UNIVERSITY OF CALIFORNIA, MERCED

On Patrolling Security Games, Modeling Agents, and Computing Viable Strategies

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering and Computer Science

by

Carlos Diaz Alvarenga

Committee in charge:

Professor Stefano Carpin, UC Merced, Chair
Professor Nicola Basilico, University of Milan
Professor Wan Du, UC Merced
Professor David Noelle, UC Merced

June 2024

The dissertation of Carlos Diaz Alvarenga is approved, and it is acceptable in quality and form for publication:

_____

_____

_____

_____

Chair

University of California, Merced

June 2024

DEDICATION

To my loving wife who has always supported me from the very first day we met and to my family who continue to help me grow as a person, brother, cousin, nephew, uncle and son.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Stefano Carpin for his continued support and guidance. Without his help and mentorship none of this would have been possible. Next, I'd like to thank professor Nicola Basilico who provided input for every publication found within this thesis. I would not have been able to complete this thesis without his advice and instruction. Additionally, I'd like to thank my thesis committee for their valuable insights and suggestions.

A warm and grateful thanks to my all of my lab-mates is necessary as well: Lorenzo, Thomas, Jose, Shuo, Marcos, Azin, and Giacomo. Their friendship and advice helped me along the way. Also, I'd like to thank the Merced community: friends, instructors, graduate students, etc. After 10 years in Merced this good-bye is bitter sweet, but the city and close-knit community has left an impression on me for the rest of my life.

Finally, I'd like to thank my wife and family members. I am the first person in my family to receive a doctorate. I know they are proud of me. Completing this doctorate is as much theirs as it is mine. It takes a village to raise a child and I am grateful for them everyday.

I also acknowledge partial financial support from the NSF under grant DGE-1633722, from the University of California under a Climate Adaptation Grant, and from the UC Merced School of Engineering through multiple teaching assistant assignments.

RELATED PUBLICATIONS

C. Diaz Alvarenga, N. Basilico, S. Carpin. "Combining Coordination and Independent Coverage in Multirobot Graph Patrolling". Proceedings of the 2024 IEEE International Conference on Robotics and Automation, 4413-4419.

C. Diaz Alvarenga, N. Basilico, S. Carpin. "Learning Generalizable Patrolling Strategies through Domain Randomization of Attacker Behaviors". Proceedings of the 2024 IEEE International Conference on Robotics and Automation, 4406-4412.

C. Diaz Alvarenga, N. Basilico, S. Carpin. "Multirobot Patrolling Against Adaptive Opponents with Limited Information". Proceedings of the 2020 IEEE International Conference on Robotics and Automation, 2486-2492.

C. Diaz Alvarenga, N. Basilico, S. Carpin. "Time-Varying Graph Patrolling Against Attackers with Locally Limited and Imperfect Observation Models". Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, 4869-4876.

C. Diaz Alvarenga, N. Basilico, S. Carpin. "Delayed and Time-Variant Patrolling Strategies against Attackers with Local Observation Capabilities". Proceedings of the 2019 International Conference on Autonomous Agents and Multiagent Systems, 1928-1930.

ABSTRACT OF THE DISSERTATION

On Patrolling Security Games, Modeling Agents, and Computing Viable Strategies

by

Carlos Diaz Alvarenga
Doctor of Philosophy in Electrical Engineering and Computer Science
University of California Merced, June 2024
Professor Stefano Carpin, UC Merced, Chair

With increasing levels of intelligence and automation mobile robots are now an enabling technology for autonomous patrolling of indoor and outdoor environments. Patrolling is a repetitive and potentially dangerous task whose execution costs can be mitigated by deploying surveillance robots in the area of interest. Because these systems are designed to be autonomous, the high-level planning of the surveillance activities emerges as one of the most critical challenges to achieving good performance and, ultimately, detecting and preventing malicious activities in the environment. Issues like how to plan efficient paths, where and when to schedule surveillance actions, and how to coordinate with teammates have been tackled by models encoding some environment representation and assumptions on agents capabilities and behaviors

This dissertation itself addresses some of the challenges in computing patrolling schedules for an agent, or team of agents, working against an intruder with limited observablility of the environment. First, we present an overview of related literature associated with the specific types of problems discussed throughout. This includes: techniques for single agent instances, techniques for multi-robot instances, machine learning for patrolling, and common metrics used for gauging a defender's performance. It is, then, divided into several chapters which each address different aspects of the aforementioned challenges.

Chapter 1

Introduction

1.1 Purpose

1.1.1 Why Security Games?

Modelling interactions between agents is a long standing research endeavour that spans many disciplines including psychology [89], cognitive science [88], computer science [92], and others. Generally, researchers are looking to: find emergent behaviors, forecast the consequences of some long running exchange between agents, or correlate individual agent actions to broader collective goals. Broadly speaking, our work falls into this category, however, with a specific application in mind. Ultimately, we study the theoretical implications and practical designs for systems aimed at *deterrence*, i.e. the act of discouraging a malicious event. Thus our work in broad terms concerns itself with computing protection strategies on one end and simulating malevolent actors on the other. Of the many applications in this area, the most interesting are the use cases that most consider societal impact, both negative and positive. Figure 1.1 presents some robot platforms that have been used for urban search and rescue, as well as for patrolling, in indistrial and commercial settings.

As recently as 2019 BBC writer Zoe Cormier [30] comments, "In a time when climate change is melting the Arctic and setting California on fire, ..., it can still come as a shock that illegal hunting of wild animals for profit is still one of the

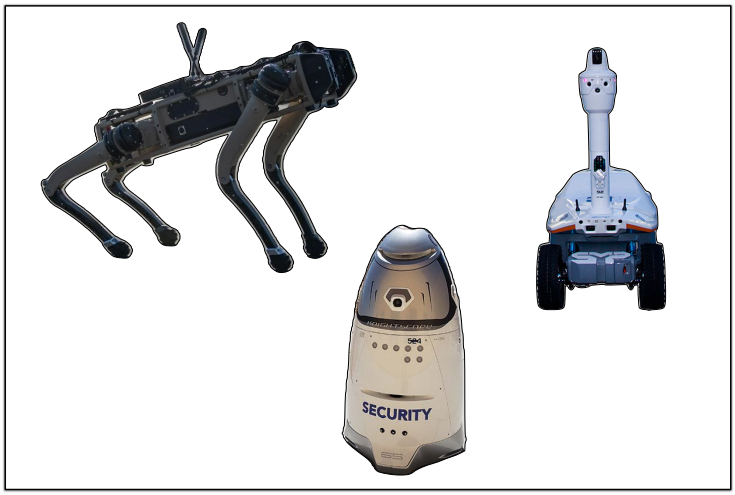


Figure 1.1: Some robot platforms that have been used for patrolling or urban search and rescue [31, 53, 94]. Example tasks include object inspection in unknown environments, collaborative mapping in harsh conditions, and patrolling a parking garage for parking infractions. Individual images grabbed from the google images search engine. The KnightScope [53] robot (bottom middle) in particular has been deployed in parking lots in San Bernardino County, CA and was recently authorized by the US government to operate in other parts of the country.

Figure 1.2: According to the BBC [30], a kilogram of rhinoceros horn can fetch up to 50,000 euros on the black market.

biggest drivers of extinction." Figure 1.2 is that a rhino generated by machine learning models. It is, also, evident that the harms caused by poaching are potentially catastrophic and avoidable. This specific use case has sparked the area known as Green Security Games [97]. Pushed by groups like AI for Social Good (Google) which is led by professor Milind Tambe [34, 93], the team's goals center around designing and implementing AI systems to help rangers counteract illegal poaching in a given area. Security Game theory allows the researchers to bridge the gap with conservationists and consequently help to protect wildlife.

The work presented in this thesis focuses mostly on the theoretical aspects of the interactions between agents. All the findings discussed in the subsequent chapters were obtained by modelling and implementing agents in our own security game simulator written in python. In this way, we hope to share our insights into strong agent strategy and agent design.

1.1.2 A Single Agent Patrolling

In [20], the authors provide a general definition of *patrolling*: "the act of walking or traveling around an area, at regular intervals, in order to protect or supervise it". Moreover, the discipline of *robotic patrolling* concerns itself with the study and application of robots, or autonomous agents, to perform patrols. Recent advances in robotics [54, 58, 65, 85, 97] and artificial intelligence, have sparked renewed interest in the already mature area of robotic patrolling. Many formulations and abstractions exist for the problem and this work will be focusing on representations that abstract the environment as a weighted un-directed graph, $G = (V, E, d)$. Here the set V is known as the set of *target locations*, or areas of interest, the edge set, E, denotes connections between locations and the function d maps individual elements of the edge set to scalar real values which model the cost of travelling the edge. Furthermore, in all subsequent cases we will consider *values* associated to target locations. A location's value represents how important that particular location is.

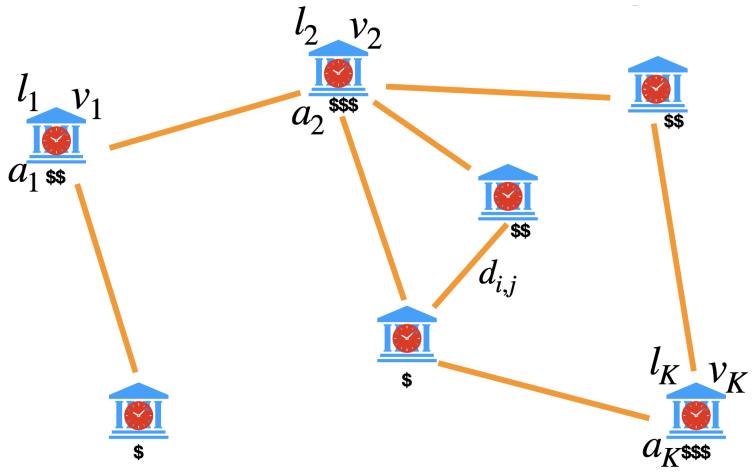


Figure 1.3: Example of a patrolling instance. Each vertex has a value v and an attack time a representing the effort needed to compromise it. A patroller moving from vertex v_i to v_j will spend time $d_{i,j}$.

All formulations define at least one agent, though some consider more. The first agent to consider is the so-called *defender* or *patroller*. The defender is tasked with repeatedly surveying the area, i.e. patrolling, so as to protect the environment from

any possible breaches. One class of problem which we call *optimization techniques*, discussed in chapter 2.1 and chapter 6, attempts to solve the patrolling issue for the defender by framing the problem as an optimization of some metric over the graph. Usually considered, is the instantaneous *idleness* [28] over the set of vertices, V, or the entropy generated by the patroller's schedule [32]. One can think of the instantaneous idleness of a vertex as a timer, associated to that specific vertex, that runs to infinity. If the patroller ever visits the vertex then the timer is reset to zero. In other words, a high idleness for a vertex equates to an under-patrolled target location.

Once a metric is established then algorithms are developed to search for patroller trajectories that either minimize the idleness or maximize the entropy. In this work I will use the words trajectories, schedule, or path interchangeably when referring to the ordered sequence of vertices that the patroller will visit to ensure safety. Finally, it is worth mentioning the overlap this patrolling formulation has with operations research and vehicle routing. A potential solution to the question of agent patrolling, particularly for small instances, could be to compute the *TSP* (travelling salesman problem) tour over the particular graph and report the length of the tour as the worst-case idleness for any vertex. Indeed, this is a common approach, however, the solution for a patroller does not necessarily need to be a tour, and potentially needs to consider that not all the target location have the same importance. For instance, the patroller could perform a *walk* over the vertices, i.e. not visiting every location only once when moving through the graph to ensure protection.

Chapter 6 will detail my work formulating a team optimization version of the patrolling problem. The question we seek to answer in that work is one of analysis: Given a limited number of patrolling resources and a large area to protect, what is the optimal workload distribution for each agent? Consequently, we provide insights into the idea of combining shared and independent workloads for the team. The heuristic provided is built upon principles in graph partitioning schemes and the multiple travelling salesman problem (m-TSP).

1.1.3 Security Games, Patrolling Games, and Search Games

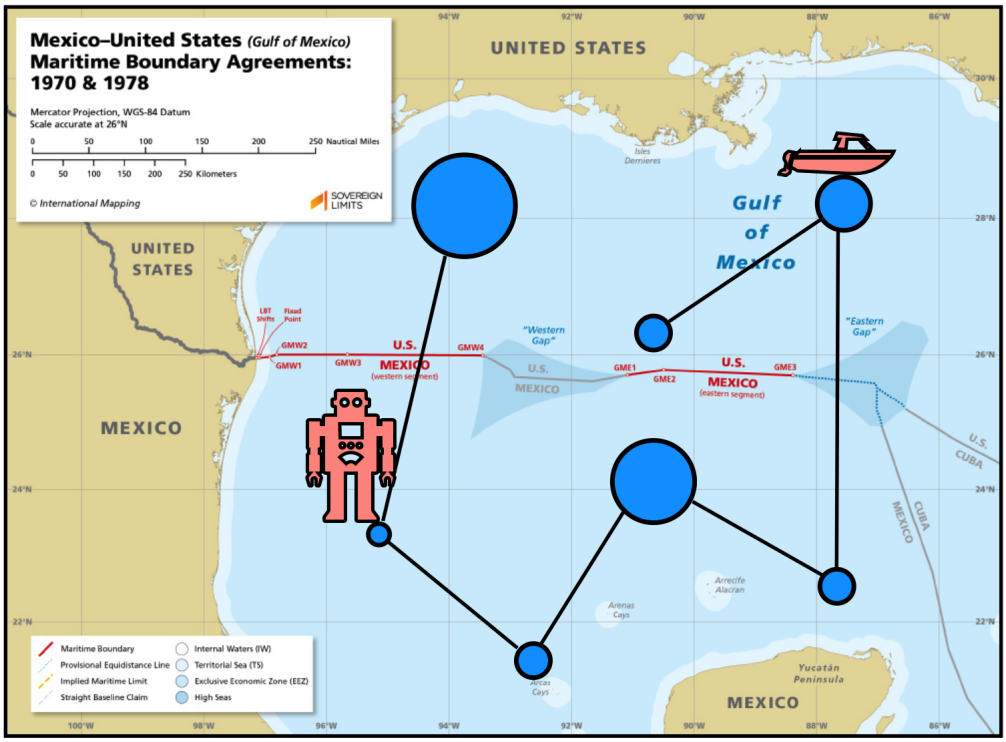


Figure 1.4: Example of a patrolling security game instance. The red agent, defender, must visit and re-visit all the vertices in the area to counteract any malicious activity. Meanwhile, the intruder (red boat) in this case has started illegal poaching activity in the top right area of the graph. If the defender can make it to the target location where the intruder is attacking before all the resources are compromised, then the defender wins. Otherwise the attacker wins and is successful in endangering the environment. The map in question is of the gulf of Mexico, where illegal poaching of red snapper fish has been a concern in recent years.

Another class of patrolling problem deals with formulations that explicitly model a malevolent agent known simply as *attacker*, or *intruder*. Generally, the defender and the attacker are modeled as rational and as competing for resources in a two player zero-sum game. In a zero-sum game, one player's gain equates to her opponent's loss. In other words, the net improvement in benefit from the game is, zero. Many formulations of the problem exist including: von Neumann's *hide-*

and-seek game [96], the *infiltration* game [7], the *search* game [37], the *interdiction* game [98], etc (see [20] for a great taxonomy of the different game formulations). The scope of this current work will be exclusively on the problem known as the *patrolling security* game (PSG) [20]. The patrolling security game differently from other formulations is defined as a two-player multi-stage game with imperfect information and infinite horizon.

The introduction of another rational agent that is competing with the defender completely changes the nature of the optimal strategy. Consider that in the previous section it is mentioned that a potential strategy for minimizing the idleness could be computing the TSP over the graph and assigning the tour as a schedule for the defender. This naive strategy would result in poor performance if say the attacker was able to even just partially observe the defender's patrol. In this case the attacker could build a belief about when the defender will be at certain target locations and calculate the optimal time to begin an attack and guarantee success. In other words, there will need to be an element of randomness in the defender's schedule so as to make it difficult for the attacker to build a belief of the patrols. Figure 1.4 depicts a typical patrolling security game scenario; the agent must repeatedly visit target location of heterogeneous importance to ensure proper protection.

The bulk of the work presented in the following thesis will belong to this category (chapters 3, 4, 5). The following will detail the assumptions and rules we apply to the patrolling security games.

1.2 Overview of Contributions

As evidenced throughout this section, the fields of robotic patrolling and security games are both very mature and share many basic assumptions with other disciplines such as in operations research. This dissertation itself addresses some of the challenges in computing patrolling schedules for an agent, or team of agents, working against an intruder with limited observablility of the environment. It is divided into seven chapters, including this chapter, which each discuss different as-

pects and challenges for robot patrolling. Firstly, chapter 2 presents an overview of related literature associated with the specific types of problems discussed throughout. This includes: techniques for single agent instances, techniques for multi-robot instances, machine learning for patrolling, and common metrics used for gauging a defender's performance.

Chapter 3 begins by detailing our general assumptions for the intruder and the security game itself. Differently from previous work in the area, our model for the intruder limits its capabilities and knowledge. A popular approach has been to model the intruder as a worst case agent that knows the defender's every move [86], commonly known as a Stackleberg Game. Our formulation instead only assumes that the attacker can make repeated observations at a single target location. We argue that this scenario captures more realistically everyday scenarios where generally we may be tasked with deterring bad actors who do not know the number of patrolling resources we employ or their respective schedules. We extend a previous solution [19], shown to be optimal within a loose bound, by exploiting the principle of a *stationary distribution* for ergodic Markov chains. Chapter 4 dives into the application of recent advancements in reinforcement learning (RL) and deep neural networks (DNNs) to the patrolling game. A good amount of the engineering effort was needed to design the observation space of the Markov Decision Process (i.e. the input to the network) and design the reward signal needed to facilitate learning. With some alterations to previous assumptions we made about the security games, we provide an RL-agent capable of protecting an environment even against an opponent who constantly changes their strategy.

Chapters 5 and 6, meanwhile, consider the implications and challenges of computing schedules for a team of patrollers. Chapter 5 takes some of the ideas presented in chapter 3 and expands their use case to also work with teams of agents. A common approach in literature for dealing with teams of defenders avoids the need for communication requirements by partitioning the environment (graph) into k-subsets - one subset for each available patrolling resource (i.e. agent). In the chapter, we present a heuristic method for computing partitions quickly and also discuss an alternative scheme. Partitioning the environment into k-subsets, while

efficient is also less robust to agent failures. For example if a particular agent breaks down, then a portion of the environment will lack adequate protection. This failure will most likely require a re-partitioning of the environment with $k-1$ agents. Thus, in chapter 5, we also introduce an approach to completely overlap every agent's workload, or said differently have every agent patrol the entire graph at the same time. The randomness of the Markov chain framework helps with the redundancy and we discuss the pros and cons of either approach. Chapter 6 continues this line of thinking and presents a method for separating the environment into a *core* set and a *periphery* set. The core set is visited by all the patrolling agents, while the periphery is partitioned into k-subsets and then each subset is assigned to exactly one agent. Moreover, we present empirical evidence and analysis for the benefits of employing this shared, unshared workload strategy. In many cases, it is more efficient resource management to have the multiple agents patrol the core, i.e. high value vertices, together and leave lesser value target locations under protection of a single patrolling resource.

The core/periphery heuristic serves as the only work presented herein that does not explicitly model an opponent and where performance is judged on worst case idleness. Finally, Chapter 7 provides some concluding remarks about the methods presented in this dissertation and discusses a few avenues of future research.

Chapter 2

Related Work

The field of robotic patrolling is very mature. Many formulations and applications exist, each with their own assumptions and frameworks. Broadly, the literature falls into work that explicitly models and opponent and work that instead tries optimize a certain performance metric for the defender. More recently, the field has seen renewed interest because of advances in machine learning and artificial intelligence. Here we discuss first the non-intruder orientated techniques, followed by the intruder orientated techniques, and then focus on our own model for the game. For a survey on recent trends, see [17], and for a more comprehensive survey on the topic consult the work in [46].

2.1 Optimization Techniques

In recent years, algorithms for graph patrolling, especially in autonomous surveillance using mobile robots, have received great attention [17]. This involves modeling environments as graphs and devising strategies to guide patrollers across vertices, mirroring real-world scenarios [79] where vertices represent locations of interest, patrollers are robots with surveillance capabilities, and visits entail checking locations for threats and taking action if needed. The use of mobile robots for surveillance is one of the flagship applications of the field attracting a significant amount of research [69]. Here, the central planning problem is that of computing a *patrolling strategy* for orchestrating, in space and time, the surveillance activities

of a robot.

Our work in chapter 6, belongs to the group of methods that adopt an optimization criterion that has been extensively studied in the literature: the *idleness* between visits [28, 69]. Solving these optimization problems is usually hard, often leading to high computational costs when seeking exact solutions. Prior research has therefore proposed heuristics and approximations to efficiently find sub-optimal strategies that offer practical performance. This is especially crucial in MultiRobot Patrolling (MRP) scenarios where the problem's complexity grows significantly with the number of robots. Nonetheless, employing multiple patrollers remains an effective means to increase performance and robustness. In our work, we propose a novel offline approach to computing MRP strategies by combining two commonly used techniques, which are typically considered mutually exclusive.

Assuming that m is the number of robots, the first technique involves assigning each robot the patrolling task over the whole environment and using a coordinated strategy [57]. This strategy typically relies on a single Hamiltonian cycle (or traveling salesman problem – TSP cycle) that all robots follow. Coordination entails synchronizing the traversal of this path by the m robots, resulting in a collective reduction of maximum idleness on any vertex by a factor of m when robots are uniformly spaced along the tour [28]. While this approach enhances patrolling compared to a single robot, it necessitates computing a complete graph tour and does not leverage the advantages that a potential division of effort among the robots might induce.

The second technique is instead an implementation of the *divide et impera* principle [33, 59, 73]. The idea is to partition the environment into m subsets of the sites to protect, one for each robot, i.e., if V is the set containing all the graph vertices, the partition is defined as $P = \{V_1, \ldots, V_m\}$, with $\cup_i V_i = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. The set P can prescribe a division of effort among the robots thus allowing to reduce the MRP problem to m simpler single-robot instances. In this setting, each robot independently patrols its assigned sub-graph. Since sub-graphs are disjoint, coordination among robots cannot introduce improvements and is therefore not necessary. Typically, each robot covers the TSP cycle (exact

or approximate) on the vertices it is in charge of.

The two techniques can be characterized in terms of overlap between partition elements. The first case corresponds to full overlap (and hence a single element identical to V) while the second case would be associated with an empty overlap, namely a partition of V into m disjoint subsets. In chapter 6, we introduce an intermediate approach to take advantage of *both* the coordination of patrollers (enabled by some overlap over portions of the environment) and the division of effort (requiring no overlap). Starting from the methods described above, we compute a partition of V into up to $m+1$ subsets, $P^+ = \{V_0, V_1, \ldots, V_m\}$ with $|V_0| \geq 2$, i.e., subset V_0 will always have at least two vertices (we motivate this constraint in the next sections.) In such partition, subset V_0 represents a portion of the environment that is patrolled by all m robots in a coordinated fashion, while the remaining subsets V_k define subsets patrolled by exactly one robot, i.e., V_k is patrolled exclusively by the k-th robot. The resulting patrolling strategy will be obtained by combining a set of Hamiltonian paths, computed on each partition's elements, to enable a scaling factor of m for the maximum weighted idleness over the shared vertices (V_0) while allowing independent patrolling over the non-shared parts (see also [28] for more details). By assuming such a structure in the joint patrolling strategy, we leverage situations where it is convenient to coordinate shared efforts on a selection of critical vertices that we dub "the core" while applying a disjoint division of effort over less important ones. Our formulation however allows for some or all of the V_k ($k > 0$) to be empty. When that happens, the corresponding k-th robot just patrols V_0. Note that in the limit, if all $V_k = \emptyset$ for $k > 0$, the patrolling strategy coincides with the classic coordinated strategy where each robot patrols the entire graph.

2.2 Adversarial Patrolling

A significant limitation of the contributions described in the previous section is the fact that they neglect the adversarial nature of many patrolling problems which often model the presence of a rational observer who can learn how patrolling

is carried out, predict the next moves up to some uncertainty, and devise a best attacking response that takes into account such knowledge. Despite such shortcomings and the availability of more sophisticated methods dealing with it (see below), these techniques still enjoy widespread use in real-world implementations of robotic patrolling systems [76].

Meanwhile, adversary modeling, as studied by Albrecht et al. [6], [5], and others has advanced our understanding of adversary behavior, from rule-based adversaries to adaptive agents capable of learning and strategic adaptation. These works help shed light on best practices for the modelling of intruders and help answer questions about what we an engineers and designers are trying to capture when considering different attacker strategies. These interdisciplinary efforts underscore the multifaceted nature of patrolling security games and highlight the importance of integrating insights from game theory, optimization, and multi-agent systems to address contemporary security challenges effectively. All of the next chapters consider an opponent of some kind except, chapter 6. Opponents are taken to be rational, and thus trying to build beliefs of the defender's planning procedure so as to maximize their own utility. Some of the approaches we employ for attackers include: k-nearest neighbor, maximum likelihood, deep neural networks, and greedy idleness attackers.

Security games [93] introduced game-theoretical models for strategic resource allocation in the presence of adversaries. Robotic patrolling can be seen as one of these problems where the resource to allocate is a robot moving on a graph while the adversary (attacker) tries to compromise some vertex. One common assumption is to adopt a *Leader-Follower* interaction model [29], where the patrolling strategy is common knowledge since the attacker can observe its execution for an arbitrary amount of time. The approach prescribes to commit to the best patrolling strategy given that the attacker will best respond to it.

The Leader-Follower assumption implies full observability of the environment and unlimited observation capabilities of the attacker, two features that properly define a worst case scenario but that are rarely satisfied in reality. A number of works challenged these assumptions, like some of the chapter presented in this

thesis. In [3], the case of perimeter patrolling is considered. In such work, different degrees of knowledge possessed by the attacker are analyzed, from a zero-knowledge attacker to a fully informed one and some resolution methods are proposed and compared. However, the work considers gathered knowledge from a general point of view without adopting an explicit observation model for the attacker. Similarly, other works have investigated the presence of noise in the patrolling strategy observed by the attacker and devised robust resolution methods. One example has been proposed in [67] where bounded rationality is also considered for the attacker. A similar approach is adopted in [102] where the authors assume that the attacker cannot always observe the current allocation of patrolling resources in the environment. In [12] the attacker is allowed to perform a limited number of observations from which it can propagate a belief over the patrolling strategy, while in [100] the patrolling strategy is assumed to be known but the attacker has no access to its real-time realization (that is, it cannot assess the protection status of the targets).

This last work introduces leakage as a way to episodically obtain such a knowledge from the target of interest. Other examples are found in [13] where the attacker constructs a belief of the patrolling strategy by performing costly observations and dealing with a trade-off between cost of observation and risk of capture. Finally, in [23] the authors consider a similar belief-based observing attacker and show that planning against the strongest observer might induce limited losses of utility.

