POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAM IN INFORMATION TECHNOLOGY

PATROLLING ADVERSARIAL ENVIRONMENTS EXPLOITING AN ALARM SYSTEM

Doctoral Dissertation of:
Giuseppe De Nittis

Supervisor:
Prof. Nicola Gatti
Co-supervisor:
Dr. Nicola Basilico

Tutor:
Prof. Francesco Amigoni

Chair of the Doctoral Program:
Prof. Andrea Bonarini

2017 – Cycle XXX

Giuseppe De Nittis
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
e-mail: giuseppe.denittis@polimi.it

To those who strive every day
to leave this world a little better than they found it.

Abstract

Physical security is one of the most important challenges of our times. Due to the terrible events happened in the last decades, e.g., on September 11, 2001, or nowadays in several European countries, new techniques and methods are being developed to face new threats and dangers. On the other side, security means also saving lives and helping people, e.g., detecting desperate migrants that are moving across the Mediterranean Sea and rescuing them.

Algorithmic Game Theory is a fundamental tool in these settings, giving us the possibility to scientifically investigate them, modeling the interaction between law enforcers and terrorists or criminals as a mathematical problem, and designing suitable algorithms to deal with these threats.

Often, in large environments or infrastructures, a constant surveillance of every area is not affordable: to cope with this issue, a common countermeasure is the usage of cheap but wide-ranged sensors, able to detect suspicious events that occur in large areas, supporting patrollers to improve the effectiveness of their strategies. This is a two-phase approach: first, there is a surveillance phase, by means of sensors or drones to detect the intrusion and, only when an attack is actually detected, the patrollers rush towards the threat to block it. Given this scenario, how would you exploit the information that something is happening in some areas if you do not know the exact position of the attack? How would you coordinate multiple guards to catch the attacker? And how would you react if, once started moving towards the attack location, another alarm is triggered, telling you that multiple attacks are ongoing?

To tackle these problems, we propose the first Security Game model with the presence of an alarm system able to trigger alarm signals, which carry the information about the set of targets that can be under attack and it is described by the probability of being generated when each target is attacked. We investigate two main research lines.

On one side, we study the uncertainties that may affect a real-world alarm system: spatial uncertainty, i.e., the alarm system is uncertain about the exact target under attack. In other words, when an attack is occurring, a signal is sent to the Defender, saying that something bad is happening in an area but without specifying the exact location. We show that without false positives and missed detections, the best patrolling strategy reduces to stay in a place, wait for a signal, and respond to it at best. Then, we introduce a significant positive missed detection rate, i.e., no alarm signal is generated even though an attack is occurring. In particular, we show that standing still and waiting for a signal is no more the best response, while movements among the areas of the environment to protect should be considered. Moreover, we introduce the notion of uncertainty also in the type of attacker we are playing against. Specifically, we tackle the problem of facing an unknown adversary, whose profile is just known to be in a list of possible profiles she can assume. The problem is completely different, requiring to identify the Attacker we are facing to exploit such information in the future and be able to prevent her from performing other attacks.

The other direction we investigate is about the dimension of the problem, i.e., the number of resources both the Attacker and the Defender can control. Thus, first we tackle the problem of finding the minimum number of resources assuring non-null protection to every target, which is of high relevance in practice due to resource costs and to the need for assuring a minimum level of protection to each target. Then, we study how the Defender should move such resources, taking into account also the level of coordination among such defensive resources, e.g., they cannot communicate because of a radio silence constraint since they are operating on an undercover mission. On the other side, we investigate the opportunities the Attacker can take when she is allowed to perform multiple attacks, simultaneously or sequentially. Then, since the computation of the equilibrium strategies requires the common knowledge about the number of Attacker's resources but this information is unlikely to be common, we assume that the Defender makes a guess about the number of resources and plays her best strategy accordingly, and we evaluate the worst-case inefficiency of this strategy when the Attacker has a different number of resources.

Contents

CHAPTER 1

Introduction

If the pages of this book contain some successful verse, the reader must excuse me the discourtesy of having usurped it first.

— Jorge Luis Borges

Physical security is one of the most important challenges of our times. Due to the terrible events happened in the last decades, e.g., on September 11, 2001, or nowadays in several European countries, new techniques and methods are being developed to face new threats and dangers. On the other side, security means also saving lives and helping people, e.g., detecting desperate migrants that are moving across the Mediterranean Sea and rescuing them.

Algorithmic Game Theory is a fundamental tool in these settings, giving us the possibility to scientifically investigate them, modeling the interaction between law enforcers and terrorists or criminals as a mathematical problem, and designing suitable algorithms to deal with these threats.

Both in the scientific community and for real-world applications, finding the optimal schedule of scarce resources to face strategic adversaries willing to commit crimes against valuable targets is nowadays one of the most

challenging opportunities provided by Artificial Intelligence [67]. Often, in large environments or infrastructures, a constant surveillance of every area is not affordable: to cope with this issue, a common countermeasure is the usage of cheap but wide-ranged sensors, able to detect suspicious events that occur in large areas, supporting patrollers to improve the effectiveness of their strategies. This is a two-phase approach: first, there is a surveillance phase, by means of sensors or drones to detect the intrusion and, only when an attack is actually detected, the patrollers rush towards the threat to block it. Real-world applications include UAVs surveillance of large infrastructures [18], wildfires detection with CCD cameras [74], agricultural fields monitoring [58], surveillance based on wireless sensor networks [129], and border patrolling [112]. Now, we describe a realistic security scenario, where the goal is the protection of a huge environment by means of a scarce number of resources. In the rest of the dissertation, we will necessarily take a general stance, but we encourage the reader to keep in mind this case as a reference for the problems we are going to solve and the techniques we are going to provide.

1.1 Problem: Securing Large Buildings and Environments

Art museums are huge spaces, often characterized by big buildings and wide gardens. The Louvre museum and the Tuileries gardens, with their 782,910 square feet, constitute the world's largest museum and a historical monument in Paris, France. The museum has a collection of $460,000$ works, $38,000$ of which are on display and million of visitors per year (7.4 millions of visitors in 2016, 8.6 millions in 2015) [76]. The whole museum is protected by security officers, who are part of a highly structured, close-knit team responsible for welcoming the public, ensuring visitor safety, protecting the collections, palace buildings, and grounds. Security officers account for over half of the Louvre's 2,000-member staff. There are gallery security officers, permanent security officers attached to the various curatorial departments, *district* heads, emergency and mobile teams, luggage check staff, *ushers*, i.e., security officers at the museum entrances, and technical and administrative posts, e.g., staff responsible for the 24-hour surveillance of closed-circuit TV cameras and intruder-detection equipment. Security officers working in the galleries are usually assigned to one of the museum's *districts*, each covering between ~ 3200 and ~ 8600 square feet and incorporating areas grouped according to criteria such as type of artwork, mode of presentation, level of fragility of the artworks, and the architectural characteristics of the galleries. *Regional* officers are assigned to posts within a particular wing of the mu-

seum. *Interregional* officers are assigned to posts throughout the museum, in response to changing daily requirements.

To detect possible attacks or damages, Louvre employs security cameras, e.g., CCTVs or 360-degree cameras, motion detectors, smoke detectors and noise detectors, which have some degree of uncertainty. Given this scenario, how would you exploit the information that something is happening in some areas, if you do not know the exact position of the attack? How would you coordinate multiple guards to catch the attacker? And how would you react if, once started moving towards the attack location, another alarm is triggered, telling you that multiple attacks are ongoing?

1.2 Solution: Adversarial Patrolling with an Alarm System

To tackle these problems, we propose the first Security Game model that integrates a spatially uncertain alarm system in game-theoretic settings for patrolling.

Security Games have been adopted to model and solve lots of real-world problems [70]. However, in almost all the cases, the interaction between the two playing agents is very limited: given the environment and the target to protect or attack, respectively by the Defender and the Attacker, these models propose a static approach, not taking into account dynamic information that may arise during the defending phase.

We originally provide a security game with the presence of an alarm system able to trigger alarm signals, which carry the information about the set of targets that can be under attack and it is described by the probability of being generated when each target is attacked. A rough model of an alarm system has been introduced in [82], where sensors have no spatial uncertainty in detecting attacks on single targets. The work analyzes how sensory information can improve the effectiveness of patrolling strategies in adversarial settings. It is shown that, when sensors are affected neither by false negatives nor by false positives, the best strategy prescribes that the patroller just responds to an alarm signal rushing to the target under attack without patrolling the environment. As a consequence, in such cases the model treatment becomes trivial.

Our work takes a big step from this very basic model, investigating two main research lines. On one side, we study the uncertainties that may affect a real-world alarm system: spatial uncertainty, i.e., the alarm system is uncertain about the exact target under attack, and missed detections, i.e., no alarm is triggered even if an attack is occurring. On the other, we develop models and design algorithms for scenarios in which the number of resources

available to the players, either the Defender or the Attacker, increases, thus allowing a coordinated defense or multiple attacks.

More specifically, at first we consider the scenario in which the Defender can control a single patroller and the alarm system is affected by spatial uncertainty. In other words, when an attack is occurring, a signal is sent to the Defender, saying that something bad is happening in an area but without specifying the exact location. We divide the problem into two phases: signal response and patrolling. For the signal response phase, we provide a complexity analysis, showing that finding the optimal strategy is FNP-hard even in tree graphs and APX-hard in arbitrary graphs. We provide two (exponential time) exact algorithms and two (polynomial time) approximation algorithms. We also show that without false positives and missed detections, the best patrolling strategy reduces to stay in a place, wait for a signal, and respond to it at best. This strategy is optimal even with non-negligible missed detection rates, which, unfortunately, affect every commercial alarm system. Some preliminary results have been published in [20].

Then, we introduce a significant positive missed detection rate, i.e., no alarm signal is generated even though an attack is occurring. This new scenario is in favor of the Attacker, who can exploit such flaw of the alarm system. This is why such new element puts the problem in a different perspective and requires a different approach to be solved. We deeply analyze security games in which the alarm system is both characterized by detection uncertainty and spatial imperfection, tackling the challenge of designing tractable algorithms for real-life scenarios. In particular, we show that standing still and waiting for a signal is no more the best response, while movements among the areas of the environment to protect should be considered. In fact, we prove that Markovian strategies are arbitrarily worse than optimal non-Markovian ones, and thus we resort to a deterministic approach. Some preliminary results have been published in [19].

The other direction we investigate is about the dimension of the problem, i.e., the number of resources both the Attacker and the Defender can control. Thus, we study how the Defender should behave if she can control multiple resources, taking into account also the level of coordination among such defensive resources, e.g., they cannot communicate because of a radio silence constraint since they are operating on an undercover mission. The challenge is designing algorithms able to scale up with the number of resources. The problem of finding the best Defender's strategy when a number of resources are given is FNP-hard from the case with a single resource. Conversely, we would expect that the problem of finding the minimum number of resources assuring non-null protection to every target, which is of high relevance in

practice due to resource costs and to the need for assuring a minimum level of protection to each target, being easier, since it does not require to compute the strategies. Nevertheless, we show that this problem is log-APX-complete on arbitrary graphs, while it is in FP in tree and cycle graphs, usually representing docks and borders respectively. Then, we study the problem of finding the best strategy to respond to any alarm signal once an allocation of resources in the environment is given, according to different degrees of coordination among the resources, each described by an adversarial team game with different forms of strategies (correlated or mixed). Some preliminary results have been published in [16].

On the other side, we will investigate the opportunities the Attacker can take when she is allowed to perform multiple attacks, simultaneously or sequentially. We show that, when we restrict the Attacker to use all her resources simultaneously, there is no algorithm that requires polynomial time in the number of Attacker's resources unless $\mathsf{P} = \mathsf{NP}$. Conversely, when the number of resources is a fixed parameter, the problem admits a polynomial time algorithm. Furthermore, fixing the number of attacking resources, the unrestricted model in which the Attacker can perform different sequential attacks admits a non-trivial polynomial time algorithm based on dynamic programming, able to find the strategies on the equilibrium path (instead, off the equilibrium path, the game may have infinite horizon). The computation of the equilibrium strategies requires the common knowledge about the number of Attacker's resources. This information—as well as a Bayesian prior required in Bayesian games—is unlikely to be common. For this reason, we follow a different approach that we believe to be more suitable in practice, but largely unexplored in Game Theory. We assume that the Defender makes a guess about the number of resources and plays her best strategy accordingly, and we evaluate the worst-case inefficiency of this strategy when the Attacker has a different number of resources. We show that the inefficiency can be arbitrary even when the guess is a wrong estimate—both over and under—for just a single resource. This suggests the use of online algorithms that do not need any guess. We provide a tight upper bound over the competitive factor when non-stochastic online algorithms are used and we show that the factor can be improved by resorting to randomization.

Finally, we will tackle the problem of facing an unknown adversary, whose profile is just known to be in a list of possible profiles she can assume. The problem is completely different, requiring to identify the Attacker we are facing to exploit such information in the future, and be able to prevent her from performing other attacks. We define a novel scenario in which the Defender plays against an Attacker whose behavior is unknown

but it belongs to a set of known profiles. We show that state-of-the-art bandit and expert algorithms—suitable for our problem—suffer from a linear and logarithmic regret, respectively, in the length of the time horizon. Thus, we introduce two novel approaches to deal with our problem, bridging together game-theoretical techniques and online learning tools. In the first approach, the Defender has a belief about the Attacker and updates it during the game and we provide a finite-time analysis showing that the regret of the algorithm is constant in the length of the time horizon. In the second approach, the learning policy is driven directly by the estimated expected regret and is based on a backward induction procedure. We provide a thorough experimental evaluation in concrete security settings, comparing our algorithms with the main algorithms available in the state of the art of the online learning field and showing that our approaches provide a remarkable improvement in terms of expected pseudo-regret minimization. Some preliminary results have been published in [28].

1.3 Structure of the Thesis

The rest of the thesis is organized as follows.

- In Part I, we present the foundations of Algorithmic Game Theory and Online Learning, providing all the notions and concepts that will be useful to better understand the results showed in the thesis (see Chapter 2). Then, in Chapter 3, we describe the main steps that have been undertaken in Security Games, then focusing on Patrolling Games and Learning Games.

- Part II studies the security problem from the perspective of the uncertainties that may affect the alarm system. Specifically, Chapter 4 introduces the basic model, with the Attacker able to perform a single attack and the Defender able to control just a mobile resource, supported by a spatially uncertain alarm system, i.e., the information carried by the system is about some areas in which the attack is occurring, and not the exact location. Then, in Chapter 5, we insert a positive missed detection rate, meaning that the system may not work even though the Attacker is performing a malicious action.

- Part III deals with multiple resources, studying how the problem changes, becoming more complex, if both agents can control more assets. Specifically, in Chapter 6 we allow the Defender to control multiple homogeneous patrolling resources, which can be coordinated on three different

levels. We will prove the benefits of full coordination w.r.t. other types and provide the best behavior for the Defender in such settings. Conversely, in Chapter 7, we give the Attacker the possibility of carrying on multiple attacks, either simultaneous or sequential. We provide an effective way of dealing with this problem when the number of possible attacks is known *a priori*, but we will also prove that, if the Defender makes a wrong guess about such number, under/over estimating it, the loss can be arbitrary.

- Part IV lead us in a new context, learning. In fact, the problem we pose here is how to face an unknown adversary, who can assume one among multiple profiles known to the Defender. We formulate this problem in an online learning framework, showing the effectiveness, in terms of loss incurred by the Defender, of identifying the type of Attacker to exploit it in successive interactions with the Attacker, giving us the possibility of best responding to her attacks.
- Finally, Chapter 9 concludes this thesis and presents some interesting future research lines that may be undertaken starting from what has been developed, proved and implemented in this work.

Part I

Starting Point

CHAPTER 2

Foundations of Algorithmic Game Theory and Online Learning

No one can pass through life, any more than he can pass through a bit of country, without leaving tracks behind.

— Robert Baden-Powell

Algorithmic Game Theory (AGT) bridges together two sciences, Mathematics, more specifically Game Theory, and Computer Science, more specifically Algorithms and Computational Complexity. In this chapter, we introduce the foundations of Algorithmic Game Theory, providing notions and concepts that will be adopted throughout the thesis.

2.1 Algorithms and Computational Complexity

Computational complexity

Here, we present the main concepts related to the algorithmic side of our approach, introducing notions about algorithms and computational complexity.

Before doing so, we want to give the rationale behind the employment of such techniques.

The motivation lies in the fact that it is not enough knowing and proving that a solution exists, we also want to find it. Finding a solution means we are able to compute in reasonable time w.r.t. the size of the problem we are going to solve.

Now, we are ready to formally define what a problem is.

Definition 2.1 (Problem)**.** *A problem Π is described by giving:*

- *a general description of all its parameters;*
- *a statement of what properties the answer, or solution, is required to satisfy.*

An instance of a problem is obtained by specifying particular values for all the problem parameters.

Example 1. *Let us consider the Traveling Salesman Problem (TSP).*

Definition 2.2 (TSP)**.** *The Travelling Salesman Problem (TSP) asks the following question: given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?*

The parameters of this problem consist of a finite set $C = \{c_1, c_2, \ldots, c_m\}$ of cities and, for each pair of cities $c_i, c_j \in C$, the distance $d(c_i, c_j)$ between them. A solution is an ordering $\langle c_{\tau(1)}, \ldots, c_{\tau(2)}, c_{\tau(m)} \rangle$ of the given cities that minimizes:

$$\left(\sum_{i=1}^{m-1} d(c_{\tau(i)}, c_{\tau(i+1)}) \right) + d(c_{\tau(m)}, c_{\tau(1)})$$

This expression gives the length of the tour that starts at $c_{\tau(1)}$ visits each city in sequence, and then returns directly to $c_{\tau(1)}$ from the last city $c_{\tau(m)}$.

A computational analysis of a problem allows us to whether, beside the existence, we are actually able to provide a solution that can be then implemented. To find a solution to a problem, we design algorithms.

Definition 2.3 (Algorithm)**.** *An algorithm is a general, self-contained, step-by-step procedure for solving problems that, given an input, produces an output.*

Example 2. *A simple (but computationally heavy) algorithm for the TSP is the following.*

- *input: an instance of TSP constituted by the corresponding graph;*

- *output: a sequence of vertices;*
- *procedure: try all the permutations of cities and then extract the one with the associated tour with the minimum cost. Return such sequence as the output.*

2.1.1 Computational Complexity in a Nutshell

In general, given a problem, we are interested in finding the most *efficient* algorithm for solving that problem. Usually, different algorithms can be proposed to solve a problem, but it may happen that, due to the nature of the problem itself, we cannot be more efficient than a certain threshold.

The efficiency of an algorithm is defined evaluating the time and memory space needed by its execution. Usually, time is the most used indicator, so by the *most efficient algorithm* one normally means the fastest.

The computing time of an algorithm is expressed in terms of the number of elementary operations (arithmetic operations, comparisons or memory accesses) needed to solve a given instance I, assuming that all such operations require one time unit. The time requirements of an algorithm are conveniently expressed in terms of a single variable, the *size* of a problem instance, which is intended to reflect the amount of input data needed to describe the instance. For the TSP, the number of cities is commonly used for this purpose. However, an m-city problem instance includes, in addition to the labels of them cities, a collection of $m(m-1)/2$ numbers defining the inter-city distances, and the sizes of these numbers also contribute to the amount of input data.

