaxed by a factor α. This is referred to as a $(\alpha, 1)$-bi-criterion algorithm [48] for Problem 3.2.2.

Proposition 3.5.3. *If there exists an α-approximation algorithm for Problem 3.5.1, then there exists a $(\alpha, 1)$-bi-criterion approximation algorithm for Problem 3.2.2.*

We will need the following lemma relating the two problems to prove the proposition. Given an instance of the decision version of Problem 3.2.2 with R robots, let us define an instance of Problem 3.5.1 by assigning $\phi(v) = \frac{r_{\mathtt{min}}}{r(v)}, \forall v \in V$, where $r_{\mathtt{min}} = \min_v r(v)$.

Lemma 3.5.4. *An instance of the decision version of Problem 3.2.2 is feasible if and only if the optimal maximum weighted latency is at most $r_{\mathtt{min}}$ for the corresponding instance of Problem 3.5.1.*

Proof. If the optimal set of walks $\mathcal{W}$ has a cost more than $r_{\mathtt{min}}$, then $L(\mathcal{W}, v) > r_{\mathtt{min}}/\phi(v) = r(v)$ for some vertex v. Hence the latency constraint for that vertex is not satisfied and the set of walks $\mathcal{W}$ is not feasible.

If the optimal set of walks $\mathcal{W}$ has a cost at most $r_{\mathtt{min}}$, then $L(\mathcal{W}, v)\phi(v) \leq r_{\mathtt{min}}$ for all v. Hence, $L(\mathcal{W}, v) \leq r_{\mathtt{min}}/\phi(v) = r(v)$. So, the latency constraints are satisfied for all vertices and $\mathcal{W}$ is feasible. □

Proof of Proposition 3.5.3. If a problem instance of Problem 3.2.2 with R robots is feasible, then by Lemma 3.5.4 the optimal set of walks has a cost at most $r_{\mathtt{min}}$. The α-approximation algorithm for the corresponding Problem 3.5.1 will return a set of walks $\mathcal{W}$ with a cost no more than $\alpha r_{\mathtt{min}}$. Hence, $L(\mathcal{W}, v) \leq \alpha r_{\mathtt{min}}/\phi(v) = \alpha r(v)$, for all v.

Hence, we can use binary search to find the minimum number of robots for which the α-approximation algorithm for the corresponding Problem 3.5.1 results in a latency at most $\alpha r(v)$ for all v. This will be the minimum number of robots for which the problem is feasible. □

3.6 Simulation Results

We now present the empirical performance of the algorithms presented in this chapter. For the approximation algorithm, we used the LKH implementation [43] to find the TSP of the graphs instead of the Christofides approximation algorithm [25]. This results in the loss of approximation guarantee but gives better results in practice. The orienteering paths in Algorithm 2 were found using the ILP formulation from [59] and the ILP's were solved using the Gurobi solver [41].

3.6.1 Patrolling an Environment

The graphs for the problem instances were generated randomly in a real world environment. The scenario represents a ground robot monitoring the University of Waterloo campus. Vertices around the campus buildings represent the locations to be monitored and a complete weighted graph was created by generating a probabilistic roadmap to find paths between those vertices. Figure 3.5 shows the patrolling environment. To generate random problem instances of different sizes, n random vertices were chosen from the original graph. The latency constraints were generated uniformly randomly between TSP$/k$ and kTSP where k was chosen randomly between 4 and 8 for each instance. Here TSP represents the TSP length of the graph found using LKH.

For each graph size, 10 random instances were created. The average runtimes of the algorithms are presented in Figure 3.6. The runtimes for the RecursiveGreedy algorithm are not shown in the figure as the its run times were very similar to the Greedy algorithm and the slight difference in run times does not depict the efficiency of the algorithms. As expected, Algorithm 2 is considerably slower than the approximation and simple greedy

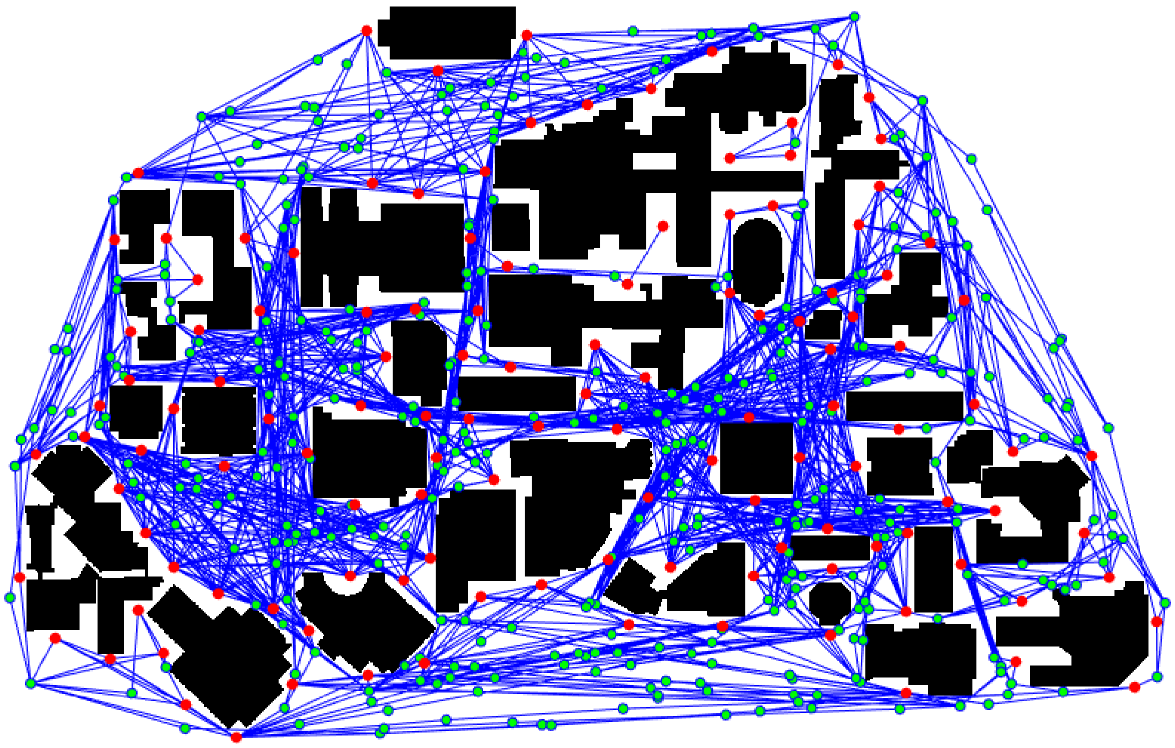


Figure 3.5: The environment used for generation of random instances. The red dots represent the vertices to be monitored and the green dots represent the vertices in the Probabilistic Roadmap used to find shortest paths between red vertices.

algorithms due to multiple calls to the ILP solver. However, as shown in Figure 3.7, Algorithm 2 also gives the lowest number of robots required for most of these instances. The trend of the number of robots returned by the greedy algorithm, the recursive greedy algorithm and the orienteering based greedy algorithm shows that the idea of visiting more vertices on the way to the greedily picked vertex works well in practice.

3.6.2 Persistent 3D Scene Reconstruction

Another application of our algorithms is in capturing images for 3D reconstruction of a scene. Since existing algorithms focus on computing robot paths to map a static scene [13,

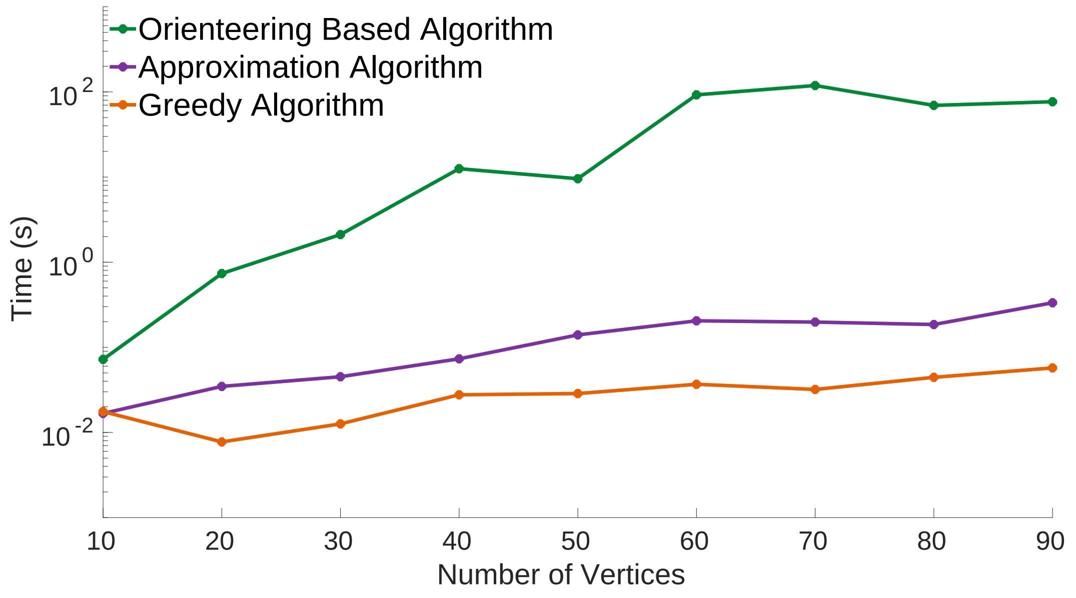


Figure 3.6: Average runtimes of the algorithms. The line plot shows the mean over 10 random instances for each graph size.

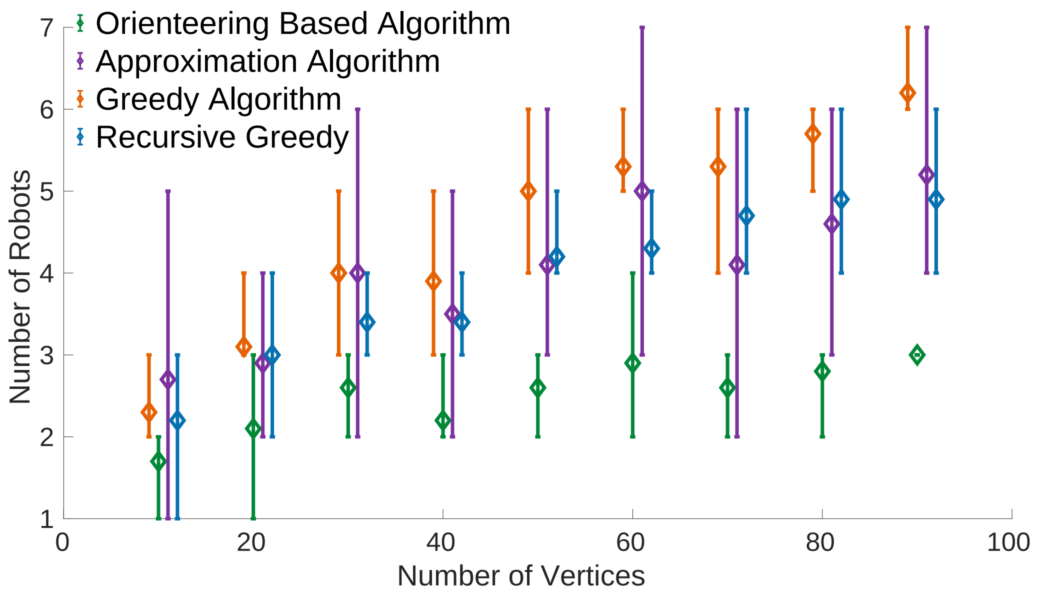


Figure 3.7: Average number of robots returned by each algorithm. The marker shows the mean over 10 random instances for each graph size. The error bars show the minimum and maximum number of robots required for a graph size.

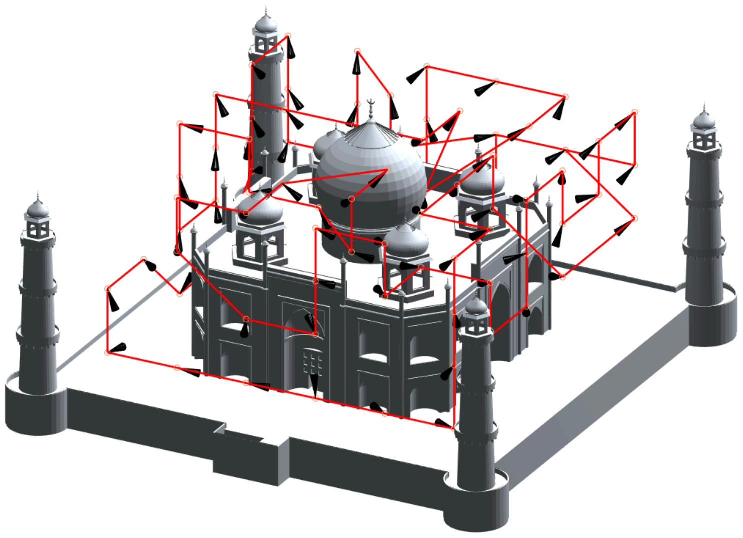
84], our algorithms could be applied to persistently monitor and thus maintain an up-to-date reconstruction of a scene that changes over time. To demonstrate this, we create problem instances using a method similar to [84]. The viewpoints were generated on a grid around the building to be monitored. For each viewpoint, five camera angles were randomly generated, and best angle was selected for each viewpoint based on a view score that was calculated assuming a square footprint for the camera. For each camera angle, equally placed rays were projected onto the building within the footprint and a score was calculated based on the distance and incidence angle of the ray. This calculation is similar to that in [84], although they used a more detailed hemisphere coverage model.

After selecting the viewing angles, the final score of a camera pose was evaluated as in [84] by greedily picking the best viewpoint first and evaluating the marginal score of other viewpoints. The resulting graph had 109 vertices. The latencies were set such that the most informative viewpoint is visited every 8 minutes and on average each viewpoint is visited every 50 minutes. Algorithm 2 found a solution in 150 seconds using two walks, as shown in Figure 3.8. Note that Algorithm 1 returned a solution with 3 robots.

3.6.3 Comparison with Existing Algorithms in Literature

In [30, 31] the authors propose an SMT (Satisfiability Modulo Theory) based approach using the Z3 solver [29] to solve the decision version of the problem. The idea is to fix an upper bound on the period of the solution and model the problem as a constraint program. The authors also provide benchmark instances for the decision version of the problem. We tested our algorithms on the benchmark instances provided and compare the results to the SMT based solver provided by [31].

Out of 300 benchmark instances, given a time limit of 10 minutes, the Z3 solver returned 182 instances as satisfiable with the given number of robots. We ran our algorithms for each instance and checked if the number of robots returned is less than or equal to the number of robots in the benchmark instance. The approximation algorithm satisfied 170 instances whereas Algorithm 2 satisfied 178 instances. The four satisfiable instances that Algorithm 2 was unable to satisfy had optimal solutions where the walks share the vertices, and Algorithm 2 returned one more robot than the optimal in all those instances. The drawback of using the constraint program to solve the problem is its poor scalability. It spent an average of 3.76 seconds on satisfiable instances whereas Algorithm 2 spent 3 ms on those instances on average. Moreover, on one such instance where Algorithm 2 returned one more robot than the Z3 solver, the Z3 solver spent 194 seconds as compared to ~ 5 ms for Algorithm 2. Note that these differences are for benchmark instances having up to 7



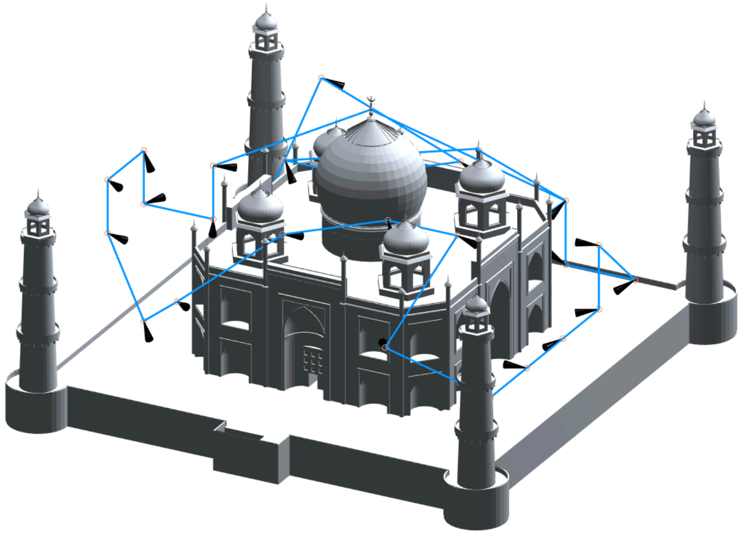


Figure 3.8: The walks returned by Algorithm 2 to continually monitor the mausoleum of the Taj Mahal. The cones at each viewpoint show the camera angle. Note that the walk on the top is not a tour, as it visits the vertex with least latency twice within a period. The walk on the bottom is a tour and it visits the vertices that the first robot was unable to cover.

vertices. As shown in Figure 3.6, Algorithm 2 takes $\sim$ 100 seconds for 90 vertex instances whereas we were unable to solve instances with even 15 vertices within an hour using the Z3 solver. Hence, the scalability of the Z3 based solver hinders its use for problem instances of practical sizes.

3.6.4 Patrolling Application

As shown in [6], one possible application of the problem is patrolling to control crime. The patrolling agents' objective is to persistently visit the locations in an area to deter crime. The locations that are more prone to criminal activities should be visited more often and hence the duration between visits to such locations should be less as compared to that of a safer location. In this section, we construct a problem instance using real world locations and data, and solve the problem using Algorithm 2.

For our example, we consider a region of downtown Toronto to be patrolled. For the input graph to the problem, 42 major intersections in the area are considered as vertices and driving times between them were queried from the Google maps API. The crime data for each of the intersections was taken from Toronto Police Service Public Safety Data Portal [1]. The data provides major crime indicators for each intersection in the area. We used the crime indicators for the calendar year 2017. For each vertex, the crimes for the intersections near that vertex were accumulated to find the crime index of that vertex. This data was then converted to a latency constraint such that the latency constraints were inversely proportional to the crime index and 150 crimes a year for a vertex translates to a latency constraint of 1 hour. The map of the area to be patrolled, along with the vertices and the latency constraints is shown in Figure 3.9.

The patrolling walks returned by the Algorithm 2 are also shown in Figure 3.9. The algorithm returns two walks and hence two patrolling agents are required to patrol the given area. Note that if patrolling agents are placed at equal distance on a TSP path of the vertices, three agents will be required to satisfy all the constraints. The approximation algorithm also returned three walks.