The approaches discussed above cast the limited observation capabilities of the attacker to some degree of uncertainty in the knowledge it considers in computing its best response. However, even if affected by errors or sometimes not accessible, observations are assumed to entirely span an environment whose structure is known by the attacker (with the exception of [100] where the patrolling strategy is assumed to be known). In chapters 3-5, we adopt a *local* observation model that allows the attacker to collect information only for a single target. As a consequence, the environment and the patrolling strategy are always unknown while its realization can only be accessed at a single and fixed vertex. Attacker's locality has been

investigated in [14], but not in relation to the observation process. Instead, it is exploited in fixing one attacker behavior according to when it positions itself at a target and then, exploiting a local view, starts its attack as soon as the patroller leaves for other targets.

Markov chains where states encode the current patroller's location have been extensively used to encode patrolling strategies on graph-represented environments (see, for example, [2, 20, 40]) despite their poor scalability when extending states to a history of previous visits [20] (a limitation recently addressed in [51] where response to sequential attacks is studied) and despite their possible failure to describe the optimal patrolling strategy [49]. To the best of our knowledge, the use of time-variance in the transition matrix to deal with an observing attacker, as described in chapters 3 and 5, has never been investigated before.

Multirobot patrolling has also received considerable attention from the robotics community in the last decade. The literature encompasses models for different real-world domains and also provides algorithms to obtain patrolling strategies which guide a robot's visits to areas of an environment to be protected [69]. Many of these solutions are deployed as fully or partially autonomous multirobot surveillance systems [76]. In other words, we are considering agent teams that must cooperate to ensure a given area is protected against malicious activity.

Techniques based on game-theoretical models usually come from the sub-field of security games [93] where patrolling is performed considering a worst-case fully informed attacker. Building on top of these basic works, many recent contributions have relaxed the worst-case assumptions on the attacker in favor of more realistic settings. Examples can be found in the use of bounded rationality for the attacker [67] and, more relevantly, in the introduction of limited observation capabilities where it is assumed that the attacker can condition its decisions on a belief constructed by observing the realization of the patrolling strategy [13, 23]. In [19], Basilico et al. introduced a single-robot patrolling setting where the attacker can only gather information from a single location of the environment. This assumption adheres to many real-life situations where the costs of adversarial intelligence are prohibitive. In chapter 5, we study the properties of the model

proposed in [19] by extending it to the multirobot case.

When compared against their single-robot counterparts, multirobot settings introduce additional challenges. Scalability is a central one, since the computation of joint optimal policies is generally computationally expensive. Customary approaches try to shrink the strategy spaces by limiting the allowed coordination among robots at run time [21]. Conflicts among patrollers could arise [35] while environmental dynamics can pose the need for non-trivial adaptive task-reallocation methods [50]. Sometimes, online coordination among robots might not be even possible due, for example, to the lack of a suitable communication infrastructure [77].

Environment partitioning is one way to mitigate such problems by diverting part of the coordination issues to the offline dimensions. This basic *divide et impera* idea of allocating robots to sub-regions of the environment has been proven relevant in the multirobot patrolling domain (see, for example, [66, 68, 72]). In chapter 5, we try to leverage such an idea to extend our novel patrolling model from [19] for multiple robots. We devise and evaluate two scenarios depending on whether partitions are used or not. We propose a heuristic and exact method to compute partitions and we carry out a theoretical analysis of how the patrolling performance is affected when more patrollers are concurrently deployed. Moreover, to address strengths and limitations of the partitioning methods we propose, we contrast these solutions with a solution where multiple robots independently patrol the whole graph without any sort of coordination or load balancing.

2.3 Patrolling Security Games

An early and seminal work on security games in von Neumann's hide-and-seek game [96]. In his formulation von Neumann describes a two player 2D game played on a grid. One player chooses a cell to "hide" in and the other player, ignorant of the first player choice, attempts to "find" which cell the first player is hiding in by also choosing a cell. Von Neumann characterizes optimal strategies for the both players and provides proofs for their optimality. Starting from this work several

meaningful innovations have been proposed in literature.

The formalism for the patrolling security game (PSG), however, was provided by Basilico et al. in [20]. Differently from previous game formulations and assumptions, the PSG is defined as a two player multi-stage game with *infinite* horizon. The advent of an infinite horizon uniquely presents a more realistic use case because the game model can incorporate intruders who wait to strategize. Indeed in the games we present in subsequent chapters happen in discrete time steps wherein at each turn the patroller decides which target location to visit next and the attacker decides whether to begin an attack or instead wait longer. Using the PSG as a starting point, Carpin et al [19] develop a strategy for the defender to beat a rational opponent and present a new metric for evaluating a defender, dubbed *protection ratio*. The two key ideas exploited by the patrollers are: a delayed movement strategy and Markov chain scheduling. Much of the work we present in the next sections are built on top of the very same assumptions and principles.

More recent work in the domain of patrolling security games has drawn upon a diverse array of disciplines and methodologies to address the complex challenges inherent in the safeguarding of target locations. Game-theoretic approaches to security, as explored by Tambe et al. [92, 93, 103], and Pita et al. [67], have provided innovative insights into the strategic interactions between defenders and adversaries. Their work explores how the two competing agents (when admissible) may reach an equilibrium and how the defender should strategize against the intruder. Moreover in [93], Tambe et al. provide insights into the challenges of deploying these systems in the real-world as evidenced by their use-case at Los Angeles International Airport.

2.4 Reinforcement Learning Techniques for Security Games

A multi-agent formulation of the patrolling problem has been introduced in [28] where idleness is proposed as a cost metric. It measures the time passed since the

last surveillance inspection of any given location. Idleness is a natural proxy for the effectiveness of a patrolling strategy since it is inversely proportional to the frequency of visits, and hence to the amount of surveillance, each node of the environment receives. Approaches built around the optimization of idleness (typically the average or the maximum over the environment) are typically combined with other metrics and domain-dependent constraints. In [87], for example, robots' sensing capabilities are confronted with a cost function that grows in each node if this is not falling in the range of any robot and decreases otherwise. The approaches presented in [62] and [82] are examples where idleness is coupled with the cost of establishing situational awareness between the robots by communication, either with teammates or with a base station. Optimizing this metric is typically hard. Other works explored the adoption of optimization paradigms for dealing with maximum idleness constraints. For example, [60] proposed approximated and heuristic algorithms for the problem of computing the minimum number of robots under idleness constraints [15], and devised approximated methods [1].

Supervised learning has been evaluated to synthesize decentralized multi-agent patrolling strategies by leveraging historical data of surveillance tours computed by a reference ideal method [58]. Despite showing promise, the need for a training dataset makes this type of approach difficult to deploy. Reinforcement learning (RL) provides a more natural environment-driven paradigm. A first RL formulation has been proposed in [81] where idleness plays a central role in shaping the reward function providing feedback to an agent moving in the environment. This idea has been reused under the latest advances in RL by some recent works. In [65] and [52] deep reinforcement learning is adopted to learn patrolling policies that maximize a reward defined as a variable interest metric on each node. Such an interest increases, at a node-dependent rate, as long as the node is not patrolled by the agent. Deep R-learning is exploited to solve a persistent area coverage problem in [85]. The task is characterized by the need to sense stochastic events in the environment as soon as they appear. This problem, sharing in its definition some of the features of the patrolling problem, is solved by adopting a reward function based on the average detection rate of the events. The main difference

with our work is that events are not modeled as the product of some adversarial process, a defining feature for the class of adversarial patrolling problems called Security Games [20].

Works in this last field use attacker models best responding to patrolling strategies [43, 64]. This task might be cast in an online framework where the patroller executes a strategy for some time and then receives feedback [16]. When compared to ours, this class of approaches, despite adopting an online setting, models the problem under a game theoretical perspective according to which attackers often comply with some level of rationality. Recent advancements in so-called "green security games" proposed the use of deep Q-learning for computing optimal strategies. In [97] deep RL is exploited to approximate best responses in an iterative resolution of a patrolling game where real-time observations suggesting how the attacker is operating are exploited. In our work, we do not necessarily assume rationality for the attackers and hence we do not rely on an underlying game-theoretical formulation.

2.5 Metrics of Evaluation

2.5.1 Optimizations

A formalization and analysis of the patrolling problem in multi-agent terms has been proposed in [28]. This, and other seminal works dealing with patrolling on a graph-represented environment, framed the search for effective patrolling strategies as optimization problems where *idleness*-based metrics were adopted as initial candidate objective functions. The rationale behind this class of works is the optimization of the inter-visit delays on some vertices of the graph, each under specific modeling and operational assumptions. Typical goals include following the graph's topology, ensuring a minimum frequency of visits on vertices [33], or complying with latency constraints [15]. Cumulative costs are normally associated with traveling along edges and, in some cases, with vertex visits as well. Different approaches have been applied to solve these problems. These range from the minimization of selected optimization criteria [1, 44] to using game-theoretical

frameworks where the behavior of a rational adversary is considered [20, 32]. This idleness metric measures the temporal difference between subsequent visits (attack-clearing actions) to a vertex. By measuring the idleness in each vertex of the graph and computing an aggregate function of the obtained values (maximum, average, or variants) it is possible to assess how well a strategy protects an environment. The rationale is that the lower the idleness of a vertex, the less likely that vertex is to be subject to an attack. The idleness of a vertex is often scaled by the vertex's importance obtaining what is often referred to as the *weighted* idleness [4].

Other metrics, proposed recently, include maximizing the entropy of the return time [32, 42], or the average entropy rate [39]. These metrics leverage principles from information theory, particularly entropy, to guide the movement of robotic agents in their environments. They aim to maximize the entropy of *return times* while still providing adequate coverage of the area. The return time reflects the duration taken by the patroller to revisit specific target locations. Consequently, the task of predicting when the defender will return to a target location becomes considerably more complex. These approaches, however, fall somewhat between the two categories, as they optimize a performance metric that accounts for the challenges of patrolling against an adversary.

2.5.2 Intruder Orientated

In the context of patrolling security games we are, to the best of our knowledge, the first to introduce the so-called *protection ratio* [19]. This refers to a metric that evaluates the effectiveness of security patrols in neutralizing attacks within a designated environment. Specifically, the protection ratio is calculated as the number of neutralized attacks divided by the total number of attacks attempted by adversaries weighted by the value of the target location. The protection ratio provides valuable insights into the overall success of the patroller's strategy in protecting critical assets. A high protection ratio indicates that a significant proportion of attempted attacks are successfully intercepted, by the security patrols, thereby reducing the overall impact of security threats and enhancing the security posture of the protected area. The protection ratio serves as a key performance indicator for

assessing the efficacy of patrolling strategies and resource allocation decisions. By monitoring the protection ratio, one can evaluate the effectiveness of different patrol strategies, adjust resource allocation based on emerging threats, and identify areas for improvement in security protocols. Other metrics for competing against adversaries include the *probability of capture* [80] which measure the empirical likelihood of capturing the opponent. Both these metrics try to asses the chances of the defender winning the game and thus can be said to capture, in general, how well the patroller is performing.

Chapter 3

Single Agent Patrolling Against Adaptive Opponents

In this chapter, the Patrolling Security Game used throughout is introduced. A strategy is devised for the problem of a single agent protecting a graph. The work shown here was originally presented in [8].

3.1 Security Game Model

We adopt a patrolling model built on a customary setting that has been adopted in numerous works dealing with patrolling robots (see, for example [20]). We consider a discretized environment consisting of n locations of interest that we call *targets* and that will be indicated by the set $T = \{t_1, t_2, \ldots, t_n\}$. The targets represent locations that might be under risk of attack and that must be kept under surveillance. Our discrete representation abstracts away from other possible non-target locations and from the topology with which target and non-target locations are connected. Instead, we assume that given two (not necessarily different) targets t_i and t_j there always exists at least one path connecting them, that any path between them can be traveled in both directions, and that the shortest path between them has a temporal traveling cost known in advance and indicated as $d_{ij} \in \mathbb{R}_0^+$ in the following.

The *value* of a target t_i, a quantity proportional to its importance, is denoted

as $v_i \in \mathbb{R}^+$ while the target's resilience to attacks is given by the *attack time* $a_i \in \mathbb{R}^+$. The attack time measures the temporal cost that an attacker must spend to successfully compromise the target's security. While it is reasonable to assume that most valuable targets have the highest attack times, we do not make any a priori assumption regarding this aspect.

In this context, a patrolling task is carried out by a single mobile robot/agent traveling from target to target. We assume that the robot has the capability of detecting the presence of an ongoing attack and undertaking some action to potentially stop it (for example, raising an alarm or alerting a human). Such a capability is localized to the target currently visited by the robot. Thus, if the robot visits target t_i at time τ and an attack on that target has started at a time within the interval $[\max\{0, \tau - a_i\}, \tau)$, then the attack is detected and neutralized. The *status* of a target is *protected* if the patroller is located in it and it becomes *unprotected* when the patroller leaves it.

The threat we assume to face is modeled as coming from an *attacker* agent that, at any time τ, can start an attack to a target t_i. Once the attack is started, the time window $[\tau, \tau + a_i)$ represents an *exposure interval* during which the attacker can be stopped if the patrolling robot visits t_i. As commonly done in security games, we assume an underlying constant-sum interaction between the patroller and the attacker. More precisely, in the case where the patroller stops an attack on target t_i it receives a utility of $U = \sum_{i=1}^{n} v_i$ while the attacker gets 0. On the contrary, if the attack on target t_i is successfully completed the patroller receives a utility of $U - v_i$ while the attacker will conquer the target's value v_i. The patroller's movements in the environment are defined by a Markov chain process where a state represents the currently visited target. The $n \times n$ transition matrix $\mathbf{P}$, where the entry p_{ij} represents the probability of transitioning from target t_i to target t_j, defines the patrolling strategy used to protect the environment. In the following, we shall assume that $\mathbf{P}$ is irreducible and aperiodic [61, 74]. Figure 3.1 depicts a Markov chain along with its corresponding transition matrix, $\mathbf{P}$.

With respect to the classic literature on security games, we propose two relaxations (introduced in [19], here extended with observation errors) that allow us to

capture some realistic aspects encountered in real-world applications. The first is about the patroller's movement model. The mainstream approach prescribes that temporal traveling costs of shortest paths (in our setting defined as d_{ij}s) should always correspond to the actual times spent by the patroller for moving between targets. Indeed, in standard strategical settings it can be relatively easy to show how following such rationale weakly dominates the opposite. In our model, we embrace only this basic requirement:

label=*)* we interpret d_{ij} as a lower bound for the time spent traveling between t_i and t_j allowing for occurrences where the patroller takes some extra additional time to transfer.

As we will show, this feature can be useful in scenarios when dealing with a non-fully informed attacker.

The second relaxation we introduce involves the model of the attacker. Typically the attacker is modeled as a rational and fully informed agent that has access to the environment topology, the patrolling strategy being executed by the robot, and its current position. The derived game is hence solved according to a leader-follower paradigm where the attacker substantially best responds to the observed patrolling strategy and the patroller's current position. In our model we instead assume an attacker that is still rational, but that is not fully informed. More specifically, our attacker model is characterized by the following features:

lbbel=*)* the environment topology and, as a consequence, the values of v_i and d_{ij} for all t_i, t_j are not known and not accessible;

lcbel=*)* the patrolling robot cannot be observed while it executes its task in any location of the environment, meaning that the current position is, in general, unknown and no observation-induced belief over the patrolling strategy can be maintained;

ldbel=*)* the attacker[1] is hidden and ready to attack at an unknown target where it can gather local observations under the assumptions described below.

[1]Despite we will generically refer to a single attacker, our model would work also under the interpretation of up to n equally defined and uncoordinated attackers.

When observing a target during a time where it is *unprotected*, the collected information will not be affected by errors. In other words, we assume a null false-positives rate $\alpha = P(protected \mid unprotected) = 0$. On the contrary, if the attacker is observing a target whose state is *protected*, with probability β it will not detect the presence of the patroller independently of how long the patroller stays on that target an it will be mislead into believing that the target has been *unprotected* for the whole time. In other words, we assume a non-null false-negative rate $\beta = P(unprotected \mid protected) > 0$.

With the model *a)–d)*, we relax some of the assumptions made in security games, namely, that the patrolling setting is fully observable. Instead, the attacker model we consider does not have any prior knowledge on the patrolling setting but only relies on locally limited and noisy observations of a single target. These features capture realistic settings where the planning activities of an attacker take place locally to the target itself and, at the same time, the context in which the patroller is operating (its current position, the set of targets, and the patrolling strategy) are out of reach due to inaccessibility or high intelligence costs. As a concrete example consider a large site protected by an autonomous patrolling unit. Our attacker does not have enough power to monitor the whole site for a long time because it would take too much effort. As a consequence it cannot derive the environment discretization used by the patroller and the surveillance strategy. What it can do, instead, is to loiter at the chosen target area and evaluate its chances on the basis of what it observes there.

3.2 Patrolling Against a Local Observer

The scenario described above induces a situation where the patrolling agent travels from target to target by following a transition matrix $\mathbf{P}$. The attacker, hidden at an unknown target that we denote as t_j, observes a sequence of state changes on that target: from *uncovered* to *protected* as soon as the patroller visits t_j and the opposite when it leaves (up to false negatives). Since the success or failure of an attack depends on the patroller's visit within an exposure interval,

the attacker is incentivized to log state changes with a timestamp and to extract a time-series defined as subsequent realizations of a random variable R_j modeling the patroller's return time (or inter-arrival time) to target t_j. In the long run, the attacker will take advantage of such knowledge by deriving a belief on $P[R_j > a_j]$, that is the probability that the target will stay uncovered for enough time to complete an attack. In short, we shall call it *attack probability*. Due to assumptions *a)* and *b)*, no inference on the environment topology can be exploited. Specifically, notice that assumption *a)* also applies to self loops, allowing the patroller to leave t_j and then returning to it after an arbitrarily small amount of time. For such reason, the attacker does not have incentives in waiting some extra time after the target has become uncovered, meaning that, if the attack must start, it will initiate as soon as the patroller has left the target (thus justifying the use of R_j).

The problem we face when computing a patrolling strategy for such a local observer is a standard constant-sum setting, namely finding $\mathbf{P}$ such that the maximum attack probability is minimized. Recall that, due to assumption *d)*, the target at which the attacker is hidden is unknown. Because of the difficulty in computing the exact probability, in [19] we provided a first, approximate answer to this question by using the upper bound given by Markov's inequality [55]

$$P[R_j > a_j] \leq \frac{\mathbb{E}[R_j]}{a_j} = U_j$$

and in this work we use this same solution[2].

The objective we seek for the patrolling task is twofold. From one side the patrolling strategy should provide the maximum protection at convergence. That is, it must optimize the attack probability when working under the condition in which the attacker has managed to derive a correct belief over it. At the same time, it is desirable that such condition is hard to meet by the attacker, making the construction of a belief from the observations of R_j as difficult as possible.

One first method that we introduced in [19] is based on the idea of decoupling spatial and temporal decisions when patrolling the environment. This is achieved by an iterative two-step decision process. Let us suppose that the current target

[2]A numerical algorithm iteratively converging to the correct attack probability is the subject of ongoing research.

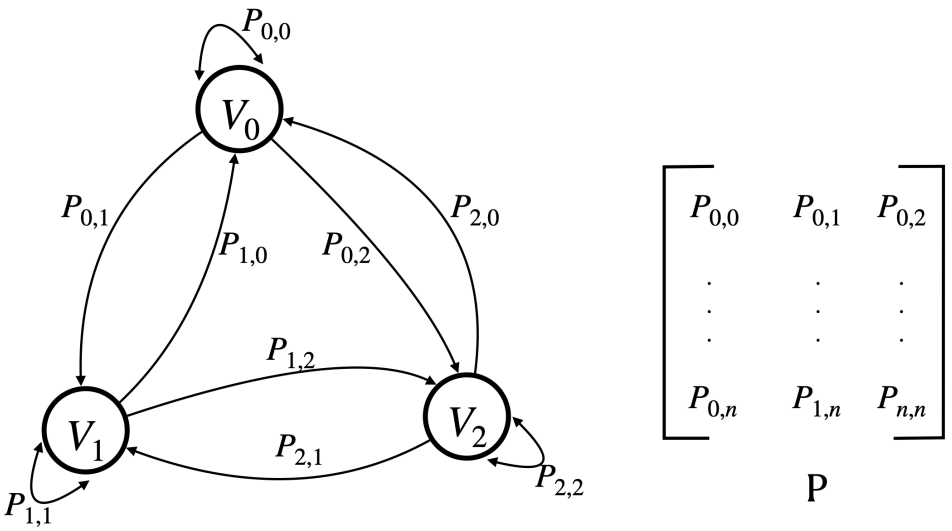


Figure 3.1: This figure depicts the Markov chain definition. The states V_0, V_1 and V_2 are connected by edges that represent transition probabilities. The transition probabilities themselves, are encoded in the *transition matrix*, $\mathbf{P}$. For our purposes the transition matrix serves as the schedule for the patroller. Since each row represents a probability distribution (each row in $\mathbf{P}$ adds up to one), starting from any initial vertex the patroller can sample the probability distribution of the row of the vertex it is currently present at to determine which area to observe next.

occupied by the patroller is t_i. In the first step the next target t_j is selected according to a Markov chain strategy expressed by $\mathbf{P}$. In the second step the patroller draws a value δ_{ij} from the uniform distribution $\mathcal{U}[d_{ij}, d_{ij} + \Delta]$ where $\Delta \in \mathbb{R}_0^+$ is a parameter representing the maximum delay to be applied. Finally, the patroller constraints itself to spend δ_{ij} to reach t_j starting from t_i. In [19] we have shown that the random delay causes the attacker to see a sequence of return times that have the characteristics of white noise. This strategy is summarized in Algorithm 1, where with $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)$ we denote the stationary distribution induced by $\mathbf{P}$. Notice that since we assumed that $\mathbf{P}$ is irreducible and aperiodic, $\boldsymbol{\pi}$ is unique and guaranteed to exist.

Algorithm 1: Time-delayed patrolling strategy.

1 **Input**: T, distances d_{ij}s, values $v_1, v_2, \ldots, v_n$, attack times $a_1, a_2, \ldots, a_n$, Δ, transition matrix $\mathbf{P}$;
2 Select the start target $t_i \sim \boldsymbol{\pi}$;
3 **while true do**
4 Select the next target t_j with probability p_{ij};
5 Generate a random time $\delta_{ij} \sim \mathcal{U}[d_{ij}, d_{ij} + \Delta]$;
6 Move to t_j spending time δ_{ij};
7 $t_i \leftarrow t_j$;

When adopting a strategy of the form described in Algorithm 1 the upper bound on the attack probability can be written as

$$U_j = \frac{1}{\pi_j a_j} \sum_{i=1}^{n} \pi_i \sum_{h=1}^{n} p_{ij} \frac{2d_{ih} + \Delta}{2}$$

and solving for the optimal stationary distribution $\boldsymbol{\pi}^*$, that is the one that minimizes the maximum upper bound of the attack probability, gives the following (see [19] for details):

$$\pi_i^* = \frac{\mu_i}{\sum_{j=1}^{n} \mu_j} \; \forall t_i \in T, \text{ with } \mu_i = \frac{v_i}{a_i}$$

Given the stationary distribution $\boldsymbol{\pi}^*$, a time-delayed patrolling strategy can be implemented by running the Metropolis-Hastings algorithm to obtain $\mathbf{P}$ (see details in the next section). The rationale behind this strategy is that by tuning the parameter Δ we can lower the autocorrelation in the time series of inter-arrival times observed by the attacker, hence making it harder to forecast.

3.3 Time-Variant Strategies

An additional way to increase the difficulty of forecasting inter-arrival times is to introduce time-variance in the transition matrix derived with the approach described in the previous section. We introduce time-variant patrolling strategies which are obtained by changing the matrix $\mathbf{P}$ used to determine which vertex to visit next. The key fact to implement this approach is that we can determine an infinite number of transition matrices $\mathbf{P}$ all having the same optimal stationary distribution $\boldsymbol{\pi}^*$. To see this, we just use the Metropolis-Hastings algorithm [78] with a random proposal. More precisely, for a given $\boldsymbol{\pi}^*$ to generate $\mathbf{P}$ we start with a random[3] transition matrix $\mathbf{Q}$ whose entries are all strictly positive. Let $q_{i,j}$ be the (i, j) entry of such matrix, and let

$$\alpha_{i,j} = \min\left\{1, \frac{\pi_j^* q_{j,i}}{\pi_i^*, q_{i,j}}\right\}$$

Then, for $i \neq j$ we set $p_{i,j} = q_{i,j}\alpha_{i,j}$, and p_{ii} is set so that all rows add up to one.

From this premise, we consider four different methods, each defined by the rule used to decide when to drop the current transition matrix $\mathbf{P}^{(k)}$ and switch to a new matrix $\mathbf{P}^{(k+1)}$. Furthermore, we introduce the concept of *duration* of a transition matrix, a quantity that determines the time span during which the same transition matrix is used.

In the **constant-transition** strategy the duration of a transition matrix is defined as a constant number of transitions, say M. That is to say that the patroller keeps an internal counter that is increased every time a new target is

[3]This can be easily created by repeatedly calling a uniform random number generator with strictly positive support to fill up the matrix, and then normalizing each row to add up to one.

visited. When the counter reaches M, a new transition matrix is generated and the counter is reset. Evidently, M is a parameter that needs to be picked upfront.

The **random-transition** strategy is a modification of the previous strategy where the duration of each transition matrix is not given by a constant number of transitions, but is rather distributed according to a Poisson distribution with parameter λ. Every time a new transition matrix is generated, its duration is sampled from the Poisson distribution. In other words, each transition matrix is associated with its own randomly defined duration.

The **constant-time** strategy relates the duration of a transition matrix to the time spent by the patroller to move from vertex to vertex (differently from the previous two methods that use the number of transitions to determine when to change transition matrix). Specifically, when moving from t_i to t_j the patroller spends time δ_{ij} (recall Algorithm 1). In the constant time approach, a fixed threshold T_{MAX} is defined, and as the patroller moves from vertex to vertex a cumulative variable T is used to add all the various δ_{ij}. When T exceeds T_{MAX}, a new transition matrix is generated and the variable T is reset to 0. In this strategy T_{MAX} is a constant parameter playing a role similar to M in the constant transition strategy.

Finally the **random-time** strategy works exactly as the previous one, except that T_{MAX} is not a constant, but is rather sampled from an exponential distribution with parameter ψ[4]. Similarly to the random transition strategy described above, every time a new transition matrix is generated, its duration is determined sampling from the exponential distribution and therefore every transition matrix has a different duration. The time-variant strategies are summarized in Algorithm 2.

All the above strategies depend on a parameter defining when to change the transition matrix. This choice could be made in different ways. One approach, embraced in the next section, is to perform a preliminary search for a given set of benchmark graphs, and then pick the value of the parameter that maximizes a performance function (see later discussion about possible metrics). This method is off-line, i.e., the parameter is decided upfront and remains constant throughout

[4]For the exponential distribution we use the parametrization where ψ is the expectation.

Algorithm 2: Time-variant patrolling strategy.