Definition 2.4 (Size of an instance)**.** *The size of an instance I, denoted by $|I|$, is the number of bits needed to encode I.*

We look for a function $f(n)$ such that, for every instance I of size at most n, the number of elementary operations to solve instance $I \leq f(n)$. We observe that we are considering the worst case since $f(n)$ is an upper bound.

Definition 2.5 (Polynomial algorithm)**.** *An algorithm is* polynomial *if it requires, in the worst case, a number of elementary operations $f(n) = O(n^d)$, where d is a constant and $n = |I|$ is the size of the instance.*

Observation 1. *If the number of elementary operations required by an algorithm is $f(n) = O(n^d \cdot \log^c n)$, the algorithm is still polynomial.*

Definition 2.6 (Exponential algorithm)**.** *An algorithm is* exponential *if it requires, in the worst case, a number of elementary operations $f(n) = O(2^n)$, where $n = |I|$ is the size of the instance.*

Observation 2. *In real life scenarios, polynomial algorithms with a big exponent, e.g., ≥ 5 may require a lot of time to be executed, becoming not usable.*

2.1.2 Complexity of Problems

As customary in the literature, we divide problems in three main groups: decision, optimization and function problems.

Definition 2.7 (Decision problem)**.** *A decision problem is a question with a yes-or-no answer, depending on the values of some input parameters.*

Definition 2.8 (Function problem)**.** *A function problem Π is defined as a relation $R(x, y)$ over a cartesian product over strings of an arbitrary alphabet Σ: $R \subset \Sigma^* \times \Sigma^*$. An algorithm solves Π if for every input x such that there exists a y satisfying $(x, y) \in R$, the algorithm produces one such y.*

Definition 2.9 (Optimization problem)**.** *An optimization problem is the problem of finding the best solution from all feasible solutions.*

Observation 3 (Problem formulation)**.** *Each problem can be formulated in each form: decision, function and optimization.*

Example 3. *We report three different formulations of the TSP problem, according to the definitions given above.*

- *Decision TSP: Given a list of cities, the distances between each pair of cities and a length L, is there a possible route that visits each city exactly once and returns to the origin city with a cost smaller or equal to L?*
- *Function TSP: Given a list of cities, the distances between each pair of cities and a length L, find a route that visits each city exactly once and returns to the origin city with a cost smaller or equal to L.*
- *Optimization TSP: Given a list of cities and the distances between each pair of cities, minimize the length of the route that visits each city exactly once and returns to the origin city.*

In the following, we present the fundamental complexity classes related to the group of problems we introduced above. Such classes consider the intrinsic difficulty of such problems to be solved despite the algorithm adopted to solve them.

Complexity classes

We report the definitions for the two main complexity classes of each group of problems: decision, function and optimization problems.

First, we define the three main classes of decision problems.

Definition 2.10 (P)**.** P *is the set of all decision problems that can be solved by a deterministic Turing machine using a polynomial amount of computation time.*

Definition 2.11 (NP)**.** NP *is the set of all decision problems such that, for each instance with* yes *answer, there exists a concise certificate, called proof, which allows to verify in polynomial time that the answer is* yes.

Definition 2.12 (coNP)**.** *A decision problem* Π *is a member of co*NP *if and only if its complement* $\overline{\Pi}$ *is in the complexity class* NP.

Then, we define the three main classes of function problems.

Definition 2.13 (FP)**.** *The complexity class* FP *is the set of function problems which can be solved by a deterministic Turing machine in polynomial time. A binary relation* $P(x, y)$ *is in* FP *if and only if there is a deterministic polynomial time algorithm that, given* x*, can find some* y *such that* $P(x, y)$ *holds.*

Definition 2.14 (FNP)**.** *A binary relation*$P(x, y)$*, where* y *is at most polynomially longer than* x*, is in* FNP *if and only if there is a deterministic polynomial time algorithm that can determine whether* $P(x, y)$ *holds given both* x *and* y*.*

Definition 2.15 (coFNP)**.** *A function problem* Π *is a member of co*FNP *if and only if its complement* $\overline{\Pi}$ *is in the complexity class* FNP.

Finally, we define the three main classes of optimization problems.

Definition 2.16 (PO)**.** PO *is the set of all decision problems that can be solved by a deterministic Turing machine using a polynomial amount of computation time.*

Definition 2.17 (NPO)**.** NPO *is the set of all optimization problems such that, for each instance with* yes *answer, there exists a concise certificate, called proof, which allows to verify in polynomial time that the answer is* yes.

Definition 2.18 (coNPO)**.** *An optimization problem* Π *is a member of co*NPO *if and only if its complement* $\overline{\Pi}$ *is in the complexity class* NPO.

In the following, we will focus on decision problems. Similar concepts and results also hold for function and optimization problems.

Now, we want to understand how to study the complexity of a new problem. The first step is to understand whether or not we can exploit the knowledge of the complexity of other problems. The answer is positive: we resort to *reduction among problems*.

Definition 2.19 (Reduction)**.** *A reduction is a transformation of one problem* Π *into another problem* Π'.

The reduction captures the informal notion of a problem being at least as difficult as another problem. For instance, if a problem Π can be solved using an algorithm for Π', Π is no more difficult than Π', and we say that Π reduces to Π'. The most commonly used reduction is the polynomial-time reduction. This means that the reduction process takes polynomial time.

Example 4. *The problem of squaring an integer can be reduced to the problem of multiplying two integers. This means an algorithm for multiplying two integers can be used to square an integer. Indeed, this can be done by giving the same input to both inputs of the multiplication algorithm. Thus we see that squaring is not more difficult than multiplication since squaring can be reduced to multiplication.*

This motivates the concept of a problem being hard for a complexity class. A problem Π is hard for a class of problems Γ if every problem in Γ can be reduced to Π. Thus no problem in Γ is harder than Π since an algorithm for Π allows us to solve any problem in Γ.

Definition 2.20 (NP-hardness)**.** NP*-hardness (non-deterministic polynomial-time hard) is the class of decision problems that are at least as hard as the hardest problems in* NP*. More precisely, a problem* Π *is* NP*-hard when every problem* Π' *in* NP *can be reduced in polynomial time to* Π*.*

Definition 2.21 (NP-completeness)**.** *A decision problem* Π *is* NP*-complete if:*

1. Π *is in* NP*;*
2. *every problem in* NP *is reducible to* Π *in polynomial time.*

Thus, the class of NP-complete problems contains the most difficult problems in NP, in the sense that they are the ones most likely not to be in P. Moreover, the following theorem holds.

Theorem 2.1. *Proving that a problem is* NP*-hard does not imply proving that the problem is in* NP*.*

We list in Table 2.1 some problems that appear to be similar but whose complexity is completely different [59].

P	NP-complete
Shortest path between two vertices	Longest path between two vertices
Edge cover	Vertex cover
TSP on unitary graph	TSP on $1-2$ graph

Table 2.1: *Examples of similar problems with different complexity.*

All the problems in NP are reducible between themselves: the difference is in the reduction, which makes the encoding problem emerge. In the next section, we provide some NP-reductions among problems.

Analyzing a problem from a computational perspective allows us to know in advance how much time we need to solve a problem. If a problem is NP-hard, then we already know the time needed to solve is way too long and we have to resort to approximation algorithms.

2.1.3 Approximability of a Problem

When we want to solve a problem, the best case would be having an exact polynomial algorithm, since in a reasonable amount of time can provide the solution to our problem. Unfortunately, as we have seen, sometimes it is not possible to solve *exactly* some problems in polynomial time. In these cases, we turn to approximation algorithms, i.e., polynomial algorithms that provide guarantees on the optimization value. Of course, such value will be smaller or equal to the value that we would obtain applying the exact algorithm.

The notation we use in the following is referred to a *maximization* problem.

Definition 2.22 (Approximation factor)**.** *Let OPT be the value of the optimal solution, returned by the exact but exponential algorithm, and APX the value of the solution returned by an approximation algorithm for some problem Π. We call* approximation factor*: $\rho = \frac{APX}{OPT} \leq 1$*[1].

Among all the polynomial algorithms, we are looking for the best in terms of approximation factor, that is:

$$\max_{algorithms} \quad \min_{istances} \frac{APX}{OPT} = \rho^*.$$

[1]The notation we use in the rest of the section is referred to a *maximization* problem.

We must look for a polynomial algorithms: in fact, if we could search among *all* the algorithms, then we could simply select an optimal exponential algorithm, having $\rho = 1$.

In principle, we would like to have $\rho = 1$, meaning that we have found the optimal solution in polynomial time. In our cases, it is not possible since we are focusing on problems which cannot be efficiently solved in polynomial time and thus we can only aspire to OPT.

Let us denote with ε: the approximation precision we want to obtain and with $\Omega = \Omega(f(\varepsilon), n)$ the approximation lower bound (we are solving a maximization problem). Thus, the following inequality holds:

$$\text{MAX: } \frac{APX}{OPT} \leq 1 - \varepsilon$$

We are ready to define the main complexity classes w.r.t. approximation of optimization problems.

Definition 2.23 (APX)**.** APX *is the complexity class of all the optimization problems* Π *that are in* NP *and that admit polynomial approximation algorithms with an approximation factor bounded by a constant value:* $\rho^* = \frac{1}{c}$.

Definition 2.24 (APX-hardness)**.** *A problem is said to be* APX*-hard if there is a polynomial-time reduction, within every multiplicative factor of the optimum other than 1, from every problem in* APX *to that problem.*

Definition 2.25 (APX-completness)**.** *A problem is said to be* APX*-complete if it is* APX*-hard and belongs to* APX*.*

Example 5. *The TSP problem, as in Definition 2.2, is* APX*-complete.*

Definition 2.26 (log-APX)**.** APX *is the complexity class of all the optimization problems* Π *that are in* NP *and that admit polynomial approximation algorithms with an approximation factor bounded by a logarithmic approximation factor:* $\rho^* = \frac{1}{log^c(n)}$.

Definition 2.27 (log-APX-hardness)**.** *A problem is said to be log-*APX*-hard if there is a polynomial-time reduction, within every multiplicative factor of the optimum other than 1, from every problem in log-*APX *to that problem.*

Definition 2.28 (log-APX-completness)**.** *log-*APX*-complete consists of the hardest problems that can be approximated efficiently to within a factor logarithmic in the input size and belongs to log-*APX*.*

Example 6. *Let us consider one of the classical 21 problems studied by Richard Karp [71], the Set Cover Problem.*

Definition 2.29 (SET-COVER problem)**.** *INPUT: a set of element $U = \{u_1, \ldots, u_n\}$, a collection $S = \{s_1, \ldots, s_m\}$ of sets whose union equals U; QUESTION: is there a sub-collection of S whose union equals the universe and whose cardinality is smaller than or equal to K?*

Theorem 2.2. *SET-COVER problem is log-*APX*-complete.*

Definition 2.30 (poly-APX)**.** APX *is the complexity class of all the optimization problems Π that are in* NP *and that admit polynomial approximation algorithms with an approximation factor bounded by a polynomial approximation factor: $\rho^* = \frac{1}{n^c}$.*

Definition 2.31 (poly-APX-hardness)**.** *A problem is said to be log-*APX*-hard if there is a polynomial-time reduction, within every multiplicative factor of the optimum other than 1, from every problem in poly-*APX *to that problem.*

Definition 2.32 (poly-APX-completness)**.** *Poly-*APX*-complete consists of the hardest problems that can be approximated efficiently to within a factor linear in the input size and belongs to poly-*APX*.*

Example 7. *Let us consider another of the classical 21 problems studied by Richard Karp [71], the Independent Set Problem.*

Definition 2.33 (INDEPENDENT-SET problem)**.** *Given an undirect graph $G = (V, E)$ and an integer K, is there a subset S with at least K not adjacent vertices, i.e., there are no edges connecting any pair of vertices in S?*

Theorem 2.3. *INDEPENDENT-SET problem is poly-*APX*-complete.*

In the above cases, as $n \to \infty$, $\Omega \to 0$: Ω is not independent of the dimension of the problem.

2.1.4 Solving a Problem

When we want to solve a problem, we follow these consecutive steps.

1. There exists an exact polynomial algorithm. The problem is in P.
2. There are no exact polynomial algorithms and the problem is NP-hard. Thus, we want to design a polynomial algorithm to better approximate the solution. First, we try to design an algorithm whose solution can be as close as we want to the optimal. There are three possible cases.
   (a) $EPTAS$: *Exactly Polynomial Time Approximation Scheme*. We find a polynomial algorithm.

- Complexity: $O(n^c)$, with c constant.

(b) $FPTAS$: *Fully Polynomial Time Approximation Scheme*. We find a fully polynomial algorithm.

- Approximation factor: $\rho^* = \Omega(1-\varepsilon)$.
- Complexity: $O(n^c(\frac{1}{\varepsilon})^k)$, with c, k constants.

(c) $PTAS$: *Polynomial Time Approximation Scheme*.

- Approximation factor: $\rho^* = \Omega(1-\varepsilon)$
- Complexity: $O(n^{f(\frac{1}{\varepsilon})})$, with c, k constants. Fixed that the complexity is polynomial, the scheme keeps being a $PTAS$ even if for different instances we have different ε functions.

Observation 4. *$FPTAS$ and $PTAS$ are approximation schemes in which Ω can go arbitrarily to zero.*

3. If we cannot provide an algorithm with an approximation factor bounded by a constant value, then we look for an algorithm with an approximation factor bounded by a logarithmic function of n.
4. If we cannot provide an algorithm with an approximation factor bounded by a logarithmic function of n, then we look for an algorithm with an approximation factor bounded by a polynomial function of n.
5. If we cannot provide an algorithm with an approximation factor bounded by a polynomial function of n, then the only thing we can do is to adopt the exact exponential algorithm.

We now report some useful results about approximating a problem.

Theorem 2.4. *If we can provide a $FPTAS$ or $PTAS$ for a problem Π, then we have proved the membership of Π to* APX.

Theorem 2.5. *If Π does not admit a pseudo-polynomial algorithm, then it does not admit a $FPTAS$.*

Theorem 2.6. *If a problem is* APX*-hard, then it does not admit $PTAS$.*

Theorem 2.7. *If a problem is* NP*-hard and all its solutions are in $\mathbb{Z}^+$, then it does not admit a $PTAS$.*

Besides $FPTAS$ and $PTAS$ there are *constant approximation* algorithms.

Theorem 2.8. *If Π does not admit a $PTAS$, then it can still admit a constant approximation algorithm.*

Theorem 2.8 suggests that we can still approximate a difficult problem that does not admit a $PTAS$ but the approximation will not depend on a parameter ε, meaning that it will be constant. Thus, constant approximation is worse than polynomial approximation.

Since we are interested in defining the approximation factor for new problems, we wonder if we can exploit already known results, as we have done for NP-hardness. The answer is positive since the notion of reduction also applies for approximating a problem. We use the following notation:

- Π is a known APX-complete problem;
- Π' is the problem we want to reduce;
- ρ is the approximation factor of Π';
- ρ' is the approximation factor of Π;
- $\bar{\rho}'$ is the threshold beyond which Π cannot be approximated anymore.

We show now the steps of a reduction technique, known as *approximation preservation*.

1. For every instance I of Π we have to construct an instance I' of Π' in polynomial time;
2. We assume to have an approximated algorithm f to solve I' with an approximation factor ρ. We call $\varsigma = f(I')$ the ρ-approximation of OPT of Π';
3. We must prove that we can construct s, the ρ'-approximation of OPT of Π, in polynomial time, with $\rho' > \bar{\rho}'$.

So, if we would be able to approximate Π' with ς, then we would be able to approximate Π with s. But s has an approximation factor which is higher than the threshold beyond which Π can be approximated.

This is a contradiction.

Theorem 2.9. *Proving that a problem is* APX*-hard does not imply proving that the problem is in* APX.

As for NP, for a problem to be in APX, we need to prove the membership. To do this, we need to provide a constant ratio algorithm, i.e., an algorithm that provides a ratio that does not depend on the size of the problem but only on ε. On the other side, to prove that a problem is not in APX, we can provide an algorithm that gives a ratio that depends on the size of the problem. Depending on *how* it depends on the size, we have the membership to the classes log-APX and poly-APX.

2.2 Game Theory

At the beginning of the chapter, we said we split the presentation of foundations of Algorithmic Game Theory in two parts. Here, we present the main concepts related to the game-theoretic side of our approach, introducing notions about games and solution concepts.

Game Theory can be thought as a river with two heads, operations research and microeconomics. It has a vary wide range of applications, from medicine to politics, from psychology to computer science. Actually, it is employed by each of us every day without even realizing it: every decision we take to feel better, it is unconsciously taken applying game theory concepts. More formally, according to Roger Myerson, we can state that Game Theory studies mathematical models to represents conflict or cooperation among rational agents that can undertake decisions [85]. The goal of Game Theory is to describe the evolution of a game, prescribing strategies, i.e., how to play, to agents, also called *players*, to maximize their utility, playing according to the rules of the game, which constitutes the *mechanism*.

2.2.1 A Little Bit of History

Game Theory, anticipated by a paper of John Von Neumann in 1928 [117], was officially born in 1944, with the publication of the book *Theory of Games and Economic Behavior* [118], by Von Neumann and Oskar Morgenstern. This book actually deals mainly with cooperative games in which the focus is on the behavior of groups of individuals that have to collaborate. The big step was taken in 1950, when Merrill Flood and Melvin Dresher [13] present the *Prisoner's Dilemma*, formalized by Albert W. Tucker [114] as follows.

Definition 2.34 (Prisoner's Dilemma)**.** *Two members of a gang, say A, B get arrested and put in jail. Each prisoner is put in isolation, without any possibility of communicating with the other. Police lack evidence to convict the pair on the principal charge. Thus, they hope to get both sentenced to a year in prison on a lesser charge. In the meantime, the following offer is made to both prisoners.*

Each convict has the chance to betray the other, confessing they committed the crime together, or cooperate with her partner by defecting. Specifically:

- *If A, B betray each other, they have to serve two years in jail;*
- *If A betrays B but B defects, A will be set free since she cooperated*

with justice but B will be condemned to three years in prison (and vice versa);

- *If they both defect, they will stay in jail for one year, being sentenced for a lesser crime.*

The prisoners can reward or punish their partner only by means of their choice, which will be reflected in the verdict: the decision they take now will not affect their reputation in the future.

In the same years, John Nash was developing what will be called *Nash Equilibrium* [87], a new way of computing the players' strategies in non-cooperative games, where the players are rational and selfish, competing against each other to maximize their wellness.

In 1965 Reinhard Selten introduced the concept of *Sub-game Perfect Equilibrium* [105] while John Harsanyi in 1967 introduces *Bayesian Games*, inserting the possibility of having incomplete information in games [64].

Other fundamental contributions are developed by Thomas Schelling, who developed the *Evolutionary Game Theory* [102] and by Robert Aumann, who studies *Repeated Games* and *Correlated Equilibrium* [12].

More recently, a lot of efforts have been done in Mechanism Design, a.k.a. reverse game theory. In fact, the goal here is to design the mechanism underneath the game that will be played by the agents. Some of the most important scientists who contributed developing it are Leonid Hurwicz [66], Eric Maskin [44] and Roger Myerson [84].