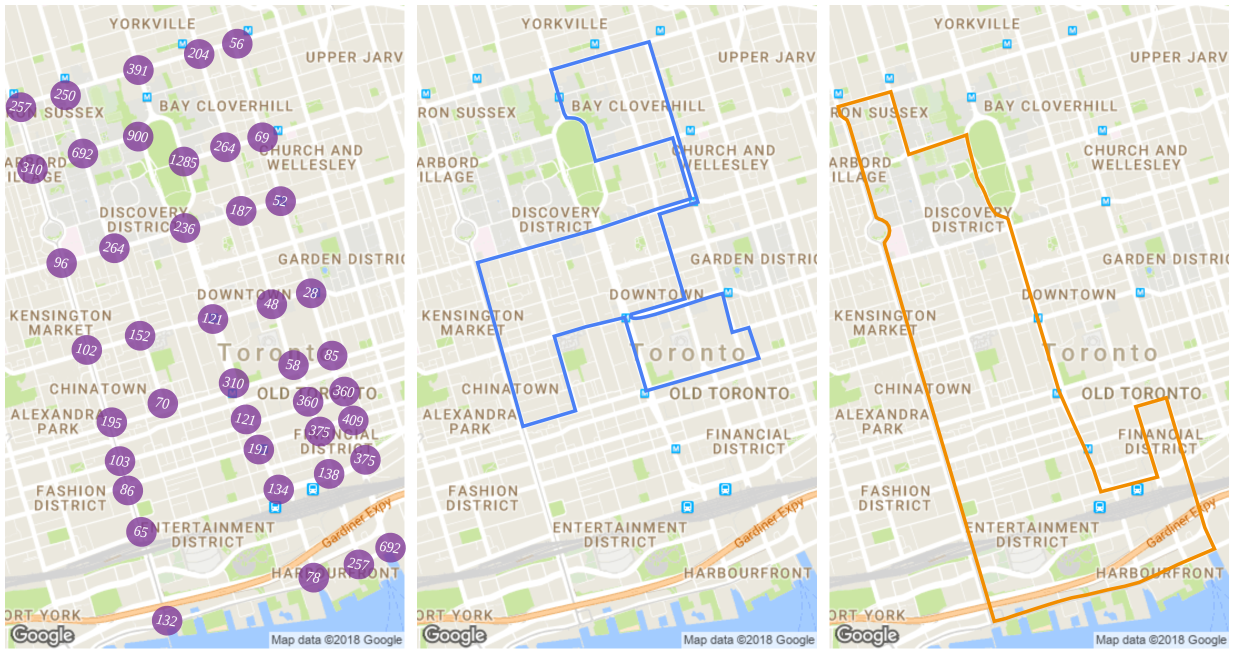


Figure 3.9: Patrolling in downtown Toronto. The figure on left shows the vertices with latency constraints in minutes and the right two figures show the walks returned by Algorithm 2.

Chapter 4

Multi-Robot Stochastic Patrolling

In this chapter we discuss the persistent monitoring problem when the number of robots is fixed. Due to the constraint on the number of robots used, all the events appearing in the environment may not be observed by the team of robots. Moreover, to define the latency constraints used in the previous chapter, the event durations must be fixed. However, in certain scenarios, the events that arrive at a location may not always stay there for a fixed amount of time. For example, if the monitoring task is to observe wildlife at a certain location, then we might have a distribution of staying times of the animals arriving there, but we will not have an exact latency constraint. In these scenarios, a measure of how well the monitoring task is carried out the expected number of observed events by the team of robots. In this chapter we focus on the problem of maximizing the expected value of the observed events, when the distribution of event duration for each location is given.

The organization and contributions of this chapter are as follows. We review the literature related to this problem in Section 4.1. The problem is formally defined and Markov chain based random patrolling policies are motivated in Section 4.2. In Section 4.4 we characterize the hardness of the problem and show that the objective function is submodular. In Section 4.3 the expected reward of observed events is evaluated as a function of the transition matrices of the Markov chains used for patrolling. In Section 4.5 we provide an approximation algorithm for the single robot problem when the event durations are the same for all the vertices and using submodularity extend this algorithm to an approximation algorithm for the multi-robot problem. We propose an algorithm for general problem instances in Section 4.6 and give a distributed algorithm to solve the problem in Section 4.7. In Section 4.8 we consider adversarial events that can choose where and when to appear in order minimize the expected reward of the patrolling team. Finally we give simulation results in Section 4.9

4.1 Related Work

The robotic surveillance problem is very well studied and there is a substantial amount of work related to design of patrolling policies for monitoring of environments. For deterministic paths, there is a breadth of work considering both single and multiple robots [88, 81, 95, 24]. In [95] the authors consider cyclic policies to detect randomly arriving events in an environment. Partitioned and Cyclic policies for multiple robots are compared in [24]. The problem discussed in this chapter is different from these works as we consider events that stay at locations for a certain amount of time. Moreover, we also consider adversarial events.

We use random patrolling paths to monitor environments in this chapter. Random paths have also been extensively studied in the literature. In [89] the authors advocate random patrolling paths by showing that it is hard to find deterministic strategies that satisfy the surveillance criterion of visiting vertices proportional to their importance. In [38] the authors consider stochastic surveillance paths so that the intruders cannot easily exploit the predictability of a deterministic path. In [4] the authors look at intelligent intruders in a perimeter patrolling problem. We deal with a general version of the problem as we deal with environments represented as graphs, for which border patrol is a special case. In [19] the authors find maximum entropy random walks such that all the paths of length t with the same endpoints have equal probability of being traversed. In [55] the authors provide a strategy for the attacker that disables sensors in a sensor selection problem and then they give a resilient sensor selection method for the adversarial setting. Our approach is similar in the sense that we also consider events that act adversarially.

Markov chains are often used to model random paths on a graph [89, 38, 78]. For example, [78] represent the environment as a graph and design Markov chains that generate randomized paths with desired behaviors. They use semidefinite programming to minimize the mean first passage time from one state of the Markov chain to any other state, which is constant for a given chain and is called the Kemeny constant [51]. The problem can be formulated as a semidefinite program for reversible Markov chains only. They also use the Markov chains on weighted graphs, where the probability of traversing an edge is independent of the weight of that edge in the graph. This formulation is more useful in the case of surveillance since the environment is usually represented as a weighted graph and the Markov chain is then defined on that graph. The problem of finding a Markov chain for a single robot to detect events is also studied in [12]. However, they consider fixed event durations only and the events can only arrive when the robot is at a vertex. We consider distributed event durations, and we allow for the events to arrive at any time.

In [15] the authors design a Markov chain with fastest mixing time which approaches the steady state distribution as quickly as possible. They also formulate the problem as a semidefinite problem. A finite set of Markov chains is used to formulate the problem as a *Bayesian Stackelberg Game* in [5], where the patroller first picks a mixed optimal strategy and the adversary then picks the region to attack to maximize its payoff.

4.2 Problem Statement

A team of m robots is patrolling an environment to detect events appearing at certain locations of the environment. As in the previous chapter, the environment is represented by a directed weighted graph $G = (V, E)$[1] where the set of vertices $V = \{1, 2, \ldots, n\}$ represents the locations to be monitored and the set of edges E represents the links using which the robots can travel between the vertices. The notation ℓ_{ij} or ℓ_e is used to represent integer travel time on edge $e = (i, j) \in E$. Each vertex of the graph $i \in V$ also has a weight associated with it given by ϕ_i and it represents the reward for detecting an event at that vertex. Once an event appears at a vertex i, say at time t_1, it stays there for $L_i \in \mathbb{R}_+$ amount of time. The event can only be observed by the robots if one of the robots visits the vertex i within time $[t_1, t_1 + L_i]$. Otherwise the event is missed by the robots. The time L_i is not known and is drawn from a known probability distribution, i.e., $L_i \sim g_i(l_i)$. Note that different vertices may have different distributions for the event duration time L_i. The probability of an event arriving at vertex i is given by ψ_i. Also note that the robots do not know if an event has arrived at a vertex unless one of the robots visits that vertex. The objective for the team of robots is to maximize the expected reward of observed events.

4.2.1 Patrolling Policies

In [85], the authors empirically show that introducing randomness into Travelling Salesman based tours of a graph can increase the probability of detecting the events if the events are adversarial, i.e., the events have information about the patrolling paths and they can use this information to appear at vertices of the graph to increase their chances of remaining undetected. This observation is intuitive because it is easier for adversarial events to avoid

[1]Note that the problem can be defined for heterogeneous robots, where each robot r is given a different edge set E_r. That can be useful when we have different types of robots, e.g, aerial and ground robots working together to patrol the environment. In such cases the edge set for an aerial robot will be different from that of a ground robot. The algorithms proposed in this chapter can be used for the case of heterogeneous robots. To keep the analysis and notation easy to follow, we will assume the same edge set E for all the robots in the rest of this chapter.

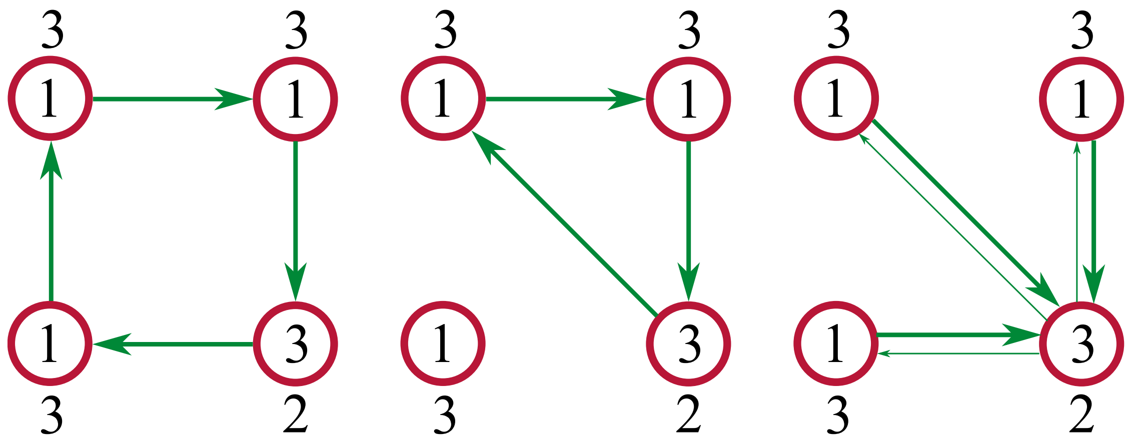


Figure 4.1: The vertex weights are written inside the vertices, and the event duration for a vertex is written beside that vertex. The expected rewards of the two deterministic paths shown visiting four and three vertices are 0.9375 and 1 respectively. The random path depicted at right has an expected reward of 1.083.

a deterministic path. However, in their experiments the vertices of the graph did not have vertex weights and the event duration was same for all the vertices. Due to this restriction, the best solution found in their experiments for random, non-adversarial events was the TSP cycle. We now provide a simple example to show that even when the events are random, difference in vertex weights and duration of events can lead to random solutions performing better as compared to simple cyclic solutions.

Example 4.2.1. *Consider the graph with four vertices as shown in Figure 4.1, where the distance between each pair of vertices is one. The vertex weight ϕ_i for each vertex i is written inside that vertex, and the fixed duration of event for each vertex is written beside that vertex. The probability of an event arriving at any vertex is* $1/4$. *The cyclic TSP solution gives an expected reward of* 0.9375 *whereas the deterministic solution visiting three vertices gives an expected reward of* 1. *Also note that any other deterministic cycle covering any two or three vertices will give* ≤ 1 *reward. Now consider the random patrolling path, in which if the robot is at the vertex with vertex weight* 3, *it goes to one of its neighbors with equal probability, and then comes back to that vertex. This random path gives an expected reward of* 1.083 *which is* 8.3% *improvement over simple cyclic deterministic solutions.*

Therefore, we consider possibly random patrolling policies in this chapter, and we use Markov chains to represent those patrolling policies. The Markov chain for a single robot will define the probability of the robot traversing an edge given a certain history of the path. The length of the history considered depends on the order of the Markov chain. Note that simple deterministic cycles can also be written as Markov chains.

In general, a Markov chain based policy can have a memory of order $t \in \mathbb{Z}_+$. Increasing the value of t cannot make the policy worse as the robot will have more information about its path history to make the next decision. In the problems of this form studied in game theory literature, there exists a maximum value $\bar{t}$ for t, such that the objective value does not increase anymore for $t \geq \bar{t}$ [12, 36]. However the value of $\bar{t}$ can be very large as shown in the following result that gives a lower bound on $\bar{t}$ for a set of problem instances.

Proposition 4.2.2. *There exists a family of instances $\mathcal{I}_k$ for $k > 0$ with $O(k^2 \ln k)$ vertices such that the minimum required order for the Markov chain of the optimal policy is $\Omega(k)$.*

Proof. Consider a single robot problem and the graph shown in Figure 4.2. This example is from [44] and the description and solution are reproduced here for completeness. Let the weight of each vertex be one. The probability of an event arriving at any vertex is same. The length of each edge shown without an edge length written beside is one. In the graph, there are k sections, and each section x has p_x branches where p_x is the x-th prime number. The event duration for each vertex is written inside that vertex. Here $T = 4k$. The optimal solution that observes all possible events, must be a deterministic solution that goes from v_t through all the sections to v_b to v_m and to v_t again, or the same path traversed in opposite direction. The robot must know what branch it took to traverse each section last time to decide which branch to take next. Since there are k sections, the order of the Markov chain that defines this deterministic policy is $\Omega(k)$. □

A Markov chain of order k will require n^k memory to store all the possible histories in general for a n vertex graph making the problem intractable. Therefore, we will consider Markov chains of order one in this chapter. Previous work dealing with Markov chain based patrolling policies also considered Markov chains of order one [12, 78, 10].

So far we have considered the policies for a single robot. In the case of multiple robots, the next vertex of a robot r, may depend on the current vertices of all the other robots. Even if the graph was unweighted or all edges in the graph had length one, the number of states required to represent such a policy is n^m where each state gives the current position of all the robots. Apart from being computationally expensive, such a policy requires coordination and communication among the robots which may not be practically desirable, specially in the presence of adversarial events. Hence, we only consider the policies where the robots are independent, and the next vertex of robot r depends only on the current vertex of robot r. The Markov chain for robot r will be represented using its transition

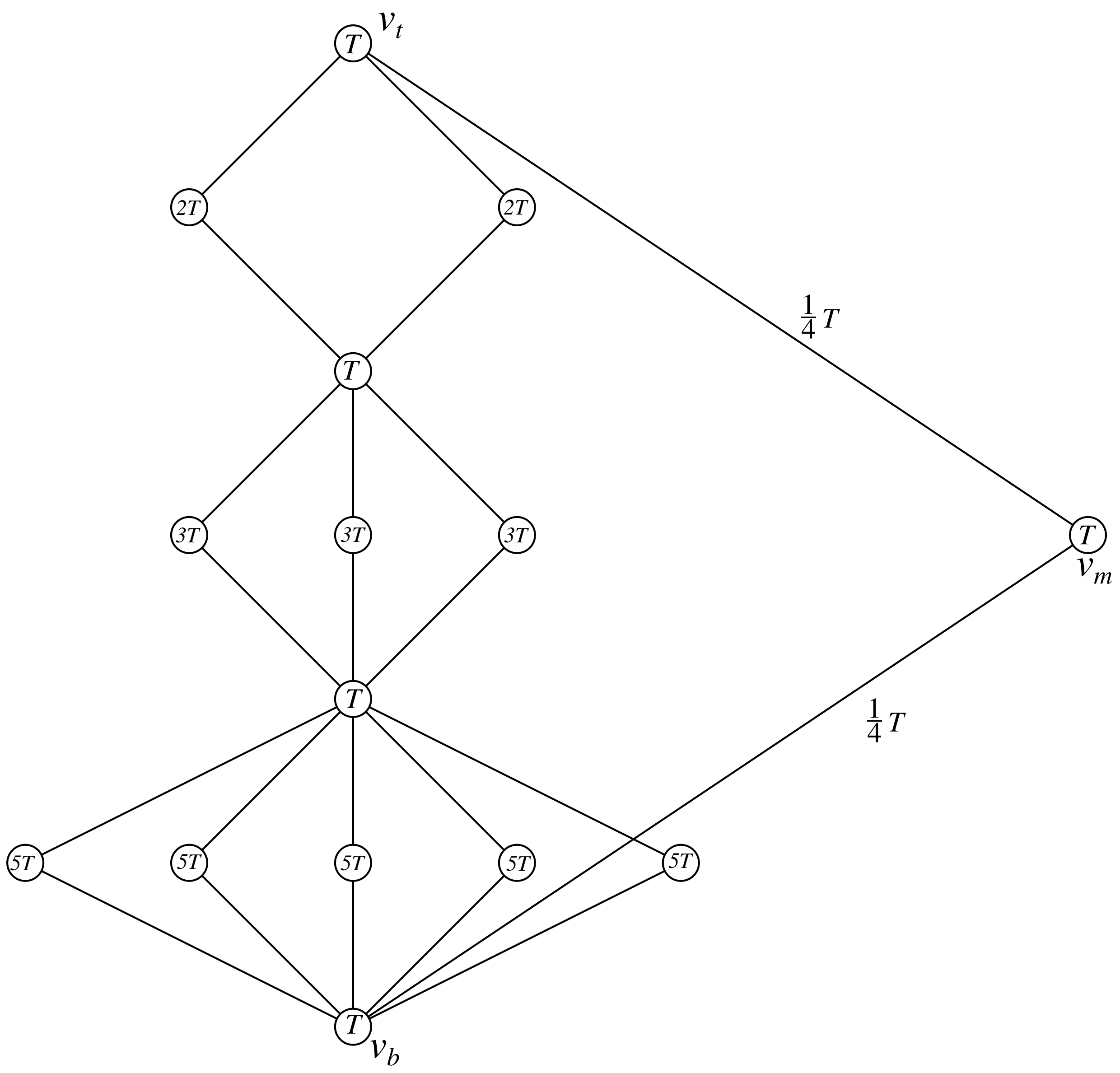


Figure 4.2: The instance $\mathcal{I}_3$ [44]. Here $T = 4k = 12$.

matrix P_r that governs the random path of the robot. The probability of the robot r going from vertex i to vertex j is given by $P_r(i,j)$. Hence, the goal of the problem is to find the set of patrolling Markov chains $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$ to maximize the expected weight of observed events. We formalize the objective function in the next section.

4.2.2 Objective Function

Given the Markov chain transition matrix P_r for robot r, let $S_r(e,j)$ represent the probability of an event missed (not observed) by the robot r, given that the event appeared at vertex j when the robot r was traversing edge $e \in E$. Also let $q_r(e)$ be the probability of robot r being on edge $e \in E$ during its patrol when an event arrives on some vertex. Note that $S_r(e,j)$ and $q_r(e)$ are functions of P_r, and since the Markov chains of the robots are independent, these probabilities do not depend on the Markov chains of other robots. We will calculate these probabilities in Section 4.3. Using the law of total probability, the probability of robot r not observing an event at j is given by

$$\mathbb{P}\left[\texttt{not observed at } j \texttt{ by } r\right] = \sum_{e \in E} q_r(e) S_r(e,j).$$

To calculate the probability of the event being observed at vertex j by the team of robots, we note that the only scenario of an event not being observed at vertex j is if none of the r robots observes that event. Since $S_r(e,j)$ and $q_r(e)$ only depend on P_r, and since Markov chains of the robots are independent, the probability of robot r not observing an event at j is independent of the probability of robot u not observing that event. Hence, the probability of an event at j not being observed is

$$\mathbb{P}\left[\texttt{not observed at } j\right] = \prod_{r=1}^{m} \Big(\sum_{e \in E} q_r(e) S_r(e,j) \Big). \tag{4.1}$$

Since the probability of an event appearing at vertex j is given by ψ_j, the expected reward for the patrolling policy $\mathcal{P}$ is given by

$$\begin{aligned} f(\mathcal{P}) &= \sum_{j=1}^{n} \phi_j \mathbb{P}\left[\texttt{event at } j\right] \mathbb{P}\left[\texttt{observed at } j\right] \\ &= \sum_{j=1}^{n} \phi_j \psi_j \mathbb{P}\left[\texttt{observed at } j\right]. \end{aligned} \tag{4.2}$$

Now we can use the complement of the probability in Equation (4.1) to get

$$
\begin{aligned}
f(\mathcal{P}) &= \sum_{j=1}^{n} \phi_j \psi_j \Big(1 - \prod_{r=1}^{m} \big(\sum_{e \in E} q_r(e) S_r(e,j)\big)\Big). \\
&= \boldsymbol{\phi}^T \boldsymbol{\psi} - (\bigodot_{r=1}^{m} (\boldsymbol{q_r}^T S_r))(\boldsymbol{\phi} \odot \boldsymbol{\psi}),
\end{aligned}
\tag{4.3}
$$

where $\bigodot$ represents element-wise product of vectors. This objective function gives the expected reward of successfully observed events for the patrolling policy $\mathcal{P}$.