Input: T, distances d_{ij}s, values $v_1, v_2, \ldots, v_n$, attack times $a_1, a_2, \ldots, a_n$, Δ, stationary distribution $\boldsymbol{\pi}$;
$k \leftarrow 0$ **while true do**
    Generate $\mathbf{P}^k$ from $\boldsymbol{\pi}$;
    while *the duration of* $\mathbf{P}^k$ *has not expired* **do**
        Set $\mathbf{P} \leftarrow \mathbf{P}^k$ and run Algorithm 1;
    $k \leftarrow k + 1$

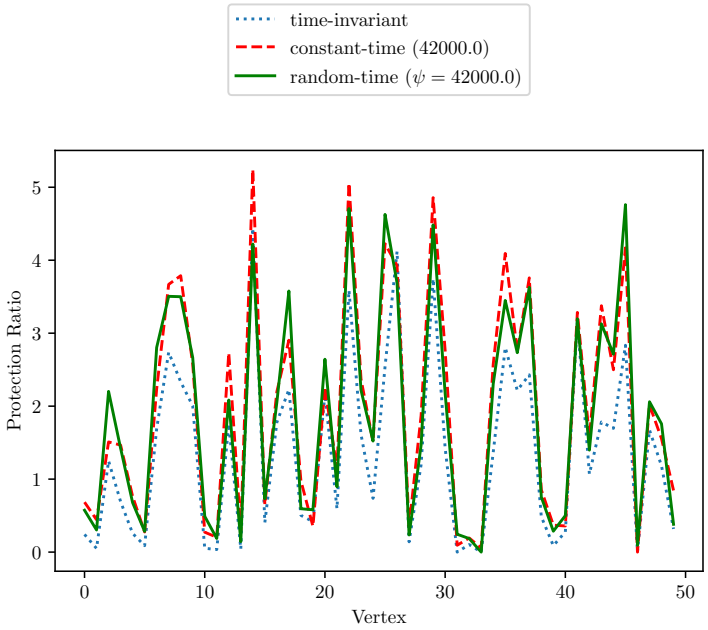
the run. In the conclusions we will discuss another possible on-line approach where the decision on when to change matrix is rather taken on the fly.

3.4 Experimental Evaluations

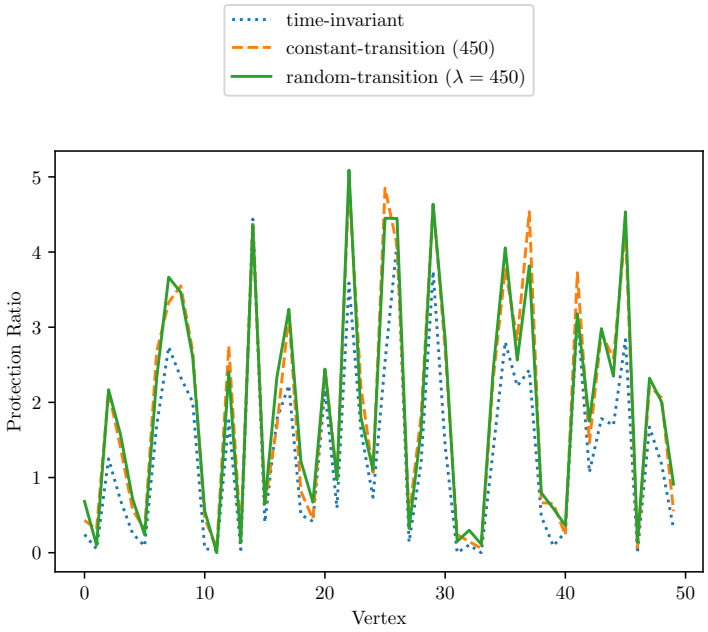
We generated 50 environments of different sizes considering complete graphs with 10, 20, 30, 40, and 50 targets, and, for each size, we generated 10 random instances. The graphs vertices were created by randomly placing points on a 100×100 plane and then setting each edge's temporal cost $d_{i,j}$ to the Euclidean distance between t_i and t_j. The value of each target v_i was drawn from $1+\mathcal{U}[0, 10]$ while the attack time was drawn from $\mathcal{U}[D, 3D]$ where D is the average of the temporal costs $d_{i,j}$.

While in [19] we considered a single strategy for the attacker, in this study we test our algorithm against three different types of observing attackers. In each case the attacker computes a prediction of the next inter-arrival time at the target as soon as a transition from *protected* to *unprotected* is observed. If the predicted value is greater than the target's attack time, the attack is attempted. Starting from its local observations, each type of attacker uses a different method to predict the time when the attacker will visit the vertex again.

The first type of attacker assumes that inter-arrival times at target t_i follow an exponential distribution with parameter λ_i and uses a maximum likelihood approach to estimate such parameter. The attacker then derives a prediction



(a)



(b)

Figure 3.2: Comparison between the protection ratios against the two attacker models. Shown here is an instance with 50 target locations.

for the next inter-arrival time by taking $1/\lambda_i$, that is the expectation of the hypothesized exponential distribution.

The second type of attacker uses a nearest neighbor (NN) approach to forecast a time series [24]. The attacker observing target t_i acquires a sequence of observations $O = (R_i^0, R_i^1, R_i^2, \ldots)$ for the inter-arrival times. It then considers all the sub-sequences of length m, where m can be thought as the attacker's finite memory, and computes the distance between each of these sequences and the last m inter-arrival times that have been observed. The closest sub-sequence (different from the last one) is selected and the inter-arrival time observed immediately after such sub-sequence is taken as the prediction. In all our experiments, $m = 10$.

The last type of attacker exploits a deep neural network (DNN) consisting of a fully connected regression network with 200 hidden units using a long short-memory layer [45]. Owing to the necessity of having enough training data, this method uses the first 500 observations to train the network and then makes predictions for the times of the remaining visits. The parameter 500 was obtained after manual tuning, and never represents more than 90% of the data available for training and testing. It shall be noted that, owing to the large number of parameters to tune, this method requires significantly more data than the other two and from a practical standpoint one could argue that when observations have a cost, an attacker would be unlikely to use it. Nonetheless, due to the vast popularity of these methods, it is interesting to use this approach to assess intrinsic strengths or weaknesses of our patrolling strategy.

We define an evaluation metric denoted as *protection ratio*. This quantity is obtained empirically, by simulating a deployment of the patrolling strategy we want to evaluate. First we simulate the strategy for a total of K transitions. (In our experiments, K was set to 10000 for the first two methods and to 100000 for the DNN, due to the necessity of generating sufficient training data). Afterwards, for each target t_i we consider K_i as the set of transitions after which the status of t_i changed from *protected* to *unprotected*. For each $k \in K_i$ we run the three attacker types feeding them with the whole history of inter-arrival times generated at t_i by all the transitions preceding k. We then record each attacker's decision and, if

that decision was to start an attack, we label the attack as successful/unsuccessful by looking at how the patrolling mission would unfold ahead of k. Call $N_A(i)$ and $N_A^u(i)$ the number of attack attempts and the number of unsuccessful attacks generated by such procedure on target t_i, respectively. The protection ratio for target t_i is then defined as $v_i N_A^u(i)/N^A(i)$. Ideally, this metric should be high, i.e., for high-value targets we prefer patrolling strategies causing the attacker to perform many unsuccessful attacks. In the experiments we present, the choice for the parameters defining the four time-variant strategies (shown at the top of the figures) were informed by a preliminary comparative evaluation of different choices. In particular, we run the algorithms on a set of benchmark problems and picked the values giving the best performance averaged across all instances.

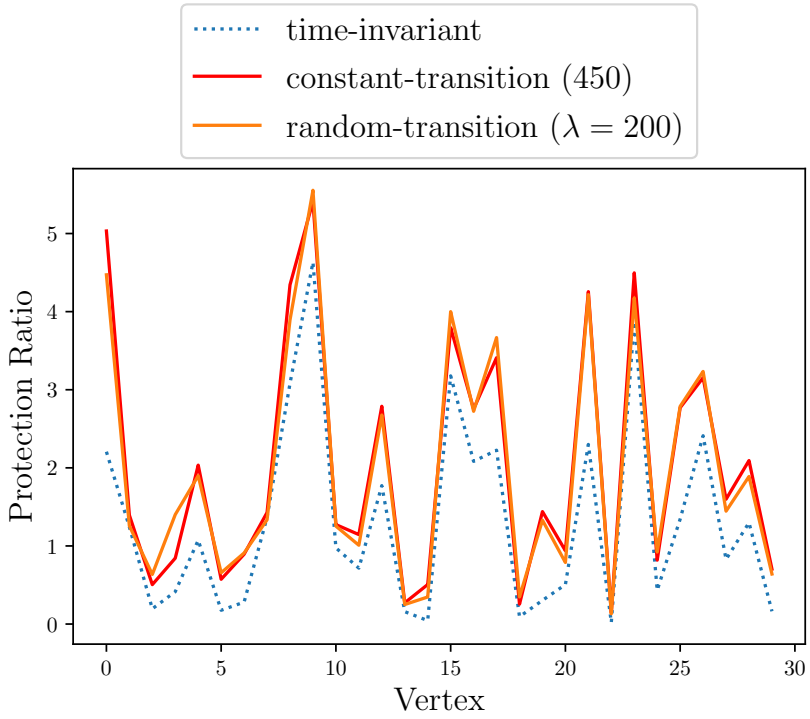


Figure 3.3: Protection ratio for each vertex with attacker using a NN strategy.

In Figures 3.3 and 3.4 we start by contrasting the time-variant strategies with the time-invariant method we proposed in [19]. In particular, we plot the protection ratio for one of the environments with 30 vertices. In this case the attacker used the NN approach. As it can be seen, the temporal variant strategies outperform

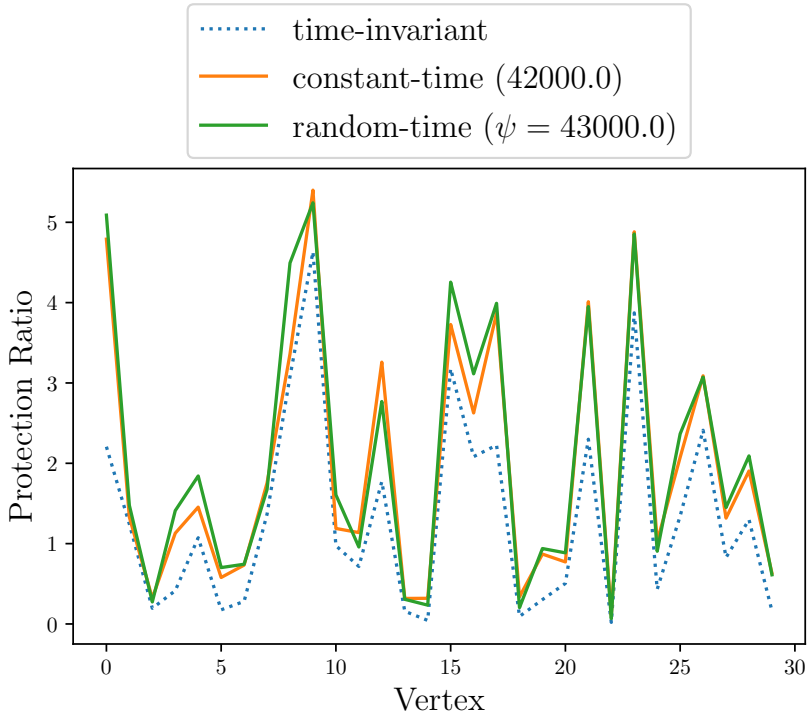


Figure 3.4: Protection ratio for each vertex with attacker using a NN strategy.

the time-invariant strategy at the majority of targets ensuring a more extensive protection. Vertices where the gaps are marginal are those typically characterized by small attack times. For such vertices, the patroller can do little to prevent an attack since they represent locations intrinsically difficult to protect.

Next, in Figures 3.5 and 3.6 we display analog results, by considering the case where the attacker uses the maximum likelihood method. It can be observed that the performance is rather similar, with time-variance outperforming the non-variant strategy. A possible explanation behind such trends is that time-variance makes these patrolling strategies *harder to learn* and hence predicting inter-arrival times is likely to be subject to errors. Slightly more complex methods such as NN do not tend to outperform simpler approaches like the prediction based on maximum likelihood). Furthermore, figure 3.2 presents the results obtained against the ML attacker on a graph with 50 target locations. As can be seen, the results are similar and show how our method can be scaled to larger graph instances.

This intuition about the hardness of forecasting these strategies is confirmed by

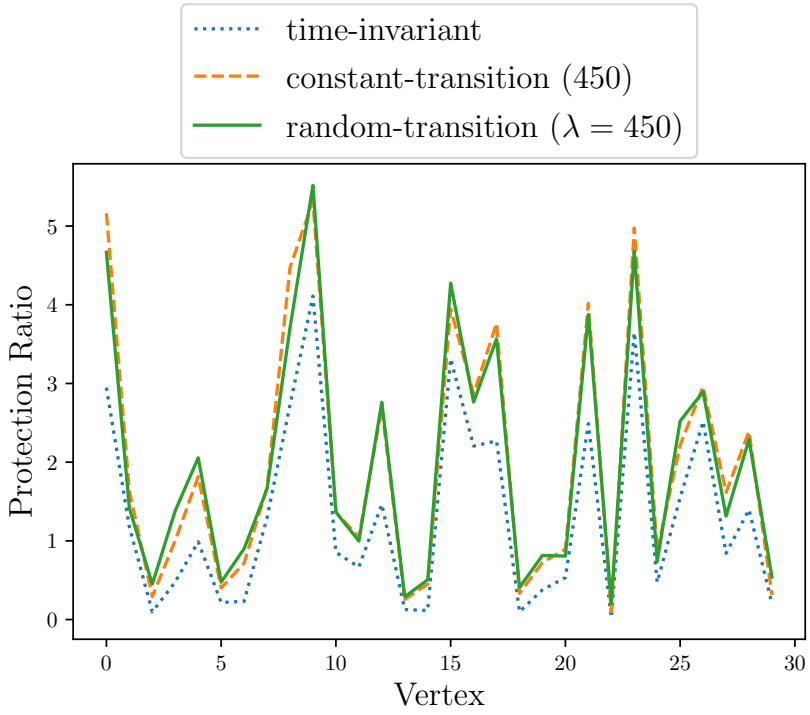


Figure 3.5: Protection ratio for each vertex with attacker using a maximum likelihood strategy.

the results obtained when the attacker uses a DNN. This is shown in Figures 3.7 and 3.8. First, observe that the absolute values for the protection ratio are essentially comparable to those obtained with the other types of attacker. Therefore even if the attacker uses a more sophisticated technique to forecast the next visit to the vertex, the performance does not improve. We explain this result with the intrinsic hardness of predicting the time series generated by our patrolling strategies. This is particularly relevant because the tests with the DNN were on purpose skewed in favor of the attacker: the network was trained with significantly more data than the samples made available to the attackers based on the NN or maximum likelihood. The only outlier case appears to be the protection ratio for vertex 23 in Figure 3.8. Due to the *black box* nature of the forecast approach implemented by the DNN, it is difficult to explain why this happens. However, this single outlier does not hinder the overall conclusions we derived, although further investigations will be done in the future. Figure 3.9 further corroborates our

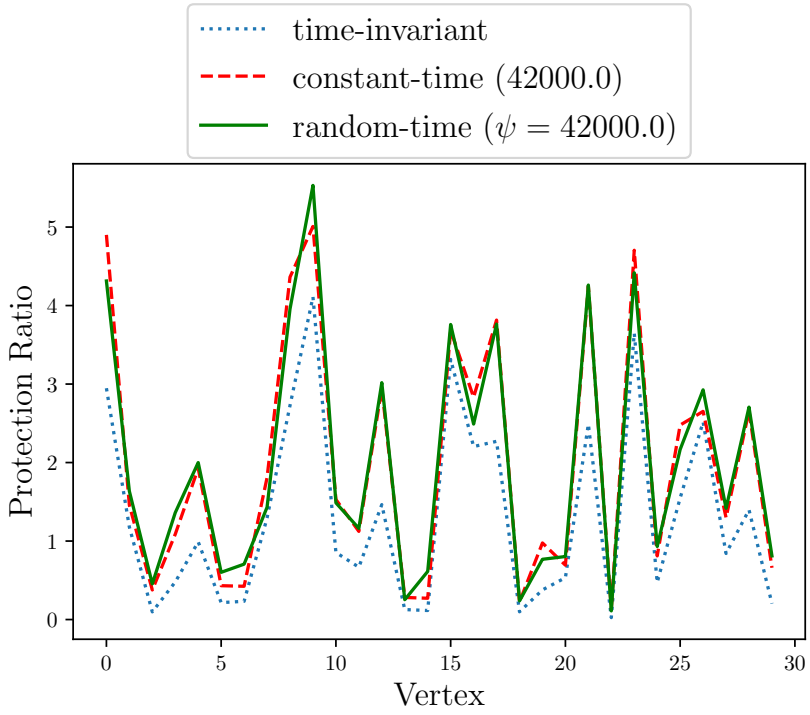


Figure 3.6: Protection ratio for each vertex with attacker using a maximum likelihood strategy.

assessment that data intensive methods like DNN do not significantly outperform simple strategies. The figure shows the protection ratio for a testcase graph when the attacker uses the three different strategies we considered. While the maximum likelihood and the NN offer comparable performance, the DNN almost uniformly performs worse that the other two strategies in terms of ability to correctly predict the next arrival time, while at the same time requiring significantly more data to train.

In the last experiment we present, we assess the performance of our strategies with respect to the false negatives rate β. We analyze how the average protection ratio varies with respect to what we denote as *observation accuracy*, defined as $1 - \beta$. The lower this accuracy, the more frequently the attacker will believe that the target has been unattended even if the patroller has actually been there to check it.

In Figures 3.10 and 3.11, we report this analysis for the random-time strategy

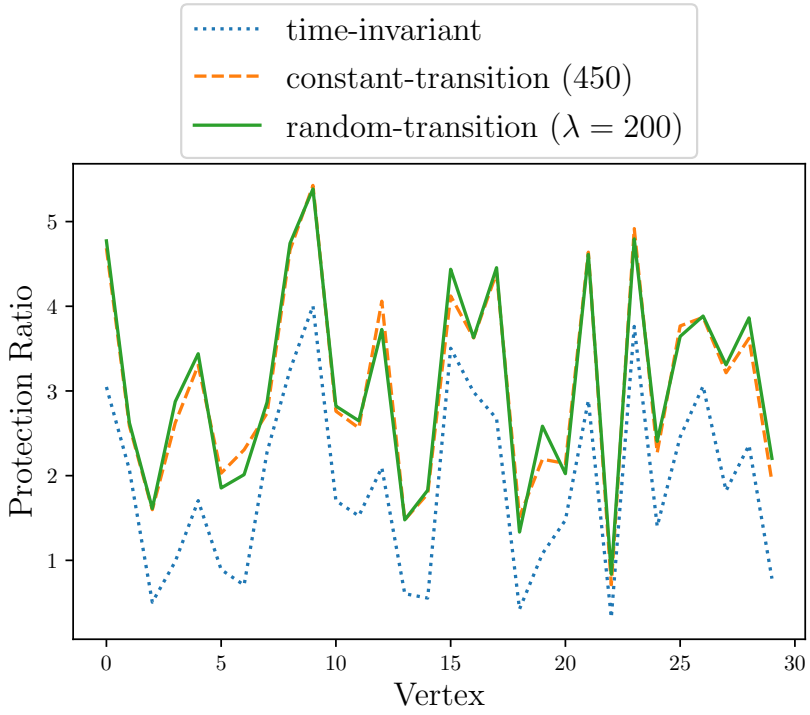


Figure 3.7: Protection ratio for each vertex with attacker using a DNN.

and for the time-invariant one against the maximum likelihood attacker. The intuitive trend that would be expected from an attacker affected by increasing levels of observation noise is that of an increase in the protection ratio. That is, attackers with higher false negatives rates should be, on average, performing worse against our patroller. However, the figures reported below draw a less intuitive and more insightful picture. We notice that for not remarkably low, and hence reasonable, values of accuracy (≥ 0.8) no significant difference can be observed in the average protection ratio for both strategies.

This counter-intuitive result can be explained by noting that a lower accuracy would inevitable result in a "reckles" attacker since false negatives will induce an optimistic overestimation of inter-arrival times. The fact that attempting more attacks results in a better performance of the attacker reveals an hidden *prevention* feature of our strategies. That is, in many occasions the attacker is induced to not attempting an attack (by inducing a belief of a small inter-arrival time) even when a winning occasion was present. Such prevention effect is clearly mitigated by

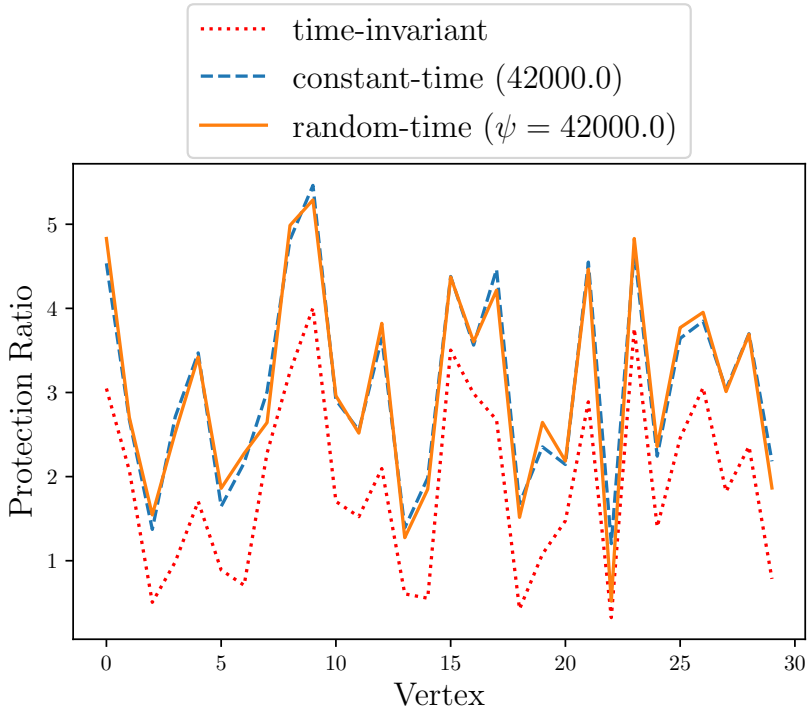


Figure 3.8: Protection ratio for each vertex with attacker using a DNN.

reckless attackers even if, as the plots show, it become significant only for very low values of accuracy.

3.5 Conclusions

This chapter presented an adversarial patrolling setting where the attacker is characterized by an observation model allowing it to gather information only locally to a target. We proposed heuristic methods to generate patrolling strategies that, using delays and time-variance, provide protection and are difficult to forecast. A compelling future direction for the present work is the study of on-line techniques to adapt time-variance. One possible approach is to have the patroller running an online model of a baseline attacker (e.g., one that runs a maximum likelihood estimator) and use it in the decision of when to switch from a strategy to another. This extension is left for future work.

Among the various directions for future work, a compelling one we already

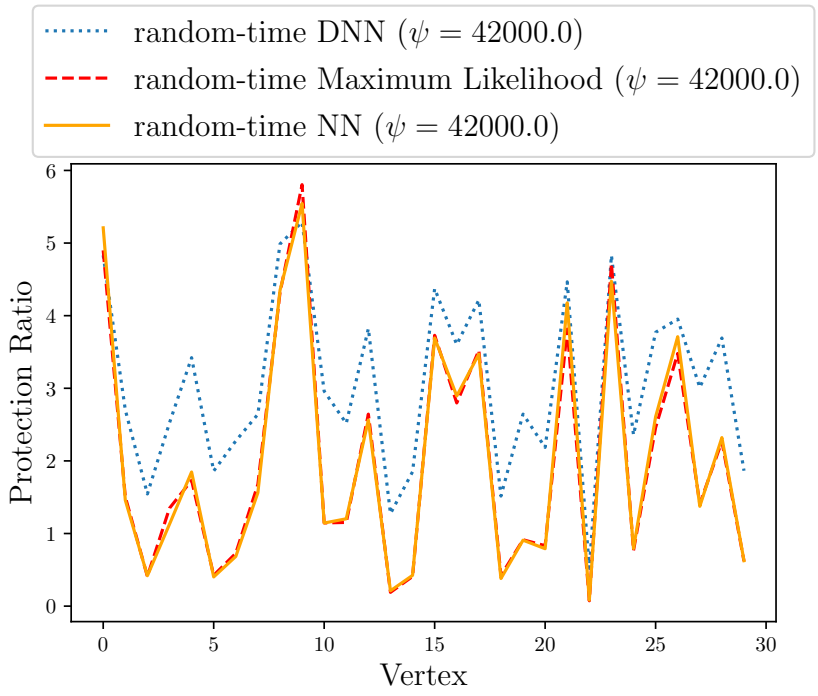


Figure 3.9: Comparison between the protection ratio achieved on a testcase graph against three different attack strategies.

mentioned earlier on is in exploring on-line approaches to control time variance. That is, devising mechanisms for determining at runtime when to change the transition matrix. One intuitive way to tackle such a problem could be as follows. As the attacker, through the observation model we defined, observes a sequence of inter-arrival times, it could try fitting such data to an exponential distribution using, for example, a maximum likelihood strategy as we discussed in the previous section. This could be considered a *baseline* attacker, i.e., the least sophisticated attacker that does not just performs random attacks (similarly to what done in [14] with attackers selecting their target with a probability proportional to their value) . Consequently, the patroller could decide to switch the transition matrix to limit the accuracy of the estimation. The problem of relating the number of samples to the estimation accuracy is well understood [75], but it requires the patroller to keep track of the interarrival times generated at each target by the current transition matrix, i.e., the decision has to be made online. This extension is left for future

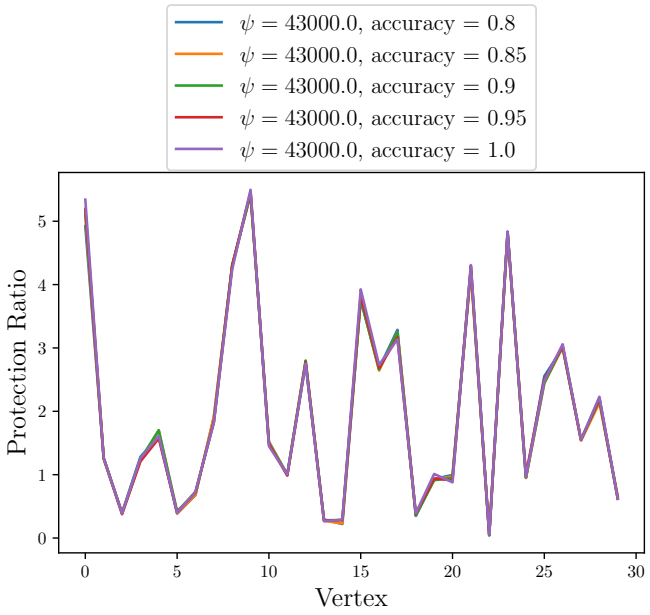


Figure 3.10: Protection ratio for each vertex with the random-time strategy and the maximum likelihood attacker for different observation accuracy.

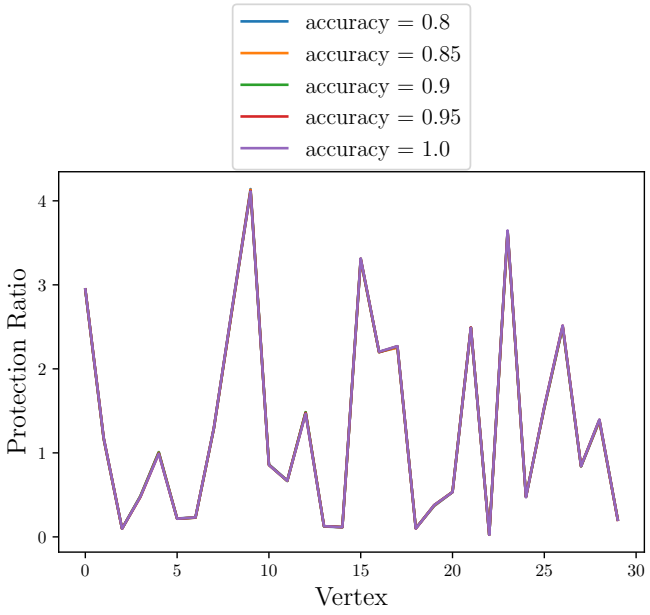


Figure 3.11: Protection ratio for each vertex with the time-invariant strategy and the maximum likelihood attacker with different observation accuracy.

work.