In conclusion, we notice that, starting from John Nash, all the scientists mentioned above have received a Nobel Prize in Economics for their contributions to Game Theory, *de facto* stating the importance of such research field, which is far beyond pure mathematics.

2.2.2 Playing a Game

We now formally introduce the main elements of Game Theory. First, we describe general concepts related to games and solution concepts, then we focus on the game models actually employed in the present work.

Definition 2.35 (Game)**.** *A* game *is a mathematical model, characterized by the following elements:*

- $N = 1, 2, ..., n$*, the set of players, where* $1, 2, ..., n$ *are the players. In the following, we use* i *to denote the* i*-th player and* $-i$ *to indicate all the players except* i*;*

- $A = \{A_1, A_2, ..., A_n\}$, *the set of possible actions, where* A_i *denotes the actions that can be undertaken by player* i*;*
- X*, the set of possible outcomes of the game;*
- $f : A_1 \times A_2 \times ... \times A_n \to X$*, the outcome function, which associate a unique outcome to each actions' profile, i.e., a sequence of actions performed by each player;*
- U_i*, player* i*'s utility. In the literature, this is also called payoff (hereafter, the two terms will be used with no distinction);*
- $u_i : X \to \mathbb{R}$*,* i*'s utility function, associating a value* $v \in \mathbb{R}$ *to each possible outcome of the game.*

We recall that, in a game, finding a solution corresponds to provide each player with the best actions they should play to maximize their utility, according to some principle.

Moreover, being our focus on non-cooperative games, the following assumptions on the players' behavior hold:

- Each player is *selfish*: this property says that rach player has the only goal to maximize her own utility, without taking into account the benefits other players could get if she would play differently;
- Every player is *rational*: rationality for all the players means that they can get the same conclusions given that they are provided with the same information;
- Every player has a preference ordering on the possible outcomes of the game: each player knows her favorite outcome and plays in order to see it realized. Utilities that players have on outcomes coherently reflect such ordering.

We now turn to the two main models we have to express games.

2.2.3 Normal Form Games

A normal form game is defined by the following tuple: (N, A, X, f, u_i).

These games are usually represented by means of matrices, and thus they are well suited to represent situations in which the agents take their actions contemporarily. With just two players, say $1, 2$, we can have an explicit representation, putting the actions of 1 on the rows and the actions of 2 on the columns. Even if it seems counterintuitive, the cells of the matrix do not contain the outcomes of the game, but the utilities the players would get when that outcome is realized.

Example 8. *Let us consider the very popular game* Rock-Paper-Scissors. *According to Definition 2.35, we can formalize it as follows.*

- $N = \{1, 2\}$*;*
- $A = \{A_1, A_2\}, A_1 = A_2 = \{R, P, S\}$*, which stay for* Rock, Paper, Scissors*;*
- $X = \{W_1, W_2, T\}$*, which means* player 1 wins, player 2 wins, tie*;*
- f*: it receives the two actions of the players and outputs the outcome, e.g., given* (R, P)*, it outputs* W_2*;*
- $u_i(x) = \begin{cases} 1 & x = W_i \\ -1 & x = W_{-i}. \\ 0 & x = T \end{cases}$

We can now fill the bimatrix representing the game:

		2		
		R	P	S
1	R	$(0, 0)$	$(-1, 1)$	$(1, -1)$
	P	$(1, -1)$	$(0, 0)$	$(-1, 1)$
	S	$(-1, 1)$	$(1, -1)$	$(0, 0)$

Table 2.2: *Bimatrix representing the game Rock-Paper-Scissors.*

Given that we know how to use this formalism, let us report the matrix of the prisoner's dilemma to understand why that game has a counterintuitive way of being played.

Example 9. *Let* A, B *be the two prisoners. We recall that each player can confess* C*, betraying her partner, of defect* D*. We also report the outcomes of the game.*

- *If* A, B *betray each other, they have to serve two years in jail;*
- *If* A *betrays* B *but* B *defects,* A *will be set free since she cooperated with justice but* B *will be condemned to three years in prison (and vice versa);*
- *If they both defect, they will stay on jail for one year, being sentenced for a lesser crime.*

The game matrix is the following.

		B	
		C	D
A	C	$(2,2)$	$(0,3)$v
	D	$(3,0)$	$(1,1)$

Table 2.3: *Bimatrix of the Prisoner's Dilemma.*

Without actually making any computation, we can guess how the game develops: no prisoner will trust her partner, confessing the crime and getting to spend two years in jail since, trusting her partner and then being betrayed, they could have been sentenced three years.

Knowing how to represent a game, we can now focus on how to solve it, namely computing the strategies the player should take.

Definition 2.36 (Game strategy)**.** *A* mixed *strategy for player i, denoted as x_i is a probability distribution over the actions of player i such that $\sum_{j=1}^{|A_i|} x_i(a_j) = 1$ and $x_i(a_j) \geq 0, \forall a_j$, where $x_i(a_j)$ is the probability with which action a_j is played by player i. If a single action is played with positive probability, the corresponding strategy is called* pure*.*

Now, we turn to the second game model we will use for our study.

2.2.4 Extensive Form Games

An extensive form game is represented by means of a tree, where each node corresponds to an information set, in which a player must take a decision and the branches flourishing outside the node are the actions the player can choose amongst. Thus, each level of the tree corresponds to a decision taken by a single player. There is a leaf node for each outcome of the game and, once such nodes are reached, utilities are distributed to the players.

Example 10 (Extensive form game)**.** *We report a tree representing 2-player game. L_i, R_i are actions of player 1 while with l_i, r_i we denote actions for player 2. Decision nodes are denoted as $x.y$, where $x.y$ refers to the y-th decision node of player $x \in \{1, 2\}$.*

We indicate with $U_k, k \in \mathbb{N}$, the utilities of the players, e.g., $U_1 = (\Upsilon_{1,1}, \Upsilon_{1,2})$, where $\Upsilon_{k,x}$ being the utility of the k-th leaf node of player x.

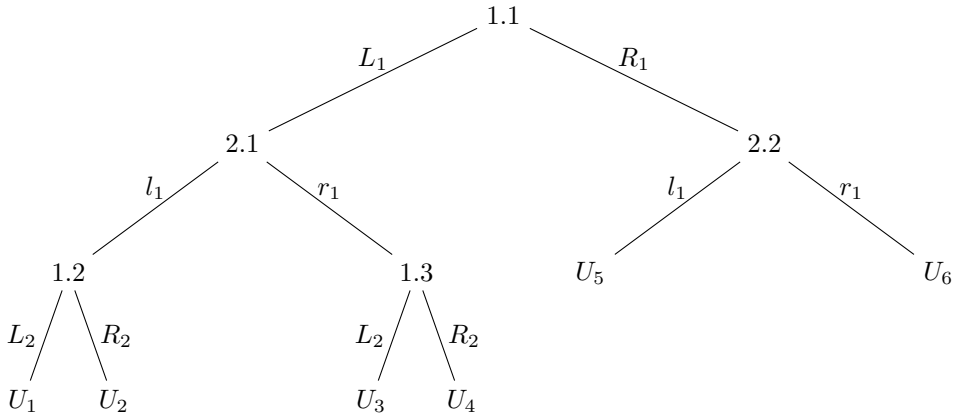


Figure 2.1: *Example of extensive form game.*

Solving Extensive Form Games

The main technique to solve extensive form games is called *Backward Induction*.

The principle is the following: starting from the leaves, we go up level by level in the tree, each time selecting the best action for the player put in the above decision node, from which the branches have been originated.

Example 3 (continue) *Let us consider again the following tree.*

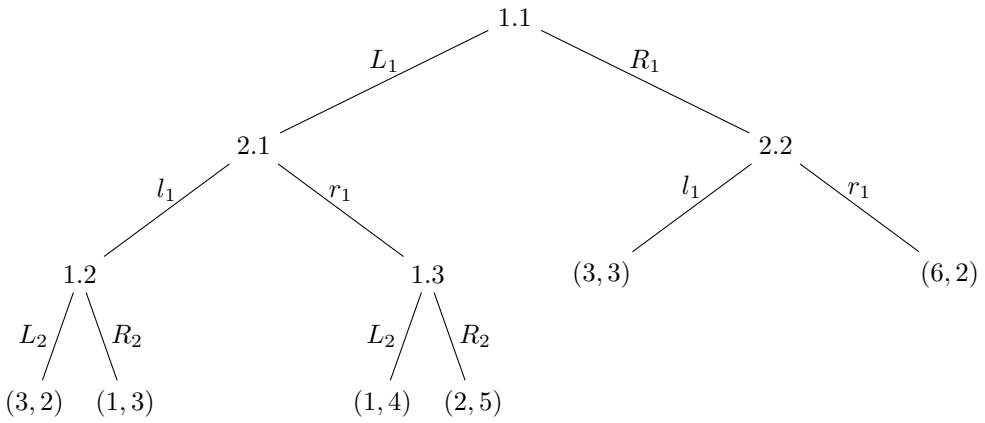


Figure 2.2: *Extensive form tree representation.*

Applying backward induction, we obatin the following results:

- *at node* 1.2*, player* 1 *undertakes* L_2*, getting a utility equal to* 3 *instead of* 1 *provided by action* R_2;
- *at node* 1.3*, player* 1 *undertakes* R_2*, getting a utility equal to* 2 *instead of* 1 *provided by action* L_2.

We can simplify our tree, obtaining the following.

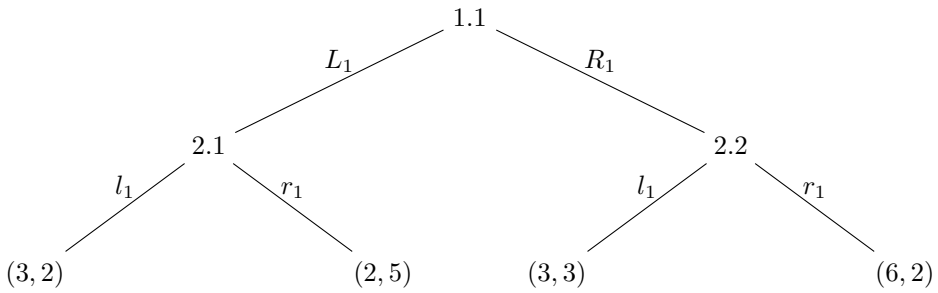


Figure 2.3: *Extensive form representation, simplified tree.*

Now, we observe that:

- *at node* 2.1*, player* 2 *undertakes* r_1*, getting a utility of* 5 *instead of* 2*, provided by* l_2;
- *at node* 2.2*, player* 2 *undertakes* l_1*, getting a utility of* 3 *instead of* 2*, provided by action* r_1.

The updated tree is the following.

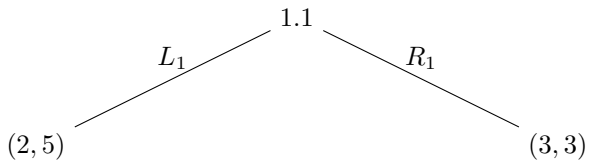


Figure 2.4: *Extensive form representation, full simplified tree.*

Now, player 1 *undertakes* R_1*, getting a utility of* 3 *instead of* 2 *provided by action* L_1*.*

Applying such technique, we are able to know how the game develops at each stage and thus also which will be the outcome and the payoffs. We observe that backward induction can be effectively applied since each player is rational and rationality is common assumption, namely each player knows each other player is rational.

While this form is very explicative, it lacks from the computational perspective, requiring a lot of information to be stored. This is why usually extensive form games are put in other forms to be solved.

In general, there three main forms in which an extensive form game is put to be solved.

- Normal form [118]: discussed above, it is the most known form for solving games. Unfortunately, the size of such game is exponential w.r.t. the size of the tree and so there is no gain in terms of how fast we can solve the game.
- Agent form [106]: introduced by Selten, it consists of considering a different player for each node of the tree while adopting the utilities of the player whose node has been taken. This representation is a step ahead, being polynomial w.r.t. the size of the tree and the number of players.
- Sequence form [120]: formalized by von Stengel, such form defines actions as sequences adding further constraints on the computation of the strategies. Such representation is linear w.r.t. the size of the tree.

2.2.5 Bayesian Games

Bayesian games are characterized by the fact that players do not have complete information about other players, e.g., they do not know their utility functions. We denote with $\Theta = \{\theta_1, \dots, \theta_t\}$ the set of t types θ_i that can be assumed by some player, and with ω the probability that such type is taken.

Example 11. *Let us consider the very well-known battle of the sexes. The interaction we want to model is the following. Two people, say player* $1, 2$*, want to go to a classical music concert but* 1 *prefers Bach while* 2 *prefers Stravinskij. However, they both prefer going together to the concert rather than going alone to listen to their favorite composer.*

Let us denote with B, S *the two actions. We can represent the game in normal form as follows.*

		2	
		B	S
1	B	$(2,1)$	$(0,0)$
	S	$(0,0)$	$(1,2)$

Table 2.4: *Bimatrix for the battle of the sexes game.*

Now, let make this game Bayesian. Let us assume that 1 *and* 2 *do not know each other, but* 1 *asks* 2 *out: being their first date,* 1 *do not know* 2 *preferences. This situation can be modeled introducing two typer for player* 2*,* $\theta_{2.1}$*, who prefers Bach, and* $\theta_{2.2}$*, who prefers Stravinskij, with* $\omega_{2.1}$ *probability of* 2 *being of type* $\theta_{2.1}$ *and probability* $\omega_{2.2}$ *of being of type* $\theta_{2.2}$*. Thus, we have the following representation.*

		$\theta_{2.1}$	
		B	S
1	B	$(2,2)$	$(0,0)$
	S	$(0,1)$	$(1,0)$

		$\theta_{2.2}$	
		B	S
1	B	$(1,0)$	$(0,1)$
	S	$(0,0)$	$(2,2)$

Table 2.5: *Bimatrices of the battle of the sexes game depending on the type of player* 2*.*

By the way, the best form to look at Bayesian game is extensive form, putting *Nature* as the root of the tree, denoted as $\mathcal{N}$, which stochastically decides the types of the players.

Example 4 (continue) *The tree corresponding to the Bayesian version of the battle of the sexes is the following.*

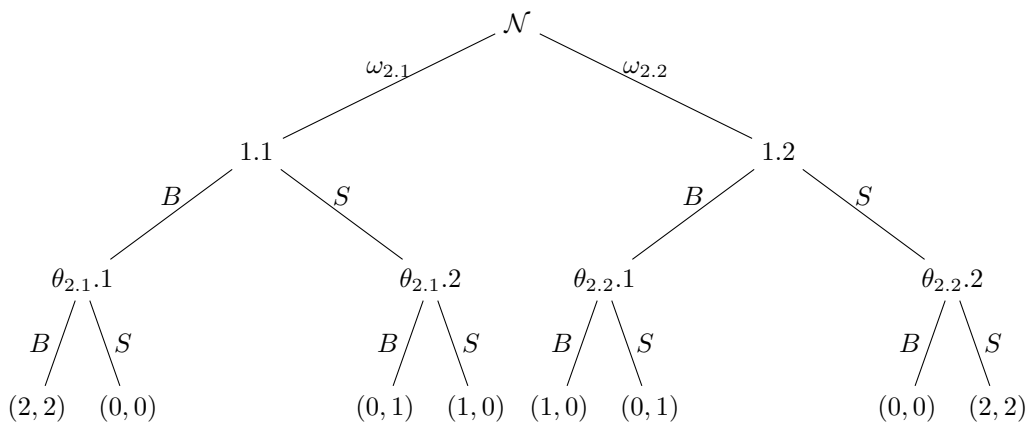


Figure 2.5: *Extensive form of the Bayesian version of the battle of the sexes game.*

Now that we have the fundamental concepts of Game Theory, we turn to the Algorithmic part, introducing the main notions of Computational Complexity.

2.3 Solution Concepts

Given a game, there exist different solution concepts that can be adopted, depending on the model we are working on. Moreover, different solutions may require different time to be computed. As we know, a solution is a set of strategies $x = (x_1, \ldots, x_n)$, one for each $i \in N$.

We focus on three main solution concepts, a.k.a., equilibrium concepts, that we will adopt in the different models: Nash Equilibrium, maxmin/minmax strategies, Leader-follower Equilibrium.

2.3.1 Nash Equilibrium

Nash Equilibrium is based on the idea of *best response*, defined as follows.

Definition 2.37 (Best response)**.** *Let $i \in N$ be a player. Her best response BR_i w.r.t. the strategies x_{-i} played by the pther players is defined as:*

$$BR_i(x_{-i}) = \{x_i | \quad x_i = argmax x_i \cdot U_i x_{-i}\}$$
$$s.t.$$
$$x_i \geq 0$$
$$\mathbb{1} \cdot x_i = 1.$$

Informally, the best response is the best action a player can undertake given what the other players will do. We can now define the Nash Equilibrium [86].

Definition 2.38 (Nash Equilibrium)**.** *A strategy profile $x*$ is a Nash Equilibrium (NE) if and only if:*

$$\forall i, \quad x_i^* \in BR_i(x_{-i}^*).$$

In an NE, no player has unilateral incentive to deviate and so each player plays the action prescribed by the equilibrium being aware the other players will do the same.

From a computational perspective, finding an NE is not in FNP unless NP=coNP. For the sake of completeness, we report two important theorems on non-cooperative games.

Theorem 2.10. *Every game played by a finite number of players, each with a finite number of actions, has at least a mixed strategies NE.*

Corollary 2.11. *If a game admits at least an NE, then it admits an odd number of NE.*

The Nash equilibrium has been successfully employed in many different applications; for this reason several variations have been defined. It is worth introducing one of them, namely the ϵ-Nash equilibrium. The idea is that players might not care about changing their strategies to a best-response when the utility that they could gain doing so is very small.

Definition 2.39. *Given $\epsilon > 0$, a strategy profile $(s_1, ..., s_n)$ is an* ϵ-Nash equilibrium (ϵ-NE) *if and only if:*

$$\forall i \in N, \forall s'_i \in \Delta(A_i), \quad u_i(s_i, s_{-i}) \geq u_i(s'_i, s_{-i}) - \epsilon$$

This solution concept has many interesting properties. First, it always exists; indeed, every Nash equilibrium is surrounded by a region of ϵ-Nash equilibria, for every $\epsilon > 0$. However, the reverse is not true, which means that there might be ϵ-Nash equilibria that are not close to any Nash equilibrium. Second, finding an ϵ-Nash equilibrium is computationally easier than searching for a Nash one.

2.3.2 Maxmin/minmax Strategies

We now turn to another solution concept, maxmin/minmax strategies. First, we provide the definition for 2-player games, where player i, the maximizer, wants to maximize her own utility, and player $-i$ is the minimizer, willing to minimize the utility of player 1.

Definition 2.40 (Maxmin optimization problem)**.** *The problem of finding a Maxmin strategy of player i against player $-i$ is formulated as:*

$$\begin{aligned}
\arg\max_{\sigma_i} \quad & \min_{\sigma_{-i}} && \sum_{a_i \in A_i} \sum_{a_{-i} \in A_{-i}} [U_i(a_i, a_{-i})\, \sigma_i(a_i)\, \sigma_{-i}(a_{-i})] && \\
& && \sum_{a_{-i} \in A_{-i}} \sigma_{-i}(a_{-i}) && = 1 \\
& && \sigma_{-i}(a_{-i}) && \geq 0 \quad \forall a_{-i} \in A_{-i} \\
\text{s.t.} \quad & \sum_{a_i \in A_i} \sigma_i(a_i) && && = \\
& \sigma_i(a_i) && && \geq 0 \quad \forall a_i \in A_i
\end{aligned}$$

The main result is that a Maxmin strategy can be found by means of linear programming, as showed in the following theorem.