4.3 Evaluating the Objective Function

In this section we will calculate the probabilities $S_r(e,j)$ (the probability of robot r missing an event that arrived on vertex j when the robot was traversing edge e) and $q_r(e)$ (the probability of robot r traversing edge e) discussed in the previous section, that are used in the evaluation of the objective function (4.3). Given the graph $G = (V, E)$ with edge lengths ℓ_e for edge e and the Markov chain transition matrix P_r for robot r, the probability mass function of the time steps taken for the first visit from vertex i to vertex j by robot r is given by [12, 10]

$$
F_{t,r}(i,j) = \begin{cases} P_r(i,j)\mathbf{1}_1(\ell_{ij}), & \text{if } t = 1 \\ \sum\limits_{h \neq j} P_r(i,h) F_{t-\ell_{ih},r}(h,j) + P_r(i,j)\mathbf{1}_t(\ell_{ij}), & \text{if } t \geq 2 \end{cases}
\tag{4.4}
$$

where $F_{t,r}(i,j) = 0$ for non-positive values of t, and $\mathbf{1}_t(\ell)$ is an indicator function which is equal to one if $\ell = t$ and 0 otherwise. $F_{t,r}(i,j)$ gives the probability of the robot r visiting vertex j for the first time from vertex i in exactly t time steps. We can use $F_{t,r}(i,j)$ to calculate the probability of observing an event of duration L at vertex j, given that the event arrives when the robot was at vertex i as follows.

Lemma 4.3.1. *If the duration of an event at vertex j is $L \in \mathbb{R}_+$, then the probability of robot r observing that event, given that the event appeared when robot r was at vertex i is $\sum_{t=1}^{\lfloor L \rfloor} F_{t,r}(i,j)$.*

Proof. Although the duration of the event may not be an integer, the edge weights being integer means that a robot starting from vertex i will reach vertex j only at integer times.

Therefore it will be able to observe the event at vertex j if it takes time less than or equal to L to reach vertex j. Since $L \in [\lfloor L \rfloor, \lfloor L \rfloor + 1)$, and since going from i to j for the first time in exactly t_1 steps, and going from i to j for first time in exactly $t_2 \neq t_1$ steps are mutually exclusive occurances, we can add the probabilities $F_{t,r}(i,j)$ for $t = 1$ to $\lfloor L \rfloor$ to find the probability of observing the event. □

An event may arrive when the robot is traversing an edge instead of being located at a specific vertex. In fact, in our patrolling model, the robot will spend all of its time on edges, moving between vertices, because if the robot has to stay at a vertex i for t time units, it can be modeled as transitioning along (i,i), a self-looping edge, repeatedly t times with $w_{ii} = 1$. Let $\bar{S}_r(e,j|L)$ represent the probability of robot r observing an event of duration L that arrived on vertex j when the robot was traversing the edge e. Since the duration of event at vertex j is distributed according to the probability distribution g_j, we can write the probability of robot r observing the event using the law of total probability as

$$\bar{S}_r(e,j) = \int_0^{L_{max}} \bar{S}_r(e,j|L) g_j(L) dL, \tag{4.5}$$

where the duration of event at vertex j is distributed between 0 and L_{max} according to the distribution g_j. Note that, without loss of generality, we can assume L_{max} to be an integer. This probability can be evaluated using the following result.

Lemma 4.3.2. *If an event arrived at vertex j when the robot r was traversing edge $e = (h,i)$, then the probability of the robot r observing that event can be evaluated using the following expression:*

$$\bar{S}_r(e,j) = \frac{1}{\ell_e} \sum_{t=1}^{L_{max}-1} F_{t,r}(i,j) \mathcal{H}_t(e,j),$$

where

$$\mathcal{H}_t(e,j) = \begin{cases} \ell_e\left(1 - \int_0^{t+\ell_e} g_j(L)dL\right) + \int_t^{t+\ell_e} (L-t) g_j(L) dL, & \text{if } t < L_{max} - \ell_e \\ \int_t^{L_{max}} (L-t) g_j(L) dL, & \text{otherwise.} \end{cases}$$

We first need the following result to calculate the probability $\bar{S}_r(e,j|L)$.

Lemma 4.3.3. *If the duration of an event at vertex j is L and the event arrived when the robot r was traversing the edge $e = (h,i)$, then the probability of the robot r observing that* *event is given by*

$$\begin{aligned}\bar{S}_r(e,j|L) = &\sum_{t=1}^{\lfloor L-\ell_e \rfloor} F_{t,r}(i,j) + \\ &\frac{1}{\ell_e}\Big[\sum_{t=1}^{\ell_e} F_{\lfloor L-(\ell_e-t)\rfloor,r}(i,j)\cdot\big(\ell_e - t + L - \lfloor L \rfloor\big)\Big].\end{aligned} \tag{4.6}$$

For the ease of notation we will use F_t instead of $F_{t,r}(i,j)$ in the forthcoming proofs to denote the probability of robot r going to vertex j from vertex i in exactly t time steps.

Proof. Suppose that when the event arrives on vertex j, the robot is at a point on edge $e = (h,i)$ that it will take k time to reach vertex i on its way from h to i. When the robot reaches vertex i, the event will stay for another $L-k$ time, and using Lemma 4.3.1, the probability of observing that event is $\sum_{t=1}^{\lfloor L-k \rfloor} F_t$. Since the arrival of the events is independent of the motion of the robot, at the time of arrival of a certain event, the robot can be anywhere on its path. Given that the robot is on edge e when the event arrives, its probability distribution of being k distance away from vertex i will be uniform and since the time taken on edge e is ℓ_e, this probability will be given by $1/\ell_e$. The probability of observing the event is then given by

$$\begin{aligned}\bar{S}_r(e,j|L) &= \int_0^{\ell_e}\sum_{t=1}^{\lfloor L-k \rfloor} F_t \mathbb{P}\left[k \texttt{ away from } i\right] dk \\ &= \int_0^{\ell_e}\sum_{t=1}^{\lfloor L-k \rfloor} F_t \frac{1}{\ell_e} dk.\end{aligned}$$

Let $k_0 = L - \lfloor L \rfloor$, then for $k \in [0, k_0]$, $\sum_{t=1}^{\lfloor L-k \rfloor} F_t = \sum_{t=1}^{\lfloor L \rfloor} F_t$. Similarly, for $k \in (k_0+q, k_0+q+1]$, $q = \{1,\dots,\ell_e - 1\}$, the integrand in the above integral remains constant, and so we can break up the integral as follows:

$$\begin{aligned}\bar{S}_r(e,j|L) =& \frac{1}{\ell_e}\Big(\int_0^{k_0}\sum_{t=1}^{\lfloor L-k \rfloor} F_t dk + \int_{k_0}^{k_0+1}\sum_{t=1}^{\lfloor L-k \rfloor} F_t dk + \ldots + \int_{k_0+\ell_e-1}^{\ell_e}\sum_{t=1}^{\lfloor L-k \rfloor} F_t dk\Big) \\ =& \frac{1}{\ell_e}\Big(\sum_{t=1}^{\lfloor L \rfloor} F_t k_0 + \sum_{t=1}^{\lfloor L-1 \rfloor} F_t + \ldots + \sum_{t=1}^{\lfloor L-(\ell_e-1) \rfloor} F_t + \sum_{t=1}^{\lfloor L-\ell_e \rfloor} F_t (1-k_0)\Big).\end{aligned}$$

Now note that all the summations go from $t = 1$ to $t = \lfloor L - \ell_e \rfloor$ and so we can rewrite the sum as follows:

$$\begin{aligned}\bar{S}_r(e, j|L) =& \frac{1}{\ell_e}\Big(\sum_{t=1}^{\lfloor L-\ell_e \rfloor} F_t(k_0 + \ell_e - 1 + 1 - k_0) + \\ & F_{\lfloor L-\ell_e+1 \rfloor}(\ell_e - 1 + k_0) + \ldots + F_{\lfloor L \rfloor} k_0 \Big) \\ =& \sum_{t=1}^{\lfloor L-\ell_e \rfloor} F_t + \frac{1}{\ell_e}\Big(\sum_{t=1}^{\ell_e} F_{\lfloor L-(\ell_e-t) \rfloor}(\ell_e - t + k_0) \Big).\end{aligned}$$

□

Now we give the proof of Lemma 4.3.2.

Proof of Lemma 4.3.2. We solve the integral in Equation (4.5) for the two summations in Equation (4.6) separately so that the derivation is easier to follow. From the first term we get,

$$\begin{aligned}\int_0^{L_{max}} \sum_{t=1}^{\lfloor L-\ell_e \rfloor} F_t g_j(L) dL =& \int_0^1 \sum_{t=1}^{\lfloor L-\ell_e \rfloor} F_t g_j(L) dL + \int_1^2 \sum_{t=1}^{\lfloor L-\ell_e \rfloor} F_t g_j(L) dL + \ldots \\ & + \int_{L_{max}-1}^{L_{max}} \sum_{t=1}^{\lfloor L-\ell_e \rfloor} F_t g_j(L) dL.\end{aligned}$$

Note that the sum $\sum_{t=1}^{\lfloor L-\ell_e \rfloor} F_t$ remains constant for L between two consecutive integers, because ℓ_e is an integer. Also note that for $L \in (0, 1)$, the integrand will be zero. Hence we can write the above expression as

$$\sum_{t=1}^{1-\ell_e} F_t \int_1^2 g_j(L) dL + \sum_{t=1}^{2-\ell_e} F_t \int_2^3 g_j(L) dL + \ldots + \sum_{t=1}^{L_{max}-1-\ell_e} F_t \int_{L_{max}-1}^{L_{max}} g_j(L) dL.$$

Since each summation in the above expression goes at least up to $t = 1 - \ell_e$, we can take

$\sum_{t=1}^{1-\ell_e} F_t$ common to get

$$\begin{aligned}
&\sum_{t=1}^{1-\ell_e} F_t\Big(1-\int_0^1 g_j(L)dL\Big)+F_{2-\ell_e}\Big(1-\int_0^2 g_j(L)dL\Big)+ \\
&\ldots+F_{L_{max}-1-\ell_e}\Big(1-\int_0^{L_{max}-1} g_j(L)dL\Big) \\
=&\sum_{t=1}^{1-\ell_e} F_t\bar{\mathcal{G}}_j(1)+F_{2-\ell_e}\bar{\mathcal{G}}_j(2)+\ldots+F_{L_{max}-1-\ell_e}\bar{\mathcal{G}}_j(L_{max}-1) \\
=&\sum_{t=1}^{1-\ell_e} F_t\bar{\mathcal{G}}_j(1)+\sum_{t=2}^{L_{max}-1} F_{t-\ell_e}\bar{\mathcal{G}}_j(t) \\
=&\sum_{t=2}^{L_{max}-1} F_{t-\ell_e}\bar{\mathcal{G}}_j(t) \\
=&\sum_{t=1}^{L_{max}-1-\ell_e} F_t\bar{\mathcal{G}}_j(t+\ell_e),
\end{aligned} \tag{4.7}$$

where $\bar{\mathcal{G}}_j(x)=1-\int_0^x g_j(L)dL$ and the second last equality is due to the fact that edge weights are integers and if the robot is spending some time in traversing an edge e, then $\ell_e \geq 1$. For the second term, first consider the following integral.

$$\begin{aligned}
&\int_0^{L_{max}} F_{\lfloor L-\ell_e+t\rfloor}(\ell_e-t+L-\lfloor L\rfloor)g_j(L)dL \\
=&\sum_{y=0}^{L_{max}-1}\int_y^{y+1} F_{\lfloor L-\ell_e+t\rfloor}(\ell_e-t+L-\lfloor L\rfloor)g_j(L)dL \\
=&\sum_{y=0}^{L_{max}-1} F_{y-\ell_e+t}\int_y^{y+1}(\ell_e-t+L-y)g_j(L)dL \\
=&\sum_{y=1}^{L_{max}-1-\ell_e+t} F_y\int_{y+\ell_e-t}^{y+\ell_e-t+1}(L-y)g_j(L)dL
\end{aligned}$$

Using this procedure for the second term in Equation (4.6), we get

$$\sum_{t=1}^{\ell_e}\sum_{y=1}^{L_{max}-1-\ell_e+t} F_y\int_{y+\ell_e-t}^{y+\ell_e-t+1}(L-y)g_j(L)dL.$$

Breaking the summation, that term becomes

$$\sum_{y=1}^{L_{max}-1-\ell_e} F_y\Big(\sum_{t=1}^{\ell_e}\int_{y+\ell_e-t}^{y+\ell_e-t+1}(L-y)g_j(L)dL\Big)+$$
$$\sum_{y=L_{max}-\ell_e}^{L_{max}-1} F_y\Big(\sum_{t=1}^{\ell_e}\int_{y+\ell_e-t}^{y+\ell_e-t+1}(L-y)g_j(L)dL\Big).$$

Combining the sum of integrals and noting that $\int_{L_{max}+k}^{L_{max}+k+1}(L-y)G_j(L)dL = 0$ for $k \geq 0$, we get

$$\sum_{y=1}^{L_{max}-1-\ell_e} F_y\Big(\int_{y}^{y+\ell_e}(L-y)g_j(L)dL\Big)+\sum_{y=L_{max}-\ell_e}^{L_{max}-1} F_y\Big(\int_{y}^{L_{max}}(L-y)g_j(L)dL\Big).$$

Using this expression along with (4.7) in Equation (4.5), we get the result. □

Using this result, the probability of robot r missing an event that arrived at vertex j when the robot r was traversing edge e can be calculated as $S_r(e,j) = 1 - \bar{S}_r(e,j)$. Note that since $F_{t,r}(i,j)$ is a function of P_r, $S_r(e,j)$ is also a function of P_r. Also note that $\mathcal{G}_{t,j}$ is not a function of P_r, and needs to be computed once for all t and j for a given instance of the problem.

The probability $q_r(e)$ used in the objective function (4.3) represents the probability of robot r being on edge e when an event arrives on some vertex. Since the arrival of events is independent of the robots' patrolling paths, sampling the position of robot r at the times of arrival of events is equivalent to randomly sampling the position of robot r. Hence $q_r(e)$ is just the probability of robot r being on edge e. This probability can be calculated using the following result.

Lemma 4.3.4. *If the robot r is patrolling the graph using the Markov chain transition matrix P_r, the probability of the robot traversing edge $e = (i,j)$ is given by*

$$q_r(e) = \hat{\pi}(i)P_r(i,j)\ell_e,$$

where the vector $\hat{\boldsymbol{\pi}} \in \mathbb{R}^{1\times n}$ is the solution to the following set of equations.

$$\sum_{i\in V}\hat{\pi}(i)\Big(\sum_{j}(w_{(i,j)})P_r(i,j)\Big) = 1 \tag{4.8}$$
$$\hat{\boldsymbol{\pi}}P_r = \hat{\boldsymbol{\pi}}$$

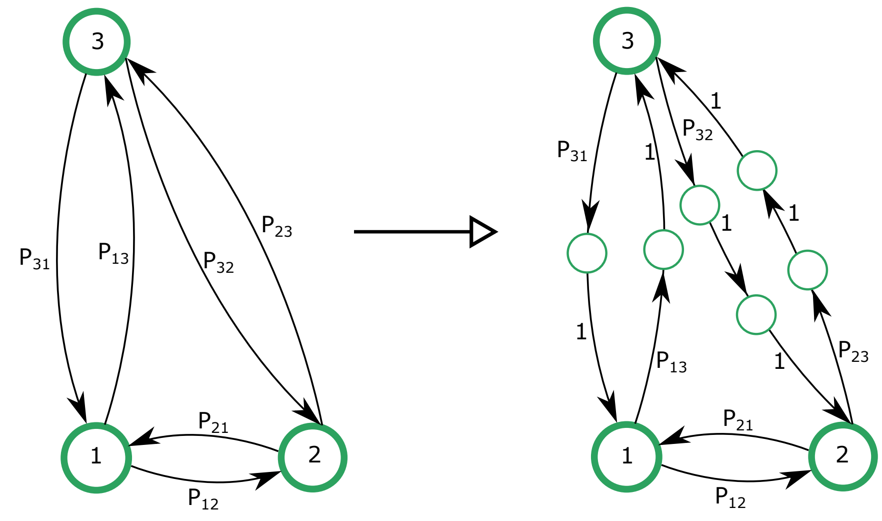


Figure 4.3: Converting an edge weighted graph into an unweighted graph. The transition probabilities for each edge are written beside that edge.

Proof. Since the robot is following a Markov chain with transition matrix P, we can use this transition matrix to find the probability of the robot being on edge e. The probability of a Markov chain making a transition from state i to state j is given by

$$\begin{aligned}\mathbb{P}\left[\text{transition } (i,j)\right] &= \mathbb{P}\left[\text{state } i\right]\mathbb{P}\left[\text{transition } (i,j)|\text{state } i\right]\\ &= \pi(i)P(i,j),\end{aligned}$$

where $\boldsymbol{\pi}$ is the stationary distribution vector of the Markov chain P. If the robot spent same amount of time on each edge, the probability of the robot being on edge (i,j) would be given by $\pi(i)P(i,j)$. However, since the transition times on edges are given by edge weights of the graph, we first convert the edge weighted graph into an unweighted one as follows. For each directed edge $e=(i,j)\in E$, we add ℓ_e-1 dummy vertices $\{d_{e,1},\dots,d_{e,\ell_e-1}\}$ to the new vertex set. The edges $(i,d_{e,1})$, (d_{e,ℓ_e-1},j), and $(d_{e,k},d_{e,k+1})$ for $k\in\{1,\dots,\ell_e-2\}$ replace the weighted edge e. The unweighted graph constructed using this method is denoted as $\hat{G}=(\hat{V},\hat{E})$. The new transition probabilities will be $\hat{P}((i,d_{e,1}))=P(i,j)$, $\hat{P}(d_{e,k},d_{e,k+1})=1$ for $k\in\{1,\dots\ell_e-2\}$, and $\hat{P}(d_{e,\ell_e-1},j)=1$. Note that this new graph $\hat{G}$ with the new Markov chain $\hat{P}$ represents the same motion of the robot as that of Markov chain P on graph G. An example of this conversion is shown in Figure 4.3.

The probability of the robot being on edge $e=(i,j)\in E$ can now be calculated as $\hat{\pi}(i)P(i,j)\ell(i,j)$ since the probability of being on the edge $(d_{e,k},d_{e,k+1})$ is the same as the probability of the robot traversing the edge $(i,d_{e,1})$. This calculation will however require the stationary distribution $\hat{\boldsymbol{\pi}}$ of $\hat{P}$ to be calculated. Since, $\hat{P}$ is a transition matrix of a $n + \sum_{e \in E}(\ell_e - 1)$ state Markov chain, the calculation of $\hat{\boldsymbol{\pi}}$ using $\hat{P}$ can be expensive. We calculate $\hat{\pi}_r$ using a more efficient method as follows.