Chapter 4

Single Agent Patrolling via Reinforcement Learning

In this chapter, the PSG is modeled as a Markov Decision Process (MDP) so that reinforcement learning techniques may be applied. The resulting framework is applied to the single agent patroller case and evaluated according to its ability to the capture the intruder. The work shown here was originally presetned in [11].

4.1 Patrolling Setting

In this chapter, we again adopt a graph-based representation of the environment where K target locations must be protected through visits by a single patrolling robot as in chapter 3. Targets are denoted as $\{t_1, ..., t_k\}$. Their topological layout is described by a weighted undirected graph $G = (V, E, d)$ where $V = \{t_1, ..., t_k\}$ and $(t_i, t_j) \in E$ indicates that the patroller can move from t_i to t_j (or vice versa) in a time equal to d_{ij}. Attack times are assumed to be derived as follows:

$$a_i = k \cdot U(lb, ub)$$

$$lb = m - \left(\frac{m}{L}\right) \qquad ub = m + \left(\frac{m}{L}\right)$$

where: L denotes a positive real number, m denotes the average travel distance on the undirected graph, the function $U(\cdot)$ denotes a random uniform sample on

the interval $[lb, ub]$, and k scales the resulting random sample by a positive real number. To generate non-trivial instances, attack times cannot be either too large (capture would be too easy) or too small (attacks would be too hard to defend against.) The above formulas capture this requirement by uniformly drawing their values from an interval (whose width is controlled by the parameter L) centered at the patroller's expected distance to be traveled by any two targets.

The patrolling mission unfolds in discrete time steps. In each of them, the patroller protects the graph by moving from one target location to the next, but may also decide to stay at the current vertex for another time step. If the patroller does decide to stay at the current vertex, the travel time is *not* zero. Instead, the patroller travels some random time along an edge and returns to the vertex it left from. It is assumed, in the model, that if the patroller travels to a vertex where an attack is taking place then the attack is neutralized and the attacker captured.

For generality, all attackers, before a game starts and after any re-spawn, derive their deadline by randomly sampling from a pre-defined real–valued set.

4.2 Deep reinforcement learning and patrolling

4.2.1 Problem Formulation

We model our robotic patrolling setting as a Markov Decision Process (MDP) and then apply Deep Reinforcement Learning (DRL) to synthesize a patrolling strategy. A Markov decision process is defined by the tuple (S, A, P, R, γ) where: S denotes the set of states, A refers to the set of available actions, P denotes the transition probabilities for the environment, R is the reward function and γ the discount factor. Figure 4.1 presents a visualization of the RL framework for the patrolling robot. The robot must decide which vertex to travel to next (action) and will then receive both a reward from the environment (security game) and a transition to a new state.

Considering that in our setting the environment is modeled as an undirected graph, we introduce a state representation with the aim of encoding key aspects of the patrolling strategy realization over a finite temporal history. Specifically, we

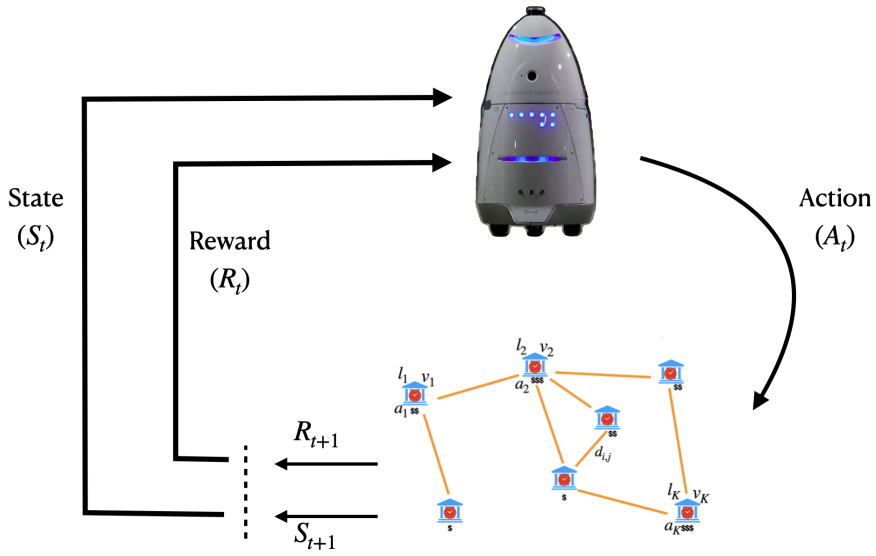


Figure 4.1: Illustration for modelling the patrolling security game as a Markov Decision Process (MDP), or in other words as a problem of sequential decision making. Given an observation from the environment and a reward signal, the defender must decide which action to take so as to maximize long term rewards. Despite the need for large amounts of data, RL techniques offer the robustness of not needing an explicit model for the *attacker*. As long as the attacker can be simulated (implemented through software), then the RL defender can play patrolling games against the attacker and learn to adapt its schedule accordingly to beat the intruder. Furthermore, we show that this framework learns to succeed even against a multitude of attacker behaviors.

define a state as a $(M + n + 1)$-dimensional vector combining several patrolling-related aspects of the graph and the patroller's past decisions:

$$[\ t_i, p_{i-M}, ..., p_{i-1}, p_i, s_0, ..., s_n\ ]$$

The first entry in the state representation (t_i) is the current time measured as the sum of the d_{ij}'s up to the ith transition in the patrolling mission. The next M entries in the state vector represent the last M vertices that the patroller visited. Finally, the last n entries denote the current (or instantaneous) *idleness* of each vertex, a common optimization criteria used in literature [81]. Idleness is defined as the time since the patroller's last visit to a particular vertex. Hence, the currently occupied vertex has always an idleness of 0 that starts increasing as long as the patroller leaves the vertex and does not return to it. The state aims to both capture the current coverage of the graph and any cycles relevant to patrolling the graph. Each idleness value encodes information about which vertices are visited more often (resulting in a lower idleness value) and which vertices are being ignored (larger idleness value). Furthermore, the history of past vertices is included to enable the patroller to make correlations between graph cycles, defined as a sequence of traveled vertices where the first and last vertex are the same, and coverage of the graph.

The use of MDP based representations is a widespread practice in patrolling [32]. Our formulation, however, adopts a richer state description by combining several attributes. The action set A for every state is the entire set of vertices, $\{l_1, ..., l_k\}$, denoting which target location the patroller will move to next. We assume deterministic transitions and use a discount factor of 0.99 for all experiments. Moreover, M in all experiments equals k, or the number of vertices.

We adopt a reward function that rewards the patroller *only when an attack is neutralized*. In other words, every action that the patroller takes receives a reward signal of 0, however, when the state transition results in the capture of the attacker the patroller receives some positive real reward, r. After a parameter sweep, we settled on a reward value of 5 for captures. A limitation of this reward function is the sparseness of the signal and from our experience generally requires that the patroller captures the attacker within a reasonable number of transitions, in

order to learn. In our experiments, this was handled by tuning the deadline of the attackers. A modest deadline (about 10 transitions) allows for the algorithm to produce adequate results during training. A few more points on the design of the reward function: the deadline for the attacker is not fixed and as explained later in this section it is sampled uniformly from a list of deadlines, a too large deadline makes the problem very difficult for the RL defender and results in very slow improvement as the RL agent trains, and finally the signal ultimately produces a patroller that attempts to capture the intruder as much as it can. The final point sheds light on the limitations of the present work and a broader debate about the appropriate measures for success in patrolling scenarios. For example consider an attacker that waits indefinitely for the best moment to attack and thus may decide to "never" attack because its interal belief model never believes it will lead to success. Our current framework would not be applicable to such an attacker since the RL agent's feedback is realized through captures and if the patient intruder waits indefinitely then the patroller never receives feedback. However, at the same time a patroller that could keep out an attacker without end because of its strategy, could also be seen as a highly successful defender. Thus our stated goal by mere design of our reward function is to capture the intruder and by virtue of our premise, our RL framework will only work against intruders that must attack eventually.

Indeed when the current framework was applied to some of the intruder models discussed in chapters 3, 5 the results were not encouraging. Despite a re-designing of the reward signal, +1 for every turn that does not result in the end of the game, the RL agent could not learn to beat K Nearest Neighbor or Maximum Likelihood attackers. Resolving these issues will require more careful design and curating.

The proposed function encourages the patroller to extend the episode for as long as possible by rewarding it for repeatedly capturing the attacker indefinitely. We opted for not including a negative reward signal at the end of the episode when the patroller is unable to counter the attack because experiments showed that including the negative signal had almost no effect on the performance. We postulate this is due to the fact that as the patrolling agent begins to capture more and more

attackers, the accumulation of positive rewards outweighs the single negative signal that could be sent to the agent during learning at the end of a failing episode (the patroller can capture many attackers in one episode, but can only lose the game once).

4.2.2 Resolution with Proximal Policy Optimization

Policy gradients [91] is a family of methods for finding an optimal policy of an MDP where the agent's policy itself is parameterized by a vector θ and the optimization happens in the policy space $\pi(a|s,\theta)$. To use policy gradient methods in our setting we define a policy as $\pi(l_j|s_t,\theta)$ i.e., the policy receives as input the state at time t and outputs the probability of selecting l_j as the next vertex to visit.

These policy gradient methods can be extended to Actor-Critic [90] methods where not only is the agent's policy parameterized (Actor) but also a value function approximation (Critic) is used to discriminate between good and bad performing actions. By adopting Proximal Policy Optimization (PPO) [84] we approximate the objective by a first-order surrogate problem we will refer to as *Clipped–PPO* agent. The agent obtains its name from the objective function used to optimize the actor:

$$L(\theta) = \mathbf{E}[\min(f_t(\theta)A_t, clip(f_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)]$$

where

$$f_t = \frac{\pi(l_t|s_t,\theta_t)}{\pi_{old}(l_t|s_t,\theta_t)},$$

A_t is an estimator of the advantage function at time step t, and ε is some small real valued scalar.

The intuition behind Clipped–PPO follows from the reasoning of the Trust Region Policy Optimization methods (TRPO) [83], i.e., that a too large update on the policy will result in a destabilization of the learning process.

Whereas TRPO methods opt to use constrained optimization techniques resulting in the need to calculate higher order approximations of gradients, Clipped-PPO moves the constraint on the update into the objective function in the form of the *clip* function which will bound the update on the policy.

Formalizing PPO to our into our patrolling game setting first requires defining a policy as $\pi(l_{t+1}|s_t, \theta)$ i.e the policy will receive as input the aforementioned state representation at time-step t (which includes the current patroller vertex) and output the next vertex for the patroller to visit.

In an adversarial patrolling setting where the attacker is able to make observations of the patroller's strategy, the defender must exhibit some non-determinism or else risk making the prediction problem too simple for the attacker. This means that the optimal strategy for the patroller will be a *mixed strategy*. Stochastic policies are a natural way to model mixed strategies over an undirected graph leading to our choice of using the Clipped Proximal Policy Optimization agent (Clipped–PPO) from [84]. Figure 4.2 denotes the architecture of our PPO network. The input to the network comes from the observation in the MDP and after passing through feed-forward layers will return a probability distribution over the vertices. Finally, the defender will sample the probabilities to decide which target location to visit next.

4.3 Modeling the Attacker's Behavior

An important departure in our work from others is the explicit modeling of attacker behavior. In general, one can have different attacker models, each representing a potential attacker to defend against. We propose six different attacker models and discuss their corresponding rationale. All attacker models draw their deadline randomly at the beginning of each game from the discrete set of values $\{100, 150, 200, 300\}$ in all experiments. The six attackers we consider are described in the following.

Max Idleness Attacker (MIA)

Once the deadline is reached, the MIA will attack the vertex with the largest idleness. The MIA model will target locations that have been left vulnerable for more since a vertex with larger idleness means that the patroller has ignored that particular vertex for longer. Generally, a larger deadline makes this model (and

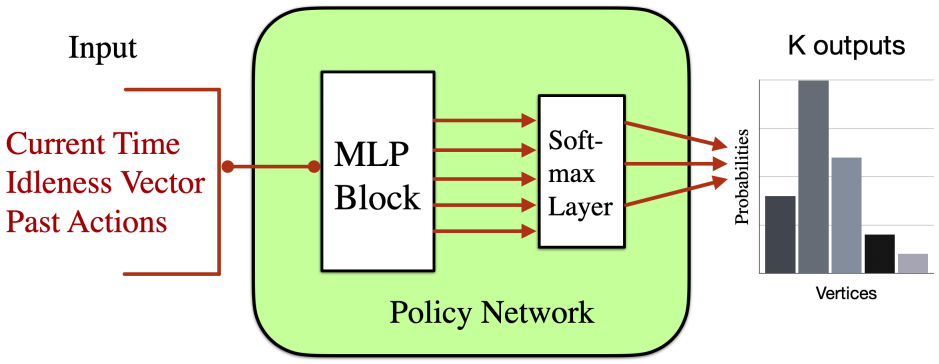


Figure 4.2: Figure depicts the network architecture used for the PPO algorithm. Input to the network corresponds to the observation received from the Markov Decision Process: the current time, the instantaneous idleness of each target location, and a history of size M (usually equal to N) of past actions. After passing through an MLP block and a final soft-max layer the network transforms the input into a probability distribution (logits) over the graph's vertex set. A function modelling the distribution acoording to the logits then returns which vertex the defender will move to next.

other idleness models) stronger since it is able to observe the patroller for a longer period of time. This model assumes that the attacker can observe all vertices in the graph.

Average Idleness Attacker (AIA)

Once the deadline is achieved this model will penetrate the vertex with the largest average idleness. Average idleness is measured as:

$$\frac{1}{t}\sum_{y=0}^{y=t} I_y$$

where I_y is the instantaneous idleness at transition y. The rationale here follows from the MIA model in that this model will penetrate more vulnerable (i.e., larger idleness) targets. However, this model will not consider instantaneous idleness but average idleness up to the current time step. This attacker observes a history of idleness and attacks the vertex that has been visited with the least regularity. Like MIA, AIA assumes full observability of the graph.

Idle Subset Attacker (MISA)

This attacker model works exactly like MIA, but it only focuses on a subset of the vertices and completely ignores the vertices not in its purview. This model's rationale follows exactly from the MIA model, but also it captures scenarios where the attacker is not interested in all the target locations or cannot observe the entire graph. Whether because of preference or restriction, this attacker models intruders who have curtailed their total attention. In all subsequent experiments, the MISA attacker focuses on a fixed subset consisting of half of the target locations.

Scaled Idleness Attacker (SIA)

This model attacks the vertex with the largest value of the product between the idleness vector and the vector of vertex values once the deadline is reached. This attacker uses the idleness information but also incorporates a preference for target locations with higher values. This model emulates intruders that are not interested in vertices that are being ignored by the patroller if these have low values.

Max Value Attacker (MVA)

Once the deadline is reached, the Max Value Attacker always attacks the vertex with the largest value. Vertex values are generated once for a particular graph and remain static. This MVA model implements cases where the attacker is only interested in the largest payout and thus repeatedly tries for the target with the most assets. This model's observational capabilities are limited to the maximum value target location and is unaware of the patroller location when it is not at the maximum value vertex.

Preference Attacker (PA)

Once the deadline is achieved this model chooses a vertex to attack according to a probability distribution over the vertices. This model does not take into consideration idleness or value. As the distribution approaches a uniform probability the problem becomes more difficult for the patroller. The PA model is a general model that can represent any preference that the attacker might have over the vertices and models an intruder who has the ability to infiltrate any target location but cannot observe the patroller's movements. All experiments presented here (where $|V| = 10$) were done with a distribution of 0.86 probability on one vertex and the remaining 0.14 distributed non-uniformly among the other vertices.

4.4 Training, Domain Randomization and Results

All the graphs presented here are created by randomly placing points on a grid of size 50 by 50; It is important to note that the attackers can be divided into categories: those that make an attack based on some function of the graph idleness induced by the patroller's schedule, and those that do not. We can expect that the policies that will yield large rewards against the idleness group will not perform well when compared to the non–idleness group since the objectives are not correlated.

Before training considering multiple attackers, we assess whether the Clipped–PPO agent is able to learn against individual attacker models. Figure 4.3 shows the learning curves for the Clipped–PPO for such a scenario.

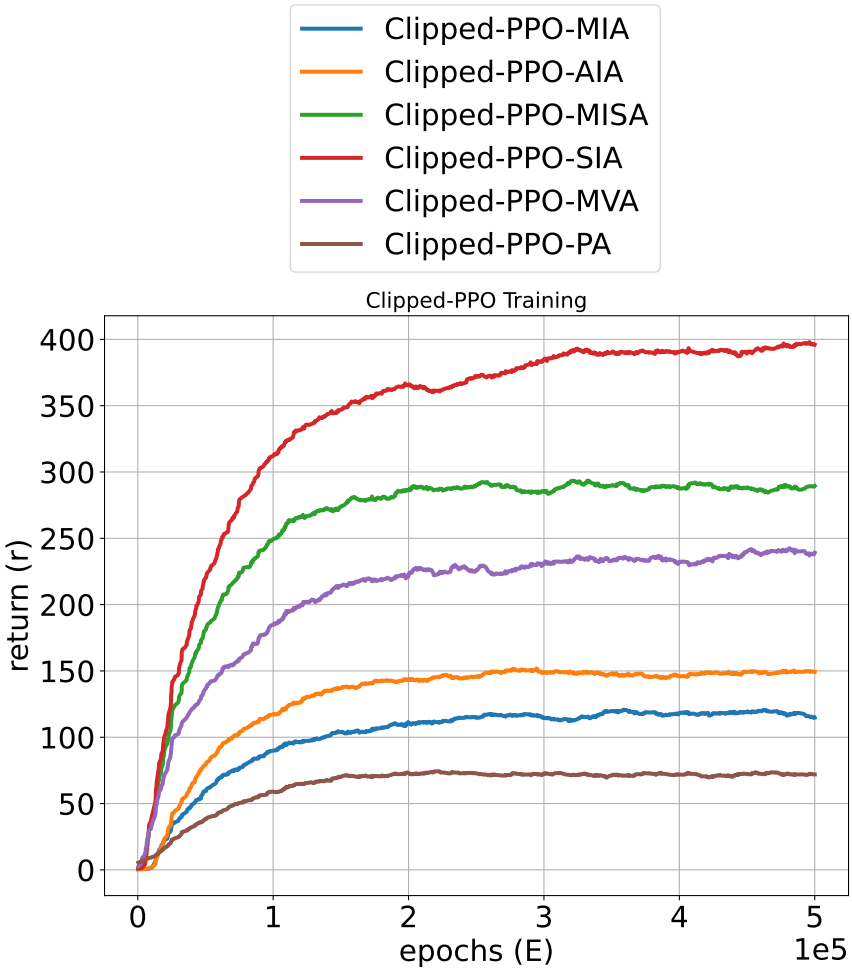


Figure 4.3: Learning curves for the Clipped-PPO agent against the different attacker models. $|V| = 10$ and all curves were smoothed using a sliding window average. The x-axis represents the number of epochs trained and the y-axis shows the returned cumulative reward. Some attacker models are easier to learn against than others as evidenced by the slope and final value of the distinct curves.

A single architecture was used to generate all of the curves. It consists of an actor and value network both with two hidden layers. For both networks, the first hidden layer contains 200 units, and the second hidden layer 100 units.

Training procedures and agent implementations are built upon the TF–Agents library [41]. An epoch consists of running 30 game simulations in parallel and then optimizing both actor and critic networks according to the loss functions and training loop presented in [84]. The learning rate starts at a value of $1 \cdot 10^{-4}$ and is annealed according to an exponential decay function. Rewards and observations were normalized and for the Clipped–PPO agent an ε value of 0.01 was used. From the slope and final values of individual curves, it is evident that some of the attacker models are easier for the Clipped–PPO agent to learn against than others. For example, the Max Value Attacker curve (in purple) shows that the patroller quickly learns a good policy for defending the graph. However, the policy that the agent settles on may not be robust to other attacker models (corroborated later empirically).

	MIA	AIA	MISA	SIA	MVA	PA
Clipped-PPO-MIA	**118.2**	29.6	93.4	93.8	0.0	1.7
Clipped-PPO-AIA	68.5	**150.1**	2.0	51.1	0.0	34.8
Clipped-PPO-MISA	0.4	0.4	**293.5**	76.8	53.3	0.0
Clipped-PPO-SIA	0.7	0.7	1.0	**400.5**	8.1	7.0
Clipped-PPO-MVA	0.2	0.2	0.4	46.6	**238.7**	2.0
Clipped-PPO-PA	0.6	0.4	0.0	0.0	0.0	**76.7**
MI-P	**24.3**	3.4	3.3	2.1	0.5	1.0
DR Clipped-PPO	37.4	35.4	43.4	**207.7**	77.6	3.6

Table 4.1: Confusion matrix for a graph with $|V| = 10$. An entry in the table is generated by first training a Clipped-PPO agent exclusively against some attacker model then deploying it against different attackers. Results are averaged over 2,500 games.

Next, we present the Domain Randomization (DR) [95] training procedure to generalize across attacker models. The approach at training time exposes the RL agent to all of the available attacker models instead of just a single model.

Tables 4.1 and 4.2 [1] show the results for the case of DR training, where before the start of any game (or episode) during training an attacker model is selected by sampling uniformly over all the available models. The first six rows of Table 4.1 form a confusion matrix for the different Clipped–PPO agents, while the last row shows the performance of the Domain Randomization (DR) Clipped–PPO agent.

	Average Performance	Standard Deviation	Worst Case
Clipped-PPO-MIA	56.1	51.9	0.0
Clipped-PPO-AIA	51.1	55.5	0.0
Clipped-PPO-MISA	**70.7**	113.9	0.0
Clipped-PPO-SIA	69.7	162.1	0.7
Clipped-PPO-MVA	48.0	95.2	0.2
Clipped-PPO-PA	12.9	31.2	0.0
MI-P	5.8	9.2	0.5
DR Clipped-PPO	67.5	72.6	**3.6**

Table 4.2: Results presented here are some statistics gathered from Table 4.1. Domain randomization helps the Clipped-PPO patroller generalize over the attacker models. As shown here, the DR Clipped-PPO agent has a reasonable trade-off between average performance and variance while at the same time has a better worst case return.

Each entry is a testing scenario where a Clipped–PPO agent is deployed against a different attacker model. The table corroborates the expectation that an agent trained exclusively on a single attacker model will perform best against the attacker it was trained on but will also generalize poorly to other attacker models. Row "Clipped–PPO–MVA" clearly shows that while the agent can learn to protect well against the MVA, the resulting policy leaves the patroller vulnerable if the attacker changes strategies. DR Clipped–PPO, however, performs well across the attacker models as shown in table 4.2. Presented in table 4.2 are the averages of

[1] Note that the results presented here differ from the original paper as there was a error in the implementation of one of the attacker behaviors (SIA).

the scores shown in table 4.1 as well as the calculated standard deviation of the scores. The DR Clipped-PPO agent has a reasonable trade-off between average performance and standard deviation while at the same time has the best worst-case performance. Thus if a user were unsure about which opponent she might face, the DR-Clipped-PPO provides the best guarantee of performance assuming that the opponent behavior is in the attacker model library used during training.

For completeness, we also compare our method against a heuristic strategy that represents an established class of approaches to the patrolling problem, [70, 71, 101] (see also chapter 2 for a discussion of idleness optimization techniques). These works tackle the patrolling problem as one of computing a route that minimizes some form of the idleness (instantaneous, average, or expected) over the lifetime of the patrol. Generally speaking, the problem of finding an optimal path is NP-hard and thus researchers have focused on developing effective heuristic methods. A popular scheme is the design of a greedy heuristic that, as the name suggests, acts optimally within a local context for the patroller.

Thus we introduce, the *Maximum Idleness Patroller* (MI-P) that will represent the aforementioned class of heuristic idleness patrollers. The MI-P attacker model, at every turn, decides to move to the vertex with the largest instantaneous idleness. In this way, the MI-P acts greedily with respect to immediate information about all the vertices and approximates a schedule that will minimize the average idleness induced over the entire graph. As can be seen from Table 4.1, the MI-P patroller performs its best when playing against the idleness–based attacker models. Our Clipped–PPO–MIA agent does better by more than a factor of 10 meaning it learns ways to manipulate the idleness so as to capture the attacker. The MI-P meanwhile does not respond or react to the attacker. A point of detail here is that, like other works [70] we set the initial instantaneous idleness for all the verities as 0. We programmed the MI-P to break ties between the vertices randomly and as such will initially visit all the of vertices once and then realize a deterministic route along the graph according to the idleness.

We also present table 4.3 with the normalized values found in the table 4.1. Here we see similar trends with the DR Clipped-PPO agent achieving the best

	MIA	AIA	MISA	SIA	MVA	PA	AVG.	STD.
Clipped-PPO-MIA	1.00	0.20	0.32	0.23	0.00	0.02	0.30	1.77
Clipped-PPO-AIA	0.58	1.00	0.01	0.13	0.00	0.45	0.36	2.17
Clipped-PPO-MISA	0.00	0.00	1.00	0.19	0.22	0.00	0.24	1.42
Clipped-PPO-SIA	0.01	0.00	0.00	1.00	0.03	0.09	0.19	1.14
Clipped-PPO-MVA	0.00	0.00	0.00	0.12	1.00	0.03	0.19	1.15
Clipped-PPO-PA	0.00	0.00	0.00	0.00	0.00	1.00	0.17	1.01
MI-P	0.21	0.02	0.01	0.01	0.0	0.01	0.04	0.26
DR Clipped-PPO	0.32	0.24	0.15	0.52	0.33	0.05	0.26	1.59

Table 4.3: Normalized (by column) confusion matrix for a graph with $|V| = 10$. An entry in the table is generated from the values found in table 4.1. Each value in the table is obtained by dividing the score by the largest score in the column.

worst-case performance, all while obtaining a good trade-off between its average normalized performance and standard deviation. Note that a lower variance value here is not necessarily a better performing model, since an agent that always scores poorly against all the attacker models will have a lower variance.

DR Clipped–PPO was able to learn a patrolling strategy that can cope against what from the perspective of the patroller is an unknown and non–stationary attack–generation process. Particularly significant is the fact that the attacker behaviors we combined exhibit substantially different dynamics. Idleness attackers (MIA, AIA, MISA) exploit observations of the patrolling strategy's realization. Value–related ones (MVA, PA) follow a set of predetermined and arbitrary preferences, which depend on the environment, not on the patroller, and the remaining one (SIA) combines idleness and values. In traditional patrolling applications, these dynamics always result in conflicting objectives to be optimized. The results obtained with DR Clipped–PPO show how our method is a first promising step to overcome this limitation and build robust patrolling agents.