Theorem 2.12 (Maxmin strategy formulation)**.** *The Maxmin strategy and the corresponding value of player i against player $-i$ can be found by solving the following Linear Program (LP):*

$$
\begin{array}{lll}
\arg\max\limits_{\sigma_i, v_i} & v_i & \\
\textit{s.t.} & v_i - \sum\limits_{a_i \in A_i} \Big[U_i(a_i, a_{-i})\, \sigma_i(a_i) \Big] & \leq 0 \quad \forall a_{-i} \in A_{-i} \\
& \sum\limits_{a_i \in A_i} \sigma_i(a_i) & = 1 \\
& \sigma_i(a_i) & \geq 0 \quad \forall a_i \in A_i
\end{array}
$$

Example 12. *Let us compute the maxmin/minmax strategies for the prisoner's dilemma. The normal form of the game is the following.*

		B	
		C	D
A	C	$(2,2)$	$(0,3)$
	D	$(3,0)$	$(1,1)$

To solve the game, we observe that prisoner A, the maximizer, do not trust her partner, prisoner B. Thus, we extract from each row of the matrix the minimum value, corresponding to the choice that prisoner B would make and, among such outcomes, we select the best.

We recall that the numerical values reported in the matrix correspond to years that should be spent in jail. So, from the first row, a 2 is extracted a 3 from the second one, thus resulting in two years to spend in prison. We observe that this solution coincides with the outcome (C, C) we found previously.

A similar rationale holds for prisoner B.

Now, we generalize what expressed above, focusing on the case in which there are multiple max players and only one min player. We underline that this scenario makes sense only when the utility functions of all the max players are the same, otherwise it would not be clear which objective function the max players should maximize. Therefore, in this case, the max players are forming a team and searching for the maxmin strategy is equivalent to searching for a Team-Maxmin strategy. Also here, we can distinguish the case in which the team plays in correlated strategies from the case in which the team plays in mixed strategies. We initially focus on the first case.

Definition 2.41 (Maxmin optimization problem (with correlated min players))**.** *The problem of finding a Maxmin strategy of a team of $n-1$ players, playing in correlated fashion, against player i is formulated as:*

$$\begin{array}{lll} \arg\max_{\boldsymbol{\sigma}_{-i}} \min_{\sigma_i} \sum_{\mathbf{a}_{-i}\in A_{-i}} \sum_{a_i\in A_i} \left[U_{-i}(a_i,\mathbf{a}_{-i})\,\sigma_i(a_i)\,\boldsymbol{\sigma}_{-i}(\mathbf{a}_{-i})\right] & & \\ \quad s.t. \sum_{a_i\in A_i} \sigma_i(a_i) & = 1 & \\ \quad \sigma_i(a_i) & \geq 0 & \forall a_i \in A_i \\ s.t. & & \end{array}$$

We can easily derive the mathematical program solving the above problem.

Theorem 2.13 (Maxmin strategy formulation (with correlated min players))**.** *The Maxmin strategy and the corresponding value of a team of $n-1$ players, playing in correlated fashion, against player i can be found by solving the following Linear Program:*

$$\begin{array}{lll} \arg\max_{\boldsymbol{\sigma}_{-i},v_{-i}} & v_{-i} & \\ \quad s.t. & v_{-i} - \sum_{\mathbf{a}_{-i}\in A_{-i}} \left[U_{-i}(a_i,\mathbf{a}_{-i})\,\boldsymbol{\sigma}_{-i}(\mathbf{a}_{-i})\right] \leq 0 & \forall \mathbf{a}_{-i} \in A_{-i} \\ & \sum_{\mathbf{a}_{-i}\in A_{-i}} \boldsymbol{\sigma}_{-i}(\mathbf{a}_{-i}) = 1 & \\ & \boldsymbol{\sigma}_{-i}(\mathbf{a}_{-i}) \geq 0 & \forall \mathbf{a}_{-i} \in A_{-i} \end{array}$$

Let us consider the case in which the max players play mixed strategies and therefore they cannot correlate.

Definition 2.42 (Maxmin optimization problem (with non-correlated min players))**.** *The problem of finding a Maxmin strategy of a team of $n-1$ players, playing in mixed strategies, against player i is formulated as:*

$$\begin{array}{lll} \arg\max_{\boldsymbol{\sigma}_{-i}} \min_{\sigma_i} \sum_{a_i\in A_i} \sum_{\mathbf{a}_{-i}\in A_{-i}} \left[U_{-i}(a_i,\mathbf{a}_{-i}) \prod_{j\in N} \sigma_j(a_j)\right] & & \\ \quad s.t. \sum_{a_i\in A_i} \sigma_i(a_i) & = 1 & \\ \quad \sigma_i(a_i) & \geq 0 & \forall a_i \in A_i \\ s.t. \sum_{a_j\in A_j} \sigma_j(a_j) & = 1 & \forall j \in N\setminus\{i\} \\ \quad \sigma_j(a_j) & \geq 0 & \forall j \in N\setminus\{i\}, \\ & & \forall a_j \in A_j \end{array}$$

We can easily derive the mathematical program solving the above problem.

Theorem 2.14 (Maxmin strategy formulation (with correlated min players))**.** *The Maxmin strategy and the corresponding value of a team of $n-1$ players,*

playing in mixed strategies, against player i can be found by solving the following Non-Linear Program:

$$\begin{aligned}
\arg \max_{\boldsymbol{\sigma}_{-i}, v_{-i}} \quad & v_{-i} & & \\
s.t. \quad & v_{-i} - \sum_{\mathbf{a}_{-i} \in A_{-i}} \left[U_{-i}(a_i, \mathbf{a}_{-i}) \prod_{j \in N \setminus \{i\}} \sigma_j(a_j) \right] & \leq 0 & \quad \forall \mathbf{a}_{-i} \in A_{-i} \\
& \sum_{a_j \in A_j} \sigma_j(a_j) & = 1 & \quad \forall j \in N \setminus \{i\} \\
& \sigma_j(a_j) & \geq 0 & \quad \forall j \in N \setminus \{i\}, \forall a_j \in A_j
\end{aligned}$$

2.3.3 Leader-follower Equilibrium

We now introduce the leader-follower equilibrium, the main solution concept adopted throughout the work. This equilibrium has been introduced for a particular class of games, called Stackelberg Games [119].

The rationale behind the Leader-follower equilibrium is that a player can play as a leader committing (announcing) her strategy, while another player can first observe the commitment (or announcement) and then play. Remarkably, the commitment may be mixed.

The game develops as follows: first, the leader publicly commits to a strategy, then the follower, after having observed the leader's commitment, undertakes her action. This situation introduces a strong asymmetry in the game, giving the leader a stronger position than that when the commitment is not possible. The conditions for a Leader-follower equilibrium can be formulated as follows.

Definition 2.43 (Leader-follower (Stackelberg) equilibrium (single leader and single follower))**.** *Strategy profile (σ_l, σ_f) is a Leader-follower equilibrium where player l is the leader and player f is the follower if:*

$$\begin{aligned}
\sigma_l \in \arg \quad & \max_{\sigma'_l \in \Delta(A_l)} \mathbb{E}_{\mathbf{a} \sim (\sigma'_l, \sigma_f)}[U_l(\mathbf{a})] \\
\text{s.t.} \quad & \sigma_f \in \arg \max_{\sigma'_f \in \Delta(A_f)} \mathbb{E}_{\mathbf{a} \sim (\sigma'_l, \sigma'_f)}[U_f(\mathbf{a})]
\end{aligned}$$

Observation 5. *The optimization problem is actually a bi-level optimization problem, in which the leader maximizes her expected utility given that, once the strategy of the leader is fixed, the follower maximizes her expected utility.*

Given its nature, such equilibrium is usually adopted for applicative settings, especially in economics and physical security. In fact, it was originally formulated for situations in which a company that aims at planning the production of a product and has to decide when and how it is convenient to enter the market when another company is already the leader in such a market. Nowadays, it is more exploited in game models for physical security,

where the goal of the leader, a.k.a., the *Defender*, is to protect some valuable targets from the follower, a.k.a., the *Attacker*, aiming at compromising them. In this thesis, such equilibrium is adopted in these sense for security scenarios.

2.3.4 Zero-sum Games

We have seen that maxmin/minmax strategies chracterize situations in which the goal of one player is hurting the other. If the goal of one player is exactly the opposite of the other, we are dealing with a *zero-sum game* [118], i.e., a game in which the sum of the utilities of the players is equal to 0[2].

Example 13. *Rock-Paper-Scissors is a classical zero-sum game. In fact, its game matrix is the following*

		2		
		R	P	S
1	R	$(0,0)$	$(-1,1)$	$(1,-1)$
	P	$(1,-1)$	$(0,0)$	$(-1,1)$
	S	$(-1,1)$	$(1,-1)$	$(0,0)$

where we observe that the sum of the utilities for each cell is equal to zero.

Now, we report some useful results for zero-sum games [110, 122].

Theorem 2.15. *Every zero-sum finite game admits at least an NE in mixed strategies.*

Theorem 2.16. *In a zero-sum game, NE, maxmin/minmax strategies and leader-follower equilibrium coincide.*

Theorem 2.17. *In non zero-sum games, the utility obtained by the leader from a Leader-follower equilibrium is greater than or equal to the utility provided by an NE.*

Now, we change topic, introducing the fundamental concepts of Online Learning, which will be exploited to study and develop a learning security game.

[2]Zero-sum games are a special class of the most general class of *constant sum games*, in which the sum of the utilties of the players is equal to a constant value.

2.4 Online Learning

Learning means taking decisions in time with the possibility of better knowing the problem we are facing after having taken each decision. More formally, we can say that learning is a *sequential decision making* problem, i.e., undertaking decisions based on a past history of observations, with the goal of minimizing some function, called *loss function*. When dealing with these situations, we usually do not know the full structure of the problem and thus we are not able to solve it exactly. To do this, we may resort to some *Online Learning* technique. In this section we introduce two frameworks: Predictions with Expert Advice (for which we adopt the notation of [37]) and Multi-Armed Bandit. Since we are not able to specify the states of the system, and we cannot use Reinforcement Learning tools, but we implicitly assume that the system has only one state, and, at each time instant, we are asked to choose among the same set of actions, based on what happened in the past.

2.4.1 Prediction with Expert Advice

The Prediction with Expert Advice is based on the following protocol: the decision maker is a forecaster whose goal is to predict an unknown sequence $y_1, y_2, \ldots$ of elements of an outcome space $\mathcal{Y}$. The forecaster's predictions $\hat{p}_1, \hat{p}_2, \ldots$ belong to a decision space $\mathcal{D}$, which we assume to be a convex subset of a vector space. In general $\mathcal{D}$ may be different from $\mathcal{Y}$. The forecaster computes his predictions in a sequential fashion, and his predictive performance is compared to that of a set of reference forecasters that we call *experts*. More precisely at each time t:

- the forecaster has access to the set $\{f_{E,t} : E \in \mathcal{E}\}$ of expert predictions $f_{E,t} \in \mathcal{D}$, where $\mathcal{E}$ is a fixed set of indices for the experts;
- on the basis of the experts' predictions, the forecaster computes his own guess $\hat{p}_t$ for the next outcome y_t;
- after $\hat{p}_t$ is computed, the true outcome y_t is revealed.

The predictions of forecaster and experts are scored using a non-negative *loss function* $l : \mathcal{D} \times \mathcal{Y} \to \mathbb{R}$. This prediction protocol can be naturally interpreted as a *repeated game* between the *forecaster* that makes some guesse $\hat{p}_t$, and the *environment*, which chooses the expert advice $\{f_{E,t} : E \in \mathcal{E}\}$ and sets the true outcomes y_t, thus determining a certain loss for the first player.

The forecaster's goal is to keep as small as possible the *cumulative regret*, a.k.a., the loss, with respect to each expert.

Definition 2.44 (Cumulative Regret)**.** *The* cumulative regret *(or simply* regret*) for expert E is equal to:*

$$R_{E,n} = \sum_{t=1}^{n} (l(\hat{p}_t, y_t) - l(f_{E,t}, y_t)) = \hat{L}_n - L_{E,n} \tag{2.1}$$

where $\hat{L}_n = \sum_{n_t=1}^{n} l(\hat{p}_t, y_t)$ denotes the forecaster's cumulative loss while $L_{E,n} = \sum_{t=1} l(f_{E,t}, y_t)$ represents the cumulative loss of expert E. Hence, $R_{E,n}$ is the difference between the forecaster's total loss and that of expert E after n prediction rounds. We also define the instantaneous regret with respect to expert E at time t as follows:

$$r_{E,t} = l(\hat{p}_t, y_t) - l(f_{E,t}, y_t) \tag{2.2}$$

Thus, $R_{E,n} = \sum_{t=1}^{n} r_{E,t}$. One may think of $r_{E,t}$ as the regret the forecaster feels of not having listened to the advice of expert E right after the t^{th} outcome y_t has been revealed.

Follow the Leader

The simplest forecasting strategy, namely *fictitious play*, consists in choosing, at time t, an expert that minimizes the cumulative loss over the past $t-1$ time instances. In other words, the forecaster always follows the expert that cumulated the smallest loss up to that time instant.

Algorithm 1 `FollowTheLeader`

1: **for all** $t \in \{1, \ldots, n\}$ **do**
2: Compute $E = \arg\min_{E_k \in \mathcal{E}} \sum_{i=1}^{t-1} l(f_{E_k,i}, y_i)$
3: Play expert $\hat{p}_{k_t} = f_{E,t}$
4: Observe y_t

Under some conditions (e.g. square loss function or convex losses with constant experts, see [37]), the regret of this algorithm grows as slowly as $O(\ln(n))$. However, in general, it can suffer in some case of linear regret. For example, consider $N = 2$ actions such that the sequence of losses $l(1, y_t)$ of the first is $(\frac{1}{2}, 0, 1, 0, 1, \ldots)$ while the values of $l(2, y_t)$ are $(\frac{1}{2}, 1, 0, 1, 0 \ldots)$. Then $L_{i,n}$ is about $\frac{n}{2}$ for both, but follow the Leader suffers a loss of $O(n)$.

Follow the Perturbed Leader

As pointed out in [62], with a simple modification we can overcome the previous problem. In fact, we can add a small random perturbation to the

cumulative losses, following the *perturbed* leader. Formally, let $Z_1, Z_2 \ldots$ be independent, identically distributed random N-vectors with components $Z_{i,t} = (i = 1, \ldots, K)$, where $K = |\mathcal{E}|$. Let us also assume that Z_t has a uniform distribution. At time t, the *follow-the-perturbed-leader* forecaster selects an action:

$$I_t = \arg\min_{i=1,\ldots,n}(L_{i,t-1} + Z_{i,t}) \tag{2.3}$$

Algorithm 2 `FollowThePetrurbedLeader (FPL)`

1: **for all** $t \in \{1, \ldots, n\}$ **do**
2: **for all** $E_k \in \mathcal{E}$ **do**
3: Sample $Z_{k,t} \sim U(0, \sqrt{nK})$
4: Compute $E = \arg\min_{E_k \in \mathcal{E}} \sum_{i=1}^{t-1}[l(f_{E_k,i}, y_i)] + Z_{k,t}$
5: Play expert $\hat{p}_{k_t} = f_{E,t}$
6: Observe y_t

If $Z_{i,t}$ is uniformly distributed on $[0, \sqrt{nK}]$ then the actual regret, with probability at least $1 - \delta$, satisfies [37]:

$$R_n \leq 2\sqrt{nK} + \sqrt{\frac{n}{2}\ln(\frac{1}{\delta})} \tag{2.4}$$

2.4.2 Multi-Armed Bandit (MAB)

Here, we turn to another approach, whose rationale is the opposite w.r.t. Prediction with Expert Advice. The Multi-Armed Bandit name derives from one of the examples used to describe it informally, after the first formalization in [99]: a gambler has to play with some slot machines (which were once called *one-armed-bandit*) and wants to minimize his cumulative loss. Given the rewards she got playing different "arms", she has to decide the next slot machine to play with, i.e., the next arm to be pull. More formally, in the multi-armed bandit setting, we define a finite set of possible choices, $\mathcal{D}$, where the possible choices are called *arms*. At each turn, the player selects arm $\hat{p}_t$ from $\mathcal{D}$ and observes the associated loss l_t. The objective is to minimize the sum of the collected losses over the horizon H, i.e. the number of rounds that have to be played after the beginning. The regret R after n rounds is defined as the expected difference between the sum of the collected losses and the loss sum associated with an optimal strategy:

$$R = \sum_{t=1}^{n} l_t - nl^* \tag{2.5}$$

where * is the minimal loss mean and l_t is the loss at time t. This problem can also be seen as a one-state *Markov Decision Process* or as a modified version of the Prediction with Expert Advice Problem, where the latter differs from the MAB on the received feedback. In fact, in the expert context, we are always informed of which expert (arm) is the best choice, while in the MAB setting we receive a loss, but we have no means of knowing what would have been the best arm to select at that round. In other words, we can call the expert feedback a *complete feedback* and the MAB one a *partial feedback*, because it gives us information only on the *pulled* arm, but not on the others. Unfortunately, all MAB algorithms are subjected to a lower bound:

Theorem 2.18. *Given a MAB stochastic problem any algorithm satisfies:*

$$\lim_{t\to\infty} L_t \geq \log t \sum_{i|\Delta_i} \frac{\Delta_i}{KL(\mathcal{R}_i, \mathcal{R}^*)} \tag{2.6}$$

where $\mathcal{R}_i$ represents the distribution of the rewards of arm i, $\Delta_i = \mathbb{E}[\mathcal{R}^*] - \mathbb{E}[\mathcal{R}_i]$, and $KL(\mathcal{R}_i, \mathcal{R}^*)$ is the Kullback-Leibler *divergence between the two distributions $\mathcal{R}_i$ and $\mathcal{R}^*$.*

Upper Confidence Bound - UCB1

Algorithm 3 `UCB1`

1: **for all** $t \in \{1, \dots, n\}$ **do**
2: **for all** $p_k \in \mathcal{D}$ **do**
3: Compute $\hat{L}_t = \sum_{i=1}^{t} l_{k,i} \mathbb{I}[p_k = p_{k_i}]$
4: Compute $B_{k,t}$
5: Play arm $\hat{p}_{k_t} = \arg\min_{p_k \in \mathcal{D}} LB_{k,t}$

While in the Expert's setting we do not need exploration, in MAB's one we cannot resort to algorithm like `FollowTheLeader`, because we are compelled to explore also the other choices. A way to do it is considering a *lower bound* $LB_{k,t}$ over the expected loss $L_{k,t}$ such that, with *high probability* $LB_{k,t} = \hat{L}_{k,t} + B_{k,t} \leq L_{k,t}$, where $B_{k,t}$ is a bound which depends on how much information we have on an arm, which is embodied by the number of times we have pulled arm k so far. Specifically, we want to have a large bound if we have chosen the arm a few times, and a tight bound if the arm has been largely pulled. To set this bound we resort to a concentration inequality called Hoeffding Bound:

Definition 2.45 (Hoeffding Inequality Bound)**.** *Let $X_1, \dots, X_t$ be i.i.d random variables with support in $[0, 1]$ and identical mean $\mathbb{E}[X_i] = X$ and let $\overline{X} = \sum_{i=1}^{t} X_i$ be the sample mean. Then:*

$$\mathbb{P}(X < \overline{X} + u) \leq e^{(-2tu^2)})$$

This inequality can be applied to each arm:

$$\mathbb{P}(L_{k,t} < \hat{L}_{k,t} + B_{k,t}) \leq e^{(-2tB_{k,t}^2)}) \tag{2.7}$$

Picking a probability p that the real value exceeds the bound $e^{(-2tu^2)} = p$ we can solve this equation to find $B_{k,t} = -\sqrt{\frac{\log(p)}{N_{t,k}}}$, where $N_{k,t}$ is the number of times we pulled arms k. UCB1 algorithm employs a typical *frequentist* approach to the problem, in fact does not assume any underlying distribution for some arms' parameters. This algorithm has an upper bound:

Theorem 2.19. *At time T, the expected total regret of UCB1 algorithm applied to a stochastic MAB problem is:*

$$L_T \leq 8 \log T \sum_{i|\Delta_i > 0} \frac{1}{\Delta_i} + \left(1 + \frac{\pi^2}{3}\right) \sum_{i|\Delta_i > 0} \Delta_i \tag{2.8}$$

Thompson Sampling

The other option w.r.t. a frequentist approach is a *Bayesian* one, adopted by Thompson Sampling. The algorithm associates to each arm a Bernoulli distribution, with a relative Beta prior. Every time we pull an arm, we observe the loss l_t, and, in case of success ($l_t = 0$) we update the α parameter of its Beta prior with $\alpha_{k,t+1} = \alpha_{k,t} + 1$, while in case of failure ($l_t = 1$) we update the parameter β with $\beta_{k,t+1} = \beta_{k,t} + 1$. Then, at each turn, we sample from each Beta prior, and choose the arm which has the maximum sampled probability of success.