By the definition of stationary distribution, we have

$$\begin{aligned}\hat{\pi}(j) &= \sum_{i \in \hat{V}} \hat{\pi}(i)\hat{P}(i,j) \\ &= \sum_{i \in V} \hat{\pi}(i)\hat{P}(i,j) + \sum_{i \in \hat{V} \setminus V} \hat{\pi}(i)\hat{P}(i,j).\end{aligned}$$

Notice that if for some $e = (i,j) \in E$, $\ell_e = 1$, then $\hat{P}(i,j) = P(i,j)$. Otherwise, if for some $e = (i,j) \in E$, $\ell_e > 1$, then $e \notin \hat{E}$, making $\hat{P}(i,j) = 0$, and the edge $(d_{e,\ell_e-1}, j) \in \hat{E}$ with $\hat{P}(d_{e,\ell_e-1}, j) = 1$. Also $\hat{\pi}(d_{e,\ell_e-1}) = \hat{\pi}(i)\hat{P}(i,j)$. Therefore, we can write

$$\hat{\pi}(j) = \sum_{i \in V} \hat{\pi}(i)P(i,j).$$

Hence we can write $\hat{\boldsymbol{\pi}}P = \hat{\boldsymbol{\pi}}$ to calculate $\hat{\pi}(j)$ for all $j \in V$. Since $\hat{\boldsymbol{\pi}}$ is the left eigenvector of P with eigenvalue one, any vector $c\hat{\boldsymbol{\pi}}$ for some constant $c \in \mathbb{R}$ is also a solution to $\hat{\boldsymbol{\pi}}P = \hat{\boldsymbol{\pi}}$. We will need the following constraint to uniquely define $\hat{\boldsymbol{\pi}}$.

$$\sum_{i \in \hat{V}} \hat{\pi}(i) = 1.$$

Since, $\hat{\pi}(d_{e,k}) = \hat{\pi}(i)\hat{P}(i,j)$ for $e = (i,j)$ we can write this constraint as

$$\sum_{i \in V} \hat{\pi}(i)\Big(1 + \sum_{j}(\ell(i,j) - 1)P(i,j)\Big) = 1.$$

We can further simplify this by noting that $\sum_j P(i,j) = 1$, and get

$$\sum_{i \in V} \hat{\pi}(i)\Big(\sum_{j}(\ell(i,j))P(i,j)\Big) = 1. \tag{4.9}$$

Therefore, given a transition matrix P, we use $\hat{\boldsymbol{\pi}}P = \hat{\boldsymbol{\pi}}$ with Equation (4.9) to find $\hat{\pi}(i)$ for $i \in V$, and then find the probability of the robot being on edge $e = (i,j)$ using

$$q(e) = \hat{\pi}(i)P(i,j)\ell_e.$$

□

Note that the equation $\hat{\boldsymbol{\pi}} P_r = \hat{\boldsymbol{\pi}}$ means that $\hat{\boldsymbol{\pi}}$ is the eigenvector of P_r with eigenvalue one. Equation (4.8) normalizes $\hat{\boldsymbol{\pi}}$.

Given Lemmas 4.3.2 and 4.3.4, we can calculate $q_r(e)$ and $S_r(e, j)$ given the transition matrix P_r, and hence we can evaluate the objective function $f(\mathcal{P})$ given a patrolling policy $\mathcal{P}$. In [10], the probability of an event being detected by a robot was calculated assuming that the events can only appear when the robot is at some vertex. Moreover, the duration of events was fixed. That probability was a sum of the terms $F_{t,r}(i, j)$ given in Equation (4.4). It is interesting to note that allowing for distributed durations of events, and considering a more general and practical case where events can arrive when the robots are traversing edges, the probability of an event being detected is a weighted sum of the terms $F_{t,r}(i, j)$ as shown in Lemma 4.3.2. Also note that the weights $\mathcal{H}_t(e, j)$ only need to be calculated once as they only depend on the distributions of event durations and the graph, and are independent of the patrolling policy.

4.4 Properties of the Problem

Given the objective function $f(\mathcal{P})$ in Equation (4.3), the patrolling problem is to find the set of patrolling policies $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ to maximize $f(\mathcal{P})$. Each of the Markov chains P_r should be a valid transition matrix that the robot can use for patrolling. This results in the following constrained optimization problem.

$$\begin{aligned}
\text{maximize:} \quad & f(\{P_1, P_2, \ldots, P_m\}) & & \\
\text{subject to:} \quad & \sum_{j=1}^{n} P_r(i, j) = 1, & \forall i, r & \qquad (4.10)\\
& P_r(i, j) \geq 0, & \forall i, j, r & \\
& P_r(i, j) = 0, & \text{if } (i, j) \notin E, \forall r & \\
& P_r \text{ has one communicating class,} & \forall r. &
\end{aligned}$$

The first two constraints make sure that P_r is a valid transition matrix. The third constraint means that the robot will only be able to transition from i to j if there is an edge (i, j) in the graph. The last constraint makes sure that there is only one communicating class and the robot can visit all the vertices in that communicating class. If the communicating class does not include all the vertices in the graph, then the probability of missing an event arriving on the vertices that are not in the communicating class is one.

4.4.1 Computational Complexity

The following result shows that maximizing $f(\mathcal{P})$ is NP-hard even in the case of a single robot.

Proposition 4.4.1. *The Problem (4.10) is NP-hard.*

Proof. We will show the hardness using a reduction from the HAMILTONIAN CYCLE problem. Given a graph $G = (V, E)$ as an instance of HAMILTONIAN CYCLE, we construct an instance of Problem (4.10) as follows. Set the number of robots as $m = 1$, the edge weights as $\ell_e = 1, \forall e \in E$ to get a weighted graph. Set the vertex weights as one for all the vertices, and $g_i(l) = \delta(|V|)$, i.e., the duration of events at each vertex is equal to the number of vertices in the graph. Also let $\psi_i = 1/|V|$ for all $i \in V$.

If the Problem (4.10) returns a solution with objective value less than 1, then there does not exist a Hamiltonian cycle in the graph $G = (V, E)$. That is because if there was a Hamiltonian cycle in the graph, it could be written as a Markov chain, and that Markov chain will return an objective value of 1 because all the events can be observed by a robot following the Hamiltonian cycle of the graph, as the duration of events and the length of the Hamiltonian cycle is $|V|$.

If the Problem (4.10) returns a solution with objective value 1, then there exists a Hamiltonian cycle in the graph $G = (V, E)$, because if there was no Hamiltonian cycle in the graph, there is a non-zero probability of the events being missed and the objective function will be less than 1. □

Remark 4.4.2. *Note that maximizing the objective function $f(\mathcal{P})$ is equivalent to minimizing the function $(\bigodot_{r=1}^{m} \boldsymbol{q_r}^T S_r)(\boldsymbol{\phi} \odot \boldsymbol{\psi})$ because $\boldsymbol{\phi}^T \boldsymbol{\psi}$ is a constant for a given instance. However, if the problem was to minimize this function, then we can show that no polynomial time approximation algorithm exists for this problem unless P=NP. This is because we can use the same reduction as in the proof of Proposition 4.4.1 from the* HAMILTONIAN CYCLE *to show that a Hamiltonian cycle exists if and only if the problem has the objective value* 0. *Since, any approximation algorithm of the problem would also result in the objective value* 0, *this implies that if a polynomial time approximation algorithm exists,* HAMILTONIAN CYCLE *could be solved.*

4.4.2 Non-Convexity

The objective function $f(\mathcal{P})$ is not a convex function in general. The following example demonstrates this.

Example 4.4.3. *Consider a single robot version of the problem ($m = 1$) with $P = P_1$. The functions $S(e, j)$ for all $e \in E$ and $j \in V$ are functions of the problem variables $P(x, y)$ for $x, y \in V$. Now consider a specific instance of the problem with all the edge weights equal to one ($\ell_e = 1, \forall e \in E$), and same duration of event for all the vertices fixed as 3. Then using Equation (4.6), for $e = (h, i)$ we can write*

$$S(e, j) = 1 - F_1(i, j) - F_2(i, j).$$

The elements in the Hessian of $S(e, j)$ will be given by

$$\frac{\partial^2 S(e, j)}{\partial P(x, y)\partial P(k, l)} = \frac{-\partial^2 F_1(i, j)}{\partial P(x, y)\partial P(k, l)} - \frac{\partial^2 F_2(i, j)}{\partial P(x, y)\partial P(k, l)}.$$

Consider edge $e = (h, 1)$ for some neighbor h of vertex 1, and the variables $P(1, 3)$ and $P(3, 2)$ as the first two variables in the indexing of the Hessian of $S(e, 2)$. Then

$$\begin{aligned}\frac{\partial^2 (F_1(1, 2) + F_2(1, 2))}{\partial P(1, 3)\partial P(3, 2)} =& \frac{\partial^2 (P(1, 2) + \sum_{k \neq 2}^{n} P(1, k)P(k, 2))}{\partial P(1, 3)\partial P(3, 2)} \\ =& 1,\end{aligned}$$

and

$$\begin{aligned}\frac{\partial^2 F_1(1, 2)}{\partial P(1, 3)\partial P(1, 3)} + \frac{\partial^2 F_2(1, 2)}{\partial P(1, 3)\partial P(1, 3)} =& \frac{\partial^2 F_1(1, 2)}{\partial P(3, 2)\partial P(3, 2)} + \frac{\partial^2 F_2(1, 2)}{\partial P(3, 2)\partial P(3, 2)} \\ =& 0.\end{aligned}$$

So, the first and the second principal minor of the Hessian of $S(e, 2)$ for $e = (h, 1)$ evaluate to 0 and -1 respectively. Hence $S(e, j)$ is not a convex function of P and hence $f(P)$ is not in general a convex function of P.

Therefore, we cannot use constrained convex optimization to solve the problem. However, the objective function is submodular as shown in the following section.

4.4.3 Submodularity

The function $f(\mathcal{P})$ is a set function that is defined on a subset of the set of Markov chains.

Proposition 4.4.4. *Let the set $\mathcal{C}$ denote the set of all possible Markov chain transition matrices defined on the graph $G = (V, E)$, i.e. $\mathcal{C} = \{P \in \mathbb{R}^{n \times n} | \sum_j P(i,j) = 1, P(i,j) \geq 0, (i,j) \notin E \implies P(i,j) = 0\}$. The objective function (4.3) defined on the subsets of $\mathcal{C}$ has the following properties.*

1. *$f(\{\}) = 0$.*
2. *$A \subseteq B \subset \mathcal{C} \implies f(A) \leq f(B)$.*
3. *$f(A \cup \{P\}) - f(A) \geq f(B \cup \{P\}) - f(B)$, for $A \subseteq B \subset \mathcal{C}$, $P \notin B$.*

Proof. The first two properties are straightforward, since if no robot is monitoring the environment, the expected reward collected will be zero. Also adding more robots cannot decrease the expected reward.

Now consider sets $A = \{P_1, P_2, \ldots, P_a\}$ and $B = \{P_1, P_2, \ldots, P_b\}$ where $b \geq a$. Then for $P_k \notin B$,

$$\begin{aligned}
&f(A \cup \{P_k\}) - f(A) \\
=&(\bigodot_{r=1}^{a} \boldsymbol{q_r}^T S_r)(\boldsymbol{\phi} \odot \boldsymbol{\psi}) - (\bigodot_{r=1}^{a} \boldsymbol{q_r}^T S_r) \odot (\boldsymbol{q_k}^T S_k)(\boldsymbol{\phi} \odot \boldsymbol{\psi}) \\
=&(1 - \boldsymbol{q_k}^T S_k) \odot (\bigodot_{r=1}^{a} \boldsymbol{q_r}^T S_r)(\boldsymbol{\phi} \odot \boldsymbol{\psi}) \\
\geq&(1 - \boldsymbol{q_k}^T S_k) \odot (\bigodot_{r=1}^{a} \boldsymbol{q_r}^T S_r) \odot (\bigodot_{r=a+1}^{b} \boldsymbol{q_r}^T S_r)(\boldsymbol{\phi} \odot \boldsymbol{\psi}) \\
=&(1 - \boldsymbol{q_k}^T S_k) \odot (\bigodot_{r=1}^{b} \boldsymbol{q_r}^T S_r)(\boldsymbol{\phi} \odot \boldsymbol{\psi}) \\
=&f(B \cup \{P_k\}) - f(B).
\end{aligned}$$

□

We use the submodularity of the objective function in the following section to give an approximation algorithm for a certain set of instances of the problem.

4.5 Approximation Algorithm for Fixed Event Duration

In this section we first give an approximation algorithm for the single robot problem when the event durations are fixed to L for all the vertices. Although we are considering the set of all Markov chains for our patrolling policy, the following result shows that a deterministic tour returned by the Orienteering Problem gives an approximate solution to the problem.

Proposition 4.5.1. *Consider an instance $\mathcal{I}$ of Problem 4.10 defined on an undirected metric graph with one robot i.e., $m = 1$ and the duration of event fixed as L for each vertex. Let $\{v_1, v_2, \dots, v_k, v_1\}$ represent an α-approximate solution to the Orienteering Problem on a metric graph $G = (V, E)$ with vertex weight on vertex $i \in V$ given by $\phi_i\psi_i$, and the maximum tour length L. Then the deterministic Markov chain following the orienteering tour ($P(i, j) = 1$ if $(i, j) = (v_i, v_{i+1})$ or $(i, j) = (v_k, v_1)$ and 0 otherwise) will give the objective value $f(P) \geq \alpha f^*/2$, where f^* is the objective value of the optimal solution to $\mathcal{I}$.*

Proof. For $m = 1$, the objective is to maximize $f = \boldsymbol{\phi}^T\boldsymbol{\psi} - (\boldsymbol{q}^T S)(\boldsymbol{\phi} \odot \boldsymbol{\psi})$. In this objective ψ_j is the probability of an event arriving at vertex j, and ϕ_j is the value of detecting an event at the vertex j. Therefore we can get the same objective function by assuming that the probability of an event arriving at a vertex is $1/n$ for all the vertices, and setting the value of detecting an event at vertex j to $n\psi_j\phi_j$. Hence, in this proof, without loss of generality, we assume that $\tilde{\phi}_j = \psi_j\phi_j$.

Now, consider any deterministic tour visiting vertices $U \subset V$ with maximum tour length at most L. Since, the robot follows a deterministic tour of vertices in U with the tour length not greater than the duration of events L, the probability of missing events on the vertices in U is 0, whereas all the events arriving on vertices not in U will be missed, i.e., $S(e, j) = 1, \forall j \in V \setminus U$, and $S(e, j) = 0, \forall j \in U$ and for all the edges e belonging to the tour. $S(e, j)$ for $j \in U$ and e not in the tour does not matter since $q(e)$ for those edges will be zero. Hence, the objective function is

$$\begin{aligned} f &= \sum_{j \in V} (\tilde{\phi}_j) - \sum_{j \in V \setminus U} (\tilde{\phi}_j) \\ &= \sum_{j \in U} (\tilde{\phi}_j). \end{aligned} \tag{4.11}$$

Hence for any deterministic tour of length at most L, the objective is the sum of the weights of the vertices in the tour. Since, the orienteering problem by definition maximizes that sum, any other deterministic tour with length at most L will be sub-optimal. Also note that any random path that visits only the vertices in U cannot perform better than the orienteering tour. Let U_α be the vertices in an α approximation of the orienteering tour of the graph with maximum tour length L and vertex weights $\tilde{\phi}$. U_1 represents the vertices in the optimal orienteering tour.

Note from Equation (4.3) that the single robot objective function can be written as

$$f = \sum_{e \in E} q(e) \Big[\sum_{j \in V} \tilde{\phi}_j \big(1 - S(e,j)\big) \Big].$$

Since, the events arrive on the vertices with equal probability, the term $\sum_{j \in V} \tilde{\phi}_j \big(1 - S(e,j)\big)$ represents the average value of the detected events when the robot starts from edge e. For any general Markov chain P, whenever the robot starts from edge e and comes back to the same edge e in at most L time, by the definition of the orienteering problem, it could not have visited the vertices with more collective value than an orienteering tour, i.e.,

$$\sum_{j \in V} \tilde{\phi}_j \big(1 - S(e,j)\big) \leq \sum_{j \in U_1} \tilde{\phi}_j \leq \frac{\sum_{j \in U_\alpha} \tilde{\phi}_j}{\alpha}.$$

In the case when the robot starts from edge e and comes back to edge e in more than L time, it could not have visited vertices with total value more than twice that of an orienteering tour by the following claim.

Claim 4.5.2. *The total vertex weight collected by an orienteering path of length L is at most twice the total vertex weight collected by an orienteering tour of length L.*

Proof. Split the orienteering path into two segments of length $L/2$. One of those segments must have a total vertex weight at least half of the total weight collected by the path. Since the graph is metric, a tour can be constructed collecting at least half the total weight of the orienteering path using that half segment. □

Hence, for the case when the robot comes back to edge e in more than L time, we have

$$\sum_{j \in V} \tilde{\phi}_j \big(1 - S(e,j)\big) \leq 2 \sum_{j \in U_1} \tilde{\phi}_j \leq \frac{2}{\alpha} \sum_{j \in U_\alpha} \tilde{\phi}_j.$$

Therefore we can write,

$$f \leq \sum_{e \in E} q(e) \big(\frac{2}{\alpha} \sum_{j \in U_\alpha} \tilde{\phi}_j \big) = \frac{2}{\alpha} \sum_{j \in U_\alpha} \tilde{\phi}_j,$$

where the second equality follows because $\sum_{e\in E} q(e) = 1$. Noting that $\sum_{j\in U_\alpha} \tilde{\phi}_j$ is the objective function obtained by an α-approximation of the orienteering problem as shown in Equation (4.11) completes the proof. $\square$

Remark 4.5.3. *The approximation ratio given in Proposition 4.5.1 is tight. Consider a ring graph with $n \geq 8$ vertices where each vertex is connected to its two neighbors with edge length 1. Let the vertex weights be one for all the vertices and the duration of event at each vertex be $n/2$. An orienteering tour of length $n/2$ will result in the objective function $1/4 + 1/n$. A deterministic tour of the whole ring will result in the objective function being $1/2$. As the number of vertices grows, we get the approximation ratio $1/2$ with the orienteering tour approximation ratio being $\alpha = 1$ in this example.*

Since the set function $f(\mathcal{P})$ is submodular, we can greedily add Markov chains to maximize the marginal reward of adding the new Markov chain in order to get an approximation algorithm to the multi-robot problem.

Algorithm 4 ApproximationAlgorithm

```
Input: Graph $G = (V, E)$ with edge length $\ell_e$, vertex weights $\phi_i$ and arrival probabilities
$\psi_i$, $\forall i \in V$, event stay duration $L$, and number of robots $m$.
Output: A set of $m$ Markov chains $\mathcal{P} = \{P_1, \dots, P_m\}$.

1: $\bar{V} \leftarrow V$
2: for $r = 1$ to $m$ do
3:     $P_r \leftarrow$ Orienteering Tour of $\bar{V}$ with vertex weights $\phi_i\psi_i$ for $i \in \bar{V}$ and tour length $L$.
4:     Remove vertices covered by $P_r$ from $\bar{V}$
5: end for
```

Proposition 4.5.4. *If the duration of events is fixed as L for all the vertices, i.e., $g_i(l) = \delta(L)$, then given an α-approximation algorithm for Orienteering Tour Problem, Algorithm 4 is a $1 - \frac{1}{e^{\alpha/2}}$ approximation algorithm for Problem 4.10.*

Proof. For a submodular, non-negative and monotone set function defined over the subsets of a *finite* set, the following algorithm gives $1 - \frac{1}{e^{\beta}}$ approximation for maximizing the set function [69]. Start from an empty set and iteratively select an element to add to the solution set such that the marginal gain of the added element is at least β times the maximum marginal gain of adding a new element.