4.4.1 Ablation Study

Our state representation (and input to our function approximators) can be partitioned into the tuple (t, P, S) where: t is a real–valued scalar representing the current time, P is a vector of size K denoting the last K vertices visited, and S is also a vector of size K denoting the instantaneous idleness of each vertex. Building our state representation equates to concatenating t, P, and S. As discussed in Section 4.2, the P vector is a history of the previous actions taken by the patrolling agent which helps the agent identify cycles and relevant patrolling patterns. The S vector (also called idleness vector) while also containing information related to past actions identifies areas of the environment that are under–visited. To corroborate the individual usefulness of S and P we study the training performance of the Clipped–PPO agents: first by removing only the history vector P from the state and then by removing only the idleness vector S from the state. For all experiments, the size of vector P is equal to the number of vertices in the graph. We, in this section, focus on two attacker models that are representative of our attacker library: the Max Idleness Attacker (MIA) and the Preference Attacker (PA).

Figures 4.4 and 4.5 show the training performances under different state representations against MIA and PA, respectively. Focusing on Figure 4.4 (MIA) first, we see that removing the idleness component from the state diminishes the performance of the RL agent more than when solely removing the history S. Intuitively, this makes sense as the RL patroller is inferring correlations between the idleness component in the state and the attacker's decisions. Removing one or the other, however, does have a negative impact on the RL agents' training performance. Figure 4.5 (PA) demonstrates a similar pattern wherein removing the idleness information significantly harms the training performance of the RL patroller. From both charts, we may deduce that the idleness component in our state representation provides vital information for the patroller during the learning process. The idleness vector provides ongoing information about which vertices have been visited most recently, thus also capturing information about past actions. From both figures, we see that the actual history of actions alone is

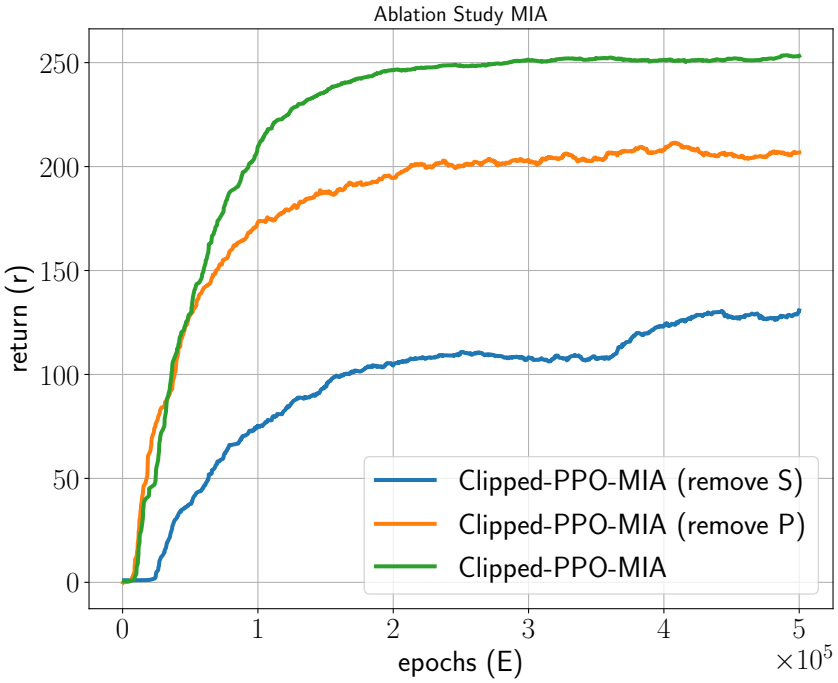


Figure 4.4: Ablation study of the state representation when training Clipped–PPO against the Maximum Idleness Attacker. The orange curve represents the performance after removing the history of actions, P, from the state and the blue curve shows the performance after removing the idleness vector, S from the state. Removing either results in a decreased performance.

not sufficient for learning to patrol the graph, but as is the case with Figure 4.4 its addition can improve performance. Note that removing both the components S and P would leave the state vector as a single scalar and thus we did not perform any experiments as such.

4.5 Conclusions

In this chapter, we proposed a method to deal with adversarial patrolling using deep reinforcement learning. We cast the problem in an RL setting where the reward function is based on the realization of attacks that can follow arbitrary logic that is unknown to the patroller. Our main contribution is the combination of a Proximal Policy Optimization agent and Domain Randomization training

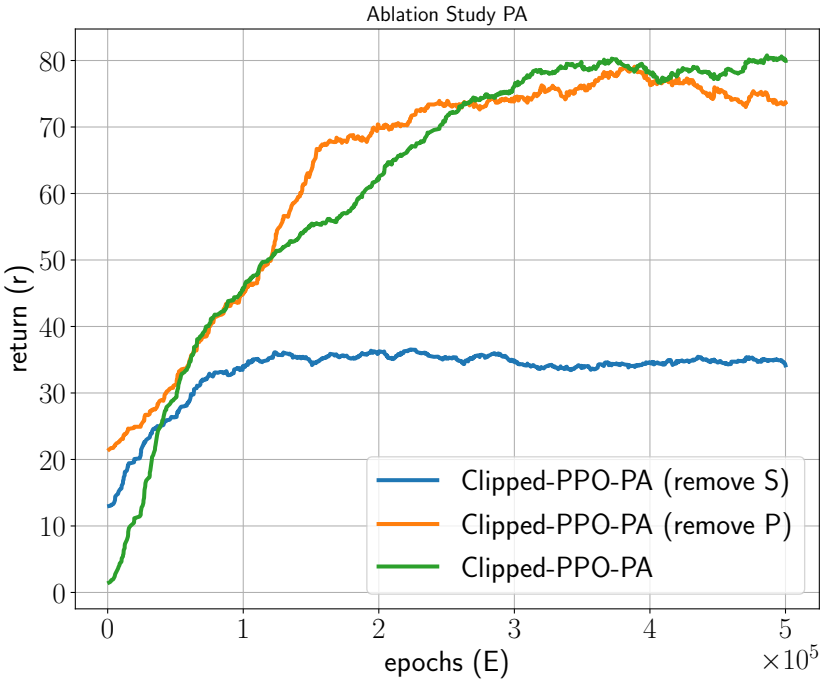


Figure 4.5: Ablation study of the state representation when training Clipped–PPO against the Preference Attacker. The orange curve represents the performance after removing the history of actions, P, from the state and the blue curve shows the performance after removing the idleness vector, S from the state. PA uses the same distribution as described in section 4.3.

techniques to generate patrollers that are robust to changing attacker strategies. Key to the method is changing attacker models before the start of every episode which effectively exposes the RL patroller to different MDPs during training. This method pushes the reinforcement learning algorithm (PPO) to converge to a sort of *average* policy that is able to generalize across different environments.

To the best of our knowledge, this is the first work that applies a framework and presents significant results against a mixture of very distinct attacker behaviors. Our ablation study revealed that including a history of past actions in the state is less important than including the current instantaneous idleness, yet it is needed. Future directions include the application of graph neural networks to the actor and critic networks and multi-patroller settings where a team of patrollers must cooperate to patrol an environment. This involves studying different network architectures for the graph neural network and applying Centralized Training and Decentralized Execution (CTDE) methods to handle the multi-robot setting. With respect to the GNN, we could obtain a model that does not need re-training for every new graph instance since the new network could handle variable graph size input. Furthermore, the CTDE technique will encourage agents to spread out so as to cover the entire graph and avoid redundancies.

Chapter 5

Multi-Agent Techniques Against an Opponent with Limited Information

The previous two chapters focused on approaches for a single defender playing the patrolling security game. The subject of the next two chapters, however, extends the problem to incorporate teams of patrollers. The team patrolling problem requires more complex solutions and strategies for cooperation, thus making it more challenging. This chapter, compares specifically different schemes for coordination and cooperation. Results are presented and discussed that show no single method consistently outperforms any other. A good designer must tailor the appropriate solution to their specific use-case.

Moreover, for the work in this chapter and in chapter 6 we assume that the agents do not have access to infrastructure for agent to agent communication. This assumption allows us to focus our attention on offline centralized approaches, but does limit the system in some particular ways.

Firstly, because the agents do not communicate any plans amongst themselves there is the potential for redundancy, or inefficient use of patrolling resources, if two agents visit the same target location. In this chapter we slightly mitigate this effect by computing a separate Markov chain for each agent. In chapter 6 we instead restrict each agent to have the same maximum velocity and also have the

agents start along a set of initial positions that are equally spaced along the given patrol path.

The work presented here was originally published in [9].

5.1 Multi-Agent Patrolling Setting Definition

Embraced here is the usual graph-based setting which adopts the model we recently studied in chapter 3 for the single-robot case. Next we introduce the presence of multiple patrollers. Assume an environment where K target locations, *targets* for short, must be protected by means of repeated visits. We assume that this graph is connected and we will always work on its transitive closure.

Consider $M \leq K$ *attackers*, each with the objective of compromising a given target. To successfully compromise a target l_i, an attacker must spend a time greater or equal to a_i on such a target without being detected by any patroller. Attacks are non-preemptive, meaning that they will terminate either with a success or a capture. The interaction between the team of patrollers and each single attacker can be thought as driven by a constant-sum revenue distribution where the patrollers get the total amount of protected value and the attacker gets the value of the target it compromises. Formally, patrollers and any attacker will get $(\sum_{i=1}^{K} v_i, 0)$ in case of a failed attack, and $(\sum_{i=1}^{K} v_i - v_j, v_j)$ in case of a successful attack on target l_j, respectively. We will not use such payoff structure in a game-theoretical model, but embrace the underlying interpretation where v_i can be seen as the value *stored* in l_i, while a_i encodes a measure of resiliency of the target.

To protect the targets, $N < K$ patrollers are deployed. We assume that when a patroller visits a target under attack it neutralizes the attacker: if one attacker attacks l_i at time t and the next visit to l_i by any of the patrollers occurs before time $t + a_i$, the attacker fails its attempt. If instead no patroller visits l_i before $t + a_i$, l_i is compromised.

The motion of the patrollers in the graph is governed by strategies that are not known to the attackers. However, through repeated observations, each attacker can construct a belief based on the following assumptions:

1. each attacker has local observation capabilities spatially confined to a single given target (as in chapter 3);

2. patrollers are indistinguishable;

3. attackers are unaware of each other.

Assumption 1) allows each attacker to know the times at which a patroller arrives and leaves the target it is observing. This target is chosen before gathering any observation of the patrollers and hence it is not influenced by them. In other words, the strategic decision that the attacker has to perform is whether to attack a given target under observation or not. We assume that the M targets under observation are unknown to the patrollers. This assumption reflects our choice of locally limited observation capabilities: our attackers cannot gather observations from multiple targets and use such a knowledge to strategically choose which target to attack.

Assumption 2) means that an attacker observing a target can just see that one of the N patrollers is visiting but it cannot distinguish which one.

Assumption 3) implies that attackers cannot share the observations they gather, nor they can coordinate their decisions to maximize their chances.

As in chapter 3, we assume that the patrolling strategy followed by the i-th patroller can be modeled by a time-invariant Markov chain described by a $K \times K$ transition matrix $\mathbf{P}^i$ whose entries are all strictly positive. As a consequence of the properties of the transition matrix, all agents will eventually visit every vertex in the environment. As we will see in the following, we will consider both the case where all the patrollers follow the same strategy and the case where each patroller follows a different one. The (i, j) entry in $\mathbf{P}^i$ represents the probability that the i-th patroller will visit l_j after having visited l_i. When a patroller moves from l_i to l_j, it will take a time of at least d_{ij}. In chapter 3 we introduced a new class of strategies characterized by *motion delays*. The main idea builds upon the assumption that when a patroller moves from i to j it will introduce a random traveling delay so that the overall transition time will be $d_{ij} + \zeta$ where ζ is a random variable drawn from a uniform distribution over $[0, \Delta]$. As we showed in chapter 3, such

random delays could make it more difficult for an attacker to precisely forecast the next visit to a target. As a result, these strategies outperform other non-delayed approaches against attackers that condition their attack on patroller observations they have collected on the observed target.

The transition matrix $\mathbf{P}^i$ is assumed to be given and computed from an optimization taking the patrolling setting as input and returning the optimal stationary distribution $\boldsymbol{\pi}^i$ from which $\mathbf{P}^i$ can be reconstructed via the Metropolis-Hastings algorithm (exactly the same procedure as presented in chapter 3). Notice that, since $\mathbf{P}^i$s have positive entries, each of them has a stationary distribution $\boldsymbol{\pi}^i$ (a K-dimensional vector such that $\sum_{j=1}^{K} \pi_j^i = 1$ and $\boldsymbol{\pi}^i = \boldsymbol{\pi}^i \mathbf{P}^i$.)

Against this background, we study adversarial patrolling when this is carried out by a team of N patrollers executing the above defined patrolling strategy over one or more sub-graphs that span the whole environment. Patrollers execute their strategies independently, meaning that we do not consider any online coordination taking place. Instead, we allow *offline coordination*, seen as a subdivision of the efforts over different sub-regions of the environment before the mission starts, and how it can impact on the resulting level of protection. To do this, we will consider and compare two configurations:

- **Partitioned Patrollers**: the patrolling effort is distributed according to a partition of the environment (computed offline). Each patroller is assigned to one sub-region (sub-graph); the sub-regions do not overlap and their union covers the whole environment. Here offline coordination is performed.

- **Non-Partitioned Patrollers**: no patitioning is performed, each patroller potentially covers the whole environment. Here offline coordination is not performed. If patrollers do not move in sync, this approach will in general shorten the time between successive visits to a vertex, thus making it harder for attackers to succeed.

Each configuration has its advantages and disadvantages. The partitioned case allows the patrollers to allocate more resources where needed. For example, the environment could be partitioned into small sub-regions when the total value of

the covered targets is high. In general, the sub-regions allow frequent revisits, thus making it harder for an attacker to succeed. On the other hand, this approach is less robust to faults. If one of the patrollers were to stop working, the sub-region that was assigned to it would remain uncovered. The non-partitioned approach instead, is more resilient to failures because even if one patroller fails, all other patrollers can periodically revisit any location.

5.2 Partitioned Patrollers

We start by considering the configuration where the environment is partitioned into N sub-graphs and each patroller is assigned to one of them. The rationale here is to balance as much as possible the workload among the patrollers.

A partition of $G = (V, E)$ can be represented with a set $\{V_1, V_2, \ldots, V_N\}$ where $V_i \subseteq V$, for any $i \neq j$ it holds that $V_i \cap V_j = \{\emptyset\}$, and such that $\cup_{i=1}^{n} V_i = V$. Given a partition element V_i, we denoted as $G[V_i]$ the subgraph induced by V_i on G. Such an induced subgraph has V_i as the set of vertices, the set of edges is given by $E_i = \{(l_i, l_j) \in E \text{ s.t. } l_i \in V_i \wedge l_j \in V_i\}$, and it represents the sub-region of the environment that will be assigned to the i-th patroller. To compute partitions, we need to quantify the patrolling workload. For any partition element V_i, we choose the cumulative temporal cost of the edges in E_i:

$$I(G[V_i]) = \sum_{(l_i,l_j)\in E_i} d_{ij}$$

Ideally, it is desirable to seek partition elements with low values of I. A small cumulative temporal cost is an indicator that the total time needed to cover $G[V_i]$ can be limited. These features can ease the patrolling task on the environment's sub-region associated to $G[V_i]$: protecting an area where targets tend to be close to each other can ensure higher frequencies of visits and, hence, better protection. Clearly, this is an heuristic principle which does not guarantee any optimality on the patrolling performance and that does not constitute the only possible choice (for example, the maximum temporal cost could be an alternative.) We decided to opt for such a metric for its simplicity of definition and, more importantly, for the

fact that it will allow us to devise two different methods for computing partitions.

Given I, our partitioning problem can be stated as the following: compute a partition $\{V_1, V_2, \ldots, V_N\}$ of G such that $\max\{I(G_1), I(G_2), \ldots, I(G_N)\}$ is minimized. This problem shares core features with the well-known Graph Partition (decision) Problem (GPP) defined as follows: given a weighted, undirected graph $G = (V, E)$ is there a $V' \subset V$ such that $I(G[V']) = I(G[V \setminus V'])$? As shown in [38] (problem SP12), the GPP is $\mathcal{NP}$-complete from the *partition* problem. An immediate consequence of this result is that our partitioned patrollers problem is $\mathcal{NP}$-hard to solve optimally. In the following we introduce and evaluate two methods to solve this problem. Figure 5.1 depicts the outcome of partitioning a graph with 3 total agents.

5.2.1 Multilevel Graph Partitioning

The first partitioning approach we propose is a heuristic based on the multi-level graph partitioning (MLGP) discussed in [47] and [68]. This method is divided into three steps: graph coarsening, partitioning, and uncoarsening. The final partition is both balanced in terms of the number of vertices in each sub-graph and also minimal with respect to the graph cut or inter-sub-graph cumulative edge weight. The balanced k-way minimum-cut partition performed by the MLGP algorithm does not directly solve our partitioning problem. Indeed, our formulation seeks partitions that minimize the cumulative weight of graph edges that, under the MLGP formulation, are left uncut. Because of this, we introduce two transformations of the edge weights. Figure 5.2 visualizes the process of the Multi-Level Graph Partition algorithm. Graphs are coarsened, cut, and then un-coarsened to produce partitions for the agents.

The underlying idea in both methods is to re-write the edge weights in such a way that the cut minimization performed by MLGP will produce a partition similar to a division based on the cumulative weight of uncut edges (those that, in our problem, are summed up to obtain I in each partition element.) Because edges that are part of the cut of the graph are discarded, we remove edges that will have a high weight cost (i.e. travel time). By inverting the edges weights of the

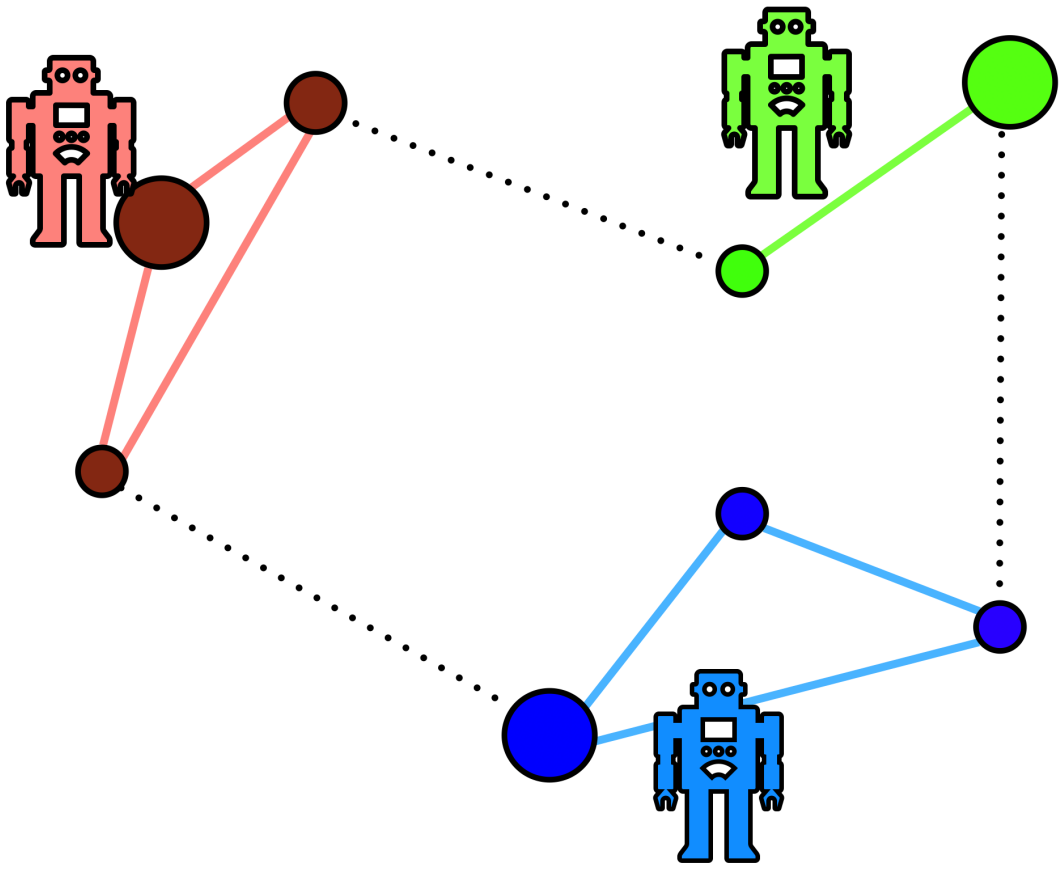


Figure 5.1: A common approach to the problem of team patrolling or team coverage involves partitioning the environment. In the discrete case (over a graph) the locations are separated into groups according to the number of patrolling resources (agents) available. Consequently, the problem reduces to a single agent instance for each member of the team. While effective at dividing the workload requirements for individual members of the team, the approach lacks robustness to agent failures. If a member of the team breaks down then a subsection of the environment is left unguarded until a new partition, with $k-1$ agents, is computed. The black dotted lines above represent edges in the original graph, that are not used after a partition is computed and employed.

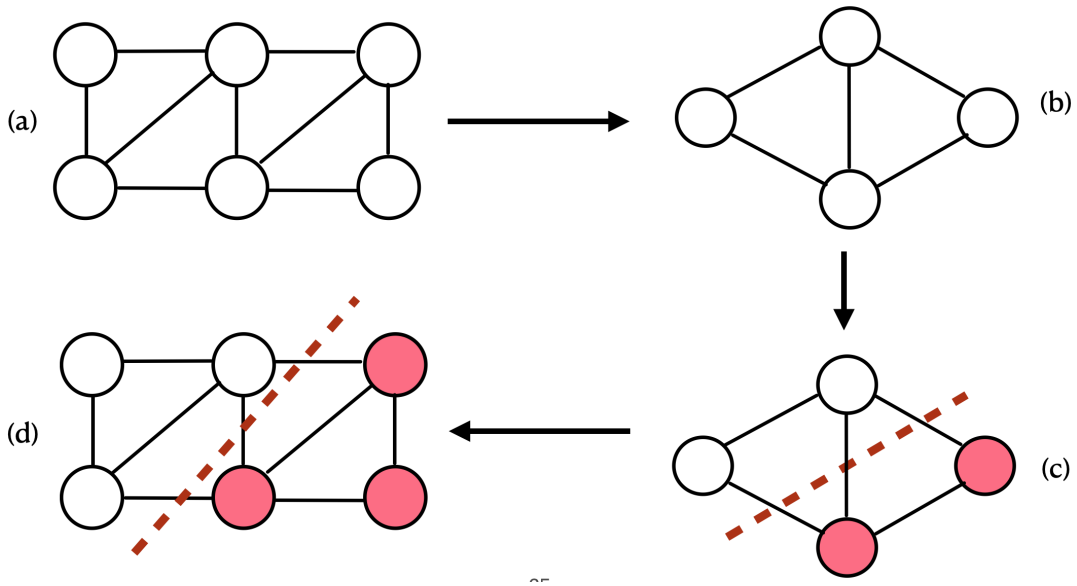


Figure 5.2: Illustration depicting the Multi-Level Graph Partitioning procedure. Sub-figure *a* (top left corner) depicts a regular input graph to the partitioning algorithm. Next, in sub-figure *b* the number of vertices in the graph is reduced by combining adjacent vertices and aggregating attributes. Then, in sub-figure *c* a maximum cut is computed on the smaller graph. In our case, the transformation computed on the edge weights pushes vertices far from each other into other sides of the k-way cut. Finally, the graph is restored to its original size via graph uncoarsening and partitions are taken according to the k-way cut. Sub-figure *d* presents the output of the procedure, a partitioned graph according to k number of sub-graphs.

graph, when minimizing the cut of the graph, the MLGP algorithm will remove edges with large weights, leaving those with small costs uncut (inside a partition element).

The first method for graph inversion is the following transformation of the graph edges:

$$d_{ij} = \left\lfloor \left(\max_{(l_i,l_j)\in E} \{d_{ij}\} - d_{ij} + 1 \right)^2 \right\rfloor \quad \forall l_i, l_j \in V$$

This transformation will leave the heaviest edges in the graph with the lowest weights, meaning that targets that in the original graph are very far apart will have, after the transformation, low edge weights. The k-way minimum-cut partition of the transformed graph will tend to keep vertices that were far apart out of the same sub-graph in the resulting partition. In other words, the edges in the minimum cut of the transformed graph will tend to have large weights in the original graph, which means that sub-graphs in the resulting partition will be characterized by low cumulative edge costs.

The second method for transforming the edge weights is based on a normalization of both the edge weights and each target's value-to-attack-time ratio defined, for a target l_i, as $\rho_i = v_i/a_i$. The normalization of these quantities is performed in this way:

$$d_{ij}^{norm} = \frac{d_{ij}}{\max\limits_{(l_i,l_j)\in E}\{d_{ij}\}}, \quad \rho_i^{norm} = \frac{\rho_i}{\max\limits_{l_i\in V}\{\rho_i\}} \quad \forall l_i, l_j \in V$$

$$d_{ij}^{norm} = \frac{d_{ij}}{\max\limits_{(l_i,l_j)\in E}\{d_{ij}\}} \quad \forall l_i, l_j \in V$$

$$\rho_i^{norm} = \frac{\rho_i}{\max\limits_{l_i\in V}\{\rho_i\}} \quad \forall l_i \in V$$

Then, we consider, for each edge, its normalized weight multiplied by the average normalized value-to-attack-time ratios of the two associated vertices. Thanks to the normalization, this operation will return a value between 0 and 1:

$$d_{ij} = d_{ij}^{norm} \frac{p_i^{norm} + \rho_j^{norm}}{2}$$

Finally the values are scaled by an arbitrary factor s and the ceiling is taken to avoid null weights:

$$d_{ij} = \lceil (s - s \cdot d_{ij} + 1)^2 \rceil$$

The idea behind this second transformation is to integrate in the new computed weights also a bias related to the value-to-attack-time ratios of the targets connected by an edge. The value ρ_i provides a measure of the *critical level* of a target. The higher the target's value the more critical it is because if it gets compromised the value loss will be large. Similarly, the lower the attack time the easier will be for the attacker to compromise that target. As a consequence, this second transformation not only will induce MLGP to push targets that are far apart into different sub-graphs, but also to try to separate targets that have a high critical level.