In principle, this algorithm should be used only with Bernoulli rewards (losses), but here we present an adjustment (citation needed) that make it work with every loss domain. We take the normalized observed loss $\bar{l}_t$ and then we interpret $1 - \bar{l}_1$ as the probability of success of a Bernoulli and after we have established this value we sampled from it, to classify the observation as a success or a failure.

There exists an upper bound for this algorithm:

Algorithm 4 `ThompsonSampling`

for all $p_k \in \mathcal{D}$ **do**
    Initialize $\alpha_k = 1$, $\beta_k = 1$
for all $t \in \{1, \dots, n\}$ **do**
    for all $p_k \in \mathcal{D}$ **do**
        Sample $\hat{l}_{k,t} = -\hat{r}_{k,t} \sim Beta(\alpha_k, \beta_k)$
    Play arm $\hat{p}_{k_t} = \arg\min_{p_k \in \mathcal{D}} \hat{l}_{k,t}$
    Observe l_t
    Sample $s \sim Bernoulli(p = 1 - \bar{l}_t)$
    if $s = 1$ **then**
        $\alpha_{k_t} = \alpha_{k_t} + 1$
    else
        $\beta_{k_t} = \beta_{k_t} + 1$

Theorem 2.20 (Thompson Sampling Upper Bound)**.** *At time T, the expected regret of Thompson Sampling applied to a stochastic MAB problem is:*

$$L_T \leq O\left(\sum_{i|\Delta_i>0} \frac{\Delta_i}{KL(\mathcal{R}_i, \mathcal{R}^*)}(\log T + \log\log T)\right) \tag{2.9}$$

CHAPTER 3

State of the Art

And those tracks may often be helpful to those coming after him in finding their way.

— Robert Baden-Powell

In this work, we will focus on a class of games known as *Security Games*, which model the task of protecting physical environments as a non-cooperative game between a *Defender*, say $\mathcal{D}$, and an *Attacker*, say $\mathcal{A}$. In the present chapter, we provide a survey and discuss the main results in Security Games. First, we introduce Security Games in general (Section 3.1), then we focus on Patrolling Games (Section 3.2) and Learning Games (Section 3.3), both adopted to study the problems tackled in this dissertation.

3.1 Security Games

The birth of the first *Security Game* (SG) is due to John Von Neumann, with the *Hide and Seek Game* [52]. The scenario is the following: a player hides in a place among a finite number of them, and the other player should find her. It is modeled by means of a normal form zero-sum game, since the goals of the two players are exactly one the opposite of the other.

From this simple game, many hints were born, originating many papers in the following years, where some fugitives are escaping from pursues that want to reach them [1]. If the fugitive tries to reach a target, e.g., a vanishing point, while the pursuer has to stop her, we have *Ambush Games* [100] while if the fugitive hides and the pursuers look for her, we have *Search Games* [55]. Finally, if both fugitives and pursues can *move* in the environment, we call them *Infiltration Games* [5].

These studies evolved in research about strategic resource allocation for security, which has been a very prolific domain in the field of algorithmic game theory during the last years. The research in this domain led to the development of what today are commonly called *Security Games*: game-theoretical frameworks for computing resource allocation strategies against adversarial security threats. Security is one of the most important issues every country and every person deals with every day: the protection of airports, ports, banks, monuments and museums, but also containing urban attacks, controlling poaching of endangering species, preventing the diffusion of misinformation and guaranteeing cybersecurity [70].

Customarily, SGs are a mathematical tool to model the protection of infrastructure or open environments as a non-cooperative game between a *Defender* and an *Attacker*. Given the setting, these scenarios take place under a *Stackelberg* (a.k.a. *leader-follower*) paradigm [122], where the Defender (*leader*) commits to a strategy and the Attacker (*follower*) first observes such commitment, then best responds to it. From a computational perspective, as discussed in [39], finding a leader-follower equilibrium is computationally tractable in games with one follower and complete information, while it becomes hard in Bayesian games with different types of Attacker. The availability of such computationally tractable aspects of Security Games led to the development of algorithms capable of scaling up to large problems, making them deployable in the security enforcing systems of several real-world applications.

There have been several applications based on such games, we report some of them. The first one to be deployed is ARMOR, consisting in the strategic placement of checkpoints on the streets leading to the Los Angeles International Airport (LAX) and the management of the patrolling units across the terminals [95, 94]. The authors cast the problem as a Bayesian Stackelberg game, giving the guards the possibility to assign different and appropriate weights to their actions and tune them w.r.t. the types the adversary.

In [113], the problem of scheduling undercover air marshals on U.S. domestic flights has been tackled with the project IRIS. Here, more constraints

have been introduced, since the agents must fly among cities such that the next day they will depart from the same city they landed the day before. Moreover, the agents are scheduled in order to have a list of cities such that the first and last cities are the same, so that they actually fly around following a circle.

In the same year, new models and algorithms were designed to deal with more complex and more realistic instances [72]. The authors proposed a compact way of representing a security game, thus obtaining an exponential improvement both in the running time and the memory space needed by the algorithms to solve such problems.

In 2012, game-theoretic techniques were employed to secure ports, bridges and ferries, giving the U.S. Coast Guard patrols the best tours they should follow to protect such areas [10]. Since patrols have a fixed duration during the day, it was not possible to model each target as a node in a graph, as it is customarily done, because of the size of the problem, having a large number of targets to be considered. The solution was to define patrol areas, i.e., groups of nearby targets, which lead the Coast Guard to redefine the set of defensive activities to be performed. Moreover, this is one of he first deployed projects in which there is an assumption of non perfect rationality w.r.t. the Attacker.

GUARDS is another interesting project, which builds upon ARMOR and IRIS [96]. Starting from the idea of protecting an airport at a national scale, the authors introduced the possibility of dealing with hundreds of heterogeneous security activities to be able to prevent several potential with a system that has been designed for hundreds of end-users.

With TRUSTS [45], the authors designed a security game to schedule city guards to stop fare evasion in Los Angeles Metro. Each day, TRUSTS generates a patrol schedule for a team of inspectors, consisting of a sequence of fare-check operations, alternating between in-station and on-train operations. Each operation indicates specifically where and when a patrol unit should check fares. They used Markov Decision Processes (MDPs) as a compact representation to model each Defender unit's patrol actions.

Starting from these projects, which are still deployed, Security Games have been adopted for many different situations. W.r.t. airport security, new games have been studied to perform an effective screening for threats, both objects and people, before they actually enter the airport (similarly, they could be applied to cargo container or stadium screening) [34, 103]. The challenge is to find a dynamic approach for randomized screening, allowing for more effective use of limited resources while improving the level of security. The authors designed Threat Screening Games (TSGs), where there

is a set of scarce resources to screen and check several individuals or objects. The proposed approach, GATE, applies to Bayesian general-sum TSGs.

The protection of water from toxic materials has been studied in [54]. In some countries, despite government regulations on leather tannery waste emissions, inspection agencies do not have enough resources to control the problem, and tanneries' toxic wastewaters have a destructive impact on surrounding ecosystems and communities. NECTAR has been proposed as the first security game application to generate environmental compliance inspection schedules. Still related to water-life, in [130], the authors proposed a method to protect coral reefs, which are valuable and fragile ecosystems, constantly under threat from human activities, e.g., coral mining. Many countries have built marine protected areas and protect their ecosystems through boat patrol, and efficiently schedule these patrols is a perfect application for security games.

In [60], a new game to study the behavior of the Defender w.r.t. multiple independent adversaries is proposed. In fact, collusion among malicious agents is very common in every domain, from airport to wildlife security. The authors study whether it could be more convenient for a group of Attackers to break up collusion by playing off the self-interest of individual adversaries.

Preventing crimes or terrorist attacks in urban areas is the problem that has been tackled in [131]. Guards must respond very quickly to be able to intercept and catch a potential Attacker on her escaping route, which could depend on time-dependent traffic conditions on transportation networks. The main challenge here consists of the presence of time constraints both on the Defender and the Attacker side.

Very recently, security games have been applied to stop the nuclear smuggling in international container shipping through advanced inspection facilities [124]. Efficiency and efficacy are fundamental for this task, given that there are millions of containers, which should be screened, that are imported to the U.S. This work models the interaction between an inspector and a smuggler by means of a security game, formulating the smuggler's sequential decision behavior as a Markov Decision Process.

All the above works necessarily work with a discrete number of strategies per player, often not explicitly taking into account the underlying topology of the space in which targets are located. However, in many practical security settings, defense resources can be located on a continuous plane, and so defense solutions are improved by placing resources in a space outside of actual targets (e.g., between targets) [57]. To address this limitation, the authors proposed Security Game on a Plane, where targets and defensive

resources are distributed on a 2-dimensional plane, able to protect targets within a certain effective distance.

The papers listed above deal with physical security. Recently, game-theoretic techniques have also been applied to *cyber* security. In [47, 46], the authors study the problem of protecting a network in which an administrator may decide the best security measures to adopt to improve the safety of the network. This is achieved by means of honeypots, i.e., decoy services or hosts, placed by the Defender, while the Attacker chooses the best response as a contingency attack policy.

Still in a cyber security dimension, [104] proposed an approach to investigate whether alerts generated by potential cyber attacks are real attacks or just false positives. Also here, the magnitude of the problem is high, with a number of alerts that is overwhelming w.r.t. the number of analysts that can check the authenticity of such attacks.

It is immediate to observe that Security Games have tons of applications. In the next section, we focus on Patrolling Games, i.e., games in which the Defender controls a mobile resources that is able to catch the Attacker.

3.2 Patrolling Security Games

In this work, we focus on a specific class of security games, called *Patrolling Security Games* (PSGs). These games are modeled as infinite-horizon extensive form games in which the Defender controls one or more *patrollers* moving within an environment, represented as a finite graph. In [22], the authors provide the formulation of a Security Game in which the Defender controls a mobile resource that can move in the environment between adjacent areas, while the Attacker, besides having knowledge of the strategy to which the Defender committed to, can observe the movements of the patrollers at any time and use such information in deciding the most convenient time and target location to attack

When multiple mobile resources are available to the Defender, coordinating them at best is in general a hard task which, besides computational aspects, must also keep into account communication issues [25]. In this work, the authors determined the smallest number of robots needed to patrol a given environment and computed the optimal patrolling strategies along several coordination dimensions, i.e., the strategy of a robot can or cannot depend on the strategies of the other robots and the environment can or cannot be partitioned.

However, the patrolling problem is tractable, even with multiple patrollers, in border security (e.g., linear and cycle graphs), when patrollers have ho-

mogeneous moving and sensing capabilities and all the vertices composing the border share the same features [4]. Scaling this model involved the study of how to compute patrolling strategies in scenarios where the Attacker is allowed to perform multiple attacks [111]. Similarly, coordination strategies among multiple Defenders are investigated in [3].

In [123], the authors study the case in which there is a temporal discount on the targets, i.e., the value of the targets diminishes as time passes by, both for the Attacker and the Defender.

Extensions are discussed in [108], where coordination strategies between Defenders that must execute joint activities are explored, in [56], where a resource can cover multiple targets, and in [2] where attacks can be detected at different stages with different associated utilities.

Patrolling has always been a fundamental research line in robotics: in [7], for the first time, a security game has been explicitly cast in this domain. Protecting sites against intrusions is a topic of increasing importance, and robotic systems for autonomous patrolling have been developed in the last years. However, unpredictable strategies are not always efficient in getting the patroller a large expected utility. Here, exploiting a model of the adversary in a game theoretic framework, the authors provide a method to find the optimal strategies, modeling a given patrolling situation as an extensive-form game. In [23], the model is refined, capturing patroller's augmented sensing capabilities and a possible delay in the intrusion. In [8], the authors conducted realistic experiments by using USARSim [36], with the goal of studying the behavior of the optimal patrolling strategy both in situations that violate its idealized assumptions and in comparison with other patrolling strategies.

An important contribution comes from [21], where it is proposed the first study on the use of abstractions in security games (specifically for PSGs) to design scalable algorithms. The authors defined some classes of abstractions and provided parametric algorithms to automatically generate such abstractions, which allow one to relax the constraint of patrolling strategies' Markovianity, a customary assumption in PSGs, and to solve large game instances.

Higher degrees of interaction between the players by means of a sequential structure were explored, e.g., in [6, 92]: the authors analytically determined the value of the game or bounds on the value, for various classes of graphs, especially where the network is a line, which models the problem of guarding a channel or protecting a border from infiltration.

Finally, [82] proposes a skeleton model of an alarm system where sensors have no spatial uncertainty in detecting attacks on single targets. The authors analyze how sensory information can improve the effectiveness of

patrolling strategies in adversarial settings. They show that, when sensors are not affected by false negatives and false positives, the best strategy prescribes that the patroller just responds to an alarm signal rushing to the target under attack without patrolling the environment. As a consequence, in such cases the model treatment becomes trivial.

3.3 Learning Security Games

The other class of games we will investigate in this dissertation is *Learning Games* (LSGs). In general, the task of computing an equilibrium in a game is an open problem, in which the most suitable techniques to adopt strictly depend on information available to the agents. Two extreme situations can be distinguished: when all the information about the game is common to the players (e.g., utility functions and rationality-either perfect or bounded), the problem is basically an *optimization problem*, solvable by means of techniques from *operations research* [109], conversely, when players have no information about the opponents, the problem is a *multi-learning problem*, and *learning* techniques are commonly employed [115]. Some attempts were also done to pair these two approaches, allowing agents to play at the equilibrium if the opponent is rational and to play off the equilibrium learning to exploit her at best otherwise [40].

As discussed in the previous sections, the success of security games in real-world applications is due to a number of reasons: committing to a strategy is the best the Defender can do, the equilibrium finding problem is conceptually simple since the Attacker can merely play her best response to the commitment of the Defender without any strategic reasoning about her behavior, and the solution is unique except degeneracy. The crucial issue is that in real-world applications the follower may be not perfectly rational, not necessarily playing her best response to the leader's commitment. For instance, a terrorist could decide either to attack a target that is not patrolled, since she is sure to not be caught, or a target not so valuable itself, but that would cause a huge panic reaction in the population (e.g., this is what happened in November 2015 in Paris attacks at the Bataclan theatre). The same challenge may be faced by a company that aims at planning the production of a product and has to decide when and how it is convenient to enter the market when another company is already the leader in such a market-this is the well-known *Stackelberg duopoly* [119]. Whenever the assumption of perfect rationality is not met, each agent may in principle exploit her opponent's strategy.

Here, we will design Security Games in which the Attacker may be not rational. The literature provides a number of models of bounded ratio-

nality [9, 89]. Probably, the most elegant one is the *Quantal Response* (QR) [80], which fixes the probability distribution over the non-optimal actions of an agent on the basis of their optimality gap. The crucial issue is that all the works on bounded rationality make an assumption about the specific behavior of the opponent and this assumption could never be met in real-world applications. In that case, such an assumption may lead to an arbitrarily loss for the leader.

Bounded rationality has been introduced in security games models in the so called Green Security Games (GSGs), a generalization of Stackelberg games [49]. A remarkable example is [97], in which the problem of protecting natural resources from illegal extraction is studied: since such extractions are frequent, it is possible for the Defender to *learn* the distribution of the resources analyzing the Attacker's behavior. A recent application in which an *ad hoc* adaption of the QR function, named *Subjective Utility Quantal Response* (SUQR) [89], has been employed is the prevention from poachers, who hunt endangered species [53, 127]. Here, the QR is employed to model the non-rational behavior of the poachers. In a similar setting, [98] analyze the problem in which the Defender is aware only of the attack activities at targets they protect, modeling it with Restless Multi-Armed Bandit and using Whittle index policy to compute patrol strategies.

In 2014, in [29], the authors analytically proved that in zero-sum security games, lazy Defenders, i.e., guards that simply optimize against perfectly informed Attackers, are almost optimal against diligent Attackers, who try to gather as more observations as possible before carrying on their attack.

In security games, [14, 30] deal with a single rational Attacker whose preferences may be of multiple types in Bayesian fashion. Specifically, the different Attackers are discriminated according to the evaluations they give to the targets, thus leading to the problem of solving Bayesian Stackelberg Games.

The main limitation of all the aforementioned works is that the Defender plays against an Attacker whose behavioral profile is *a priori* known, while in real-world situation it may be unknown. When dealing with sequential decision learning problems, a customary approach consists in exploiting Multi-Armed Bandit (MAB) techniques, as done by [73, 126]. Even though both works focus on minimizing the expected regret, the different actions corresponding to the arms are the possible targets that may be chosen, while in our work we are discriminating among different Attacker types.

In [61], the authors deal with cyber security scenarios, exploiting the strategic placement of honeypots in a network to entice malicious entities into attacking in order to waste Attacker resources and learn information

about attack behavior or previously unknown exploits. In this work, they assume that the Attackers leverage various exploits, such that the value they provide dynamically change over time as more information is gathered about them. To tackle this novel problem, they mapped it to a Multi-Armed Bandit (MAB) problem.

Even if Leader-Follower Equilibrium is perfectly suitable to solve this kind of problems, this solution needs to have all the parameters of the problem. In particular it is necessary to determine the payoffs of the targets, the Attacker's resources and the Attacker's rationality degree.