The set of all Markov chain transition matrices defined on the graph $G = (V, E, W)$, is an uncountably infinite set. Therefore to use the greedy algorithm, we construct a finite set as follows. Take the m optimal Markov chains $P_1^*, \ldots, P_m^*$, and m Markov chains picked by the greedy algorithm $P_1, \ldots, P_m$ to form a set of Markov chains with $2m$ elements. If we run the greedy algorithm on this finite set, then using Proposition 4.5.1 we get the result that $f(\{P_1, \ldots P_m\}) \geq (1 - \frac{1}{e^{\alpha/2}}) f(\{P_1^*, \ldots, P_m^*\})$. Since $\{P_1^*, \ldots, P_m^*\}$ is the optimal solution in $\mathcal{P}$, we get the desired approximation ratio. $\square$

In the next section we solve the general problem where the duration of events can be distributed.

4.6 Centralized Algorithm

Motivated by the submodular structure of the problem, we propose an algorithm similar to Algorithm 4. We will first present an algorithm to solve the single robot problem and then give a greedy algorithm to construct a solution for the multi-robot problem. The following section focuses on finding a solution for the single robot problem.

4.6.1 Single Robot Solution

Since the single robot objective function is non-convex and continuously differentiable, we can use gradient descent methods to find a locally optimal Markov chain. However, due to the recursive nature of the objective function, calculating the gradient of the objective function is computationally expensive with runtime $O(n^5 L_{\texttt{max}})$ where n is the number of vertices in the graph and $L_{\texttt{max}}$ is the maximum duration of event. Therefore, we use a gradient free direct search method (Algorithm 5) that converges to a Markov chain P' such that $\nabla f(P') = 0$ [60]. We present the details of the direct search algorithm below.

At iteration k of Algorithm 5, a set of search directions $\mathcal{D}_k$ is chosen and the function is evaluated at a given step length γ_k along these directions. As soon as a better point is found, it is chosen as the current point and the method proceeds to the next iteration. The step length is decreased or increased depending on whether a better point is found. The sufficient increase function $\rho(\gamma_k)$ is $M\gamma_k^{3/2}$ where M is a constant, as suggested by [72] for the algorithm to converge.

Direction set: For each row i of the transition matrix, the constraints define a simplex in $\mathbb{R}^{|\mathcal{N}(i)|}$ dimensions where $\mathcal{N}(i)$ is the set of neighboring vertices of vertex i. So, the set $\mathcal{D}_k$ includes $|\mathcal{N}(i)| - 1$ orthonormal directions on the simplex for each row i.

Algorithm 5 PATROLLINGSTRATEGY

```
Input: Graph $G = (V, E)$ and function $f(P)$.
Output: Markov chain $P$.
1: Pick a starting transition matrix $P_o$ and $\gamma_o$
2: for $k = 0, 1, 2, \ldots$ do
3:     if $\gamma_k < \gamma_t$ then
4:         stop
5:     end if
6:     $flag \leftarrow 0$
7:     for $d \in \mathcal{D}_k$ do
8:         $Q \leftarrow P_k + \gamma_k d$
9:         if $Q$ not feasible then
10:            $Q \leftarrow$ PROJECT($Q$)
11:        end if
12:        if $f(Q) > f(P_k) + \rho(\gamma_k)$ then
13:            $P_{k+1} \leftarrow Q$
14:            $\gamma_{k+1} \leftarrow \phi_k \gamma_k$
15:            $flag \leftarrow 1$
16:            break
17:        end if
18:    end for
19:    if $flag = 0$ then
20:        $\gamma_{k+1} \leftarrow \theta_k \gamma_k$
21:    end if
22: end for
```

The motivation for the search directions in the direct search method comes from the 2-opt or k-opt local search heuristic for the travelling salesman problem [62]. The Markov chain transition matrix P for a deterministic tour will have $P(i, j) = 1$ if and only if the vertex j comes after the vertex i in the tour. Suppose a deterministic tour is given by $(v_1, \ldots, i, j, \ldots, x, y, \ldots, v_n)$. A 2-opt move will be to set $P(i, j)$ and $P(x, y)$ to 0, and set $P(i, x)$ and $P(j, y)$ to 1. Since in a Markov chain we can have transition probabilities between zero and one, we can generalize 2-opt by getting a transition matrix Q from a given transition matrix P by setting $Q(i, j) = P(i, j) - \epsilon$, $Q(i, x) = P(i, x) + \epsilon$, $Q(x, y) = P(x, y) - \epsilon$ and $Q(j, y) = P(j, y) + \epsilon$ for some $\epsilon > 0$. Also note that we can also have 1-opt local search directions by modifying only one row of the transition matrix P. In our implementation, we use 1-opt and some random directions in $\mathcal{D}_k$.

Projection onto the feasible set: The function PROJECT in Line 10 of Algorithm 5 projects the matrix Q onto the convex set defined by the first three constraints of Problem 4.10. Since each row i of matrix P lies in a $|\mathcal{N}(i)|$ dimensional simplex, we can use the algorithm from [32] to project each row of Q onto the desired simplex. This method employs simple vector operations and is therefore more efficient as compared to projection using convex optimization.

4.6.2 Multi-Robot Solution

Now we can use the single robot algorithm presented in the last section to construct a multi-robot solution. We start from random patrolling Markov chains for all the robots. At each iteration of the multi-robot algorithm, the Markov chain of one of the robots, say r, is updated. The following observation shows that the problem of finding P_r to maximize $f(\mathcal{P})$ is equivalent to solving the problem for a single robot. Therefore, we can use Algorithm 5 to update P_r.

Observation 4.6.1. *If the Markov chains $\{P_1, \ldots, P_{r-1}, P_{r+1}, \ldots, P_m\}$ are given, the problem of finding the Markov chain P_r to maximize the objective function $f(\mathcal{P})$ given in (4.3) is equivalent to solving the problem for a single robot.*

Proof. The term $\boldsymbol{\phi}^T\boldsymbol{\psi}$ in the objective function in Equation (4.3) is independent of the patrolling policy, hence maximizing $f(\mathcal{P})$ is equivalent to minimizing

$$(\bigodot_{k=1}^{m} \boldsymbol{q_k}^T S_k)(\boldsymbol{\phi} \odot \boldsymbol{\psi}). \tag{4.12}$$

We can write this objective as

$$\begin{aligned} &\Big((\bigodot_{k\neq r} \boldsymbol{q_k}^T S_k) \odot \boldsymbol{q_r}^T S_r\Big)(\boldsymbol{\phi} \odot \boldsymbol{\psi}) \\ =&(\boldsymbol{q_r}^T S_r)\Big((\bigodot_{k\neq r} \boldsymbol{q_k}^T S_k)^T \odot \boldsymbol{\phi} \odot \boldsymbol{\psi}\Big). \end{aligned}$$

Since $\boldsymbol{q_k}$ and S_k are fixed for $k \neq r$, we can define new vertex weights as

$$\tilde{\phi}_j = \prod_{k\neq r} \big(\sum_{e\in E} q_k(e) S_k(e,j)\big)\phi_j$$

and the objective function becomes $(\boldsymbol{q_r}^T S_r)(\tilde{\boldsymbol{\phi}} \odot \boldsymbol{\psi})$, the minimization of which is equivalent to minimizing the function (4.12) with $m = 1$ and modified vertex weights. □

Since Algorithm 5 is an ascent algorithm, and because the Markov chains of all the other robots remain the same during this iteration, $f(\mathcal{P})$ increases if a new Markov chain for robot r is found. The algorithm stops when none of the robots can increase $f(\mathcal{P})$.

4.7 Online Distributed Algorithm

In the problem and its solutions studied so far, the parameters ϕ_j (importance of vertex j), ψ_j (probability of event arrival on vertex j), and g_j (the distribution of the duration of events on vertex j) are provided beforehand as an input to the problem. In a practical setting, these parameters may not remain constant throughout the monitoring mission, or the parameters may be unknown beforehand. In these cases, the robots may use some estimation and filtering techniques to update these parameters from the observed data while patrolling the environment. As the parameters change, the patrolling policy should change accordingly. In this section we present a distributed online algorithm to solve the problem.

Given the current patrolling policy $\mathcal{P} = \{P_1, \dots, P_m\}$, where robot r is patrolling the vertices $V_r \subseteq V$, let $\mathcal{M}_r$ represent the neighboring robots of robot r. The neighboring robots can be defined based on the communication protocol used by the robots. Our proposed algorithm is independent of the definition of $\mathcal{M}_r$ as long as two non-neighbor robots do not visit the same vertex on their patrolling paths. One such definition of the neighbors of robot r is given in [99] where the neighbors of robot r are the robots that are patrolling at least one vertex that is a neighboring vertex of the vertices in V_r. We assume that the robot r knows the current Markov chains P_k for their neighboring robots $k \in \mathcal{M}_r$.

Algorithm 6 DistributedAlgorithmMove(r, k)

```
1: for $i \in V_r$ and $j \in V_k$ do
2:     Find new vertex sets $\bar{V}_r$ and $\bar{V}_k$ using Local Move 1, 2 or 3
3:     Find new Markov chains $\bar{P}_r$ and $\bar{P}_k$ on the new vertex sets
4:     Evaluate objective function on the set $V_r \cup V_k$
5:     if objective function improves then
6:         Return $\bar{V}_r$, $\bar{V}_k$, $\bar{P}_r$ and $\bar{P}_k$
7:     end if
8: end for
```

4.7.1 Local Moves

In the distributed algorithm, a pair of neighboring robots r and $k \in \mathcal{M}_r$ communicate and try some local moves to improve the local objective function, i.e., the objective function defined on the vertices $V_r \cup V_k$. The neighboring robots that execute the local move can be selected in a distributed manner using a token passing algorithm [58], or any other method that ensures each pair of robots gets a turn at making a local move. One such randomized algorithm is presented in [34]. The algorithm terminates when no pair of robots can improve the local objective function. The possible local moves that can be implemented in Line 2 of Algorithm 6 are given below.

Local Move 1 (Vertex Removal): Robot r removes a vertex i from its subset V_r, which is added to the subset of the neighboring robot $k \in \mathcal{M}_r$, resulting in new subsets $\bar{V}_r \leftarrow V_r \setminus \{i\}$ and $\bar{V}_k \leftarrow V_k \cup \{i\}$.

Local Move 2 (Vertex Exchange): Robot r and robot $k \in \mathcal{M}_r$ swap vertices $i \in V_r$ and $j \in V_k$ to get the subsets $\bar{V}_r \leftarrow (V_r - \{i\}) \cup \{j\}$ and $\bar{V}_k \leftarrow (V_k - \{j\}) \cup \{i\}$.

Local Move 3 (Vertex Sharing): A vertex $i \in V_r$ is shared with robot $k \in \mathcal{M}_r$, so that both the robots r and k will cover vertex i. The subset V_r will remain the same, whereas $\bar{V}_k \leftarrow V_k \cup \{i\}$. This local move can be ignored if we are only interested in partitioned solutions.

In Line 3 of Algorithm 6, we find new Markov chains on the vertex sets updated using local moves. These Markov chains can be found using Algorithm 5. However, for quick evaluation of new Markov chains $\bar{P}_r$ and $\bar{P}_k$ that are 'near' to the current Markov chains P_r and P_k, we use the method described in the following section.

4.7.2 Adding and Removing Vertices

To evaluate the change in objective function when a vertex is removed or added to a partition, we need the new Markov chain. As discussed earlier, for a general objective function, finding an optimal Markov chain for a single robot is an NP-hard problem. Even for the objective functions where an optimal Markov chain can be found, e.g., a semi-definite program can find a Markov chain that minimizes the Kemeny constant [78], we will need to solve for $O(n^2)$ optimal chains since we have to evaluate the change in the multi-robot objective function each pair $i \in V_r$ and $j \in V_k$. Therefore we present a heuristic method to find new Markov chains after adding or removing a vertex from a partition.

Removing a Vertex: Let $x \in V_r$ be the vertex removed from a partition V_r to get $V_r - \{x\}$ and the new Markov chain be $\bar{P}_r$. Since x is removed from V_r, the transition probabilities between x and vertices in $V_r - \{x\}$ become zero, and we need to update the transition probabilities $P(i,j), \forall i,j \in V_r - \{x\}$. If the edge (i,j) is in the graph induced by the vertices in $V_r - \{x\}$, then its updated transition probability is given by

$$\bar{P}_r(i,j) = P_r(i,j) + P_r(i,x)\Big(P_r(x,j) + \frac{\phi(j)}{\sum_{v \in \mathcal{N}(i)} \phi(v)}\big(\sum_{w \notin \mathcal{N}(i)} P_r(x,w)\big)\Big), \tag{4.13}$$

where $\mathcal{N}(i)$ is the set of outgoing neighbors of i in the graph induced by $V_r - \{x\}$.

Notice that a special case of this expression is when we remove a vertex from a deterministic path or tour. In that case all the probabilities are either 1 or 0, and after removing a vertex x from the tour, the resultant tour is obtained by connecting the vertex preceding x in the tour to the vertex following x in the tour.

Proposition 4.7.1. *The Markov chain transition matrix $\bar{P}$ obtained using Equation (4.13) is feasible.*

Proof. Since we are only adding a positive term to the old transition probability, $\bar{P}_r(i,j)$ is positive. Now we show that the sum of each row in the transition matrix is one. First notice that

$$\begin{aligned} &\sum_{j \in \mathcal{N}(i)} \Big(P_r(x,j) + \frac{\phi(j)}{\sum_{v \in \mathcal{N}(i)} \phi(v)}\big(\sum_{w \notin \mathcal{N}(i)} P_r(x,w)\big)\Big) \\ &= \sum_{j \in \mathcal{N}(i)} P_r(x,j) + \frac{\sum_{j \in \mathcal{N}(i)} \phi(j)}{\sum_{v \in \mathcal{N}(i)} \phi(v)} \sum_{w \notin \mathcal{N}(i)} P_r(x,w) \\ &= \sum_j P_r(x,j) = 1. \end{aligned}$$

Since, $x \notin \mathcal{N}(i)$, $\sum_{j \in \mathcal{N}(i)} P_r(i,j) = 1 - P_r(i,x)$, therefore, using Equation (4.13),

$$\begin{aligned} \sum_{j \in \mathcal{N}(i)} \bar{P}_r(i,j) &= 1 - P_r(i,x) + P_r(i,x) \\ &= 1. \end{aligned}$$

□

Adding a Vertex: For adding a vertex y to a partition V_r, we need to choose the transition probabilities from all vertices in V_r to y, the transition probabilities from y to all the vertices in V_r, and to update the transition probabilities for all $i, j \in V_r$. We select the transition probability of going from a vertex $i \in V_r$ to y based on the vertex weight of y. The higher the $\phi(y)$ as compared to the sum of the weights of neighbors of i, the higher the transition probability of going from i to y. Let the new transition matrix after adding vertex y to V_r be given by $\bar{P}_r$, then,

$$\frac{\sum_{j \in \mathcal{N}(i)} \bar{P}_r(i,j)}{\sum_{j \in \mathcal{N}(i)} \phi(j)} = \frac{\bar{P}_r(i,x)}{\phi(x)}.$$

Since, we also want $\sum_{j \in \mathcal{N}(i)} \bar{P}_r(i,j) + \bar{P}_r(i,x) = 1$, we get

$$\bar{P}_r(i,x) = \frac{\phi(x)}{\sum_{j \in \mathcal{N}(i)} \phi(j) + \phi(x)}.$$

For the transition probabilities on the outgoing edges from x, we also use the vertex weights to get

$$\bar{P}_r(x,j) = \frac{\phi(j)}{\sum_{i \in \mathcal{N}(x)} \phi(i)}.$$

Now we re-normalize to get the transition probabilities between $i, j \in V_r$,

$$\begin{aligned}\bar{P}_r(i,j) &= \frac{P_r(i,j)}{1 + \bar{P}(i,x)} \\ &= \frac{P_r(i,j)(\sum_{v \in \mathcal{N}(i)} \phi(v) + \phi(x))}{\sum_{v \in \mathcal{N}(i)} \phi(v) + 2\phi(x)}.\end{aligned}$$

It is easy to see that that $\bar{P}_r$ is a feasible transition matrix.

4.8 Objective Functions for Adversarial Events

The objective function $f(\mathcal{P})$ gives the expected reward of successfully observed events when the events arrive at the vertices according to a given distribution, and the time of arrival of events is random and independent of the positions of the robots. Such events can model environmental phenomenon such as forest fires where historical data can be used to estimate the probability of event arriving at a location and the arrival of events is independent of the patrolling paths. However, in certain applications such as patrolling to protect locations of an environment against intruders, the events can be adversarial and they can use the information about the patrolling paths to avoid detection by the patrolling robots or to minimize the patrolling reward.

In this section, we consider two different cases of adversarial events. In the first case, the probability of arrival of events at locations is given, but the events can choose the time to appear in order to avoid being detected. These events model cases where the event selects the location to appear at beforehand, and then uses the information about the patrolling policy to decide when to appear. In the second case, the events can also choose the location to appear at based on the patrolling policy of the robots. We provide the objective functions that represent the expected value of observed events for both these cases below.

4.8.1 Adversarial Events with Given Probability of Arrival

First we give the objective function for the events that appear at location i according a known probability ψ_i and choose the time to appear in order to maximize their chances of remaining undetected. Using Equation (4.2), the expected reward for such events using patrolling policy $\mathcal{P}$ is given by

$$\begin{aligned} f_2(\mathcal{P}) &= \sum_{j=1}^{n} \phi_j \mathbb{P}\left[\texttt{event at } j\right] \mathbb{P}\left[\texttt{observed at } j\right] \\ &= \sum_{j=1}^{n} \phi_j \psi_j \mathbb{P}\left[\texttt{observed at } j\right]. \end{aligned} \tag{4.14}$$

To calculate the probability of an event arriving at j not being observed by the robots, we first note that in order to maximize its chances of remaining undetected by robot r, the event will appear as soon as robot r commits to traversing an edge e. Hence the probability of detecting such an event at vertex j given event duration L is given by

$$\bar{S}_r(e, j|L) = \begin{cases} \sum_{t=1}^{L-\ell_e} F_t(i,j), & \ell_e < L \\ 0, & \text{otherwise.} \end{cases}$$

Using this probability, $S(e, j)$ can be calculated by using Lemma 4.3.2.

Also note that in this case we do not need the probability of robot r traversing edge e when the event arrives because in the worst case scenario for the robot, the event will arrive when the robot is on the edge which maximizes the probability of missing the event, i.e., $S_r(e,j)$. Hence we get

$$\mathbb{P}\left[\texttt{not observed at } j \texttt{ by } r\right] = \max_e S_r(e,j). \tag{4.15}$$

Since the Markov chains of the robots are independent, we get

$$f_2(\mathcal{P}) = \sum_{j=1}^{n} \phi_j \psi_j \Big(1 - \prod_{r=1}^{m} \big(\max_e S_r(e,j)\big)\Big). \tag{4.16}$$

An analysis similar to Proposition 4.4.4 shows that this objective function is also submodular.