5.2.2 MILP-based partitioning

The second approach we introduce is an exact method based on the resolution of a Mixed Integer Linear Program (MILP). The following decision variables denote, for any sub-graph $G[V_i]$ induced by a partition element V_i, whether any target and edge of the original graph belong to that sub-graph, respectively. Formally, these variables are:

$$x_i^h = \begin{cases} 1 & \text{if } l_i \in V_h \\ 0 & \text{otherwise} \end{cases}, \quad y_{i,j}^h = \begin{cases} 1 & \text{if } (l_i, l_j) \in E_h \\ 0 & \text{otherwise} \end{cases}$$

Then the binary linear program reads as follows:

$$\min u \quad \text{s.t.} \tag{5.1}$$

$$\sum_{h\in\{1,\ldots,N\}} x_i^h = 1 \qquad \forall l_i \in V \tag{5.2}$$

$$\sum_{i\in V} x_i^h \geq \gamma \qquad \forall h \in \{1,\ldots,N\} \tag{5.3}$$

$$y_{i,j}^h \leq x_i^h \qquad \forall l_i, l_j \in V \tag{5.4}$$

$$y_{i,j}^h \leq x_j^h \qquad \forall l_i, l_j \in V \tag{5.5}$$

$$y_{i,j}^h \geq x_i^h + x_j^h - 1 \qquad \forall l_i, l_j \in V, h \in \{1,\ldots,N\} \tag{5.6}$$

$$u \geq \sum_{l_i,l_j\in V} d_{i,j} y_{i,j}^h \qquad \forall h \in \{1,\ldots,N\}, i > j \tag{5.7}$$

$$x_i^h \in \{0,1\} \qquad \forall l_i \in V, h \in \{1,\ldots,N\} \tag{5.8}$$

Constraints (5.2) force each target to belong to exactly one sub-graph in the partition. Constraints (5.3) require each sub-graph to include at least γ targets (in general $\gamma \geq 2$, and in our experiments we set $\gamma = 2$). This requirement translates to a minimum workload assigned to each patroller, trying to avoid situations in which one patrollers stays fixed on a given target as such a situation would be equivalent to removing that target and one patroller from the original problem. Constraints (5.4), (5.5), and (5.6) bound the x and y decision variables. These three set of constraints express, in a linearized form, the fact that and edge (l_i, l_j) belongs to a sub-graph if and only if both l_i and l_j belong to that graph too. Constraints (5.7) define the auxiliary decision variable u and any upper bound over all the cumulative temporal costs of the various sub-graphs in the partition. The objective function minimization provided in (5.1) requires to seek the minimum upper bound. Constraints (5.8) impose a binary integrality to the xs (these are the only integrality constraints needed since the joint effect of Constraints (5.4), (5.5), and (5.6) and the minimization (5.1) guarantee that, at the optimum, the ys always get a binary value).

5.3 Non-partitioned Patrollers

We consider a patrolling strategy formulated using a Markov Chain over a graph, as we did in chapter 3. The Markov Chain state space is $S = \{1, 2, \ldots, n\}$ and its transition matrix $\mathbf{P}$ has all entries strictly positive.

In this second configuration, we consider no offline coordination and also N patrolling units independently executing the same Markov strategy $\mathbf{P}$ (computed, like before, as described before but now over the whole graph. In this case no partition of the environment is computed.

One key aspect we aim at studying here is how the expected return times to a generic target l_j vary when the $N \geq 2$ patrollers are deployed. Indeed, such return times are at the core of the performances obtained by $\mathbf{P}$ against an attacker that tries to learn them by observing a single target and, when a confident estimate is reached, uses such knowledge to determine whether to attack or not. The return time to a target l_j when N patrollers use the same strategy can be formally defined as follows. For $1 \leq i \leq N$, let x_t^i be the state occupied by the i-th patroller at time t (this can correspond to being at some target or traveling along some edge). The return time to a target l_j of any of the N patrollers is then defined as the following random variable:

$$r_j^N = \inf\{k \text{ s.t. } \exists i, w, t \;\; x_{t+k}^i = x_t^w = l_j\}$$

In particular, we would like to determine how $\mathbb{E}[r_j^N]$ relates to $\mathbb{E}[r_j]$ (the expected return time with a single patroller).

Consider a Markov chain with K states and collapse it into a two-state chain where one state corresponds to state j in the original chain and state ξ represents the aggregation of all the other states different from j.

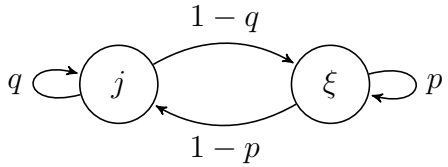


Figure 5.3: The collapsed two-states Markov chain.

Figure 5.3 depicts such a collapsed Markov Chain whose transition matrix is:

$$\mathbf{P_{coll}} = \begin{bmatrix} q & 1-q \\ 1-p & p \end{bmatrix}$$

Here, q denotes the transition probability from state j to itself and p will denote the transition probability from any state different from j (denoted by ξ) and remaining there. The expected return time to state j is given by:

$$\mathbb{E}[r_j] = \frac{(1-q)+(1-p)}{(1-p)}$$

Next, we can derive the relation between the values p, q and the non-collapsed transition matrix $\mathbf{P}$. The value q is $\mathbf{P}_{jj}$ as it corresponds to the probability of remaining in state j. To obtain p we solve for the value $1-p$. Using the total probability law $1-p$ can be derived as:

$$1-p = \sum_{i \neq j} \mathbf{P}_{ij} \mathbf{P}_i^{'}$$

where $i \in \xi$ and

$$\mathbf{P}_i^{'} = \frac{\pi_i}{\sum_{i \neq j} \pi_i}$$

$$1-p = \sum_{i \neq j} \mathbf{P}_{ij} \mathbf{P}_i^{'}, \text{ where } \mathbf{P}_i^{'} = \frac{\pi_i}{\sum_{i \neq j} \pi_i}$$

which corresponds to the probability of being in state ξ given that we are not in state j. Thus, the expected return time for any state in the new two-state Markov chain can be explicitly computed from the values in the original $K \times K$ transition matrix. Given this, we can now provide a characterization of the expected return times at the targets in the multi-patroller setting using the transition matrix $\mathbf{P}$ adopted by each of the N patrollers on the graph. We again formulate the multi-patroller problem as a two-state Markov chain. We will call *covered* the state where at least one patroller is at the target l_j, while *uncovered* is the state when

no patrollers are currently visiting l_j. The transition matrix associated to this Markov chain is:

$$\mathbf{P_{coll}} = \begin{bmatrix} q^* & 1-q^* \\ 1-p^* & p^* \end{bmatrix}$$

and

$$\mathbb{E}[r_j^N] = \frac{(1-q^*)+(1-p^*)}{(1-p^*)}$$

The value q^* in the transition matrix is associated with the probability that given there is at least one patroller on target l_j, at least one of them remains in that target in the consecutive transition. Next, p^* corresponds to the probability that given that none of the patrollers are at target l_j, all of them remain outside of l_j in the following transition. We can reason that p^* is equal to p^N, because p is the probability that one patroller in state ξ remains in the same state after one transition. Moreover, it can be shown that:

$$(1-q^*) = \sum_{k=1}^{N} \binom{N}{k} \eta \pi_j^k (1-\pi_j)^{N-k} (1-q)^k p^{N-k}$$
$$\eta = \frac{1}{1-(1-\pi_j)^N}$$

To see this, consider that, using the total probability law again, $(1-q^*)$ can be derived as,

$$(1-q^*) = \sum_{s \in S} p(s,\xi) p(s)$$

where we will now consider a binary string of length N denoted s representing whether or not a particular patroller is at target l_j. A value of 1 is assigned to index $i \in [1, ..., N]$ of the string if the i-th patroller is at target l_j and 0 otherwise. S corresponds to the set of all possible strings.

The probability of a given string can be written as:

$$p(s) = \eta \pi_j^b (1-\pi_j)^{N-b}$$

where b is the number of patrollers at target l_j and with the normalization factor η coming from the fact that the string of all 0s must be left out because we are calculating the probability of leaving l_j. In other words, a patroller must already be in l_j to be able to leave it. The probability of going from a *covered* string to an *uncovered* one, $p(s, \xi)$, is equal to

$$(1-q)^b p^{N-b}$$

i.e., the probability that all the patrollers at target l_j leave l_j, and all the agents outside of l_j remain outside of it.

Proof. The set S has a cardinality of $[2^{N-1}-1]$ meaning calculating the probability is exponential. However, we notice that given a string s_1 and string s_2 with the same number of agents at vertex j their probabilities are equal and $p(s_1, \xi) = p(s_2, \xi)$. Consequently, the exponential problem reduces to an n choose k problem wherein we introduce the binomial coefficient. □

5.4 Evaluations and Comparisons

We provide an empirical evaluation of the patrolling strategies obtained with our proposed techniques: heuristic partitions without critical levels (non-weighted, PNW) and with critical levels (weighted, PW), partitions computed with our exact method (MIP, run with a deadline of 30 minutes), and non-partitioned patrollers (NP). Varying the number of vertices in the graph from 30 to 60 (with increments of 10), for each size we generate 10 random instances and we run patrolling missions over the obtained graphs where each patroller visits 10,000 vertices. We present the cases where the number of patrollers is either 5 or 10.

We assumed one attacker observing each target and considered two different types of attackers. The first type, called Maximum-Likelihood (ML) attacker, fits an exponential distribution over the inter-arrival times observed at its target. Then, it generates a prediction for the next inter-arrival time by computing the expectation of that distribution. The second type of attacker, called Nearest-Neighbor attacker (NN), treats the sequence of inter-arrival times as a temporal

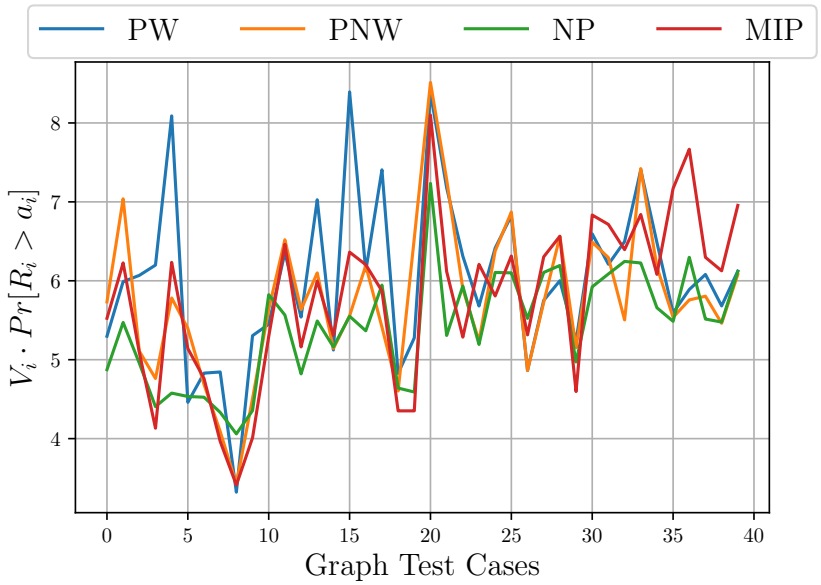


Figure 5.4: 5 patrollers. Value multiplied by the probability of the return time being larger than the attack time.

series and uses a nearest neighbor over the last 10 observed elements from the series to forecast the next one. In both cases, at each iteration where the predicted inter-arrival time is larger than the target's attack time, the attacker decides to attack. As it is evident, the patrollers could perform more frequent visits but should also try to induce an overestimation of the next attack time.

The plots will show the *protection ratio* defined as the number of the attacks intercepted by a patroller divided by the attempted ones and multiplied by the value of the target. We show also the number of attack attempts induced by a patrolling strategy and, for each l_i, the value $v_i \cdot Pr[r_u > a_i]$ which provides an indicator of the protected value not dependent on how the attacker estimates arrival times (called *intrinsic loss* in the following). Each chart features on the y axis the average quantity over all the vertices of the graph while on the x axis we have the single instances from the smallest to the biggest K (number of targets).

Figures 5.5 and 5.6 show the average protection ratios with 5 patrollers against the two types of attackers. The generally decreasing trends reflect the increasing difficulty of protecting a graph where there are more targets but the number of

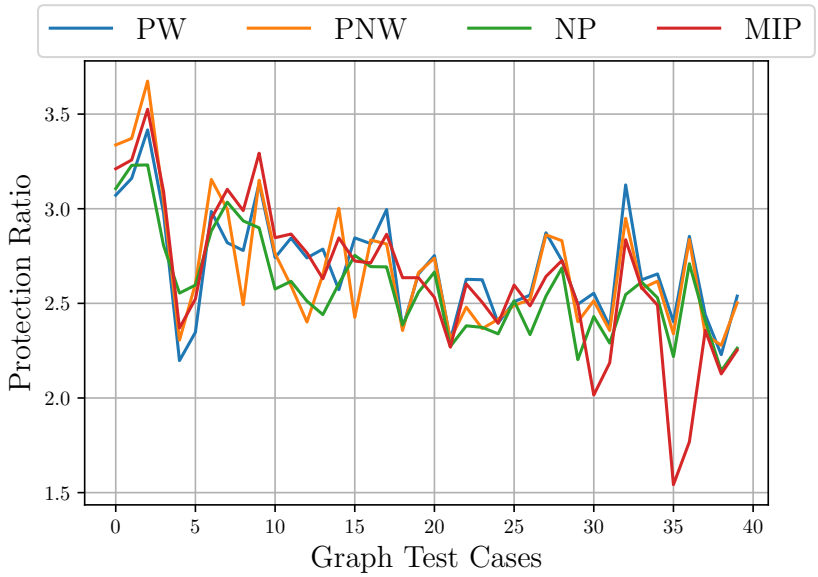


Figure 5.5: Protection ratio for five patrollers against a ML attacker.

patrollers stays the same. The NP approach seems to be slightly dominated by the others suggesting that partitioning could be profitable. The two heuristics kept pace with the MIP method, even outperforming it at times when this last met the deadline without finding the optimal partition and returning, instead, the current best solution found.

A more insightful analysis of these trends can be extracted if one considers not only the protection ratio, but also the number of attempted attacks – something we analyze for the case of 10 patrollers discussed next. A more insightful analysis for the case of 5 patrollers can be observed in Figure 5.7 where we show the number of attempted attacks, irrespective of whether they were caught or not. While the performance in terms of protection ratio did not show particularly sharp gaps, the number of attacks induced by each strategy provides a clear indication that partitioned-based methods induce fewer attack attempts, especially in those instances where computing a balanced partition is easier. This is reasonable since a well-balanced partition (recall the definition of I) favours an even distribution of values and small inter-arrival times. This acts as a deterrent on the attacker that in many occasions opts for not attempting an attack. Thus, if from one side (protection ratio) partitioned strategies seem almost comparable to the non-

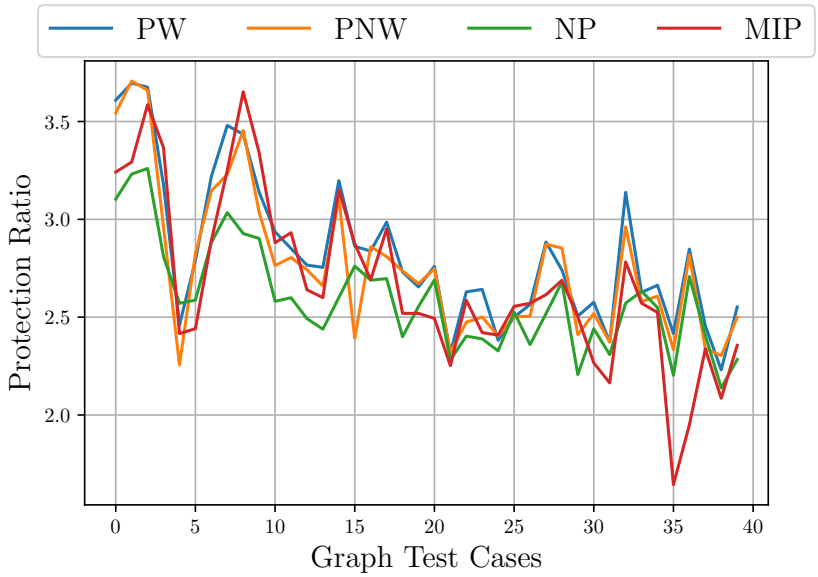


Figure 5.6: Protection ratio for five patrollers against a Nearest Neighbor attacker.

partitioned one in terms of capturing the attacker, they clearly work better in keeping the attacker out.

With 10 patrollers things become more challenging and slightly more evident gaps start to emerge as shown in Figure 5.9 where we show the intrinsic loss for 10 patrollers. The figure clearly shows that the MIP method provides better coverage to vertices with high value. Figures 5.10 and 5.11 show the protection ratio against the two type of attackers. To fully appreciate the meaning of Figure 5.10, where it seems that the MIP method is underperforming, it is useful to consider Figure 5.12 where we show the number of attempted attacks. The figure shows that the MIP strategy induces the attacker to reduce its number of attempts. In general, partitioned-based methods induce fewer attack attempts, especially in those instances where computing a balanced partition is easier. This is reasonable since a well-balanced partition (recall the definition of I) favours an even distribution of values and small inter-arrival times. This acts as a deterrent on the attacker that in many occasions opts for not attempting an attack. Thus, if based on the protection ratio partitioned strategies seem almost comparable to the non-partitioned one in terms of capturing the attacker, they clearly work better

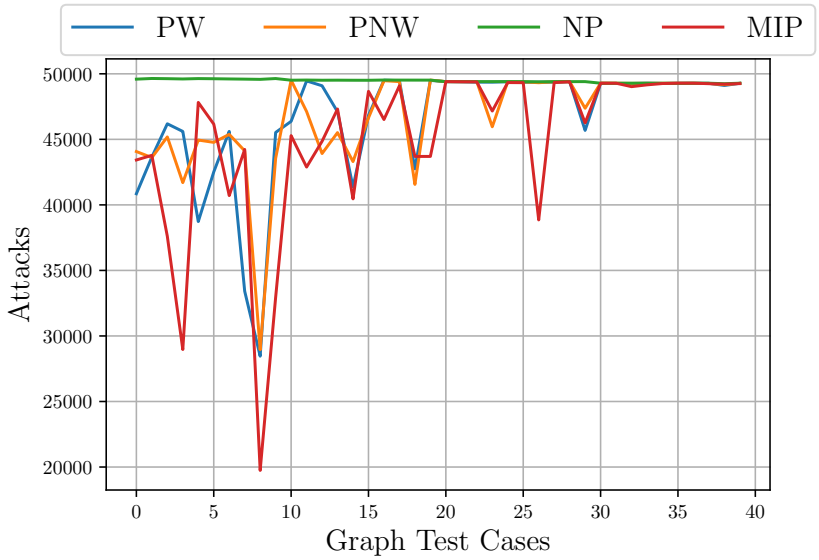


Figure 5.7: Number of attacks for 5 patrollers against a maximum-likelihood attaker.

in keeping the attacker out. Taking all into considerations, the MIP strategy is the best one. While it is more demanding, our experiments show that using the approximate solution produced upon stopping the method after 30 minutes still provides good results. PW and PNW can be considered as fairly effective heuristics since their performance was not remarkably worse than MIP's. Their lower computational burden can be leveraged when scaling to very large patrolling settings.

5.5 Conclusions

In this chapter, we studied three approaches for multirobot patrolling against an attacker that, through repeated observations, tries to predict when it is the best time to attack. Our method based on a MIP formulation turns out to be the best one, even when it is stopped before the optimal solution is found. For the two types of attacker we considered, this strategy discourages the attacker from attempting to compromise the assets being protected. In future works, we shall expand the analysis presented in this section to consider refined models of the attacker behavior

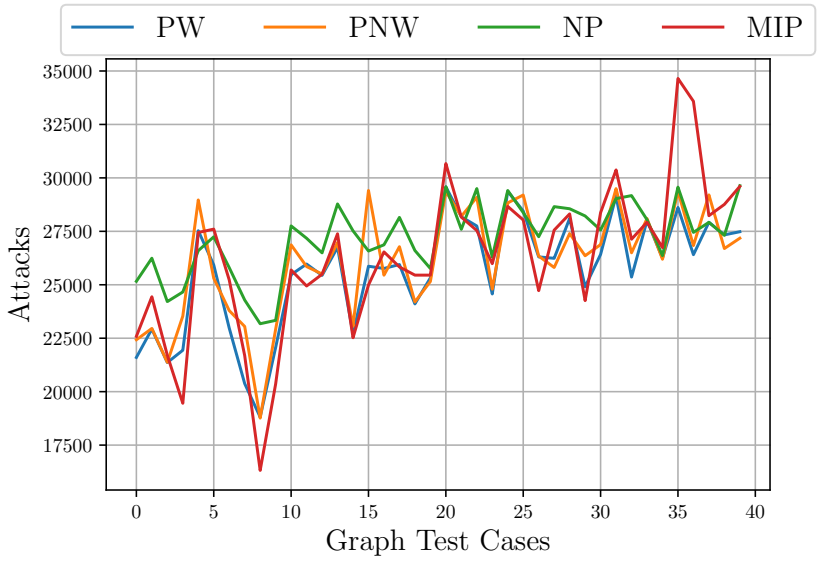


Figure 5.8: 5 patrollers, nearest neighbor attacker.

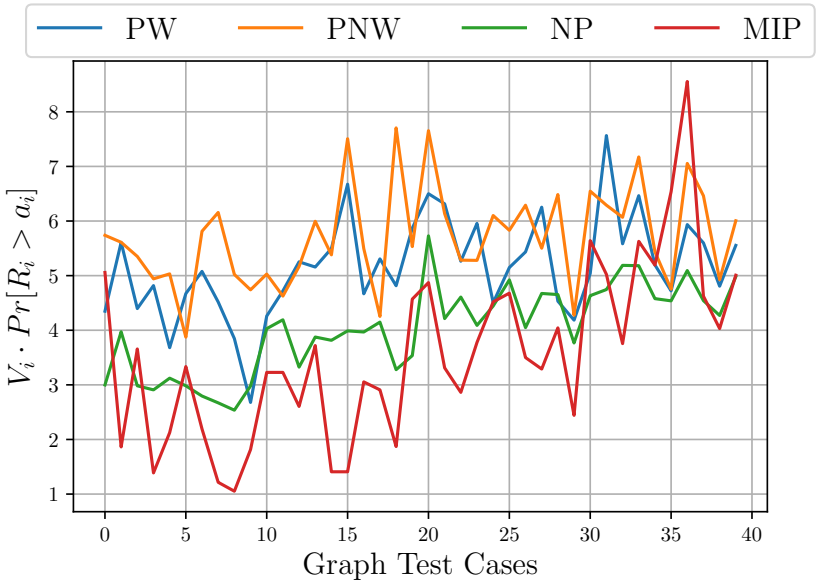


Figure 5.9: Intrinsic loss for the case of 10 patrollers.

including, for example, coordination between multiple attackers.

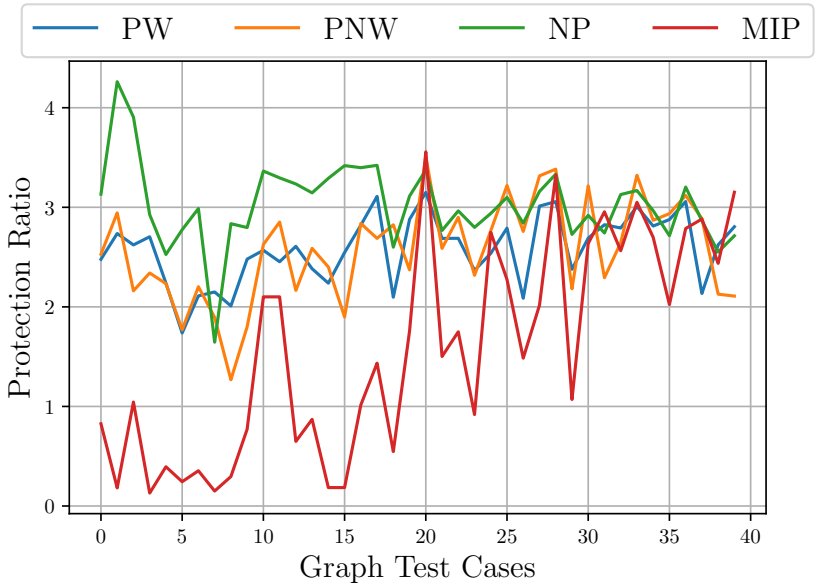


Figure 5.10: Protection ratio for ten patrollers against a ML attacker.

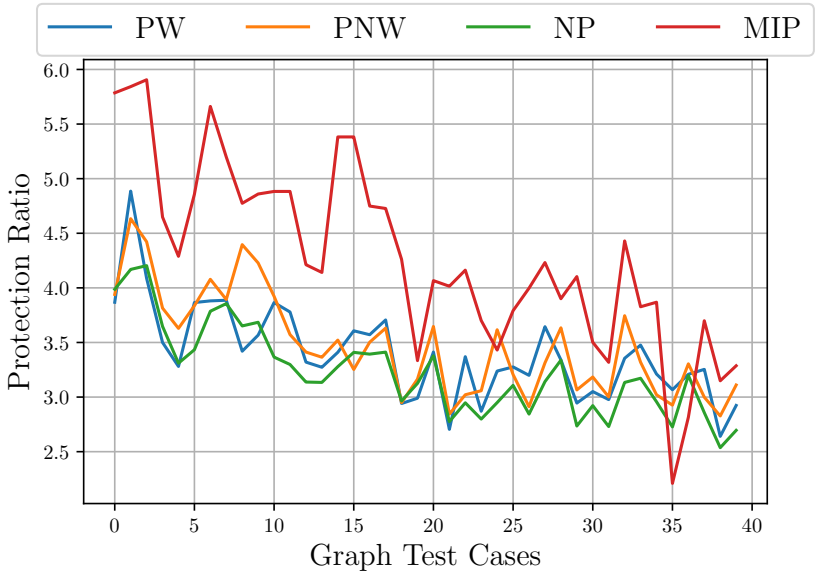


Figure 5.11: Protection ratio for ten patrollers against a Nearest Neighbor attacker.

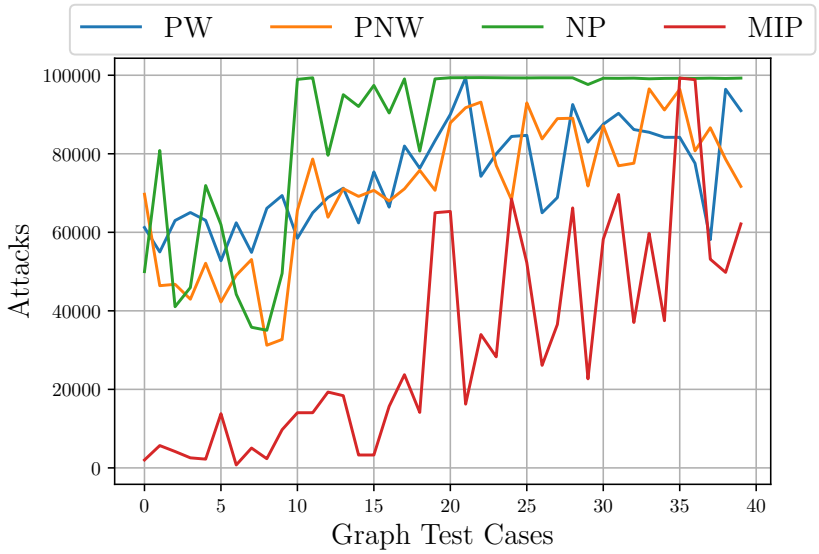


Figure 5.12: Number of attacks for 10 patrollers against a ML attacker.

Chapter 6

Methods for Optimizing a Team of Agents

Chapter 5 compared and contrasted two different strategies for multi-agent coordination: partitioned and non-partitioned patrollers. In the end, the mixed integer linear program gave some of the better results for how to divide up the work for the agents. Here in chapter 6 we consider a modification to the patrolling scenario and propose a new paradigm for designing solutions. In the following, we do not consider an explicit opponent model and opt instead for optimizing the worst-case idleness. Furthermore, given there are limited patrolling resources and some target locations are more valuable than others, we introduce a shared and un-shared workload for the team of patrolelrs. The work shown here was originally presetned in [10].