Security Games represent adversarial interactions between players. However, in general, security games need not to be zero-sum. Some reasons could be that the adversary evaluates some targets as particularly important for her audience for their symbolic value, whereas they may not be of equal importance to the police. Or an adversary may not view even a failed attack as a negative outcome because of the publicity and fear it generates. Or the adversary may need to incur a significant cost in mounting a particular attack that may not be particularly important to the police. This problem is sometimes faced with the Bayesian Stackelberg Games formalism. In this model, it is assumed that different types of Attackers (which means different payoffs matrices) appear to the Defender according to a certain distribution. Determining the values of the targets is a job for domain experts, which is not easy and subject to errors.

Attacker's rationality is another issue: in the previous sections, we always assumed *perfectly rational* Attackers, but in reality Attackers are humans who do not usually reason in a rational fashion. Many human rationality models have been applied by psychologists to describe human behavior about choices, though often these models are *parametric* or simply not suitable to describe all possible kinds of Attackers.

In general, when the Security Game model is not completely specified, we have to overcome that *uncertainties*, developing new solutions that involve *learning* or *robust optimization* techniques. These and other issues, which have been faced in literature, will be illustrated later in this chapter. However first we have to introduce the Online Learning framework that stands at the base of many of the proposed solutions and, as we shall see, has a strong connection to Game Theory itself.

3.3.1 Uncertainty in Security Games

Uncertainties can arise from multiple sources in Security Games. In this section we describe the main works in literature on this topic, comparing

them to our work.

First works in Security Games assumed a *completely rational* adversary. However, even if it can be supposed to be true for particularly critic situation, in most of the security games, the Attackers will take decisions in a different way, a human way indeed, using essentially their feelings and intuitions to choose amongst the possible targets. Quantal choice [79] is one of the classical models used to explain the bounded rational behavior. It has also been applied in games [81], bringing to the definition of a new kind of equilibrium, the Quantal Choice Equilibrium. This model has been used in security games in many ways, particularly in the *poaching* context. First it has been applied using the traditional *utility* function [128], with only one parameter λ used to measure the rationality. However, later works showed that human behavior is better described by the use of a *Subjective Utility function* [89] which weight in a subjective way the coverage probability and the target values (*SUQR*). Successive refinement of the model took in consideration non-linear subjective function [69], taking inspiration from the Prospect Theory [116], and also adding other parameters to the SU, as animal density and distance. Another article on *green security games* also considered SUQR Attackers which re-weighted the Defender commitment using a linear combination of past commitments, with a *bounded memory*. In a paper on protecting natural resources by illegal extractors [97] Attackers were modeled as *Fictitious Quantal Response* (FQR) agent, which means that these Attackers assume the Defender empirical distribution to be its strategy in the next round. Also Attackers with a stocastic strategy, stationary or non-stationary, which are unaware of the Defender commitment are taken into account in a work in the context of border patrolling ([73],[126]) and in another one . The same works also considered *fictitious player* Attackers: as for FQR, these Attackers assume the Defender to draw its move from a stochastic fixed distribution, which they estimate with their observation. They respond to it playing the pure strategy which maximize their utility function.

Determining the exact values for the payoffs matrix is a challenging task: estimates are usually obtained from risk analysis experts and historical data, but the degree of uncertainty is typically high. Consequently, researchers have introduced Bayesian frameworks [95] that capture uncertainty using a probability distribution over possible games. This framework assumes an underlying distribution of Attackers with different payoff values for the targets. However, such a hypothesis may compromise the deployed solution if the supposed distribution is far from the real one. This issue could be solved by learning the distribution, which is, unfortunately, most of the time an impossible task. Another approach consists in bounding the payoffs into fixed

intervals, and then use a robust optimization or a minimax regret algorithm [88]. Conversely, other works did not make any assumption on they payoffs, employing online learning techniques to determine them while playing. In [30] *membership-queries* to find the best response against an Attacker without knowing the exact payoffs, leveraging on the presence of a membership oracle for the optimization region, which is represented by the response of the Attacker to the commitment. In another article [126] a modified version of FPL algorithm is used to minimize the regret in a MAB way. Things can also get more complicated as we extend the domain of the security game: in a work on fighting illegal extraction of natural resources [97] the Defender is supposed to be unaware of the values of a target until it has been attacked. This happens because the area that has to be monitored is so large, and forbidden to any exploitation, that only Attackers can discover the value of its resources.

Every time it is necessary to resort to online learning in order to overcome uncertainties, learning performances strongly depend on the type of feedback received. In a security game context, a complete feedback (Section 2.4.2) for the Defender consists in observing at each turn where the adversary attacked even if the Defender was to able to catch it. This is the hypothesis made in most of the papers in literature ([127], [69], [30], [88], [97],[49]), though in some cases it is necessary to assume the opposite, namely a partial feedback. For example poachers can catch their prey without leaving any track [98] or people can pass illegally the border without being noticed [73]. Most of the time Attackers are modeled as *strategy-aware*. This means that they can see the Defender's commitment to a strategy. However in some papers they only have a complete feedback ([97], [49]), which is usually used to estimate Defender's next strategy (Fictitious Player, FQR).

3.3.2 Features of Security Game models

Here it has been made an attempt to list some of the main features of security game models in literature, based on the possible uncertainties and other characteristics not mentioned before:

- **Attacker Rationality**: Attacker can be completely rational (Stackelberg hypothesis), boundedly rational (SUQR, FQR, Fictitious Player) or he can have a stochastic behavior;
- **Payoff Knowledge**: Attacker payoffs can be known, vary amongst the various type of Attacker (Bayesian Security Games), or be completely unknown to the Defender;

- **Feedback**:the Defender is not always aware of all what happened in the last round, he can have a *partial* or a *complete* feedback. The Attacker can be *strategy-aware* or have a complete feedback. No works in the literature consider Attackers with a partial feedback;
- **Topology**: the game can have a topological structure, such that not all the actions are available to each position of players, or not;
- **Coordination**: Attackers are *coordinated* if they act as a unique entity (for example never attacking the same target at the same time). They have a *common knowledge* if they share collected information. When there are multiple defending resource, usually they have both these features, but in principle they could be, for example, not coordinated or temporary unable to share information;
- **States**: the system can be modeled with a single state or with multiple states.

The problem we will study (see Chapter 8) has *known payoffs*, *complete feedback* and a *complete coordination*. *No topology* is considered, and a *single state* is present. However, differently from the previous literature this work allows to maintain *unspecified* the Attacker *rationality*. In fact our main goal is to discover if it is possible to learn the opponent's behavior from her moves, in connection to the Defender's commitments, while minimizing the regret.

Part II

Uncertainties of the Alarm System

CHAPTER 4

Adversarial Patrolling with Spatially Uncertain Alarm Signals

The purpose and essence of the known life, the transient, is to form and decide identity for the unknown life, the permanent.
— Walt Whitman

Often, in large environments of infrastructures, a constant surveillance of every area is not affordable: to cope with this issue, a common countermeasure is the usage of cheap but wide-ranged sensors, able to detect suspicious events that occur in large areas, supporting patrollers to improve the effectiveness of their strategies. This is a two-phase approach: first, there is a surveillance phase, by means of sensors or drones to detect the intrusion and, only when an attack is actually detected, the patrollers rush towards the threat to block it. Real-world applications include UAVs surveillance of large infrastructures [18], wildfires detection with CCD cameras [74], agricultural fields monitoring [58], surveillance based on wireless sensor networks [129], and border patrolling [112]. We summarily describe two practical security problems that can be ascribed to this category. We report them as examples, presenting features and requirements that our model

can properly deal with. In the rest of the chapter, we will necessarily take a general stance, but we encourage the reader to keep in mind these two cases as reference applications for a real deployment of our model.

Fight to Illegal Poaching Poaching is a widespread environmental crime that causes the endangerment of wildlife in several regions of the world. Its devastating impact makes the development of surveillance techniques to contrast this kind of activities one of the most important matters in national and international debates. Poaching typically takes place over vast and savage areas, making it costly and ineffective to solely rely on persistent patrol by ranger squads. To overcome this issue, recent developments have focused on providing rangers with environmental monitoring systems to better plan their inspections, concentrating them in areas with large likelihood of spotting a crime. Such systems include the use of UAVs flying over the area, alarmed fences, and on-the-field sensors trying to recognize anomalous activities.[1] In all these cases, technologies are meant to work as an alarm system: once the illegal activity is recognized, a signal is sent to the rangers base station from where a response is undertaken. In the great majority of cases, a signal corresponds to a spatially uncertain localization of the illegal activity. For example, a camera-equipped UAV can spot the presence of a pickup in a forbidden area but cannot derive the actual location to which poachers are moving. In the same way, alarmed fences and sensors can only transmit the location of violated entrances or forbidden passages. In all these cases a signal implies a restricted, yet not precise, localization of the poaching activity. The use of security games in this particular domain is not new (see, for example, [53]). However, our model allows the computation of alarm response strategies for a given alarm system deployed on the field. This can be done by adopting a discretization of the environment, where each target corresponds to a sector, values are related to the expected population of animals in that sector, and deadlines represent the expected completion time of illegal hunts.

Safety of Fair Sites Fairs are large public events attended by thousands of visitors, where the problem of guaranteeing safety for the hosting facilities can be very hard. For example, Expo 2015, the recent Universal Exposition hosted in Milan, Italy, saw an average of about 100,000 visits per day. This poses the need for carefully addressing safety risks, which can also derive from planned act of vandalism or terrorist attacks. Besides security guards

[1] See, for example, `http://wildlandsecurity.org/`.

patrols, fair sites are often endowed with locally installed monitoring systems. Expo 2015 employed around 200 baffle gates and 400 metal detectors at the entrance of the site. The internal area was constantly monitored by 4,000 surveillance cameras and by 700 guards. Likely, when one or more of these devices/personnel identified a security breach, a signal was sent to the control room together with a circumscribed request of intervention. This approach is required because, especially in this kind of environments, detecting a security breach and neutralizing it are very different tasks. The latter one, in particular, usually requires a greater effort involving special equipment and personnel whose employment on a demand basis is much more convenient. Moreover, the detecting location of a threat is in many cases different from the location where it could be neutralized, making the request of intervention spatially uncertain. For instance, consider a security guard or a surveillance camera detecting the visitors' reactions to a shooting rampage performed by some attacker. In examples like these, we can restrict the area where the security breach happened but no precise information about the location can be gathered since the attacker will probably have moved. Our model could be applied to provide a policy with which schedule interventions upon a security breach is detected in some particular section of the site. In such case, targets could correspond to buildings or other installations where visitors can go. Values and deadlines can be chosen according to the importance of targets, their expected crowding, and the required response priority.

4.1 Original Contributions

In this chapter, we propose the first Security Game model that integrates a spatially uncertain alarm system in game-theoretic settings for patrolling. An alarm signal carries the information about the set of targets that can be under attack and it is described by the probability of being generated when each target is attacked. The analysis and the main results we propose in this work are devoted to the basic game model where the Defender can control a single patroller and the alarm system is immune to false positives and false negatives making spatial uncertainty its only limitation. As our results show, such assumptions do not play down the significance of the arising computational challenges whose resolution is a prerequisite for more complex settings. The game we consider can be decomposed in a finite number of finite-horizon subgames, each called Signal Response Game from v (SRG-v) and capturing the situation in which the Defender is in a vertex v and the Attacker attacked a target, and an infinite-horizon game, called Patrolling Game (PG),

in which the Defender moves in absence of any alarm signal. We show that SRG-v is FNP-hard for tree-based topologies and that, for arbitrary graphs, is APX-hard. We provide two exact algorithms. The first one, based on dynamic programming, performs a breadth-first search, while the second one, based on branch-and-bound approach, performs a depth-first search. We use the same two approaches to design two approximation algorithms.

Then, we study the PG, showing that when no false positives and no missed detections are present, the optimal Defender strategy is to stay in a fixed location, wait for a signal, and respond to it at best. This strategy keeps being optimal even when non-negligible missed detection rates are allowed. We experimentally evaluate the scalability of our exact algorithms and we compare them with the approximation ones in terms of solution quality and compute times, investigating in hard instances the gap between our hardness results and the theoretical guarantees of our approximation algorithms. We show that our approximation algorithms provide very high quality solutions even in hard instances. Finally, we provide an example of resolution for a realistic instance, based on Expo 2015, and we show that our exact algorithms can be applied for such kind of settings. Moreover, in our realistic instance we assess how the optimal patrolling strategy coincides with a static placement even when allowing a false positive rate of less or equal to 30%.

4.1.1 Chapter Structure

In Section 4.2, we introduce our game model. In Section 4.3, we study the problem of finding the strategy of the Defender for responding to an alarm signal. In Section 4.4, we study the patrolling problem. In Section 4.5, we experimentally evaluate our algorithms. Appendix A includes the most technical proofs of the work and discusses some particular results obtained for special classes of instances while Appendix B provides a table summarizing the notation symbols used in the thesis.

4.2 Problem Statement

In this section we formalize the problem we study. More precisely, in Section 4.2.1 we describe the patrolling setting and the game model, while in Section 4.2.2 we state the computational questions we shall address in this work.

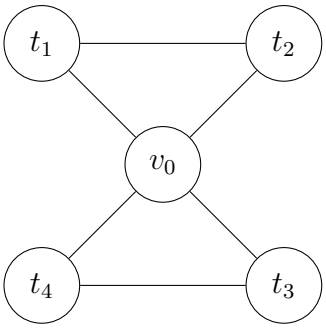
4.2.1 Game Model

Initially, in Section 4.2.1, we introduce a basic patrolling security game model integrating the main features from models currently studied in literature. Next, in Section 4.2.1, we extend our game model by introducing alarm signals. In Section 4.2.1, we depict the game tree of our patrolling security game with alarm signals and we decompose it in notable subgames to facilitate its study. To ease presentation we summarized our notation symbols in Table B.1.

Basic Patrolling Security Game

As is customary in the artificial intelligence literature [22, 123], we deal with discrete, both in terms of space and time, patrolling settings, representing an approximation of a continuous environment. Specifically, we model the environment to be patrolled as an undirected connected graph $G = (V, E)$, where vertices represent different areas connected by various corridors/roads represented through the edges. Time is discretized in turns. Edges are assigned weights representing the number of time steps needed to traverse them. If not stated otherwise, we shall assume unitary weights. With $\omega^*_{i,j}$ we shall denote the shortest time to go from i to j. We define $T \subseteq V$ the subset of sensible vertices, called *targets*, that must be protected from possible attacks. Each target $t \in T$ is characterized by a value $\pi(t) \in (0, 1]$ and a penetration time $d(t) \in \mathbb{N}^+$, which measures the number of turns needed to complete an attack to t.

Example 14. *We report in Figure 4.1 an example of patrolling setting. Here,* $V = \{v_0, v_1, v_2, v_3, v_4\}$, $T = \{t_1, t_2, t_3, t_4\}$ *where* $t_i = v_i$ *for* $i \in \{1, 2, 3, 4\}$. *All the targets* t *present the same value* $\pi(t)$ *and the same penetration time* $d(t)$.



t	$\pi(t)$	$d(t)$
t_1	0.5	4
t_2	0.5	4
t_3	0.5	4
t_4	0.5	4

Figure 4.1: *Example of patrolling setting.*

At each turn, an *Attacker* $\mathcal{A}$ and a *Defender* $\mathcal{D}$ play simultaneously having the following available actions:

- if $\mathcal{A}$ has not attacked in the previous turns, it can observe the position of $\mathcal{D}$ in the graph G^2 and decide whether to attack a target or to wait for a turn. The attack is instantaneous, meaning that there is no delay between the decision to attack and the actual presence of a threat in the selected target[3];

- $\mathcal{D}$ has no information about the actions undertaken by $\mathcal{A}$ in previous turns and selects the next vertex to patrol among those adjacent to the current one; each movement is a non-preemptive traversal of a single edge $(v, v') \in E$.

The game may conclude in correspondence of any of the two following events. The first one is when $\mathcal{D}$ patrols a target t that is under attack by $\mathcal{A}$ from less than $d(t)$ turns. In such case the attack is prevented and $\mathcal{A}$ is captured. The second one is when target t is attacked and $\mathcal{D}$ does not patrol t during the $d(t)$ turns that follow the beginning of the attack. In such case the attack is successful and $\mathcal{A}$ escapes without being captured. When $\mathcal{A}$ is captured, $\mathcal{D}$ receives a utility of 1 and $\mathcal{A}$ receives a utility of 0. When an attack to t has success, $\mathcal{D}$ receives $1 - \pi(t)$ and $\mathcal{A}$ receives $\pi(t)$. The game may not conclude if $\mathcal{A}$ decides to wait for every observed position of $\mathcal{D}$, thus never attacking. In such case, $\mathcal{D}$ receives 1 and $\mathcal{A}$ receives 0. (Another intuitive way to think at this payoff structure is to assume that $\mathcal{D}$ receives an initial utility of 1 and then loses $\pi(t)$ whenever target t is successfully compromised.) Notice that the game is constant sum and therefore it is equivalent to a zero-sum game through a positive affine transformation.

The above game model is in extensive form (being played sequentially), with imperfect information ($\mathcal{D}$ not observing the actions undertaken by $\mathcal{A}$), and with infinite horizon ($\mathcal{A}$ being in the position to wait forever). The fact that $\mathcal{A}$ can observe the actions undertaken by $\mathcal{D}$ before acting makes the *leader-follower* equilibrium the natural solution concept for our problem, where $\mathcal{D}$ is the *leader* and $\mathcal{A}$ is the *follower*. Since we focus on zero-sum games, the leader's strategy at the leader-follower equilibrium is its maxmin strategy and it can be found by employing linear mathematical programming, which requires polynomial time in the number of actions available to the players [109].

[2]Partial observability of $\mathcal{A}$ over the position of $\mathcal{D}$ can be introduced as discussed in [24].

[3]This is a worst-case assumption according to which $\mathcal{A}$ is as strong as possible. It can be relaxed by associating execution costs to the $\mathcal{A}$'s actions as shown in [23].

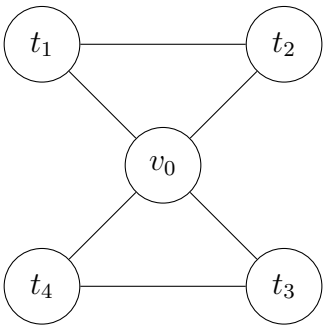
Introducing Alarm Signals

We extend the game model described in the previous section by introducing a *spatial uncertain alarm system* that can be exploited by $\mathcal{D}$. The basic idea is that an alarm system uses a number of sensors spread over the environment to gather information about possible attacks and raises an alarm signal at any time an attack occurs. The alarm signal provides some information about the location (target) where the attack is ongoing, but it is affected by uncertainty. In other words, the alarm system detects an attack but it is uncertain about the target under attack. Formally, the alarm system is defined as a pair (S, p), where $S = \{s_0, s_1, \cdots, s_m\}$ is a set of $m \geq 1$ *signals* and $p : S \times T \rightarrow [0, 1]$ is a function that specifies the probability of having the system generating a signal s given that target t has been attacked, we denote such probability with $p(s \mid t)$. With a slight abuse of notation, for a signal s we define $T(s) = \{t \in T \mid p(s \mid t) > 0\}$ and, similarly, for a target t we have $S(t) = \{s \in S \mid p(s \mid t) > 0\}$. In this work, we assume that:

- the alarm system is not affected by false positives (signals generated when no attack has occurred). Formally, $p(s \mid \triangle) = 0$, where $\triangle$ indicates that no targets are under attack;
- the alarm system is not affected by false negatives (signals not generated even though an attack has occurred). Formally, $p(s_0 \mid t) = 0$, where s_0 indicates the null signal, i.e., no signals have been generated; in Section 4.4 we will show that the optimal strategies we compute under this assumption can preserve optimality even in presence of non-negligible false negatives rates.