4.8.2 Adversarial Events that Select Vertex

Now we consider events that can also decide where to appear in order to maximize the chances of avoiding detection by the robots. In this scenario, the probabilities of event arrival $\boldsymbol{\psi}$ are not given. Using Equation (4.15) and the fact that the robots are independent, the probability of the robots not detecting an event that arrives at the worst possible time at vertex j is given by,

$$\mathbb{P}\left[\texttt{not observed at } j\right] = \prod_{r=1}^{m} \max_e S_r(e,j). \tag{4.17}$$

Now, in the worst case the event will appear at the vertex that minimizes the patrolling reward:

$$f_3(\mathcal{P}) = \min_j \left[\phi_j \Big(1 - \prod_{r=1}^{m} \max_e S_r(e,j)\Big)\right]. \tag{4.18}$$

Observation 4.8.1. *The objective function $f_3(\mathcal{P})$ is not submodular.*

Proof. Consider a simple example with three vertices a, b and c. All the vertices have same reward equal to 1. Now consider three Markov chains P_a, P_b and P_c. Each Markov chain P_x just stays at vertex x and the robot does not move. Then adding the Markov chain

P_c to a larger set of Markov chains $\{P_a, P_b\}$ increases the expected reward from 0 to 1 as all three vertices will be covered. Adding P_c to the smaller set $\{P_a\}$ keeps the expected reward at 0 because the event will always appear at vertex b. Hence, adding a new element to a larger set offers more marginal reward and the function is not submodular. $\square$

This objective function can also be interpreted as the worst case reward even if the events are not adversarial. The algorithms presented in Sections 4.6 and 4.7 can be used to find patrolling policies for the adversarial objective functions as well.

4.9 Simulations

In this section we evaluate the performance of the algorithms presented in this chapter.

4.9.1 Problem Instance

The problem instance was constructed by taking a planar environment with obstacles, and picking random points as vertices in the free environment. To create edges each vertex was connected to its 4 nearest neighbors and the euclidean distance was rounded off to get integer edge weights on the edges. The product of the probability of arrival and vertex weight, i.e., $\phi_i\psi_i$, was drawn randomly from a uniform distribution between 1 and 10. The resulting instance with 20 vertices is shown in Figure 4.4. The duration of events was distributed between 0 and a random number chosen from a uniform distribution between 10 and 20 for each vertex. The experiments were performed for three robots, i.e., $m = 3$.

4.9.2 Centralized Algorithm

The performance of the centralized algorithm presented in Section 4.6 is shown in Figure 4.5. Each color in the figure represents a different robot, e.g., green represents the direct search perfromed by robot 1. The graph shows the value of the objective function as the robots perform the direct search from Algorithm 5 iteratively.

4.9.3 Distributed Algorithm

For the distributed algorithm, we considered the objective function in Equation (4.18), i.e., the events that can choose the time to appear adversarially. We did not consider local

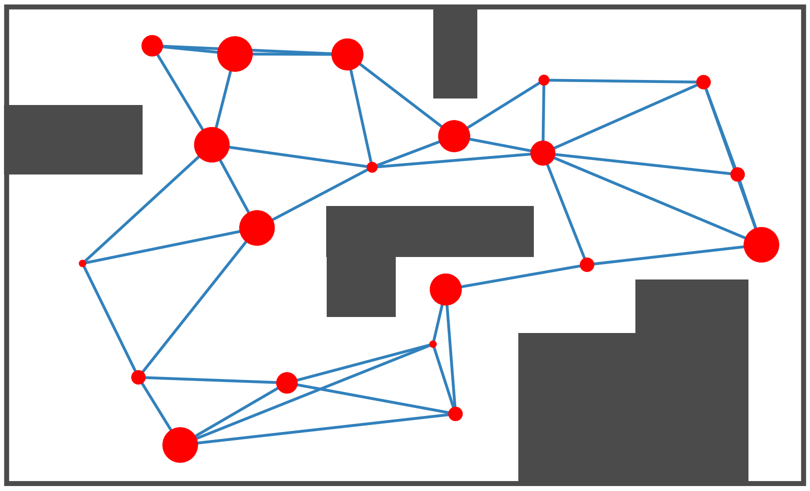


Figure 4.4: The graph with 20 vertices used for simulations. The size of the vertices shows the product of vertex weight and arrival probability.

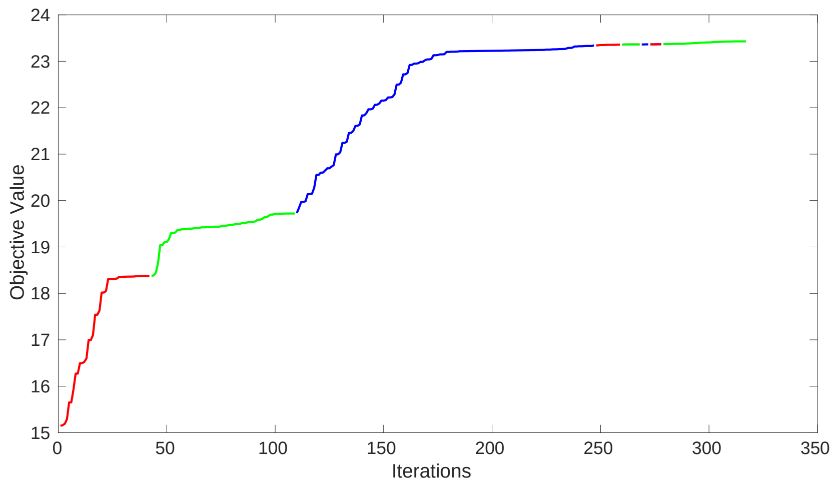


Figure 4.5: The objective value as a percentage of the total reward available in the environment is shown as a function of the iterations of the direct search method performed by each robot. Different colors show the progress of the direct search method for different robots.

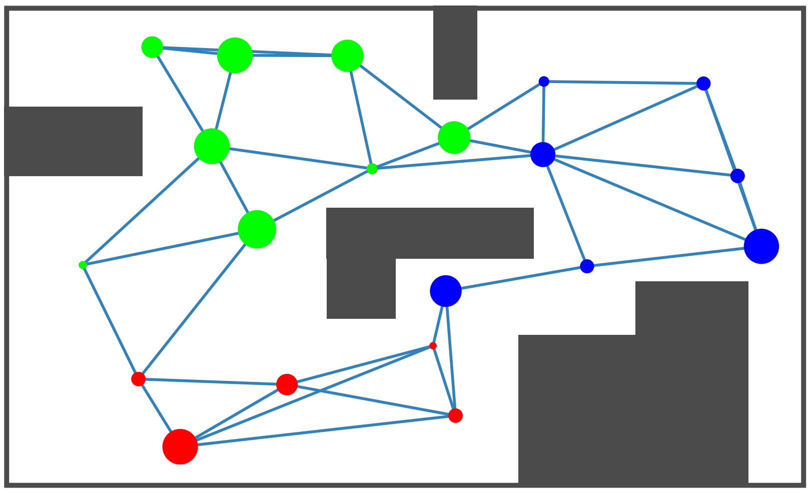


Figure 4.6: The final partitions returned by the distributed algorithm.

move 3 and hence only partitioned solutions were considered. The partitions returned by the algorithm are shown in Figure 4.6. The vertices of one color represent the partition of a robot. Note that the robots have a transition matrix defined on each of these partitions.

Chapter 5

Adversarial Events with Limited Observation Time

The problem of patrolling locations of importance to protect against possible attacks by an adversary arises in several security applications including ports, airports and environmental resources. An adversary can observe the patrolling path over time and use its knowledge about the path to its advantage. In Chapter 4, we discuss the scenario where the adversary has perfect knowledge of the patrolling paths. Unless the patrolling strategy is leaked to the adversary, the adversary will have to observe the patroller long enough to derive the strategy [12]. However, this might not always be true in practice [53], as the adversary might not have enough time to perfectly learn the patrolling strategy. In this chapter we focus on the scenarios where the adversary has a limited time to observe the patrolling path to infer the patrolling strategy. It then decides to attack or renege based on that learned strategy. Designing a patrolling strategy that takes longer to learn will increase the chances of the adversary reneging. However, the strategy should also minimize the chances of a successful attack in case the adversary attacks. We study the single robot version of the problem in this chapter. We will refer to the adversary as the intruder or attacker throughout this chapter.

The organization and contributions of this chapter are as follows. We start by reviewing the literature related to the problem in Section 5.1. We present a game model for the scenario where the adversary has a limited time to learn the patroller's strategy in Section 5.2. We devise a strategy for a rational adversary and characterize the expected payoffs for the players in Section 5.3. In Section 5.4 we set up the problem as a linearly constrained non-convex, non-smooth optimization problem and show that the objective function is locally Lipschitz so that direct search methods guarantee convergence to a Clarke Stationary point.

We also show that finding a k factor approximation to the optimal patrolling strategy is NP-hard. Finally we share the experimental results in Section 5.5.

5.1 Related Work

There is a substantial amount of work related to design of patrolling policies in robotics, control, and game theory. Random patrolling paths are advocated in [89] by showing that it is hard to find deterministic strategies that satisfy the surveillance criterion of visiting vertices proportional to their importance. Stochastic surveillance is considered in [38] so that the intruders cannot easily exploit the predictability of a deterministic path. Intelligent intruders in a perimeter patrol problem are considered in [4]. Different ways of randomizing paths are used in [85] to empirically show that randomizing the paths can decrease the probability of successful intrusions. In [61] a strategic attacker is considered that plays two player zero sum game with the patroller.

As done in Chapter 4, we will restrict the search of the optimal patrolling policy to the set of Markov chains in this chapter. Markov chains are often used to model random paths on a graph [89, 38, 78]. For patrolling, a common objective is to minimize mean first passage time of a Markov chain [78]. Different intruder models are considered in [10] to find Markov chains for patrolling in the presence of such intruders. However, these works do not consider an intruder that can observe and learn the patrolling paths.

In game theory, security games are usually posed as Stackelberg games [76, 7, 49, 94, 17]. In [5], a finite set of Markov chains is used to formulate the patrolling problem as a Bayesian Stackelberg Game where the patroller first picks a mixed optimal strategy and the adversary then picks the region to attack to maximize its payoff. In [93] the patrolling problem is studied on the graphs where traversing edges incurs different costs but the same amount of time and the attacker prefers to attack as soon as possible. The problem setting considered by [12] is closest to ours, however we do not assume that the attacker knows the patrolling strategy beforehand, and it has a limited time to learn that strategy.

Limited observation time for the adversary leads to it having uncertainty about its own payoffs. In [52] security games where the defender is uncertain about the attacker's payoff is considered. In [53] a player, dubbed 'Nature', decides whether the follower will observe the leader's actions or not and the leader does not know Nature's choice. We will also use Nature in our game model to pick the time allowed for observation. In [9] a fixed amount of resources are distributed among some locations at each step of the game and the attacker observes the defender for a limited time before attacking. The attacker's belief about the defender's strategy is represented as a Dirichlet distribution, and the attacker updates this belief using Bayes' rule based on its observations. The defender's objective function depends on all possible observation sequences made by the attacker. In [8] the authors update the model to consider observation costs for the attacker, and the optimal observation time for the attacker is evaluated. These works, however, consider placing static resources on a set of locations rather than the problem of a patroller traversing a graph.

5.2 Problem Definition

In this chapter we consider a single robot problem with settings similar to the previous chapter. The patrolling scenario involves a patrolling agent or robot patrolling along the edges of a directed weighted graph $G = (V, E)$ where the vertices $V = \{1, 2, ..., n\}$ represent the locations to be monitored. The integer travel times on edges of the graph are represented by ℓ_e for $e \in E$. Each vertex i of the graph has a value $\phi_i \geq 0$ that represents the value of that vertex. An intruder waits outside the environment observing the patrolling agent's path. It can attack any vertex of the graph and stay there for L time steps to complete the attack. We use the uniform length of attack (or event duration) L for simplicity of presentation. The event duration can be different for each vertex as in Chapter 4 and the results will hold with a slight change in payoff equations. Moroever, in this chapter we assume for simplicity that the intruder can only appear at a vertex when the patrolling robot leaves a vertex. The patrolling agent cannot observe the intruder unless it visits a vertex i while it is being attacked, in which case the intruder is captured and incurs a penalty ψ whereas the patroller receives a reward ρ. Otherwise, the attack is successful and the intruder receives the reward ϕ_i.

This scenario can be modeled as a patrolling game with two players, the robot and the intruder. Patrolling games are usually modeled as Stackelberg games where the defender chooses a strategy first, and the intruder being the follower knows the patroller's strategy and plays the best response. We consider the case where the intruder does not know the patrolling strategy beforehand. It has a fixed time budget to observe the patrolling path and it uses those observations to learn the patrolling agent's strategy. It then decides whether it will attack the environment or leave without attacking.

5.2.1 Game Model

We now model the above mentioned patrolling scenario as a multi-stage game, building on the game in [12] to include the learning aspect of the intruder. The game involves two players, the patrolling robot (defender) and the intruder (attacker). The players act simultaneously at each stage of the game. The attacker can observe the actions of the defender whereas the defender cannot observe the actions of the attacker. We also employ a move by Nature to formally model the time allowed for the attacker to learn the defender's strategy. Nature selects a time T that denotes the maximum allowed time for the intruder before it decides whether it will attack or leave without attacking. The attacker knows the time T, however, the defender only knows the probability distribution $q : t \in \mathbb{Z}_+ \to [0, 1]$ representing the probability of Nature selecting time t. The game starts at time zero.

Actions: The defender can go to a neighboring vertex of its current vertex i by taking the action *move*(j) where $j \in \mathcal{N}(i)$. Let H represent the set of all possible histories of vertices visited by the patroller, i.e. $h \in H$ is a sequence of vertices. The available actions for the attacker are *obs*, *renege* or *attack-when*(j, h) for $j \in V$ and $h \in H$. Action *obs* corresponds to observing the current action of the defender for the sake of learning its strategy. Note that *obs* cannot be played after time T has elapsed from the start of the game. Also note that time steps are not analogous to stages of the game since the graph is weighted. Traversal of an edge by the defender can take multiple time steps but spans one stage of the game since both the attacker and defender play one action during that time. Playing *attack-when*(j, h) means that the attacker will wait until the defender has followed history h and then start the attack at vertex j. The attacker cannot take any other action after playing *attack-when*(j, h) and the game will conclude either in a successful attack or the capture of the intruder. Playing *renege* means that the attacker will leave without attacking the environment.

Outcomes: In case the intruder plays *renege*, the outcome of the game will be *no-attack*. If the attacker plays *attack-when*(j, h), it can either result in *successful-attack-j* if the defender does not visit j while it was being attacked or *capture* otherwise.

Payoffs: Both the player's payoffs are given in the following table. The attacker's

outcome	attacker's payoff	defender's payoff
no-attack	0	0
successful-attack-j	ϕ_j	$-\phi_j$
capture	$-\psi$	ρ

payoff is the value gained from attacking or the penalty incurred if it is captured. Setting

$\rho = 0$ represents a defender that does not give priority to *capture* over *no-attack* and only wants to stop a successful attack.

Strategies: The defender's strategy $\sigma_d(h)$ gives the probability with which the defender will move to a vertex that is a neighbor of the last visited vertex in $h \in H$. Since the defender cannot observe the actions of the attacker, its strategy is not a function of the attacker's actions. The attacker's strategy is a function from the patrolling history to the possible actions *obs*, *attack-when*(j, h) or *renege*.

Remark 5.2.1. *We assume that T is much greater than the time the attacker waits to attack after deciding where to attack. This models scenarios where the attacker can commit significant resources/time to planning an attack relative to the time scale on which the defender traverses the graph. In this case the information gained while waiting will be negligible compared to the information gained while learning.*

5.3 Solution Approach

For the scenario where the attacker has perfect knowledge of defender's strategy beforehand, the defender's strategy will be Markovian with some finite memory length as discussed in Chapter 4. We restrict defender's strategy to Markov chains with memory one in this chapter as well. Moreover, since the defender receives no information about the attacker during the game unless the attacker is captured (at which point the game ends), we only consider stationary Markov chains. Thus, we represent the defender's strategy as the transition matrix $P = [p_{ij}]$ of a Markov chain, where p_{ij} gives the probability of the patrolling agent going to vertex j when it is in i. In the rest of the chapter, optimal strategy for the defender will refer to the optimal Markov chain.

5.3.1 Attacker's Strategy

Assuming that the players are rational, the following proposition simplifies the strategy of the attacker.

Proposition 5.3.1. *Restricting the defender's strategy to be a time homogeneous Markov chain, an optimal strategy for the attacker is to play the action obs until time T and then to either play attack-when(j, i) for some $i, j \in V$, or renege.*

Proof. Actions *attack-when*(j, h_1) and *attack-when*(j, h_2) for $h_1, h_2 \in H$ such that the last vertex in h_1 and h_2 is the same vertex i, will give the attacker the same expected payoff, since the defender's strategy only depends on the current vertex occupied by the defender. Given an observation sequence o_t of duration $t \leq T$, consider the attacker's optimal action given by $a(o_t)$. Suppose that the expected payoff for the attacker is maximized for some o_{t^*} where $t^* < T$. Then the attacker can still receive the same payoff by playing the action $a(o_t^*)$ after observing o_T because there is no cost to play *obs* and the expected payoff for action *attack-when*(j, i) is independent of the time the action is played. □

Given that the defender is using the transition matrix P as its strategy, the attacker's expected payoff for the action *attack-when*(j, i) can be calculated using the probability of robot visiting vertex j from vertex i in exactly t time steps, i.e., $F_t(i, j)$ from Equation (4.4). The probability of an action *attack-when*(j, i) being successful can then be calculated using Lemma 4.3.1 as $S(i, j) = 1 - \sum_{t=1}^{L} F_t(i, j)$. The attacker's expected payoff for this action will be

$$u_{ij} = \phi_j S(i, j) - \psi(1 - S(i, j)), \tag{5.1}$$

whereas the robot's payoff will be

$$x_{ij} = -\phi_j S(i, j) + \rho(1 - S(i, j)). \tag{5.2}$$

Note that the players' payoffs x_{ij} and u_{ij} are functions of the patrolling Markov chain P. The attacker, not knowing P, cannot calculate its expected payoffs. Therefore, the attacker uses its belief on the patrolling Markov chain to calculate its payoff. It uses the observations to learn the defender's strategy and uses the learned strategy to estimate its expected payoffs. Let $\hat{P}$ denote the estimated Markov chain transition matrix after time T with the covariance $\texttt{Cov}(\hat{P})$. If u_{ij} was a linear function of P, then the attacker could use $\hat{u}_{ij}$ (obtained by using $\hat{p}_{ij}$ instead of p_{ij} in above expressions) to estimate the expected payoff. Since, u_{ij} is not linear, maximizing $\hat{u}_{ij}$ may not be optimal and the attacker will need to consider higher moments of the estimate. The defender needs the optimal attacker response to optimize its own strategy. Since the Fisher information matrix [67] can easily provide $\texttt{Cov}(\hat{P})$, we approximate the attacker's response by only considering $\texttt{Cov}(\hat{P})$ and ignore the higher moments. We leave the true optimization for a future work. The uncertainty in the estimated strategy can then be propagated to evaluate $\texttt{var}(\hat{u}_{ij})$. Note that $\texttt{Cov}(\hat{P})$ and $\texttt{var}(\hat{u}_{ij})$ are functions of T. Given this data, the problem is to decide whether to play *renege* or *attack-when*(j, i) for some i, j. The efficient frontier for this risk-reward trade-off can be represented as a special case of Markowitz portfolio theory [64] where the attacker has to pick only one action instead of a portfolio. For a parameter $\Lambda \geq 0$, let

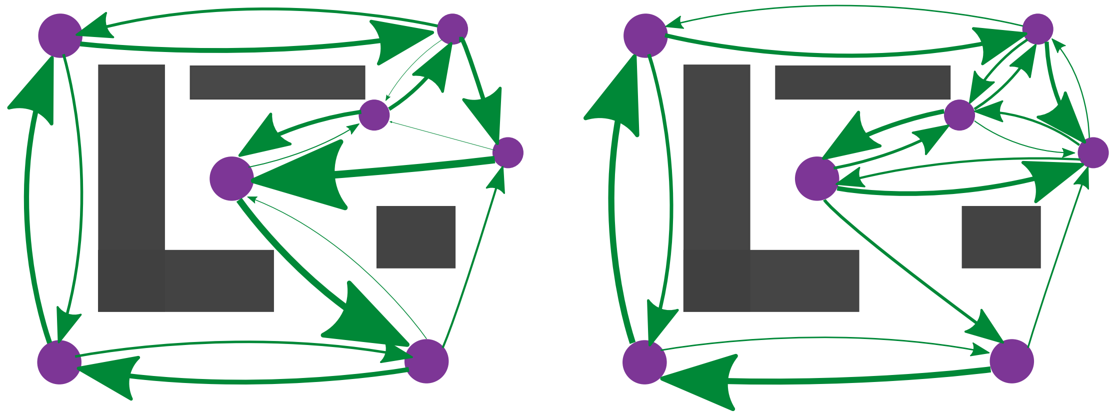


Figure 5.1: The 'hard to attack' (left) and 'hard to learn and attack' (right) strategies in an environment with obstacles. The area of a vertex represents its reward. The thickness of an edge represents the probability of traversing that edge. Notice that the traversals in the strategy on the right are less deterministic as compared to that on the left.