6.1 Patrolling Optimization Setting and Definitions

We consider the classical graph patrolling setting refining a model we formerly considered in chapter 5. The environment is modeled by a graph $G = (V, E)$, where vertices $V = \{1, 2, \ldots, n\}$ represent locations and edges $(i, j) \in E$ represent their connections. A value c_{ij} represents the traveling cost (time or distance) to

move from i to j. We assume that the graph is complete and that c_{ij} is the shortest cost. (Such a representation can always be computed from an arbitrary connected graph). Each vertex $i \in V$ is assigned an importance value $v_i > 0$, indicating the level of criticality for its protection. A set $R = \{1, 2, \ldots, m\}$ of m *patrollers* must protect the environment by moving from vertex to vertex in the graph. When a patroller visits a vertex, the vertex is protected, i.e., it cannot be compromised by an attacker, or, if the vertex is under attack, the attack is neutralized. Our optimization criterion uses the *idleness* of a vertex i, indicated as I_i and defined as the time between two successive visits to i by any of the patrollers. A common function to optimize is the weighted idleness:

$$\min \max_{i \in V} v_i I_i \tag{6.1}$$

This favors patrolling strategies where valuable vertices are visited more frequently, thus resulting in lower idleness. The m patrollers can be organized according to the following. (i) *Coordinated patrolling*: each patroller can visit any of the vertices in G. (ii) *Disjoint partitions*: the set of vertices V is partitioned into m non-overlapping subsets, and each patroller is assigned a subset of vertices (to prevent degenerate cases, we assume each subset has at least two vertices). (iii) *Overlapping partitions*: V is subdivided into m subsets that may share some vertices, each subset is then assigned to one of the patrollers.

6.2 Overlapping partitions

We propose a new way to split the workload between m patrolling agents by introducing the concept of *core* and *periphery* on the vertex set V. Figure 6.2 shows how this could work for a pair of defenders protecting a graph. The core V_0 is a subset of V patrolled by all m robots. The periphery $P = V \setminus V_0$ is the set of remaining vertices that is instead split into up to m non-overlapping subsets. Let these subsets be indicated as $V_1, V_2, \ldots, V_m$. The idea is that each robot k first patrols the core V_0 by traversing a Hamiltonian path between some start and end vertices, then patrols its assigned subset V_k, also traversing it with a Hamiltonian

path, and then goes back to the core and repeats. This strategy is sketched in Figure 6.1 for a case where $m = 2$.

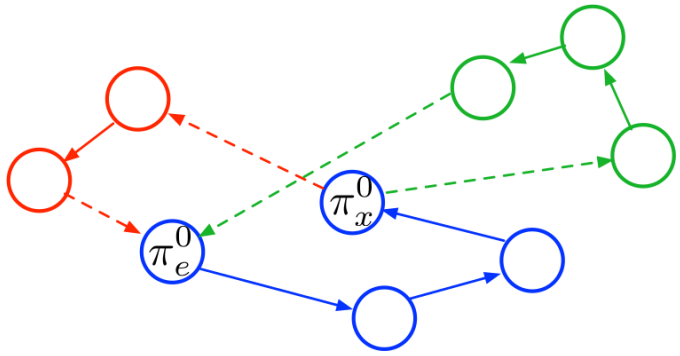


Figure 6.1: Overlapping partition for $m = 2$. Both robots patrol the core's blue vertices following the same path between π_e^0 and π_x^0. Subsequently, robot 1 patrols the red vertices, while robot 2 patrols the green vertices.

We make these assumptions about robots' coordination.

- All robots traverse V_0 following the same Hamiltonian path π^0 from π_e^0 (entry) and π_x^0 (exit).
- When exiting π_x^0, if $V_k \neq \emptyset$ robot k traverses the assigned subset V_k by following an Hamiltonian path π^k starting at π_e^k and ending at π_x^k. Then it travels back to π_e^0 and repeats. If instead $V_k = \emptyset$, robot k moves from π_x^0 to π_e^0 and resumes patrolling the core following π^0.
- Robots (i) adapt their speed so that each robot spends the same time to complete its tour (the time of the longest tour followed by any robot) (ii) uniformly distancing in time their arrivals at the core's entrance.

For a given core V_0 with entry vertex π_e^0 and exit vertex π_x^0, let t_c be the time it takes to follow the Hamiltonian path π^0. Similarly, for each of the subsets V_k, let t_k be the time to complete π^k also including the movement from π_x^0 to π_e^k (V_k's entry) as well from π_x^k (V_k's exit) to π_e^0 – see Fig. 6.1. Given this construction, the idleness of each vertex in V_k for $k > 0$ will be $t_c + t_k$ because the robot in charge of that subset must patrol, in sequence, V_0 and V_k. However, vertices in V_0 will

be subject to a lower idleness thanks to robots' coordination. If robots coordinate their starts from π_e^0, then the maximum idleness experienced by the vertices in V_0 will be scaled by a factor of m. More formally, when we impose the existence of a shared core V_0, the objective function defined in Eq. (6.1) can be rewritten as follows:

$$\min \max_{1 \leq k \leq m} \left\{ A^c \frac{t_c + t_k}{m}, A^p (t_c + t_k) \right\} \tag{6.2}$$

where

$$A^c = \max_{i \in V_0} \{v_i\}, A^p = \max_{i \in V \setminus V_0} \{v_i\}.$$

Note that, to be well-defined, this formulation requires that $|V_0| \geq 2$ to allow for at least two vertices in V_0 as π_e^0 and π_x^0 must be distinct. However, if useful towards the solution of the above minimization problem, we do allow $V_k = \emptyset$ for one or more periphery sets ($k > 0$). When that is the case, the corresponding agent just patrols the core. The following problem formulation formalizes the problem we described.

> **Overlapping Partitions Problem (OPP)** Given a weighted graph $G = (V, E)$ with n vertices, edge costs $c : E \to \mathbb{R}^+$, and vertex values $v : V \to \mathbb{R}^+$. Let m be a given number of robots. Determine a core subset $V_0 \subset V$ with at least two vertices and a partition of $V \setminus V_0$ into at most m elements that solve the minimization problem defined by Eq. (6.2).

The pivotal question therefore is how to determine the core V_0, the periphery $V_1, \ldots, V_m$, and the corresponding Hamiltonian paths, solving the minimization problem formulated in Eq. (6.2), also in light of the following theoretical result.

Theorem 1. *The OPP problem is NP-hard.*

Proof: for the special case $m = 1$ the OPP problem is equivalent to the traveling salesman problem (TSP) over the entire set of vertices V.

6.3 Exact formulation

We here provide an exact mathematical formulation for OPP. This formulation can be useful to solve small instances (i.e., graphs with few vertices), and to better

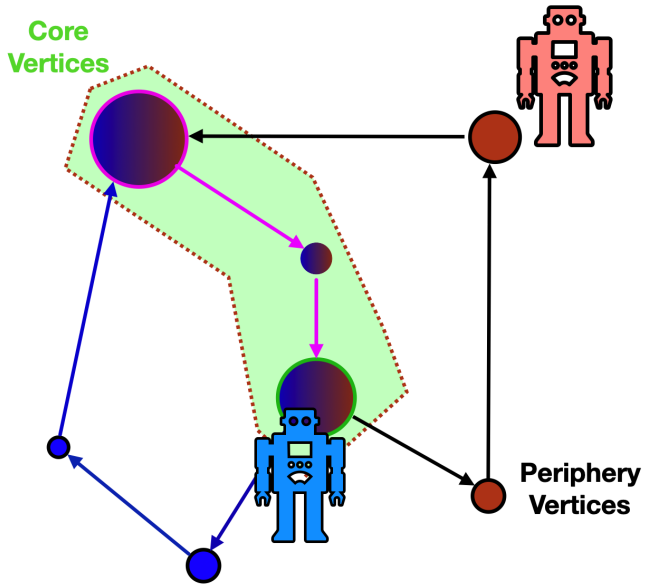


Figure 6.2: Visualization of the shared and unshared workload for the team of patrollers. The target locations that belong to the *core* set are considered high value and will visited by all of the defenders. Meanwhile, the target locations in the *periphery* set will only be patrolled by a single agent. This scheme is empirically shown to out perform a the traditional partitioning approach. Both the red and blue agent share the vertices in the core, however, only the blue agent visits the target locations on the bottom left of the graph (peripheral vertices).

understand the structure of the problem. Note that the formulation we provide allows for the case where all $V_k = \emptyset$ for $k > 0$, but imposes that $V_0 \neq \emptyset$. As will be shown in Section 6.5, in some peculiar cases this could be a disadvantage, i.e., the disjoint partitions approach could provide better solutions to the problem defined in Eq. 6.3.

Let $V^+ = V \cup \{0\}$ and $R^+ = R \cup \{0\}$ and let us consider a directed version of the edge set E, where we replace each edge with the two corresponding symmetric arcs. Notably, as we will show in the end, the formulation is nonlinear. Both sets include an additional 0 element, which will be useful for the following construction. In the set V^+, the element 0 represents an extra vertex that does not correspond to any real location but that allows us to express the resolution of the problem as finding a set of $m+1$ tours starting and ending at it. In the set R^+, the 0 element represents an extra robot which we will associate with the shared patrolling task of the core. We introduce the following binary decision variables for each $i, j \in V^+$ and $k \in R^+$.

$$x_{ij}^k = \begin{cases} 1, & \text{if robot } k \text{ will travel on edge } (i,j) \\ 0, & \text{otherwise} \end{cases}$$

An assignment of the x_{ij}^k variables defines a partition of the environment. Each element $V^k \subseteq V$ ($k \in R^+$) of the partition can be recovered as follows: $V^k = \left\{ j \in V \mid \sum_{i=1}^{n} x_{ij}^k = 1 \right\}$. Another set of similar binary variables y_{ij}^k is introduced with the same definition but a different meaning (see usage below).

Notice that the above set definition assumes that the assignment to the x_{ij}^k variables is consistent with the set of constraints that will be introduced in the formulation, which will enforce that each vertex will be visited by exactly one robot (including robot 0). The definition has the following rationale: if the robot k travels vertex (i,j) for any i then j is considered part of that robot's subset of vertices. In our problem formulation, V^0 will represent the vertices assigned to the core, i.e., the portion that is shared among the m robots. Conversely, V^k for $1 \leq k \leq m$ represents the vertices that are patrolled exclusively by robot k.

The variable t_i represents the time at which vertex i is visited within the

sequence of vertices in the partition to which i belongs. In the solution, we do not require that variables t_i will be assigned values that are consistent with traveling costs: we only require that if a vertex i is visited later than a vertex j then $t_i \geq t_j$.

$$\min U \tag{6.3}$$

s.t.

$$\sum_{j=1}^{n} x_{0j}^{k} = 1, \quad \forall k \in R^{+} \tag{6.4}$$

$$\sum_{i=1}^{n} x_{i0}^{k} = 1, \quad \forall k \in R^{+} \tag{6.5}$$

$$\sum_{k=0}^{m} \sum_{i=0}^{n} x_{ij}^{k} = 1, \quad \forall j \neq i \in V \tag{6.6}$$

$$\sum_{k=0}^{m} \sum_{j=0}^{n} x_{ij}^{k} = 1, \quad \forall i \neq j \in V \tag{6.7}$$

$$\sum_{i=0}^{n} x_{ij}^{k} = \sum_{i=0}^{n} x_{ji}^{k}, \quad \forall j \in V, k \in R^{+} \tag{6.8}$$

$$u_i - u_j + (n - m + 1) \sum_{k=0}^{m} x_{ij}^{k} \leq n - m, \quad \forall i \neq j \in V \tag{6.9}$$

$$x_{i0}^{0} + x_{0j}^{k} - 1 \leq y_{ij}^{k}, \quad \forall i \neq j \in V, k \in R \tag{6.10}$$

$$x_{0j}^{0} + x_{i0}^{k} - 1 \leq y_{ij}^{k}, \quad \forall i \neq j \in V, k \in R \tag{6.11}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \left(x_{ij}^{0} + x_{ij}^{k} + y_{ij}^{k} \right) \leq I^{k}, \quad \forall k \in R \tag{6.12}$$

$$\frac{1}{m} \sum_{i=0}^{n} x_{ij}^{0} v_j \leq A^{c}, \quad \forall j \in V \tag{6.13}$$

$$\sum_{i=0}^{n} x_{ij}^{k} v_j \leq A^{p}, \quad \forall j \in V, k \in R \tag{6.14}$$

$$A^{w} I^{k} \leq U, \quad w \in \{c, p\}, k \in R \tag{6.15}$$

Table 6.1: An exact formulation for the OPP problem.

The formulation shown in Table 6.1 is inspired by the ILP formulation for the maxmin Multiple TSP (see [56]) from which we take a basic set of constraints and

we extend it to enforce the core/periphery structure we propose in this work.

Constraints (6.4) and (6.5) require that each robot leaves and returns to the depot along a single arc. Constraints (6.6) and (6.7) require that exactly one robot shall arrive and leave each vertex (except for vertex $i = 0$, the dummy depot). Constraints (6.8) ensure that the robot that arrives is the same one that leaves. Constraints (6.9) are the Miller-Tucker-Zemlin (MTZ) subtour-elimination constraints. They eliminate solutions where one robot covers its vertices through two or more disjoint tours on the graph. Here it is assumed that $n \geq m$ and that, in the solution, each robot will cover at least one vertex.

The above basic constraints will enforce solutions whose structure is the same as that required for classical multi-TSP problems: $m + 1$ tours, each starting and ending at the depot $i = 0$. The tours are such that each vertex $i > 0$ is covered once by just one robot. The tour for robot k is composed in this way: leaving the depot, traveling an *inner path* π^k, and returning to the depot. We need to combine these $m + 1$ tours in order to obtain m tours, one for each real robot $k > 0$, each defined as start at the first vertex of π^0, follow π^0, travel to the first vertex of π^k, follow π^k, travel to the first vertex of π^0, repeat. We can show that the optimal solution to our problem can be expressed in the multi-TSP form described above provided that the required objective function is defined. Hence we embed our required structure in the costs minimized by the objective function.

Constraints (6.10) and (6.11) select, through the y variables, those arcs that will connect, for each real robot, the inner path π^0 with the inner path for just that robot π^k. Constraints (6.12) define I^k as the upper bound on the tour cost (and hence, the idleness) for a real robot $k > 0$. Constraints (6.13) and (6.14) similarly define, an upper bound for the maximum vertex value in the core (A^c) and outside of it (A^p). Notice that these upper bounds will be "pushed" to be tight since the problem is a minimization one. Finally, Constraints (6.15) provide an upper bound to the objective function we seek to minimize. This cost is the maximum weighted idleness that accounts for a discount of $1/m$ for the maximum value among the vertices in the core (those visited by π^0). Notice how this final constraint, crucial to the whole problem, is also the one that introduces a non-linearity.

As mentioned earlier, the exact formulation illustrated in this section could be used to solve small problem instances, i.e., instances with few vertices and edges, but obviously do not scale to larger ones. For this reason, in the next section, we propose various heuristics that scale with the problem's size.

6.4 Heuristics for OPP

In light of the computational complexity of OPP, in this section we introduce various heuristic methods for finding solutions to large problem instances where exact methods cannot be applied. As mentioned in the previous section our problem is related to the min-max multiple traveling salesman problem (mTSP) formulation [56] and this insight informs our solving strategy. A valid solution to OPP can be determined as follows:

- Select some vertices to form the core.
- Compute a TSP tour on the core with arbitrary starting and ending vertices (this can be achieved for example by adding a *dummy vertex* to the graph that is connected to all other vertices with edges of zero cost) and restricting the TSP tour to start and end at the dummy vertex; these arbitrary starting and ending vertices become π_e^0 and π_x^0;
- Solve the min-max mTSP problem on the periphery where all m tours must start at π_x^0 and end at π_e^0. Furthermore, the solution to the mTSP will partition the periphery vertices into disjoint sets for the robots.
- Finally by combining the Hamiltonian path on the core with each of the m Hamiltonian paths on the periphery we form a valid path for each robot and thus a valid solution to the problem.

Thus given a core set of vertices, a TSP solver, and a min-max mTSP solver[1] one can form a solution by performing the aforementioned steps. However, the question of how to select the vertices that belong in the core so as to effectively

[1]Note that we are not requiring an exact solver for these two NP-hard problems.

minimize Eq. 6.2 remains open. In the following, we propose different methods aiming at obtaining low values for Eq. (6.2) that differ in the strategy to build the core.

6.4.1 K-means core

A first heuristic is based on the intuition that to minimize the objective function one should minimize the distance traveled by the robots, i.e., reducing the sum $t_c + t_k$. These two terms are conflicting since reducing the total distance traveled on the core means removing a vertex from there and adding it to the periphery hence increasing the distance traveled by some robot on its independent workload. Because we would like every robot to have some minimum workload we propose to use k-means clustering based on the Euclidean distance between vertices with $m + 1$ clusters to identify the subsets $V_0, V_1, \ldots, V_m$. After running the algorithm, we assign the largest cluster to the core.

6.4.2 Weighted K-means core

Weighted k-means clustering [48] works similarly to the non-weighted version but exploits the fact that each of the vertices in the graph has an associated value. Since in Eq. (6.2) the values of vertices multiply the travel times, in the weighted k-means heuristic the values of the vertices are used to scale the weight of the clusters. Then just as we did with non-weighted k-means, we compute $m + 1$ clusters and denote the largest cluster as the core set of vertices.

6.4.3 Balanced Weights Heuristic (BWH)

By examining Eq. (6.2) we note some principles governing the nature of the objective function. The variables t_c and t_k are found in every term and furthermore in general minimizing one of the variables means increasing the other since a smaller t_c can only be achieved by removing a vertex from the core and thus adding said vertex into the periphery. The same is true for decreasing the largest tour on the periphery - it can only be achieved, in general, by removing a vertex from the

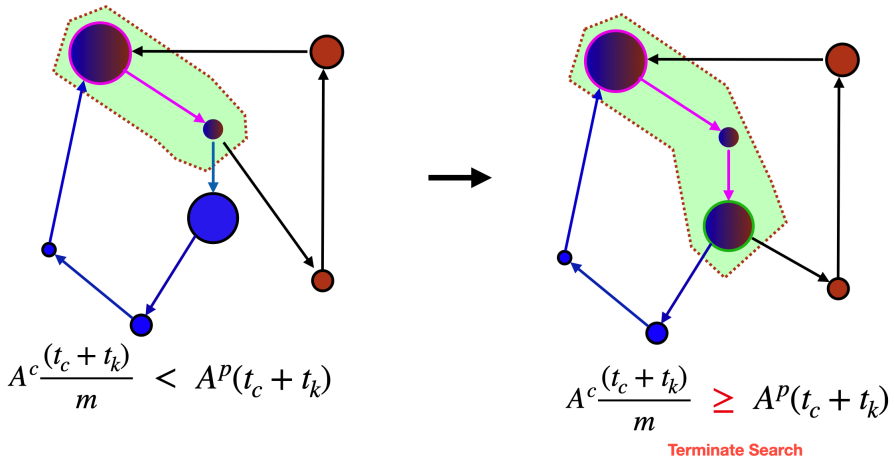


Figure 6.3: The balanced weight heuristic can be said to work in the following way: the most valuable target locations are repeatedly added to the core set until the termination condition is met. This procedure is visualized above.

periphery and adding it to the core set or adding it to another partition whose t_k value will increase. This suggests that the optimal is found when the terms $\{A^0\frac{t_c+t_0}{m}, A^0\frac{(t_c+t_1)}{m}, ..., A^1(t_c+t_k), ..., A^k(t_c+t_k)\}$ are roughly similar in value. Note here that the maximum of the first k terms is simply $A^0\frac{t_c+max_k(t_k)}{m}$. Given that A^0 is the largest valued vertex in the core and each subsequent A^k is obtained from the largest valued vertex for each robot's independent workload, a simple heuristic for partitioning the set of vertices into core and periphery is obtained by adding a vertex into the core when its corresponding value satisfies the inequality:

$$v_j > \frac{max_{i\in V}(v_i)}{m}.$$

In this way, we can keep high-valued vertices in the core and, at the same time, push out into the periphery vertices with values similar to or less than the first coefficient in the objective function, $\frac{A^0}{m}$. This heuristic aiming at balancing the terms in the objective function is dubbed *balanced weights heuristic (BWH)*. Futhermore, the balanced weight heuristic can be said to work in the following way: the most valuable target locations are repeatedly added to the core set until the termination condition is met. Under this line of thinking, figure 6.3 visualizes how the core set evolves as the heuristic adds valuable vertices.

6.4.4 Local Search Heuristic (LSH)

Starting from the balanced weights heuristic, we can develop an improved version called *local search heuristic (LSH)*. Since the core is shared between k robots it follows that generally, we would like cores that include more vertices with respect to subsets in the periphery. Starting from the core proposed by BWH, the local search heuristic iteratively tries to improve the current selection of the core by evaluating random additions to it. In this way, our local search method tries to build solutions that are at least as good as those provided by the BWH approach. At each iteration for each vertex i in the periphery set, with probability p, it is removed from the periphery set and introduced into the core set ($p = 0.6$ in our experiements). We can then evaluate this new candidate core set by computing its objective value from Eq. 6.2. If the new core is better than the previous one it is saved otherwise, it is passed over. An outline of the local search heuristic is given in Algorithm 3.

Algorithm 3: Local Heuristic Search

```
Data: B search budget p vertex add probability
C ← Balanced Weight Heuristic;
o ← objective value for C (from equation 6.2);
for i ← 0 to B do
    D ← C;
    P ← V − C;
    for n in P do
        with probability p, D = D ∪ n;
    solve TSP and mTSP sub problems;
    s ← objective value for D (from equation 6.2);
    if s < o then
        o ← s    C ← D
return C, o
```

6.5 Evaluation

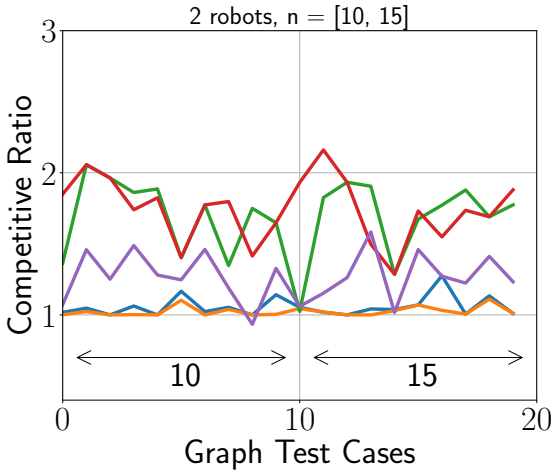
In this section, we perform two types of evaluations. First, we assess the relative merit of the heuristics we introduced in section 6.4. Second, we evaluate if the novel formulation we proposed in this chapter is advantageous when compared with the coordinated patrolling and disjoint partition methods formerly proposed in the literature. To compare with the disjoint partition method, we use the k-Max Cut algorithm we formerly proposed in [9]. This approach partitions the graph into k subsets and subsequently assigns each robot a sub-graph to patrol. For each sub-graph, a TSP tour is calculated and serves as the robot's path through the area.

For a more complete comparison we also evaluated these heuristics against our scheme for *disjoint partitions* from The partition is computed via a max-cut solver that works on the same graph but with its edge weights transformed according to their distance to the maximum edge weight. Partitioning the graph into k subsets and assigning each robot an independent tour has many advantages, but as we will show empirically for our objective function this does not always provide the best solution. More details about how the partitioning is performed can be found in [9].

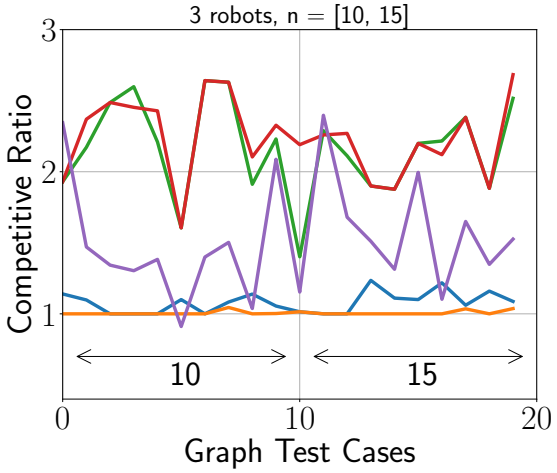
We performed evaluations on graph sizes ranging from 10 vertices to 60 vertices, with 10 instances for each graph size. Vertex locations are generated by randomly sampling points in a 50×50 square and edge weights are given by the Euclidean distance between the vertex coordinates. Finally, vertices values are sampled from the range 1 to 100 using a uniform distribution. To solve the TSP and mTSP problems we used the routing module provided by the Google OR-Tools library [36], while for k-means we use scikit-learn [63].

To get an idea of how far from the optimal solution our methods are, we first performed a set of experiments on graphs of small size comparing the solutions provided by the different heuristics and the exact solution.

Figures 6.4a and 6.4b show the *competitive ratio* of the various heuristics for graph instances of size 10 and 15. The competitive ratio is defined as the ratio between the objective value returned by the heuristic method and the optimal

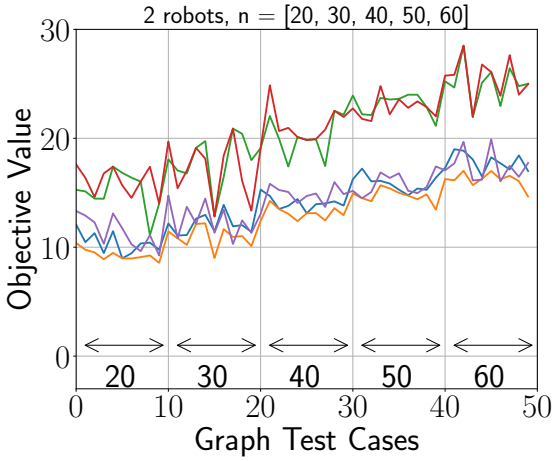


(a)

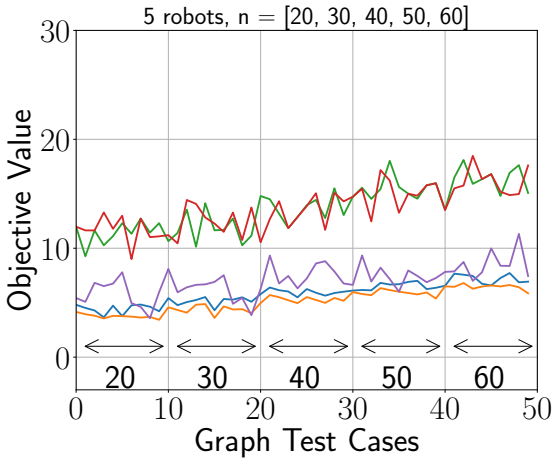


(b)

Figure 6.4: Comparison between the different heuristics for different graph sizes and number of robots. In both charts, the following color coding is used: red for k-means; green for weighted k-means; purple for k-max cut; blue for the balanced weights heuristic; and orange for the local search heuristic



(a)



(b)

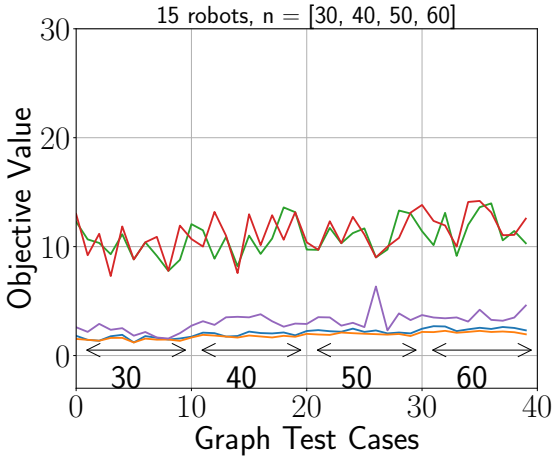
Figure 6.5: Comparison between the different heuristics for different graph sizes and number of robots. Instead of competitive ratio we analyze the minimum objective value reached by each heuristic. In both figures, the following color coding is used: red for k-means; green for weighted k-means; purple for k-max cut; blue for the balanced weights heuristic; and orange for the local search heuristic.

value (with 1 therefore being the best one can aim for). Both the charts for 2 and 3 robots show similar trends, namely that BWH and LSH almost always perform better than the other methods, and have a competitive ratio close to 1.