Example 15. *We report two examples of alarm systems for the patrolling setting depicted in Figure 4.1. The first example is reported in Figure 4.2(a). It is a low-accuracy alarm system that generates the same signal anytime a target is under attack and therefore that does not provide any information about the target under attack. The second example is reported in Figure 4.2(b). It provides more accurate information about the localization of the attack than the previous example. Here, the reception of a signal s_i implies, under a uniform strategy of $\mathcal{A}$, a high probability of an attack on target t_i. Namely, when t_i is attacked the alarm system generates s_i with high probability and a different signal otherwise.*

Given the presence of an alarm system defined as above, the game mechanism changes in the following way. At each turn, before deciding its next move, $\mathcal{D}$ observes whether or not a signal has been generated by the alarm



t	$\pi(t)$	$d(t)$	$p(s_1 \mid t)$
t_1	0.5	4	1
t_2	0.5	4	1
t_3	0.5	4	1
t_4	0.5	4	1

(a) Alarm system with a single signal for all the targets.

t	$\pi(t)$	$d(t)$	$p(s_1 \mid t)$	$p(s_2 \mid t)$	$p(s_3 \mid t)$	$p(s_4 \mid t)$	$p(s_5 \mid t)$
t_1	0.5	4	0.1	0.6	0.1	0.1	0.1
t_2	0.5	4	0.1	0.1	0.6	0.1	0.1
t_3	0.5	4	0.1	0.1	0.1	0.6	0.1
t_4	0.5	4	0.1	0.1	0.1	0.1	0.6

(b) Alarm system with multiple signals.

Figure 4.2: *Examples of alarm systems.*

system and then makes its decision considering such information. This introduces in our game a node of chance implementing the non-deterministic selection of signals.

The game tree and its decomposition

Here we depict the game tree of our game model, decomposing it in some recurrent subgames. A portion of the game is depicted in Figure 4.3. Such tree can be read along the following steps.

- *Root of the tree.* $\mathcal{A}$ decides whether to wait for a turn (this action is denoted as $\triangle$ to indicate that no target is attacked) or to attack a target $t \in T$ (this action is denoted by the label t of the attacked target).
- *Second level of the tree.* $\mathcal{N}$ denotes the alarm system, represented by a nature-type player. Its behavior is *a priori* specified by the conditional probability mass function p which determines the generated signal given the attack performed by $\mathcal{A}$. In particular, it is useful to distinguish between two cases:
  - (*a*) if $\mathcal{A}$ does not attack (action $\triangle$), then no signal will be generated under the assumption that $p(s_0 \mid \triangle) = 1$;
  - (*b*) if $\mathcal{A}$ attacks target t, a signal s will be drawn from $S(t)$ with probability $p(s \mid t)$ (recall that $p(s_0 \mid t) = 0$).

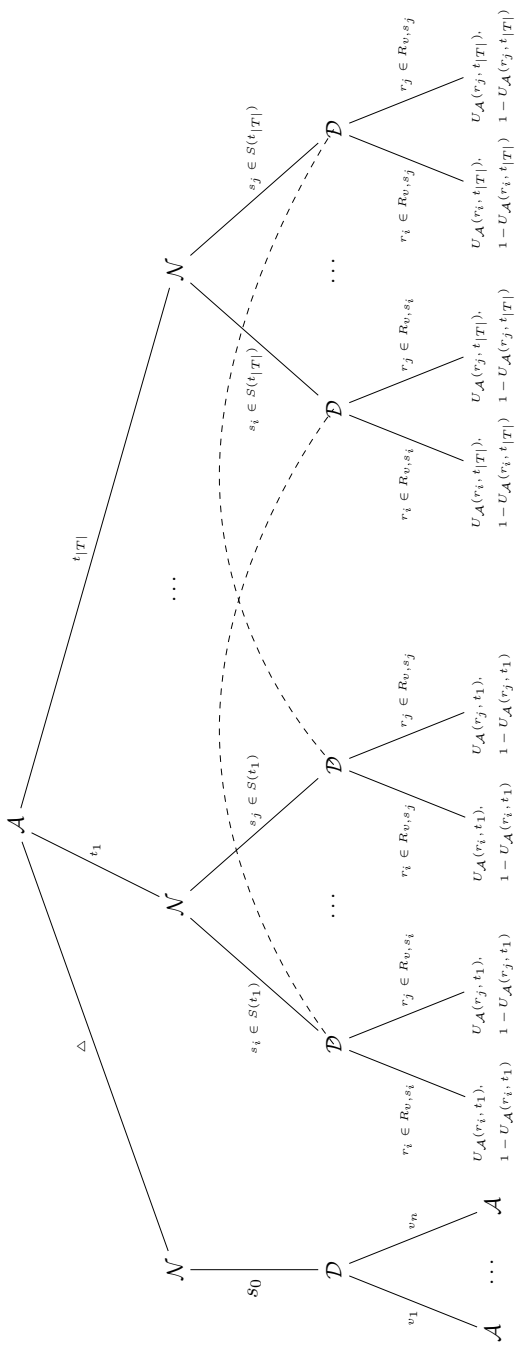


Figure 4.3: *Game tree, v is assumed to the be current vertex for $\mathcal{D}$, r is a collapsed sequence of moves, called route, that $\mathcal{D}$ performs on the graph, dashed lines indicate information sets, and function $U_{\mathcal{A}}(r_i, t_j)$ returns $\mathcal{A}$'s payoff when $\mathcal{A}$ attacks target t_j and $\mathcal{D}$ patrols on route r_i.*

- *Third level of the tree.* $\mathcal{D}$ observes the signal generated by the alarm system and decides the next vertex to patrol among those adjacent to the current one (the current vertex is initially chosen by $\mathcal{D}$).
- *Fourth level of the tree and on.* It is useful to distinguish between two cases:
  - (a) if no attack is present, then the tree of the subgame starting from here is the same of the tree of the whole game, except for the position of $\mathcal{D}$ that may be different from the initial one (notice that in this case each of $\mathcal{A}$'s decision nodes is a singleton information set, modeling the fact that $\mathcal{A}$ can observe $\mathcal{D}$'s position);
  - (b) if an attack is taking place on target t, then only $\mathcal{D}$ can act.

Such game tree can be decomposed in a number of finite recurrent subgames such that the best strategies of the agents in each subgame are independent from those in other subgames. This decomposition allows us to apply a *divide et impera* approach, simplifying the resolution of the problem of finding an equilibrium. More precisely, we denote with Γ one of these subgames. We define Γ as a game subtree that can be extracted from the tree of Figure 4.3 as follows. Given $\mathcal{D}$'s current vertex $v \in V$, select a decision node for $\mathcal{A}$ and call it i. Then, extract the subtree rooted in i discarding the branch corresponding to action Δ (no attack)[4]. Intuitively, such subgame models the players' interaction when the Defender is in some given vertex v and the Attacker will perform an attack. As a consequence, each Γ obtained in such way is finite (once an attack on t started, the maximum length of the game is $d(t)$). Moreover, the set of *different* Γs we can extract is finite since we have one subgame for each possible current vertex for $\mathcal{D}$. As a consequence we can extract at most $|V|$ different subgames. Notice that, due to the infinite horizon, each subgame can recur an infinite number of times along the game tree. However, being such repetitions independent and the game zero-sum, we only need to solve one subgame to obtain the optimal strategy to be applied in each of its repetitions. In other words, when assuming that an attack will be performed, the agents' strategies can be split in a number of independent strategies solely depending on the current position of $\mathcal{D}$.

[4]Rigorously speaking, our definition of subgame is not compliant with the definition provided in game theory [77], which requires that all the actions of a node belong to the same subgame (and therefore we could not separate action Δ from actions t). However, we can slightly change the structure of our game making our definition of subgame compliant with the one from game theory. More precisely, it is sufficient to split each node of $\mathcal{A}$ into two nodes: in the first $\mathcal{A}$ decides to attack a target or to wait for one turn, and in the second, conditioned to the fact that $\mathcal{A}$ decided to attack, $\mathcal{A}$ decides which target to attack. This way, the subtree whose root is the second node of $\mathcal{A}$ is a subgame compliant with game theory. It can be easily observed that this change to the game tree structure does not affect the behaviour of the agents.

The reason why we discarded the branch corresponding to action Δ in each subgame is that we seek to deal with such complementary case exploiting a simple backward induction approach, as explained later.

First, we call *Signal Response Game* from v the subgame Γ defined as above and characterized by a vertex v representing the current vertex of $\mathcal{D}$ (for brevity, we shall use SRG-v). In an SRG-v, the goal of $\mathcal{D}$ is to find the best strategy starting from vertex v to respond to any alarm signal. All the SRG-vs are independent and thus the best strategy in each subgame does not depend on the strategies of the other subgames. The intuition is that the best strategy in an SRG-v does not depend on the vertices visited by $\mathcal{D}$ before the attack. Given an SRG-v, we denote by $\sigma^{\mathcal{D}}_{v,s}$ the strategy of $\mathcal{D}$ once observed signal s, by $\sigma^{\mathcal{D}}_{v}$ the strategy profile $\sigma^{\mathcal{D}}_{v} = (\sigma^{\mathcal{D}}_{v,s_1} \dots, \sigma^{\mathcal{D}}_{v,s_m})$ of $\mathcal{D}$, and by $\sigma^{\mathcal{A}}_{v}$ the strategy of $\mathcal{A}$. Let us notice that in an SRG-v, given a signal s, $\mathcal{D}$ is the only agent that plays and therefore each sequence of moves can be collapsed in a single action. Thus, SRG-v is essentially a two-level game in which $\mathcal{A}$ decides the target to attack and $\mathcal{D}$ decides the sequence of moves on the graph.

Then, according to classical backward induction arguments [77], once we have found the best strategies of each SRG-v, we can substitute the subgames with the agents' equilibrium utilities and then we can find the best strategy of $\mathcal{D}$ for patrolling the vertices whenever no alarm signal has been raised and the best strategy of attack for $\mathcal{A}$. We call this last problem *Patrolling Game* (for conciseness, we shall use PG). We denote by $\sigma^{\mathcal{D}}$ and $\sigma^{\mathcal{A}}$ the strategies of $\mathcal{D}$ and $\mathcal{A}$, respectively, in the PG.

4.2.2 The Computational Questions

Our contributions focus on the design of algorithms to find an equilibrium for our game and develop along four central questions. The first one stems directly from our problem definition.

Question 1. *Which is the best patrolling strategy for $\mathcal{D}$ maximizing its expected utility?*

Clearly, this problem is related to what we called PG in our game decomposition. In order to build an answer, we pose other three questions that, instead, involve the other subgame called SRG-v.

Question 2. *Given a starting vertex v and a signal s, is there any strategy allowing $\mathcal{D}$ to visit all the targets in $T(s)$, each within its deadline?*

Question 3. *Given a starting vertex v and a signal s, is there any pure strategy giving $\mathcal{D}$ an expected utility of at least k?*

Question 4. *Given a starting vertex v and a signal s, is there any mixed strategy giving $\mathcal{D}$ an expected utility of at least k?*

These questions are not independent from each other. In particular, answering to the last three is a prerequisite to solving the first one. For this reason, we shall take a bottom-up approach answering the above questions starting from the last three and then dealing with the first one at the whole-game level. Also, Questions 3 and 4 are not easier than Question 2 so hardness results for this last one can be extended to the others.

4.3 Signal Response Game

In this section we start by dealing with the resolution of SRG-v. Specifically, in Section 4.3.1 we prove the hardness of the problem, analyzing its computational complexity. Then, in Section 4.3.2 and in Section 4.3.3 we propose two algorithms, the first based on *dynamic programming* (breadth-first search) while the second adopts a *branch and bound* (depth-first search) paradigm. Furthermore, we provide a variation for each algorithm, approximating the optimal solution. For the sake of presentation, the most technical proofs are reported in A.

4.3.1 Complexity Results

The aim of this section is to assess the computational complexity of finding an exact or approximate equilibrium for our game model. Furthermore, we aim at identifying the source of hardness of our problem and the most efficient algorithm for solving it.

Each SRG-v is characterized by $|T|$ actions available to $\mathcal{A}$, each corresponding to a target t, and by $O(|V|^{\max_t\{d(t)\}})$ decision nodes of $\mathcal{D}$. The portion of game tree played by $\mathcal{D}$ can be safely reduced by observing that $\mathcal{D}$ will move between any two targets along a shortest path. This allows us to discard from the tree all the decision nodes where $\mathcal{D}$ occupies a non-target vertex. Nevertheless, this reduction keeps the size of the game tree exponential in the parameters of the game, specifically $O(|T|^{|T|})$.[5] Notice that, the exponential size of the game tree does not constitute a proof that finding the equilibrium strategies of an SRG-v requires exponential time in the parameters of the game because it does not exclude the existence of some compact representation of $\mathcal{D}$'s strategies, e.g., Markovian strategies. The first result we provide implies that such a representation is very unlikely to exist or that,

[5] A more accurate bound is $O(|T|^{\min\{|T|,\max_t\{d(t)\}\}})$.

if it exists, it unlikely can be computed in polynomial time. Call g_v the expected utility of $\mathcal{A}$ from SRG-v ($1 - g_v$ is then the corresponding utility for $\mathcal{D}$). We define the following problem.

Definition 4.1 (k-SRG-v)**.** *The decision problem k-SRG-v is defined as:*
INSTANCE: an instance of SRG-v;
QUESTION: is there any $\sigma^{\mathcal{D}}$ such that, when $\mathcal{A}$ plays its best response, it holds that $g_v \leq k$?

Theorem 4.1. *k-SRG-v is* NP*-hard even when $|S| = 1$ and the graph is a tree.*[6]

The above result shows that with trees:

- answering to Question 1 is NP-hard,
- answering to Questions 2, 3, 4 is NP-hard.

For the sake of completeness, notice that a maxmin strategy with support upper bounded by $|T|$ (that is, the number of actions available to the min player $\mathcal{A}$) always exists [107].

Now, given that we established that the main source of hardness of the problem is computing the strategy space of $\mathcal{D}$, we focus on the problem of finding an efficient representation for it. We start with some definitions.

Definition 4.2 (Route)**.** *Given a starting vertex v and a signal s, a route r is a finite sequence of vertices where, called $r(i)$ the i-th vertex, $r(0) = v$ and $r(i) \in T(s)$ for any $i > 0$. With a slight overload of notation, call $T(r)$ the set of targets from the sequence.*

We restrict our attention on a special class of routes that we call *covering*. For a route r and $i > 0$, define $A_r(r(i)) = \sum_{h=0}^{i-1} \omega^*_{r(h),r(h+1)}$. Such value gives the time needed by $\mathcal{D}$ to arrive at target $r(i)$ after following a graph walk along the shortest paths between consecutive vertices of the sequence $r(0), r(1), \ldots, r(i)$.

Definition 4.3 (Covering Route)**.** *A covering route is a route r where $A_r(t) \leq d(t)$ for any $t \in T(r)$.*

Covering routes are abstractions for $\mathcal{D}$'s available pure strategies when the current vertex is v and a signal s has been generated. If r is a covering route for vertex v and signal s and $T(r) \subseteq T(s)$ is the set of targets in

[6] Rigorously speaking, we show NP-hardness for a more specific class of trees we call S2L-STARS. Details can be found in A.1.

r, then we can always instantiate r to a graph walk for $\mathcal{D}$ (a sequence of moves on G) which guarantees to capture $\mathcal{A}$ on any target in $T(r)$. Such walk is simply obtained by starting from $r(0) = v$ and then covering any shortest path between $r(i)$ and $r(i+1)$. The total temporal cost of such walk is expressed by $c(r) = A_r(r(|T(r)|))$. We shall informally refer to such process as *walking a route* r.

Covering routes then constitute the action space for $\mathcal{D}$ is a SRG-v game. Even when considering a single signal s, the number such of such routes is $O(|T(s)|^{|T(s)|})$ in the worst case. However, some covering routes will never be played by $\mathcal{D}$ due to any of the following two dominance arguments [91] and discarding dominated routes could be crucial in the design of an efficient algorithm.

Definition 4.4 (Intra-Set Dominance)**.** *Given a starting vertex v, a signal s and two different covering routes r, r' such that $T(r) = T(r')$, if $c(r) \leq c(r')$ then r dominates r'.*

Definition 4.5 (Inter-Set Dominance)**.** *Given a starting vertex v, a signal s and two different covering routes r, r', if $T(r) \supset T(r')$ then r dominates r'.*

Furthermore, it is convenient to introduce the concept of *covering set*, which is strictly related to the concept of covering route. It is defined as follows.

Definition 4.6 (Covering Set)**.** *Given a starting vertex v and a signal s, a covering set (covset) Q is a subset of targets $T(s)$ such that there exists a covering route r with $T(r) = Q$.*

Let us focus on Definition 4.4. It suggests that we can safely use only one route per covering set. Covering sets suffice for describing all the outcomes of the game, since the agents' payoffs depend only on the fact that $\mathcal{A}$ attacks a target t that is covered by $\mathcal{D}$ or not, and in the worst case are $O(2^{|T(s)|})$, with a remarkable reduction of the search space w.r.t. $O(|T(s)|^{|T(s)|})$. However, any algorithm restricting on covering sets should be able to determine whether or not a set of targets is a covering one, which is a difficult problem as well.

Definition 4.7 (COV-SET problem)**.** *The COV-SET problem is defined as:*
INSTANCE: an instance of SRG-v with a target set T;
QUESTION: is T a covering set? (Equivalently, does T admit any covering route r?)

Theorem 4.2. *COV-SET is* NP*-complete.*

The theorem immediately follows from the same proof given for Theorem 4.1, reported in Appendix A.1.

Therefore, computing a covering route for a given set of targets (or deciding that no covering route exists) is not doable in polynomial time unless $\mathsf{P} = \mathsf{NP}$. This shows that, while covering sets suffice for defining the payoffs of the game and therefore the size of payoffs matrix can be bounded by the number of covering sets, in practice we also need covering routes to certificate that a given subset of targets is covering. The impossibility to confine our algorithms to the space of covering sets seems to suggest a complexity worse than $O(2^{|T(s)|})$. However, in the next section we provide an algorithm with complexity $O(2^{|T(s)|})$ (neglecting polynomial terms) to enumerate all and only the covering sets and, for each of them, the associated covering route with minimum cost.

Let us focus on Definition 4.5. Inter-Set dominance can be leveraged to introduce the concept of *maximal* covering sets (and routes) which could enable a further reduction in the set of actions available to $\mathcal{D}$.