$(i^*, j^*) = \arg\max_{(i,j)}\{\hat{u}_{ij} - \Lambda\mathtt{var}(\hat{u}_{ij})\}$. Then the attacker will play

$$\begin{array}{rl} \textit{attack-when}(j^*, i^*) & \text{if} \quad \hat{u}_{i^*j^*} - \Lambda\mathtt{var}(\hat{u}_{i^*j^*}) > 0 \\ \textit{renege} & \text{otherwise.} \end{array} \tag{5.3}$$

Here Λ models how risk averse the attacker is, e.g. a large value of Λ models a risk averse intruder who will attack a vertex only if it is confident enough in its estimate of the expected payoff. We will assume that the defender does not know the value of Λ but has a probability distribution $g(\lambda)$ over the possible values of Λ.

5.3.2 Defender's Strategy

The defender's strategy should be hard to attack (minimize the worst case attacker's payoff) as well as hard to learn (make attackers renege due to high payoff variance). The following proposition formalizes the defender's payoff to design such a strategy.

Proposition 5.3.2. *If the defender's strategy is given by P and the attacker's strategy is defined by (5.3) and Proposition 5.3.1, then the expected payoff for the defender is given by*

$$\left(1 - \sum_t \int_{\lambda=\max_{i,j} \frac{\hat{u}_{ij}}{var(\hat{u}_{ij})}}^{\infty} g(\lambda) d\lambda q(t)\right) x_{i^*j^*}. \tag{5.4}$$

Proof. Using (5.3), the attacker will renege if T and Λ are drawn such that $\max_{i,j}\{\hat{u}_{ij} - \Lambda\texttt{var}(\hat{u}_{ij})\} \leq 0$. This implies $\Lambda \geq \hat{u}_{ij}/\texttt{var}(\hat{u}_{ij}), \forall i, j$ and hence, the part inside the parentheses in (5.4) is the probability that the attacker will play *attack-when*(j^*, i^*) instead of *renege*. The defender's payoff is zero in case of no attack and, using Equation (5.2), $x_{i^*j^*}$ if the attacker plays *attack-when*(j^*, i^*). □

Figure 5.1 gives an example of a hard to attack strategy that maximizes $x_{\tilde{i},\tilde{j}}$ where $\tilde{i}, \tilde{j} = \arg\max_{i,j}\{u_{ij}\}$ versus a hard to learn and attack strategy that maximizes an approximation of (5.4). Ideally, the defender should find P that maximizes its expected payoff (5.4), but the values of $\hat{u}_{ij}$, $\texttt{var}(\hat{u}_{ij})$ and $x_{i^*j^*}$ depend on a realized path that the attacker observed to calculate these quantities. Hence, maximizing the said expression as it is, is not possible for the defender. We deal with this issue as follows.

Lower Bound on Expected Payoff: Although the defender does not know i^*, j^*, it can lower bound its expected payoff using $\min_{i,j}\{x_{ij}\}$ instead of $x_{i^*j^*}$ in Expression (5.4). This is equivalent to being prepared for the worst case scenario in case the intruder attacks. Note that if learning was not involved, and intruder knew the patrolling strategy P, then the attacker would maximize its own payoff resulting in the defender maximizing $x_{\tilde{i},\tilde{j}}$ where $\tilde{i}, \tilde{j} = \arg\max_{i,j}\{u_{ij}\}$.

Attacker's Estimate: The attacker's estimate of the expected payoff $\hat{u}_{ij}$ is not known to the defender. However, since it knows P, it can calculate the actual value of the attacker's expected payoff u_{ij}. If the attacker is using a consistent estimator, the estimate $\hat{u}_{ij}$ converges to u_{ij}.

Variance in Estimate: Assuming the attacker is using an efficient estimator, the covariance in the attacker's estimate of P is given by the inverse of the Fisher information matrix of the Markov chain represented by $I_N(P)$, where N is the number of transitions observed. The average time taken during one transition of the Markov chain on a weighted graph is given by $\pi(P \odot \mathcal{L})\mathbf{e}$ where $\mathbf{e}$ is a column of ones, $\mathcal{L}$ is the matrix of edge weights, and $P \odot \mathcal{L}$ denotes the element wise product of P and $\mathcal{L}$ [78]. Hence, for large N and T,

$$N \approx \frac{T}{\pi(P \odot \mathcal{L})\mathbf{e}}. \tag{5.5}$$

Therefore, theoretically, the defender can propagate the covariance using the expressions from [14] to find $\texttt{var}(\hat{u}_{ij})$ as a function of T . However, the expression for u_{ij} includes iterative multiplications and summations of correlated variables and in practice this propagation is computationally expensive. Since the defender has access to P, it can use Monte Carlo simulations to estimate these variances. We will denote the variances calculated by the defender by $\texttt{var}(u_{ij})$ although u_{ij} is not a random variable for the defender.

Now, we are in a position to write the optimization problem to be solved by the defender. Let

$$f(P) = -\left(1 - \sum_t \int_{\lambda=\max_{i,j} \frac{u_{ij}}{\mathtt{var}(u_{ij})}}^{\infty} g(\lambda) d\lambda q(t)\right) \min_{i,j} x_{ij}, \tag{5.6}$$

then the optimization problem can be written as follows.

$$\begin{aligned}
\text{minimize:} \quad & f(P) & & \\
\text{subject to:} \quad & \sum_j p_{ij} = 1, & & \forall i \\
& p_{ij} \geq 0, & & \forall i, j \\
& p_{ij} = 0, & & \text{if } (i,j) \notin E \\
& P \text{ has one communicating class.} & & \forall r
\end{aligned} \tag{5.7}$$

Note that we will relax the last constraint in our optimization and instead we verify that the final solution satisfies this constraint. This works well in practice because a Markov chain with more than one communicating class means that the chain cannot go from a state to all other states making $S(i,j) = 1$ for all j for some value of i.

5.4 Computing the Patrolling Strategy

The function $f(P)$ is a non-convex function in general. This can be easily observed by calculating the Hessian of s_{ij} for some i, j. Moreover, the function is also non-smooth in general because it picks the minimum element among x_{ij}. We now characterize the hardness of the problem using a reduction very similar to the one used to prove the hardness of the problem discussed in Chapter 4.

Proposition 5.4.1. *Unless $\mathcal{P} = \mathcal{NP}$, for any $k \geq 1$, there does not exist a polynomial time k-factor approximation algorithm for Problem (5.7).*

Proof. We use a reduction from the instance $G = (V, E)$ of the HAMILTONIAN CYCLE to an instance of our problem with zero optimal value. Pass the graph G to the approximation version of Problem (5.7) with $\ell_{ij} = 1, \forall \{i,j\} \in E$, $\phi = \mathbf{1}$, $\psi = \rho = 0$ and $l = |V|$. Choose any distributions for $g(\lambda)$ and $q(t)$. A Hamiltonian cycle in G can be represented using a transition matrix P and it will ensure that the payoff of the attacker and the defender is zero which is optimal. If no Hamiltonian cycle exists in G, the defender cannot visit all the vertices within l time and the expected payoff for the defender will not be zero. $\square$

5.4.1 Numerical Optimization

We will first show that our objective function is locally Lipschitz under mild assumptions on $g(\lambda)$ and $q(t)$.

Proposition 5.4.2. *If the variance $\texttt{var}(u_{ij})$ is calculated by propagating the inverse of $I_N(P)$, and if the distributions $g(\lambda)$ and $q(t)$ for $t \in [T_{min}, T_{max}]$ are bounded, then the objective function $f(P)$ is locally Lipschitz at each point P.*

Proof. The functions x_{ij} and u_{ij} are continuously differentiable with respect to $p_{xy}, \forall x, y$, and hence, $\min x_{ij}$ is locally Lipschitz. The product of two bounded and locally Lipschitz functions is locally Lipschitz. The part inside the parentheses in Equation (5.6) is a probability and hence is bounded, and x_{ij} is bounded if ϕ and ρ are bounded. So proving that the probability of the attacker not reneging is locally Lipschitz will complete the proof. Using the definition of Fisher information matrix for a Markov chain along with the fact that π_i is a differentiable function of P [86], $u_{ij}/\texttt{var}(\hat{u}_{ij})$ is differentiable[1] and $\max u_{ij}/\texttt{var}(\hat{u}_{ij})$ is Lipschitz. It can be easily shown that $\int_{y(x)}^{c} g(\lambda)d\lambda$ and $\sum_{t=T_{\min}}^{T_{\max}} z(x,t)q(t)$ are locally Lipschitz for bounded $g(\lambda)$ and $q(t)$ if $y(x)$ and $z(x,t)$ are locally Lipschitz. $\square$

Given that the objective function is non-convex, non-smooth and locally Lipschitz, and the constraint set is convex, Mesh Adaptive Direct Search method (MADS) can be used to find a Clarke Stationary point [11]. Thus, we can apply standard gradient-free optimization techniques [72] to compute a patrolling strategy. Therefore we use the direct search method presented in Algorithm 5 to find a patrolling Markov chain.

5.5 Simulations

In this section we demonstrate that when intruders have limited time to learn, solving for the patrolling strategies that are hard to learn along with being hard to attack can be more useful than optimizing for the worst case.

[1]If $\texttt{var}(\hat{u}_{ij}) = 0$ for $u_{ij} > 0$, $f(P) = -\min x_{ij}$ is locally Lipschitz.

Strategy comparison: We conducted several experiments with n vertices placed in a square plane including obstacles. Each vertex was connected to a random number of its nearest neighbors and the euclidean distances between the vertices were rounded off to get integer edge weights. This construction of the graph is similar to the Probabilistic Roadmaps that are used for robotic path planning. The problem data for two of these instances is as follows.

Instance 1	**Instance 2**
$n = 7$	$n = 10$
$L = 8$	$L = 10$
$\psi = \rho = 5$	$\psi = \rho = 20$
$\phi = [10, 5, 10, 5, 10, 10, 5]$	$\phi_i \sim \mathcal{U}[20, 50]$
$T \sim \mathcal{U}[90, 110]$	$T \sim \mathcal{U}[200, 1000]$
$\Lambda \sim \mathcal{U}[0, 10]$	$\Lambda \sim \mathcal{U}[0, 10]$

We compare attacker's average payoffs $v_{\texttt{lim}}$ and v_∞ for the 'hard to learn and attack' strategy $P_{\texttt{lim}}$ (obtained by minimizing $f(P)$) and 'hard to attack' strategy P_∞ (obtained by minimizing $-x_{\tilde{i},\tilde{j}}$ where $\tilde{i}, \tilde{j} = \arg\max_{i,j}\{u_{ij}\}$). Since $\rho = \psi$, it is a zero sum game and the defender's payoff is symmetric. These average payoffs are calculated by simulating 100 attacks for each of these strategies and then taking the average of the payoff acquired by each of these attacks. The parameters T and Λ for these attackers were drawn randomly from their respective distributions.

Figure 5.1 shows the graph for Instance 1 along with both the strategies. For $P_{\texttt{lim}}$, 67% attackers renege and $v_{\texttt{lim}} = 1.96$, as compared to 1% reneging attackers and $v_\infty = 3.88$ for P_∞. If the intruder had perfect knowledge, the defender payoffs would be $v_\infty = 4.25$ and $v_{\texttt{lim}} = 6.79$. It is to be expected since the optimization for P_∞ assumes the model in which the attackers will not renege due to lack of knowledge about the patrolling policy.

For Instance 2, we classify the attackers into different types based on the drawn parameters and show the average payoffs in Table 5.1. For example, the attackers with Λ between 6.6 and 10 are risk averse attackers. Similarly the attackers with t from 200 to 466 are given less time to learn. Note that the strategy was calculated using $T \sim \mathcal{U}[200, 1000]$ and $\Lambda \sim \mathcal{U}[0, 10]$, and the attackers were classified during simulation to show the effect of T and Λ on attacker's payoffs.

For the $P_{\texttt{lim}}$ patrolling strategy, 33 out of 100 attackers reneged. Some entries in the table for this strategy are zero because all of the attackers of that category reneged when this patrolling strategy was used. Note that for P_∞, corresponding entries are non-zero,

Table 5.1: The comparison of the attacker payoffs for the patrolling strategies. The payoffs are rounded to the nearest integer.

Λ	$v_{\texttt{lim}}$	v_∞	$v_{\texttt{lim}}$	v_∞	$v_{\texttt{lim}}$	v_∞	
$6.6 - 10$	0	28	0	33	38	32	
$3.3 - 6.6$	0	31	37	17	37	27	
$0 - 3.3$	32	21	37	29	37	46	
	$200 - 466$		$467 - 733$		$734 - 1000$		T

meaning that attackers decided to attack. In fact, none of the 100 attackers reneged for P_∞. The fraction of reneging attackers agrees with our expression of probability of reneging used in (5.6). It can also be observed from the table that the risk averse attackers that are given less time to learn are more likely to renege. The average attacker payoff for $P_{\texttt{lim}}$ is 25 whereas it is 29 for P_∞. Hence, using the strategy for the attackers with limited learning time resulted in a 13.7% decrease of the attacker payoff. Also note that the starting point of the optimizations, which was a randomly generated Markov chain resulted in an average attacker payoff of 36.

Algorithm comparison: To evaluate the performance of Algorithm 5, we compare it with the MATLAB patternsearch function that is an adaptive mesh based direct search method [72]. For this experiment random graphs of different sizes were generated as described above in an obstacle free environment and the problem variables were generated as follows.

$$\begin{aligned}
&L = 3/4(\text{MST length of graph}) \\
&\phi \sim \mathcal{U}[20, 50] \\
&\psi = \rho = 20
\end{aligned}$$

For the purpose of this comparison, $-x_{\tilde{i},\tilde{j}}$ was minimized, i.e., it was assumed that the attacker already knows defender's patrolling strategy. The algorithms were timed out at 10 minutes. The attacker's expected payoff $-x_{\tilde{i},\tilde{j}}$ for different sizes of the graph is reported in Table 5.2, which shows that Algorithm 5 performs better in terms of final objective value and runtime.

Table 5.2: Comparison of MATLAB's *patternsearch* function with Algorithm 5.

	patternsearch		Algorithm 5	
n	$-x_{\tilde{i},\tilde{j}}$	runtime	$-x_{\tilde{i},\tilde{j}}$	runtime
10	44.7	108	32.1	26
30	46.4	600	38.9	600
50	47.8	600	44.5	600

Chapter 6

Conclusions and Future Work

In this thesis we studied four related persistent monitoring problems. We started with the problem of finding the minimum number of robots to satisfy the latency constraints on the vertices of a graph. The latency constraints represented the maximum amount of time the robots were allowed to stay away from a vertex. For this problem, we gave an $O(\log \rho)$ approximation algorithm, where ρ is the ratio of the maximum and minimum latency constraints. We then proposed three different heuristic algorithms to solve the problem. Our simulation results showed that the algorithm that visits more vertices on its way to the greedily picked vertex performed better in terms of the number of robots required. The results also showed that although the Orienteering based heuristic algorithm had a larger run time, it performed much better than the approximation algorithm in terms of the objective value.

The second problem we considered was very much related to the first problem, where instead of finding the minimum number of robots to satisfy latency constraints, our objective was to minimize the maximum weighted latency in the graph for a given number of robots. For this problem we gave an algorithm which has a performance guarantee as a function of the optimal cost of the single robot instance of the problem. We then presented some results that relate this problem to the problem of satisfying latency constraints.

Next, we considered the problem of finding patrolling paths for a team of robots when the durations of the events were not necessarily fixed and the objective was to maximize the expected reward of observing the events. For this problem we motivated randomized patrolling strategies, and evaluated the expected reward as a function of the Markov chain transition matrices used by the robots for patrolling. We presented some properties of the problem, one of which was submodularity and we leveraged this submodularity to present an approximation algorithm for the case where the event durations were fixed. Moreover, we also presented a centralized and a distributed online algorithm to solve the problem. We also considered adversarial events and characterized the objective value for those events.

The last problem we studied considered a patrolling game where the attacker learns the patrolling strategy for a limited amount of time before deciding whether to attack or renege. We showed that in the cases where the attackers have limited time, designing strategies that are hard to learn along with being hard to attack can be useful.

In the following section we discuss some possible directions for future work.

6.1 Future Work

We will discuss the future work directions for each of the problems studied in this thesis separately.

6.1.1 Persistent Monitoring With Latency Constraints

The heuristic algorithms presented in the thesis work for partitioned solutions. An interesting direction for future work would be to use Team Orienteering problem in order to construct solutions where the robots can share the vertices. Figuring out the time budget for each of the different walks, and their target vertices might be done using a greedy approach as well, where each robot looks at the combination of the distance to a vertex and the time to expiry for that vertex and selects a target vertex accordingly.

Another possible direction is to include the fuel constraints for each of the robots, where each robot needs to refuel after travelling for some fixed amount of time from designated refuelling depots. Adding the refuelling constraints will be very useful in practical applications for persistent monitoring, since in such tasks the robots spend extended amounts of time in the field and need to be refuelled repeatedly. The fuel constraint can be easily added in the single robot problem to satisfy latency constraints, where the fueling depot has a latency constraint equal to the fuel capacity of the robot. However, for the case of multiple robots, that latency constraint could be satisfied by any of the robots, and all the robots would not be necessarily refuelled. A possible solution to this problem would be to extend the Orienteering based heuristic algorithm, where at each step the robot keeps track of the nearest refuelling depot and its current fuel level.

For the problem of minimizing the maximum weighted latency, we have presented an algorithm in this thesis that returns a solution with the cost bounded above by a factor of the optimal cost of the single robot problem instance. Finding a relation between the optimal cost for a single robot solution and the optimal cost of multi-robot solution will be helpful in establishing an approximation ratio for the proposed algorithm.

6.1.2 Multi-Robot Stochastic Patrolling

Incorporating the refuelling constraints for the multi-robot stochastic patrolling problem is also an interesting direction for future work. We will have to deviate from the model of using Markov chains to include the refuelling constraints, because the refuelling constraints are hard constraints, and need to be met, whereas using a random path will result in a probability of the robot being refuelled in proper time. Since we want this probability to be one, we will have to look at a combination of deterministic and random patrolling paths. Assuming that refuelling is not done as frequently as visiting the other vertices of the graph, the robot can switch between random patrolling paths for detecting adversarial events, and deterministic paths when it requires to be refuelled.