Furthermore, local search can improve BWH as evidenced in Figure 6.4b. Important to note in both Figures 6.4a and 6.4b is that the k-Max Cut sometimes does even better and has a competitive ratio of less than 1. This apparent contradiction can be explained by the fact that some graph instances naturally cluster into distinct sub-graphs. This arrangement is advantageous to the k-Max Cut method since it will keep the independent robot tours small, while it is also detrimental to the heuristics and the exact method since selecting a core set of vertices will incur a large travel cost when moving between the two natural sub-graph clusters. The next set of experiments compares the objective values of the returned solutions for each of the various heuristics for larger graphs. Note that as the number of vertices exceeds 15 the exact solution becomes too costly to compute and we therefore compare the objective values between the different solutions rather than the competitive ratio. Figure 6.5a, 6.5b, 6.6a, and 6.6b show the values found for 2, 5, 10, and 15 robots and graph sizes varying from 20 to 60 vertices.

As can be seen, the two heuristics proposed in this section almost invariably are the most effective, and in particular they outperform the method based on disjoint partitions. The experiments described thus far show that the method we propose outperforms the disjoint partitions approach. However, one could wonder if it also outperforms the coordinated method with all robots patrolling all vertices. An analysis of the results produced shows that the answer is affirmative, especially in the case of constrained resources (e.g., a small number of robots and a large number of vertices). Figure 6.7 shows two examples, with the left graph showing the solution produced by the optimal method and the right one produced by LSH. In the figure, the size of the vertices is proportional to the v_i values and it outlines how important vertices are included in the core (plotted in blue). In both instances, better patrolling strategies are obtained by assigning each robot to a subset of vertices in the periphery (paths plotted in different colors) rather than including everything in the core as in the coordinated method. These two samples

(a)



(b)

Figure 6.6: Comparison between the different heuristics for different graph sizes and number of robots. In all figures, the following color coding is used: red for k-means; green for weighted k-means; purple for k-max cut; blue for the balanced weights heuristic; and orange for the local search heuristic.

are representative of the entire dataset.

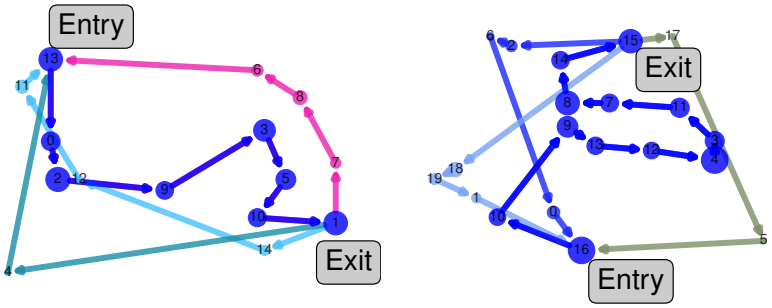


Figure 6.7: Left: optimal solution for 3 robots and 15 vertices. Right: LSH solution for 3 robots and 20 vertices.

6.6 Conclusions

In this chapter, we studied three approaches for multirobot patrolling against an attacker that, through repeated observations, tries to predict when it is the best time to attack. Our method based on a MIP formulation turns out to be the best one, even when it is stopped before the optimal solution is found. For the two types of attacker we considered, this strategy discourages the attacker from attempting to compromise the assets being protected. In future works, we shall expand the analysis presented in this section to consider refined models of the attacker behavior including, for example, coordination between multiple attackers.

Chapter 7

Final Thoughts

7.1 Conclusions

Throughout this dissertation, approaches for computing patrolling strategies for single agents and teams of agents are discussed. Chapter 1 provides the motivation for the patrolling games which involves the use of robots to automate the task of repeatedly visiting important locations of interest. Use cases for these systems include urban search and rescue, airport security [93], and conservation efforts [30]. With effective deterrence in mind these studies provide insight into the challenges associated with the deployment and design of robotic patrollers. There are different formulations for the patrolling problem: single agent, teams of patrollers, adversarial patrolling, machine learning applications etc. As well as different measures of performance such as worst case idleness, or protection ratio. Chapter 2 gives a history and overview of how these challenges have been addressed literature.

Chapter 3 introduced our patrolling security game's formulation and described the challenges associated with computing a patrol schedule for a single agent. By exploiting the structure of an ergodic Markov chains we present four different methods that outperform the previous approach in terms of the protection ratio. The approach computes a new Markov chain every so often and as such better obscures the correlations between visits as evidenced by the KNN and Maximum Likelihood attackers poor performance against it. This is all provided at no cost to

the protection provided by the patrol schedules since all of the computed Markov chains have the same stationary distribution and thus the same optimal protection program.

Chapter 4, on the other hand, delves into a different paradigm for considering the patrolling security game. By modelling the patrolling task as a sequential decision making problem, i.e. Markov decision process, we introduce adaptability and generalizability to the defender's strategy. We present, to the best of our knowledge, the first RL patroller to succeed and generalize against multiple attackers. Key to the performance is training the agent via domain randomization techniques; during training the agent playing games against multiple attackers. Best results are achieved when the agent's opponent is switched every episode. This enemy randomization pushes the agent's policy to achieve the best worst case performance against the attackers and at the same time the best average performance against the attackers.

Chapter 5 switches the composition of the defender's resources and deals with the problem of strategizing for a teams of patrollers. The addition of a team of defenders presents multiple new challenges to the issue including team communication, resource management, and energy management. In the chapter we extend the Markov chain model introduced in chapter 3 to work with a team of agents. Because multiple Markov chains can admit the same stationary distribution, we compute a single different Markov chain for each patroller to use. Randomness in the probability chain, furthermore, helps diminish any redundancy in the schedules. Redundancy occurs when two defending agents arrive at the same target location at the same time because we assume that the agents are equipped with sensors that can identify any intruder with fully certainty if seen. Next, we allow the patrollers to work over the entire graph and dub the management scheme the *non-partitioned* team of patrollers. On the other hand, we also present an alternative method for resource management dubbed *partitioned patrollers*. We introduce a measure of workload for the agents, so that we may quantify the individual effort any agent must spend to patrol her section of the graph. Finally, multi-level graph partitioning algorithms provide quick and efficient prcedures for dividing the workload

evenly amongst the defenders. Finally, we present empirical results comparing and contrasting the two approaches accordingly to the formerly introduced protection ratio. The Mixed Integer Linear Program partitioning comes out as one of the best methods for managing the patrolling resources.

Lastly, chapter 6 considers a different plan of action altogether. Seeking to study a question of *analysis* we consider: what is the best strategy a limited team of defenders can employ to reduce idleness over a vast area. Consequently, we propose a novel method for distributing the workload between the patrolling agents. While on one side one can employ partitioning schemes to divide the workload between the agents, and on the other side one could overlap completely the designated areas of the agent (as discussed in chapter 5) in chapter 6 we present a middle ground between the two sides. Thus, partial overlaps are considered wherein some of environment is under the protection of all the agent, meanwhile the other parts of the environment are only under the guardianship of a single agent. Then question then becomes: which parts of the environment should be overlapped (guarded by all resources) and which only need the guardianship of a single agent? Our formulation of the Overlapping Partitions Problem (OPP) and the balanced weight heuristic (BWH) are our answers to the aforementioned question. Finally, we empirically show on a synthetic dataset that these methods of partial overlap work, in most cases, work better than divide and conquer approaches.

7.2 Possible Future Research Directions

There is still more work to be done regarding planning algorithms for patrolling agents and teams of agents looking to deter malicious actors. Future work should focus on challenging the assumptions made in this thesis to make patrolling more robust, and engineering new algorithmic approaches that achieve better results than obtained thus far.

7.2.1 Machine Learning for Patrols

Graph Neural Networks (GNNs) and A Curriculum Approach

Chapter 4 proposed a different paradigm for considering the patrolling security game, the Markov Decision Process (MDP), and as a consequence also introduced adaptability and generalizability to the defender's strategy. Furthermore, during the RL training the agent plays games against multiple attackers. Best results are achieved when the agent's opponent is switched every episode. This enemy randomization pushes the agent's policy to achieve the best worst case performance against the attackers and at the same time the best average performance against the attackers. However, a major limitation of this work is the need for re-training the neural network when either a new attacker behavior or a new graph size is introduced. One could tackle the latter by employing graph neural network architectures [99] to the PPO agent. The GNN architecture would allow the agent to train on graphs of varied size and could thus result in an agent that could patrol any graph against multiple attackers. Moreover, curriculum learning methods [22] could be applied in order to achieve smoother and faster training results. These curriculum procedures also could reduce the amount of training needed when a new attacker model is introduced to the patroller.

Multi-agent Reinforcement Learning (MARL)

Chapter 4 focuses on designing a reinforcement learning solution for a single agent tasked with guarding an area. A natural extension would be a framework for a team of defenders. Consequently, one would need to consider the application of multi-agent reinforcement learning algorithms [27]. A common scheme is to employ the centralized training and decentralized execute approach (CTDE). For example, since the defenders are cooperating one could centralize their training have the rewards for any agent be the average of the reward signals coming through. The decentralization could then work by giving each agent a copy of the network as to patroller their own section of the graph. Issues of coordination and communication will still need to be address and some of the techniques presented in chapter 5 could

be used.

7.2.2 Adaptive Patrolling as adaptive sampling

Finally and perhaps the most interesting line of future work is the prospect of online adaptability for the defenders. Consider that *all* of the approaches introduced in this thesis compute schedules offline and then apply those same schedules in a "testing" environment where they compete against some attacker model. In general, one could establish the use of *online* methods [18] that continue to learn against the attackers even while still competing against them. Furthermore, this paradigm shares many similarities to those proposed in the area of adaptive sampling wherein an agent must collect samples at various locations of interest in a field in order to characterize a physical phenomenon. In works [25, 26], Booth et al. present algorithms for teams of agents trying to perform adaptive sampling for a phenomenon that also varies over time. By modelling the attacker's preference, one could apply these techniques to produce a map of the attacker's likely areas of interest and at the same time also react to any changes in the attacker's priorities. Furthermore, the aforementioned adaptive sampling techniques allow for a mechanism to determine when to resample target locations. In our case, this would be akin to determining the proper time that an area should be re-visited, thus further optimizing our patrolling resources. These methods would expand the applicability of the presented deterrence methods and most likely outperform methods that do not adapt online.

Bibliography

[1] P. Afshani, M. De Berg, K. Buchin, J. Gao, M. Löffler, A. Nayyeri, B. Raichel, R. Sarkar, H. Wang, and H.-T. Yang. Approximation algorithms for multi-robot patrol-scheduling with min-max latency. In *Algorithmic Foundations of Robotics XIV: Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics 14*, pages 107–123. Springer, 2021.

[2] P. Agharkar, R. Patel, and F. Bullo. Robotic surveillance and markov chains with minimal first passage time. In *53rd IEEE Conference on Decision and Control*, pages 6603–6608. IEEE, 2014.

[3] N. Agmon, V. Sadov, G. A. Kaminka, and S. Kraus. The impact of adversarial knowledge on adversarial planning in perimeter patrol. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 55–62, 2008.

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

[5] S. V. Albrecht and P. Stone. Reasoning about hypothetical agent behaviours and their parameters. In *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems*, pages 547–555, 2017.

[6] S. V. Albrecht and P. Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

[7] S. Alpern. Infiltration games on arbitrary graphs. *Journal of mathematical analysis and applications*, 163(1):286–288, 1992.

[8] C. D. Alvarenga, N. Basilico, and S. Carpin. Time-varying graph patrolling against attackers with locally limited and imperfect observation models. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4869–4876. IEEE, 2019.

[9] C. D. Alvarenga, N. Basilico, and S. Carpin. Multirobot patrolling against adaptive opponents with limited information. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2486–2492. IEEE, 2020.

[10] C. D. Alvarenga, N. Basilico, and S. Carpin. Combining coordination and independent coverage in multirobot graph patrolling. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4413–4419. IEEE, 2024.

[11] C. D. Alvarenga, N. Basilico, and S. Carpin. Learning generalizable patrolling strategies through domain randomization of attacker behaviors. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4406–4412. IEEE, 2024.

[12] B. An, D. Kempe, C. Kiekintveld, E. Shieh, S. Singh, M. Tambe, and Y. Vorobeychik. Security games with limited surveillance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 1241–1248, 2012.

[13] B. An, M. Brown, Y. Vorobeychik, and M. Tambe. Security games with surveillance cost and optimal timing of attack execution. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 223–230, 2013.

[14] A. B. Asghar and S. L. Smith. Stochastic patrolling in adversarial settings. In *Proc. ACC*, pages 6435–6440. IEEE, 2016.

[15] A. B. Asghar, S. L. Smith, and S. Sundaram. Multi-robot routing for persistent monitoring with latency constraints. In *2019 American Control Conference (ACC)*, pages 2620–2625. IEEE, 2019.

[16] M. Balcan, A. Blum, N. Haghtalab, and A. D. Procaccia. Commitment without regrets: Online learning in stackelberg security games. In *Proceedings of the sixteenth ACM conference on economics and computation*, pages 61–78, 2015.

[17] N. Basilico. Recent trends in robotic patrolling. *Current Robotics Reports*, pages 1–12, 2022.

[18] N. Basilico and S. Carpin. Online patrolling using hierarchical spatial representations. In *2012 IEEE International Conference on Robotics and Automation*, pages 2163–2169, 2012. doi: 10.1109/ICRA.2012.6224802.

[19] N. Basilico and S. Carpin. Balancing unpredictability and coverage in adversarial patrolling settings. In *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*, pages 762–777. Springer, 2020.

[20] N. Basilico, N. Gatti, and F. Amigoni. Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. *Artificial intelligence*, 184:78–123, 2012.

[21] N. Basilico, A. Celli, G. De Nittis, and N. Gatti. Coordinating multiple defensive resources in patrolling games with alarm systems. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 678–686, 2017.

[22] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[23] A. Blum, N. Haghtalab, and A. Procaccia. Lazy defenders are almost optimal against diligent attackers. In *Proc. AAAI*, pages 573–579, 2014.

[24] G. Bontempi, S. B. Taieb, and Y. Le Borgne. Machine learning strategies for time series forecasting. In *Business Intelligence*, pages 62–77, 2013.

[25] L. Booth and S. Carpin. Distributed estimation of scalar fields with implicit coordination. In *International Symposium on Distributed Autonomous Robotic Systems*, pages 466–478. Springer, 2022.

[26] L. Booth and S. Carpin. Informative path planning for scalar dynamic reconstruction using coregionalized gaussian processes and a spatiotemporal kernel. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8112–8119. IEEE, 2023.

[27] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[28] Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proc. IAT*, pages 302–308, 2004.

[29] V. Conitzer and T. Sandholm. Computing the optimal strategy to commit to. In *Proc. EC*, pages 82–90, 2006.

[30] Z. Cormier. The technology fighting poachers, 2019. URL `https://www.bbcearth.com/news/the-technology-fighting-poachers`. Accessed on May 22, 2024.

[31] S. R. Corp. Security robots for home use. URL `https://smprobotics.com/security_robot/security_home_robot/`. Accessed on June 03, 2024.

[32] X. Duan and F. Bullo. Markov chain–based stochastic strategies for robotic surveillance. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:243–264, 2021.

[33] Y. Elmaliach, N. Agmon, and G. Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Mathematics and Artificial Intelligence*, 57 (3-4):293–320, 2009.

[34] F. Fang, P. Stone, and M. Tambe. When security games go green: designing defender strategies to prevent poaching and illegal fishing. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 2589–2595, 2015.

[35] L. Freda, M. Gianni, F. Pirri, A. Gawel, R. Dubé, R. Siegwart, and C. Cadena. 3D multi-robot patrolling with a two-level coordination strategy. *Autonomous Robots*, 43(7):1747–1779, 2019.

[36] V. Furnon and L. Perron. Or-tools routing library. URL `https://developers.google.com/optimization/routing/`.

[37] S. Gal. Search games. *Wiley encyclopedia of operations research and management science*, 2010.

[38] M. Garey and D. Johnson. *Computers and Intractability. A guide to the theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[39] M. George, S. Jafarpour, and F. Bullo. Markov chains with maximum entropy for robotic surveillance. *IEEE Transactions on Automatic Control*, 64(4):1566–1580, 2018.

[40] J. Grace and J. Baillieul. Stochastic strategies for autonomous robotic surveillance. In *Proc. CDC*, pages 2200–2205, 2005.

[41] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartok, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo. TF-Agents: A library for reinforcement learning in tensorflow. `https://github.com/tensorflow/agents`, 2018. URL `https://github.com/tensorflow/agents`. [Online; accessed 25-June-2021].

[42] H. Guo, Q. Kang, W.-Y. Yau, M. H. Ang, and D. Rus. Em-patroller: Entropy maximized multi-robot patrolling with steady state distribution approximation. *IEEE Robotics and Automation Letters*, 2023.

[43] N. Haghtalab, F. Fang, T. H. Nguyen, A. Sinha, A. D. Procaccia, and M. Tambe. Three strategies to success: learning adversary models in security games. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 308–314, 2016.

[44] S. Hari, S. Rathinam, S. Darbha, K. Kalyanam, S. Manyam, and D. Casbeer. The generalized persistent monitoring problem. In *Proceedings of the American Control Conference*, pages 2783–2788, 2019.

[45] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[46] L. Huang, M. Zhou, K. Hao, and E. Hou. A survey of multi-robot regular and adversarial patrolling. *IEEE/CAA Journal of Automatica Sinica*, 6(4): 894–903, 2019.

[47] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20 (1):359–392, 1998. ISSN 1064-8275. doi: 10.1137/S1064827595287997. URL `http://epubs.siam.org/doi/10.1137/S1064827595287997`.

[48] K. Kerdprasop, N. Kerdprasop, and P. Sattayatham. Weighted k-means for density-biased clustering. In *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery*, pages 488–497, 2005.

[49] D. Klaška, A. Kučera, T. Lamser, and V. Řehák. Automatic synthesis of efficient regular strategies in adversarial patrolling games. In *Proc. AAMAS*, pages 659–666, 2018.

[50] N. Li, M. Li, Y. Wang, D. Huang, and W. Yi. Fault-tolerant and self-adaptive market-based coordination using hoplites framework for multi-robot patrolling tasks. In *2018 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 514–519, 2018.

[51] E. S. Lin, N. Agmon, and S. Kraus. Multi-robot adversarial patrolling: Handling sequential attacks. *ARTIF INTELL*, 274:1–25, 2019.

[52] S. Y. Luis, D. G. Reina, and S. L. Marín. A deep reinforcement learning approach for the patrolling problem of water resources through autonomous surface vehicles: The ypacarai lake case. *IEEE Access*, 8:204076–204093, 2020.

[53] D. L. Michaels. Knightscope deploys new autonomous security robot in southern california, 2022. URL `https://www.businesswire.com/news/home/20220316005436/en/Knightscope-Deploys-New-Autonomous-Security-Robot-in-Southern-California`. Accessed on May 22, 2024.

[54] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[55] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[56] R. Necula, M. Breaban, and M. Raschip. Tackling the bi-criteria facet of multiple traveling salesman problem with ant colony systems. In *Proceedings of the International Conference on Tools with Artificial Intelligence*, pages 873–880, 2015.

[57] Y. Oshart, N. Agmon, and S. Kraus. Non-uniform policies for multi-robot asymmetric perimeter patrol in adversarial domains. In *Proceedings of the International Symposium on Multi-Robot and Multi-Agent Systems*, pages 136–138, 2019.

[58] M. Othmani-Guibourg, A. El Fallah-Seghrouchni, and J. Farges. Path generation with lstm recurrent neural networks in the context of the multi-agent patrolling. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 430–437. IEEE, 2018.

[59] J. Palacios-Gasós, D. Tardioli, E. Montijano, and C. Sagüés. Equitable persistent coverage of non-convex environments with graph-based planning. *The International Journal of Robotics Research*, 38(14):1674–1694, 2019.

[60] J. M. Palacios-Gasós, E. Montijano, C. Sagues, and S. Llorente. Multi-robot persistent coverage using branch and bound. In *2016 American Control Conference (ACC)*, pages 5697–5702. IEEE, 2016.

[61] A. Papoulis and S. Pillai. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 4th edition, 2002.

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

[63] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

[64] A. Perrault, B. Wilder, E. Ewing, A. Mate, B. Dilkina, and M. Tambe. End-to-end game-focused learning of adversary behavior in security games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1378–1386, 2020.

[65] C. Piciarelli and G. L. Foresti. Drone patrolling with reinforcement learning. In *Proceedings of the 13th International Conference on Distributed Smart Cameras*, pages 1–6, 2019.

[66] C. Pippin, H. Christensen, and L. Weiss. Performance based task assignment in multi-robot patrolling. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 70–76, 2013.

[67] J. Pita, M. Jain, M. Tambe, F. Ordóñez, and S. Kraus. Robust solutions to

stackelberg games: Addressing bounded rationality and limited observations in human cognition. *ARTIF INTELL*, 174(15):1142–1171, 2010.

[68] D. Portugal and R. Rocha. MSP Algorithm : Multi-Robot Patrolling based on Territory Allocation using Balanced Graph Partitioning. *ACM Symposium on Applied Computing*, pages 1271–1276, 2010. ISSN 1079-0632. doi: 10.1145/1774088.1774360.

[69] D. Portugal and R. Rocha. A survey on multi-robot patrolling algorithms. In *Proc. DoCEIS*, pages 139–146, 2011.

[70] D. Portugal and R. P. Rocha. Distributed multi-robot patrol: A scalable and fault-tolerant framework. *Robotics and Autonomous Systems*, 61(12):1572–1587, 2013. ISSN 0921-8890.

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

[72] D. Portugal and R. P. Rocha. Performance estimation and dimensioning of team size for multirobot patrol. *IEEE Intelligent Systems*, 32(6):30–38, 2017.

[73] D. Portugal, C. Pippin, R. Rocha, and H. Christensen. Finding optimal routes for multi-robot patrolling in generic graphs. In *Proceedings of the IEEE/RSJ International Conference on Robots and Systems*, pages 363–369, 2014.

[74] N. Privault. *Understanding Markov Chains*. Springer, 2013.

[75] D. Rasch. Sample size determination for estimating the parameter of an exponential distribution. *Biometrical Journal*, 19(7):521–528, 1997.

[76] C. Robin and S. Lacroix. Multi-robot target detection and tracking: taxonomy and survey. *Autonomous Robots*, 40(4):729–760, 2016.

[77] M. Romeo, J. Banfi, N. Basilico, and F. Amigoni. Multirobot persistent patrolling in communication-restricted environments. In *Distributed Au-*

tonomous Robotic Systems: The 13th International Symposium, pages 59–71, 2018.

[78] S. Ross. *Introduction to Probability Models*. Elsevier, 2014.

[79] F. Rubio, F. Valero, and C. Llopis-Albert. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 16(2):1729881419839596, 2019.

[80] T. Sak, J. Wainer, and S. K. Goldenstein. Probabilistic multiagent patrolling. In *Advances in Artificial Intelligence-SBIA 2008: 19th Brazilian Symposium on Artificial Intelligence Savador, Brazil, October 26-30, 2008. Proceedings 19*, pages 124–133. Springer, 2008.

[81] H. Santana, G. Ramalho, V. Corruble, and B. Ratitch. Multi-agent patrolling with reinforcement learning. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1122–1129. IEEE Computer Society, 2004.

[82] J. Scherer and B. Rinner. Multi-uav surveillance with minimum information idleness and latency constraints. *IEEE Robotics and Automation Letters*, 5 (3):4812–4819, 2020.

[83] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[84] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[85] R. Shah, Y. Jiang, J. Hart, and P. Stone. Deep r-learning for continual area sweeping. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5542–5547. IEEE, 2020.

[86] A. Sinha, F. Fang, B. An, C. Kiekintveld, and M. Tambe. Stackelberg security games: Looking beyond a decade of success. In *27th International*

Joint Conference on Artificial Intelligence, IJCAI 2018, pages 5494–5501. International Joint Conferences on Artificial Intelligence, 2018.

[87] S. L. Smith, M. Schwager, and D. Rus. Persistent robotic tasks: Monitoring and sweeping in changing environments. *IEEE Transactions on Robotics*, 28 (2):410–426, 2011.

[88] R. Sun. Cognitive science meets multi-agent systems: A prolegomenon. *Philosophical psychology*, 14(1):5–28, 2001.

[89] R. Sun. Individual action and collective function: From sociology to multi-agent learning, 2001.

[90] R. Sutton and A. Barto. *Reinforcement Learning – An Introduction*. MIT Press, 2018.

[91] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[92] M. Tambe. Towards flexible teamwork. *Journal of artificial intelligence research*, 7:83–124, 1997.

[93] M. Tambe. *Security and game theory: algorithms, deployed systems, lessons learned*. Cambridge University Press, 2011.

[94] A. Technology. Portland angb deploys robot dog to enhance base security, April 19, 2022. URL `https://www.airforce-technology.com/news/portland-angb-deploys-robot-dog-to-enhance-base-security/?cf-view`. Accessed on June 03, 2024.

[95] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[96] J. Von Neumann. A certain zero-sum two-person game equivalent to the optimal assignment problem. *Contributions to the Theory of Games*, 2(0): 5–12, 1953.

[97] Y. Wang, Z. R. Shi, L. Yu, Y. Wu, R. Singh, L. Joppa, and F. Fang. Deep reinforcement learning for green security games with real-time information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1401–1408, 2019.

[98] A. Washburn and K. Wood. Two-person zero-sum games for network interdiction. *Operations research*, 43(2):243–251, 1995.

[99] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[100] H. Xu, A. Jiang, A. Sinha, Z. Rabinovich, S. Dughmi, and M. Tambe. Security games with information leakage: modeling and computation (2015). arxiv preprint. *Proc. IJCAI*, pages 674–680, 2015.

[101] C. Yan and T. Zhang. Multi-robot patrol: A distributed algorithm based on expected idleness. *International Journal of Advanced Robotic Systems*, 13 (6):1729881416663666, 2016. doi: 10.1177/1729881416663666. URL `https://doi.org/10.1177/1729881416663666`.

[102] Z. Yin, M. Jain, M. Tambe, and F. Ordonez. Risk-averse strategies for security games with execution and observational uncertainty. In *Proc. AAAI*, pages 758–763, 2011.

[103] C. Zhang, V. Bucarey, A. Mukhopadhyay, A. Sinha, Y. Qian, Y. Vorobeychik, and M. Tambe. Using abstractions to solve opportunistic crime security games at scale. In *Int. Conf. on Autonomous Agents and Multiagent Systems*, pages 196–204, 2016.