6.1.3 Adversarial Events with Limited Observation Time

In [9] the authors consider an attacker that observes the defender for a limited time before attacking. Although they consider assigning resources to static locations, theoretically their idea of using a belief for intruders can be extended to the patrolling problem. For our problem, the defender is using a Markov chain to patrol and the attacker observes the path of the robot to learn the patrolling strategy. The transition matrix of a Markov chain on a graph with n vertices can be interpreted as n independent Multinomial distributions. The Dirichlet distribution is the conjugate prior to the Multinomial distribution, i,e, if the prior belief of the attacker on a row of the Markov chain is represented as a Dirichlet distribution, the posterior belief after applying the Bayes' law will also be a Dirichlet distribution. Hence, we can assume that the intruder's belief about each of the n rows of the patrolling Markov chain transition matrix is a Dirichlet distribution. After the intruder observes a transition (i, j), it will update its belief for the i-th Multinomial distribution. Since, all the rows of the Markov chain transition matrix are independent, the overall belief of the intruder is the product of individual Dirichlet distributions. From the intruder's perspective, after observing the patroller for T time, the probability that the defender is using Markov chain $\tilde{P}$ is represented as $\mathbb{P}\Big[\text{ Defender is using } \tilde{P}|T\Big]$. Then the expected payoff for the intruder playing action *attack-when*(j, i) is given by

$$\int_{\tilde{P}\in\mathcal{C}} u_{ij}(\tilde{P})\mathbb{P}\left[\text{ Defender is using } \tilde{P}|T\right] d\tilde{P}$$

where u_{ij} is given in Equation 5.1 and written as $u_{ij}(\tilde{P})$ to emphasize that u_{ij} is a function of the Markov chain transition matrix. The integral is taken over the whole set of feasible Markov chains.

In [9], the payoff is a linear function of the variables that need to be estimated. Therefore, the expected reward can be calculated as the reward obtained by using the mean of the belief. In our problem, u_{ij} is a recursive, non-linear, non-convex function of the estimated Markov chain. Hence, calculating this integral is computationally very expensive, and that is why we used Markowitz portfolio theory in Chapter 5 to estimate the behavior of the intruder. A future work direction is to find a method to evaluate the above mentioned integral efficiently.

Another possible direction for future work is to consider a cost for observations instead of fixed learning time T and find the optimal time to learn the patroller's strategy from the intruder's perspective.

References

[1] Toronto police service public safety data portal. http://data.torontopolice.on.ca/.

[2] Jose Joaquin Acevedo, Begoña C Arrue, Ivan Maza, and Aníbal Ollero. Cooperative perimeter surveillance with a team of mobile robots under communication constraints. In *IEEE International Conference on Intelligent Robots and Systems*, pages 5067–5072, 2013.

[3] Pushkarini Agharkar, Rushabh Patel, and Francesco Bullo. Robotic surveillance and markov chains with minimal first passage time. In *IEEE Conference on Decision and Control*, pages 6603–6608, 2014.

[4] Noa Agmon, Gal A Kaminka, and Sarit Kraus. Multi-robot adversarial patrolling: facing a full-knowledge opponent. *Journal of Artificial Intelligence Research*, 42:887–916, 2011.

[5] Tauhidul Alam, Matthew Edwards, Leonardo Bobadilla, and Dylan Shell. Distributed multi-robot area patrolling in adversarial environments. In *International Workshop on Robotic Sensor Networks*, Seattle, WA, USA, April 2015.

[6] Soroush Alamdari, Elaheh Fata, and Stephen L Smith. Persistent monitoring in discrete environments: Minimizing the maximum weighted latency between observations. *The International Journal of Robotics Research*, 33(1):138–154, 2014.

[7] Steve Alpern, Alec Morton, and Katerina Papadaki. Patrolling games. *Operations research*, 59(5):1246–1257, 2011.

[8] Bo An, Matthew Brown, Yevgeniy Vorobeychik, and Milind Tambe. Security games with surveillance cost and optimal timing of attack execution. In *Autonomous Agents and Multi-Agent Systems*, pages 223–230, 2013.

[9] Bo An, David Kempe, Christopher Kiekintveld, Eric Shieh, Satinder Singh, Milind Tambe, and Yevgeniy Vorobeychik. Security games with limited surveillance. In *AAAI Conference on Artificial Intelligence*, 2012.

[10] Ahmad Bilal Asghar and Stephen L Smith. Stochastic patrolling in adversarial settings. In *IEEE American Control Conference*, pages 6435–6440, 2016.

[11] Charles Audet and John E Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on optimization*, 17(1):188–217, 2006.

[12] Nicola Basilico, Nicola Gatti, and Francesco Amigoni. Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. *Artificial Intelligence*, 184:78–123, 2012.

[13] Andreas Bircher, Kostas Alexis, Michael Burri, Philipp Oettershagen, Sammy Omari, Thomas Mantel, and Roland Siegwart. Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics. In *IEEE International Conference on Robotics and Automation*, pages 6423–6430, 2015.

[14] George W Bohrnstedt and Arthur S Goldberger. On the exact covariance of products of random variables. *Journal of the American Statistical Association*, 64(328):1439–1442, 1969.

[15] Stephen Boyd, Persi Diaconis, and Lin Xiao. Fastest mixing markov chain on a graph. *SIAM Review*, 46(4):667–689, 2004.

[16] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

[17] Tomáš Brázdil, Petr Hliněný, Antonín Kučera, Vojtěch Řehák, and Matúš Abaffy. Strategy synthesis in adversarial patrolling games. *arXiv preprint arXiv:1507.03407*, 2015.

[18] Giulio Brogi, Luca Pietranera, and Francesco Frau. Forest surveillance and monitoring system for the early detection and reporting of forest fires, 1998. US Patent 5,734,335.

[19] Zdzislaw Burda, Jarek Duda, Jean-Marc Luck, and Bartek Waclaw. Localization of the maximal entropy random walk. *Physical review letters*, 102(16):160602, 2009.

[20] Gonçalo Cabrita, Pedro Sousa, Lino Marques, and A De Almeida. Infrastructure monitoring with multi-robot teams. In *International Conference on Intelligent Robots and Systems*, pages 18–22, 2010.

[21] Nannan Cao, Kian Hsiang Low, and John M Dolan. Multi-robot informative path planning for active sensing of environmental phenomena: A tale of two algorithms. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 7–14, 2013.

[22] I-Ming Chao, Bruce L Golden, and Edward A Wasil. The team orienteering problem. *European Journal of Operational Research*, 88(3):464–474, 1996.

[23] Chandra Chekuri, Nitish Korula, and Martin Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms*, 8(3):23, 2012.

[24] Yann Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *International Conference on Intelligent Agent Technology*, pages 302–308. IEEE, 2004.

[25] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon University Pittsburgh, 1976.

[26] Nicos Christofides and John E Beasley. The period routing problem. *Networks*, 14(2):237–256, 1984.

[27] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[28] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Virkumar Vazirani. *Algorithms*. McGraw-Hill Higher Education New York, 2008.

[29] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[30] Nir Drucker. Cyclic routing of unmanned aerial vehicles. Master's thesis, Technion – Israel Institute of Technology, Israel, 2014.

[31] Nir Drucker, Michal Penn, and Ofer Strichman. Cyclic routing of unmanned aerial vehicles. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 125–141. Springer, 2016.

[32] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *International conference on Machine learning*, pages 272–279, 2008.

[33] Matthew Dunbabin and Lino Marques. Robots for environmental monitoring: Significant advancements and applications. *IEEE Robotics and Automation Magazine*, 19(1):24–39, 2012.

[34] Joseph W Durham, Ruggero Carli, Paolo Frasca, and Francesco Bullo. Discrete partitioning and coverage control for gossiping robots. *IEEE Transactions on Robotics*, 28(2):364–378, 2011.

[35] Yehuda Elmaliach, Noa Agmon, and Gal A Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Mathematics and Artificial Intelligence*, 57(3-4):293–320, 2009.

[36] Drew Fudenberg and Jean Tirole. *Game Theory*. Cambridge, MA: MIT press, 1991.

[37] Bruce L Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics*, 34(3):307–318, 1987.

[38] Jeremy Grace and John Baillieul. Stochastic strategies for autonomous robotic surveillance. In *IEEE Conference on Decision and Control and European Control Conference*, pages 2200–2205, 2005.

[39] Geoffrey Grimmett and David Stirzaker. *Probability and random processes*. Oxford University Press, 2001.

[40] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

[41] Gurobi. Gurobi optimizer, 2018.

[42] Erez Hartuv, Noa Agmon, and Sarit Kraus. Scheduling spare drones for persistent task performance under energy constraints. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 532–540, 2018.

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

[44] Hsi-Ming Ho and Joël Ouaknine. The cyclic-routing UAV problem is pspace-complete. In *International Conference on Foundations of Software Science and Computation Structures*, pages 328–342. Springer, 2015.

[45] Jarrod C Hodgson, Shane M Baylis, Rowan Mott, Ashley Herrod, and Rohan H Clarke. Precision wildlife monitoring using unmanned aerial vehicles. *Scientific Reports*, 6(1):1–7, 2016.

[46] Peter F Hokayem, Dusan Stipanovic, and Mark W Spong. On persistent coverage control. In *IEEE Conference on Decision and Control*, pages 6130–6135, 2007.

[47] Frank Imeson and Stephen L Smith. An SMT-based approach to motion planning for multiple robots with complex constraints. *IEEE Transactions on Robotics*, 35(3):669–684, 2019.

[48] Rishabh K Iyer and Jeff A Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *Advances in Neural Information Processing Systems*, pages 2436–2444, 2013.

[49] Manish Jain, Dmytro Korzhyk, Ondřej Vaněk, Vincent Conitzer, Michal Pěchouček, and Milind Tambe. A double oracle algorithm for zero-sum security games on graphs. In *Autonomous Agents and Multi-Agent Systems*, pages 327–334, 2011.

[50] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[51] John G Kemeny and J Laurie Snell. *Markov chains.* Springer-Verlag, New York, 1976.

[52] Christopher Kiekintveld, Towhidul Islam, and Vladik Kreinovich. Security games with interval uncertainty. In *International conference on Autonomous agents and multi-agent systems*, pages 231–238, 2013.

[53] Dmytro Korzhyk, Vincent Conitzer, and Ronald Parr. Solving Stackelberg games with uncertain observability. In *Autonomous Agents and Multi-Agent Systems*, pages 1013–1020, 2011.

[54] Jonathan Las Fargeas, Baro Hyun, Pierre Kabamba, and Anouck Girard. Persistent visitation under revisit constraints. In *International Conference on Unmanned Aircraft Systems*, pages 952–957, 2013.

[55] Aron Laszka, Yevgeniy Vorobeychik, and Xenofon Koutsoukos. Resilient observation selection in adversarial settings. In *IEEE Conference on Decision and Control*, pages 7416–7421, 2015.

[56] Eugene L Lawler. The traveling salesman problem: a guided tour of combinatorial optimization. *Wiley-Interscience Series in Discrete Mathematics*, 1985.

[57] Kian Seng Lee, Mark Ovinis, T Nagarajan, Ralph Seulin, and Olivier Morel. Autonomous patrol and surveillance system using unmanned aerial vehicles. In *IEEE International Conference on Environment and Electrical Engineering*, pages 1291–1297, 2015.

[58] Thomas Lemaire, Rachid Alami, and Simon Lacroix. A distributed tasks allocation scheme in multi-uav context. In *IEEE International Conference on Robotics and Automation*, pages 3622–3627, 2004.

[59] Adam N Letchford, Saeideh D Nasiri, and Dirk Oliver Theis. Compact formulations of the steiner traveling salesman problem and related problems. *European Journal of Operational Research*, 228(1):83–92, 2013.

[60] Robert Michael Lewis, Virginia Torczon, and Michael W Trosset. Direct search methods: then and now. *Journal of Computational and Applied Mathematics*, 124(1-2):191–207, 2000.

[61] Kyle Y Lin, Michael P Atkinson, Timothy H Chung, and Kevin D Glazebrook. A graph patrol problem with random attack times. *Operations Research*, 61(3):694–710, 2013.

[62] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[63] Parikshit Maini, Kevin Yu, PB Sujit, and Pratap Tokekar. Persistent monitoring with refueling on a terrain using a team of aerial and ground robots. In *IEEE International Conference on Intelligent Robots and Systems*, pages 8493–8498, 2018.

[64] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[65] Neil Mathew, Stephen L Smith, and Steven L Waslander. Multirobot rendezvous planning for recharging in persistent tasks. *IEEE Transactions on Robotics*, 31(1):128–142, 2015.

[66] Luis Merino, Fernando Caballero, J Ramiro Martínez-De-Dios, Iván Maza, and Aníbal Ollero. An unmanned aircraft system for automatic forest fire monitoring and measurement. *Journal of Intelligent and Robotic Systems*, 65(1-4):533–548, 2012.

[67] Kevin P Murphy. *Machine learning: A probabilistic perspective.* MIT press, 2012.

[68] USR Murty and Adrian Bondy. *Graph Theory.* Springer, 2008.

[69] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming*, 14(1):265–294, 1978.

[70] Nikhil Nigam, Stefan Bieniawski, Ilan Kroo, and John Vian. Control of multiple uavs for persistent surveillance: algorithm and flight test results. *IEEE Transactions on Control Systems Technology*, 20(5):1236–1251, 2012.

[71] Nikhil Nigam and Ilan Kroo. Persistent surveillance using multiple unmanned air vehicles. In *IEEE Aerospace Conference*, pages 1–14, 2008.

[72] Jorge Nocedal and Stephen J Wright. *Numerical Optimization.* Springer, 2006.

[73] José Manuel Palacios-Gasós, Eduardo Montijano, Carlos Sagues, and Sergio Llorente. Multi-robot persistent coverage using branch and bound. In *IEEE American Control Conference*, pages 5697–5702, 2016.

[74] José Manuel Palacios-Gasós, Eduardo Montijano, Carlos Sagüés, and Sergio Llorente. Multi-robot persistent coverage with optimal times. In *IEEE Conference on Decision and Control*, pages 3511–3517, 2016.

[75] José Manuel Palacios-Gasós, Danilo Tardioli, Eduardo Montijano, and Carlos Sagüés. Equitable persistent coverage of non-convex environments with graph-based planning. *The International Journal of Robotics Research*, 38(14):1674–1694, 2019.

[76] Praveen Paruchuri, Jonathan P Pearce, Milind Tambe, Fernando Ordonez, and Sarit Kraus. An efficient heuristic approach for security against multiple adversaries. In *Autonomous Agents and Multi-Agent Systems*, page 181, 2007.

[77] Fabio Pasqualetti, Antonio Franchi, and Francesco Bullo. On cooperative patrolling: Optimal trajectories, complexity analysis, and approximation algorithms. *IEEE Transactions on Robotics*, 28(3):592–606, 2012.

[78] Rushabh Patel, Pushkarini Agharkar, and Francesco Bullo. Robotic surveillance and Markov chains with minimal weighted kemeny constant. *IEEE Transactions on Automatic Control*, 60(12):3156–3167, Dec 2015.

[79] Dan O Popa, Koushil Sreenath, and Frank L Lewis. Robotic deployment for environmental sampling applications. In *IEEE International Conference on Control and Automation*, volume 1, pages 197–202, 2005.

[80] David Portugal and Rui Rocha. MSP algorithm: Multi-robot patrolling based on territory allocation using balanced graph partitioning. In *ACM Symposium on Applied Computing*, pages 1271–1276, 2010.

[81] David Portugal and Rui P Rocha. Multi-robot patrolling algorithms: examining performance and scalability. *Advanced Robotics*, 27(5):325–336, 2013.

[82] Michael Melholt Quottrup, Thomas Bak, and RI Zamanabadi. Multi-robot planning: A timed automata approach. In *IEEE International Conference on Robotics and Automation*, volume 5, pages 4417–4422, 2004.

[83] Mohammed Rahimi, Mark Hansen, William J Kaiser, Gaurav S Sukhatme, and Deborah Estrin. Adaptive sampling for environmental field estimation using robotic sensors. In *IEEE International Conference on Intelligent Robots and Systems*, pages 3692–3698, 2005.

[84] Mike Roberts, Debadeepta Dey, Anh Truong, Sudipta Sinha, Shital Shah, Ashish Kapoor, Pat Hanrahan, and Neel Joshi. Submodular trajectory optimization for aerial 3D scanning. In *International Conference on Computer Vision*, pages 5334–5343, 2017.

[85] Tiago Sak, Jacques Wainer, and Siome Klein Goldenstein. Probabilistic multiagent patrolling. In *Brazilian Symposium on Artificial Intelligence*, pages 124–133. Springer, 2008.

[86] Paul J Schweitzer. Perturbation theory and finite markov chains. *Journal of Applied Probability*, 5(2):401–413, 1968.

[87] Ryan N Smith, Mac Schwager, Stephen L Smith, Burton H Jones, Daniela Rus, and Gaurav S Sukhatme. Persistent ocean monitoring with underwater gliders: Adapting sampling resolution. *Journal of Field Robotics*, 28(5):714–741, 2011.

[88] Stephen L Smith, Mac Schwager, and Daniela Rus. Persistent robotic tasks: Monitoring and sweeping in changing environments. *IEEE Transactions on Robotics*, 28(2):410–426, 2011.

[89] Kunal Srivastava, Dušan M Stipanović, and Mark W Spong. On a stochastic robotic surveillance problem. In *IEEE Conference on Decision and Control and Chinese Control Conference*, pages 8567–8574, 2009.

[90] Paolo Toth and Daniele Vigo. *The vehicle routing problem.* SIAM, 2002.

[91] John N Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22(3):263–282, 1992.

[92] Alphan Ulusoy, Stephen L Smith, Xu Chu Ding, Calin Belta, and Daniela Rus. Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research*, 32(8):889–911, 2013.

[93] Yevgeniy Vorobeychik, Bo An, Milind Tambe, and Satinder P Singh. Computing solutions in infinite-horizon discounted adversarial patrolling games. In *ICAPS*, 2014.

[94] Zhen Wang, Yue Yin, and Bo An. Computing optimal monitoring strategy for detecting terrorist plots. In *AAAI Conference on Artificial Intelligence*, pages 637–643, 2016.

[95] Jingjin Yu, Sertac Karaman, and Daniela Rus. Persistent monitoring of events with stochastic arrivals at multiple stations. *IEEE Transactions on Robotics*, 31(3):521–535, 2015.

[96] Wei Yu and Zhaohui Liu. Improved approximation algorithms for some min-max and minimum cycle cover problems. *Theoretical Computer Science*, 654:45–58, 2016.

[97] Xi Yu, Sean B Andersson, Nan Zhou, and Christos G Cassandras. Optimal dwell times for persistent monitoring of a finite set of targets. In *IEEE American Control Conference*, pages 5544–5549, 2017.

[98] Xi Yu, Sean B Andersson, Nan Zhou, and Christos G Cassandras. Optimal visiting schedule search for persistent monitoring of a finite set of targets. In *IEEE American Control Conference*, pages 4032–4037, 2018.

[99] Seung-kook Yun and Daniela Rus. Distributed coverage with mobile robots on a graph: locational optimization and equal-mass partitioning. *Robotica*, 32(2):257–277, 2014.

[100] Nan Zhou, Xi Yu, Sean B Andersson, and Christos G Cassandras. Optimal event-driven multiagent persistent monitoring of a finite set of data sources. *IEEE Transactions on Automatic Control*, 63(12):4204–4217, 2018.