Definition 4.8 (MAXIMALITY)**.** *Given a covering set $Q = T(r)$ for some r, we say that Q and r are maximal if there is no other covering route r' such that $Q \subset T(r')$.*

In the best case, when there is a route covering all the targets, the number of maximal covering sets is 1 (and we can safely restrict to a single minimum cost covering route over that set), while the number of covering sets (including the non-maximal ones) is $2^{|T(s)|}$. Thus, considering only maximal covering sets allows an exponential reduction of the payoffs matrix. In the worst case, when all the possible subsets composed of $|T(s)|/2$ targets are maximal covering sets, the number of maximal covering sets is $O(2^{|T(s)|-2})$, while the number of covering sets is $O(2^{|T(s)|-1})$, allowing a reduction of the payoffs matrix by a factor of 2. Furthermore, if we knew *a priori* that Q is a maximal covering set, we could avoid searching for covering routes for any set of targets that strictly contains Q. When designing an algorithm to solve this problem, Definition 4.5 could then be exploited to introduce pruning techniques to save average compute time. However, the following result shows that deciding if a covering set is maximal is hard.

Definition 4.9 (MAX-COV-SET)**.** *The decision problem MAX-COV-SET is defined as:*
INSTANCE: an instance of SRG-v with a target set T and a covering set $T' \subset T$;
QUESTION: is T' maximal?

Theorem 4.3. *There is no polynomial-time algorithm for MAX-COV-SET unless* $\mathsf{P} = \mathsf{NP}$.

Nevertheless, we show hereafter that there exists an algorithm computing all and only the maximal covering sets and one route for each of them (which potentially leads to an exponential reduction of the time needed for solving the linear program) with only an additional polynomial cost w.r.t. the enumeration of all the covering sets. Therefore, neglecting polynomial terms, our algorithm has a complexity of $O(2^{|T(s)|})$.

Finally, we focus on the complexity of approximating the best solution in an SRG-v. When $\mathcal{D}$ restricts its strategies to be pure, the problem is clearly not approximable in polynomial time even when the approximation ratio depends on $|T(s)|$. The basic intuition is that, if a game instance admits the maximal covering route that covers all the targets and the value of all the targets is 1, then either the maximal covering route is played returning a utility of 1 to $\mathcal{D}$ or any other route is played returning a utility of 0, but no polynomial-time algorithm can find the maximal covering route covering all the targets, unless $\mathsf{P} = \mathsf{NP}$. On the other hand, it is interesting to investigate the case in which no restriction to pure strategies is present. We show that the problem keeps being hard.

Theorem 4.4. *The optimization version of k-SRG-v, say OPT-SRG-v, is* APX*-hard even in the restricted case in which the graph is metric, there is only one signal s, all targets $t \in T(s)$ have the same penetration time $d(t)$, there exists a maximal covering route covering all the targets.*

The above theorem allows us to make two important remarks.

Remark 1. *The above result does not exclude the existence of constant-ratio approximation algorithms for OPT-SRG-v. We conjecture that it is unlikely. OPT-SRG-v presents similarities with the (metric) DEADLINE-TSP, where the goal is to find the longest path of vertices each traversed before its deadline. The DEADLINE-TSP does not admit any constant-ratio approximation algorithm [32] and the best-known approximation algorithm has logarithmic approximation ratio [15]. The following observations can be produced about the relationships between OPT-SRG-v and DEADLINE-TSP:*

- *when the maximal route covering all the targets in the OPT-SRG-v exists, the optimal solution of the OPT-SRG-v is also optimal for the DEADLINE-TSP applied to the same graph;*
- *when the maximal route covering all the targets in the OPT-SRG-v does*

not exist, the optimal solutions of the two problems are different, even when we restrict us to pure-strategy solutions for the OPT-SRG-v;

- *approximating the optimal solution of the DEADLINE-TSP does not give a direct technique to approximate OPT-SRG-v, since we should enumerate all the subsets of targets and for each subset of targets we would need to execute the approximation of the DEADLINE-TSP, but this would require exponential time. We notice in addition that even the total number of sets of targets with logarithmic size is not polynomial, being $\Omega(2^{\log^2(|T|)})$, and therefore any algorithm enumerating them would require exponential time;*

- *when the optimal solution of the OPT-SRG-v is randomized, examples of optimal solutions in which maximal covering routes are not played can be produced, showing that at the optimum it is not strictly necessary to play maximal covering routes, but even approximations suffice.*

Remark 2. *If it were possible to map DEADLINE-TSP instances to OPT-SRG-v instances where the maximal covering route covering all the targets exists, then it trivially follows that OPT-SRG-v does not admit any constant-approximation ratio. We were not able to find such a mapping and we conjecture that, if there is an approximation-preserving reduction from DEADLINE-TSP to OPT-SRG-v, then we cannot restrict to such instances. The study of instances of OPT-SRG-v where mixed strategies may be optimal make the treatment very challenging.*

4.3.2 Dynamic-programming Algorithm

We start by presenting two algorithms. The first one is exact, while the second one is an approximation algorithm. Both algorithms are based on a dynamic programming approach.

Exact Algorithm

In this section we provide an algorithm to compute the strategies available to $\mathcal{D}$ when v is the current vertex and signal s has been generated by the alarm system. The idea is to adopt a dynamic programming method that enumerates covering sets of increasing cardinalities. Each covering set is obtained by a sequence of expansions which, starting from the empty set, add one target at each iteration. We denote a covering set by $Q^k_{v,t}$ where k indicates its cardinality while v and t denote the starting vertex of $\mathcal{D}$ and the last target added to the set, respectively. The algorithm operates in such a way that the sequence of targets corresponding to the expansions made to

obtain $Q^k_{v,t}$ is a covering route for that set. Informally, we shall call it the *generative route* of $Q^k_{v,t}$ and we will denote with $c(Q^k_{v,t})$ its temporal cost. We choose to obtain this by requiring any expansion to be admissible with respect to three conditions. Given a set $Q^k_{v,t}$ a new set $Q^{k+1}_{v,w} = Q^k_{v,t} \cup \{w\}$ can be obtained if:

1. w is not covered in the current covering set, that is $w \in T(s) \setminus Q^k_{v,t}$;
2. w can be covered by $d(w)$ by extending the generative route of $Q^k_{v,t}$ with a shortest walk from t to w, that is $d(w) \geq c(Q^k_{v,t}) + \omega^*_{t,w}$;
3. call t' a target visited by a shortest path from t to w, if $t' \notin Q^k_{v,t}$ then it cannot be covered, that is $d(t') < c(Q^k_{v,t}) + \omega^*_{t,t'}$.

Conditions 1 and 2 require any expansion to form a new covering set consistent with Definition 4.6, thus ensuring the algorithm's soundness. Condition 3 requires $Q^k_{v,t}$ to be a *proper* descriptor of its generative route, meaning that such route, once walked, covers uniquely the targets included in $Q^k_{v,t}$ and not targets outside it. This last requirement is an operational choice to reduce the number of expansions made in each iteration of the algorithm. Consider for instance a graph with degree bounded by 3, Condition 3 allows us to generate in the worst case 3 new covering sets at each expansion round instead of $|T|$. Notice that under Condition 3 the algorithm can still generate any maximal covering set.

Lemma 4.5. *Any maximal covering set can be generated from the empty set with a sequence of admissible expansions.*

Proof sketch. Consider an expansion which, by adding a target w, violates Condition 3 for a target t'. Such expansion can always be split in two admissible ones. The first adding t', the second adding w. The same rationale works for multiple expansions and multiple t' and clearly applies to maximal covering sets which are a subset of covering sets. □

Condition 3 can be exploited to reduce the required compute time but, rigorously speaking, it presents a drawback. To establish if target w can be added to set $Q^k_{v,t}$ the algorithm needs to check every shortest path from t to w, and such paths can be, in the worst case, exponentially many. We can cope with this by adopting the following simplification. We fix a set of canonical shortest paths by running the Floyd-Warshall algorithm. Then, given t and w, we fetch the canonical shortest path between them and we check Condition 3 assuming that such path is unique. If the condition is not verified under this assumption then it is also not verified in its original

definition. If otherwise, the condition is verified, then the algorithm (assuming validity of Conditions 1 and 2) might obtain a covering set $\bar{Q}^{k+1}_{v,w}$ which is not a proper one, meaning that by walking its generative route at least one target $t' \notin \bar{Q}^{k+1}_{v,w}$ gets covered. Let us assume that this is the case and, w.l.o.g., that t' is the only invalidating target. The algorithm's current solution representation (the set $\bar{Q}^{k+1}_{v,w}$) would then be inconsistent with the actual solution (the generative route). By Lemma 4.5 though, we know that the algorithm would generate also set $Q^{k+2}_{v,w}$ making two admissible expansions with t' and w to $Q^{k}_{v,t}$. Since $Q^{k+2}_{v,w} = \bar{Q}^{k+1}_{v,w} \cup \{t'\}$ the non proper covering set $\bar{Q}^{k+1}_{v,w}$ is removable by inter-set dominance (Definition 4.5). Obviously this workaround comes at an additional cost: the algorithm unnecessarily generates the set $\bar{Q}^{k+1}_{v,w}$ which under the proper definition of Condition 3 would have been avoided. Still, the above solution turned out very convenient since exponential multiplicity of shortest paths is very unlikely in graphs representing real environments.

In Algorithm 5 we provide full technical details. Covering sets obtained by `DP-ComputeCovSets` are grouped in collections: $\mathcal{C}^{k}_{v,t}$ denotes the collection of all covering sets of cardinality k where the last expansion added target t. After the required initialization steps (Lines 1 and 2) for any generated $Q^{k-1}_{v,t}$ we compute the set of admissible expansions Q^{+} (Line 6) and we apply each one of them (Line 8). In Step 9, we make use of a procedure called `Search`$(Q, \mathcal{C})$ where Q is a covering set and $\mathcal{C}$ is a collection of covering sets. The procedure outputs Q if $Q \in \mathcal{C}$ and $\emptyset$ otherwise. (We adopted an efficient implementation of such procedure which can run in $O(|T(s)|)$. More precisely, we represent a covering set Q as a binary vector of length $|T(s)|$ where the i-th component is set to 1 if target $t_i \in Q$ and 0 otherwise. A collection of covering sets C can then be represented as a binary tree with depth $|T(s)|$. The membership of a covering set Q to collection C is represented with a branch of the tree in such a way that if $t_i \in Q$ then we have a left edge at depth $i-1$ on such branch. We can easily determine if $Q \in \mathcal{C}$ by checking if traversing a left (right) edge in the tree each time we read a 1 (0) in Q's binary vector we reach a leaf node at depth $|T(s)|$. The insertion of a new covering set in the collection can be done in the same way by traversing existing edges and expanding the tree where necessary.) If the newly generated covering set is not present in its collection or is already present with a higher cost (Step 10), then collection and cost are updated (Steps 11 and 12).

After `DP-ComputeCovSets` completed its execution, for any arbitrary $T' \subseteq T$ we can easily obtain the temporal cost of its shortest covering route

Algorithm 5 `DP-ComputeCovSets`(v, s)

1: $\forall t \in T(s)$: $\mathcal{C}^0_{v,v} = \{\emptyset\}$
2: $c(\emptyset) = 0$
3: **for all** $k \in \{1 \dots |T(s)|\}$ **do**
4: **for all** $t \in T(s)$ **do**
5: **for all** $Q^{k-1}_{v,t} \in \mathcal{C}^{k-1}_{v,t}$ **do**
6: $Q^+ = \{w \in T(s) \mid$ Conditions 1-3 are satisfied$\}$
7: **for all** $w \in Q^+$ **do**
8: $Q^k_{v,w} = Q^{k-1}_{v,t} \cup \{w\}$
9: $U = \texttt{Search}(Q^k_{v,w}, \mathcal{C}^k_{v,w})$
10: **if** $c(U) > c(Q^{k-1}_{v,t}) + \omega^*_{t,w}$ **then**
11: $\mathcal{C}^k_{v,w} = \mathcal{C}^k_{v,w} \cup \{Q^k_{v,w}\}$
12: $c(Q^k_{v,w}) = c(Q^{k-1}_{v,t}) + \omega^*_{t,w}$
13: **return** $\{\mathcal{C}^k_{v,t} : t \in T(s), k \leq |T(s)|\}$

as

$$c^*(T') = \min_{Q \in Y_{|T'|}} c(Q)$$

where $Y_{|T'|} = \cup_{t \in T'} \{\texttt{Search}(T', \mathcal{C}^{|T'|}_{v,t})\}$ (notice that if T' is not a covering set then $c^*(T') = \infty$). For the sake of simplicity, Algorithm 5 does not specify how to carry out two sub-tasks we describe in the following.

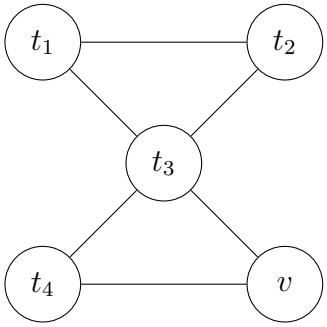
The first one is the *annotation of dominated covering sets*. Each time Steps 11 and 12 are executed, a covering set is added to some collection. Let us call it Q and assume it has cardinality k. Each time a new Q has to be included at cardinality k, we mark all the covering sets at cardinality $k-1$ that are dominated by Q (Definition 4.5). The number of sets that can be dominated is in the worst case $|Q|$ since each of them has to be searched in collection $\mathcal{C}^{k-1}_{v,t}$ for each feasible terminal t and, if found, marked as dominated. The number of terminal targets and the cardinality of Q are at most n and, as described above, the `Search` procedure takes $O(|T(s)|)$. Therefore, dominated covering sets can be annotated with a $O(|T(s)|^3)$ extra cost at each iteration of Algorithm 5. We can only mark and not delete dominated covering sets since they can generate non-dominated ones in the next iteration.

The second task is the *generation of routes*. To do this we maintain a list of generating routes by iteratively appending terminal vertex w to the generative route of $Q^{k-1}_{v,t}$ when set $Q^{k-1}_{v,t} \cup \{w\}$ is included in its corresponding collection. At the end of the algorithm only routes that correspond to non-dominated covering sets are returned. Maintaining such a list introduces a $O(1)$ cost.

Theorem 4.6. *`DP-ComputeCovSets` is an exact algorithm and has worst-case complexity of $O(|T(s)|^2 2^{|T(s)|})$ since it has to compute covering sets up to cardinality $|T(s)|$. With annotations of dominances and routes generation the whole algorithm yields a worst-case complexity of $O(|T(s)|^5 2^{|T(s)|})$.*

Notice that the algorithm is exact since it is based on an enumeration procedure.

Example 16. *Here, we provide a simple example of the execution of the algorithm `DP-ComputeCovSets`. Consider a problem instance with a single signal, arbitrary target values while topology and penetration times are as follow:*



t	$d(t)$
t_1	3
t_2	3
t_3	3
t_4	3

We report the expansions made by the algorithm for increasing cardinalities (value of k) in the following table.

$k=0$	$k=1$	$k=2$	$k=3$
$\emptyset, c=0$	$Q^1_{v,t_3}=\{t_3\}, c=1$	$Q^2_{v,t_1}=\{t_1,t_3\}, c=2$	$Q^3_{v,t_2}=\{t_1,t_2,t_3\}, c=3$ $Q^3_{v,t_4}=\{t_1,t_3,t_4\}, c=4$
		$Q^2_{v,t_2}=\{t_2,t_3\}, c=2$	$Q^3_{v,t_1}=\{t_1,t_2,t_3\}, c=3$ $Q^3_{v,t_4}=\{t_2,t_3,t_4\}, c=4$
		$Q^2_{v,t_4}=\{t_3,t_4\}, c=2$	$Q^3_{v,t_1}=\{t_1,t_3,t_4\}, c=4, \square$ $Q^3_{v,t_2}=\{t_2,t_3,t_4\}, c=4, \lozenge$
	$Q^1_{v,t_4}=\{t_4\}, c=1$	$Q^2_{v,t_3}=\{t_3,t_4\}, c=2$	$Q^3_{v,t_1}=\{t_1,t_3,t_4\}, c=3, \blacksquare$ $Q^3_{v,t_2}=\{t_2,t_3,t_4\}, c=3, \blacklozenge$

Notice that Constraint 3 intervenes both when expanding covering sets with $k=1$ and $k=2$. In the table, c indicates the temporal cost of the relative covset while the covset marked with $\blacklozenge$ dominates the one marked with $\lozenge$ while the covset marked with $\blacksquare$ dominates the one marked with $\square$.

Approximation Algorithm

The dynamic programming algorithm presented in the previous section cannot be directly adopted to approximate the maximal covering routes. We notice that even in the case we introduce a logarithmic upper bound over

the size of the covering sets generated by Algorithm 5, we could obtain a number of routes that is $O(2^{\log^2(|T(s)|)})$, and therefore exponential. Thus, our goal is to design a polynomial-time algorithm that generates a polynomial number of *good* covering routes. We observe that if we have a total order over the vertices and we work over a complete graph of the targets where each edge corresponds to the shortest path, we can find in polynomial time the maximal covering routes subject to the constraint that, given any pair of targets t, t' in a route, t can precede t' in the route only if t precedes t' in the order. We call *monotonic* a route satisfying a given total order. Algorithm 6 returns the maximal monotonic covering routes when the total order is lexicographic (trivially, in order to change the order, it is sufficient to re-label the targets).

`MonotonicLongestRoute` is based on dynamic programming and works as follows. $R(k,l)$ is a matrix storing in each cell one route, while $L(k,l)$ is a matrix storing in each cell the maximum lateness of the corresponding route (see below for the meaning of k and l). The maximum lateness of a route r captures the maximum delay between a target's first visit and its deadline. Formally, it is defined as $\max_{t \in T(s)} A_r(t) - d(t)$. The route stored in $R(k,l)$ is the one with the minimum lateness among all the monotonic ones covering l targets where t_k is the first visited target. Thus, basically, when $l = 1$, $R(k,l)$ contains the route $\langle v, t_k \rangle$, while, when $l > 1$, $R(k,l)$ is defined appending to $R(k,1)$ the best (in terms of minimizing the maximum lateness) route $R(k', l-1)$ for every $k' > k$, in order to satisfy the total order. The whole set of routes in R are returned.[7] The complexity of Algorithm 6 is $O(|T(s)|^3)$, except the time needed to find all the shortest paths.

We use different total orders (breaking ties randomly) over the set of targets, collecting all the routes generated using each total order:

- increasing order $\omega^*_{v,t}$: the rationale is that targets close to v will be visited before targets far from v;
- increasing order $d(t)$: the rationale is that targets with short deadlines will be visited before targets with long deadlines;
- increasing order $d(t) - \omega^*_{v,t}$ (we call this quantity *excess time*): the rationale is that targets with short excess time will be visited before targets with long excess time.

In addition, we use a a random restart generating random permutations over the targets.

[7] We notice that dominance can be applied to discard dominated routes. However, in this case, the improvement would be negligible since the total number of routes, including the non-dominated ones, is polynomial.

Algorithm 6 (v, s)

$\forall k, k' \in \{1, 2, \ldots, |T(s)|\}, R(k, k') = \emptyset, L(k, k') = +\infty, C_R(k) = \emptyset, C_L(k) = +\infty$
for all $\forall k \in \{|T(s)|, |T(s)| - 1, \ldots, 1\}$ **do**
  for all $\forall l \in \{1, 2, \ldots, |T(s)|\}$ **do**
    if $l = 1$ **then**
      $R(k, l) = \langle v, t_k \rangle$
      $L(k, l) = \omega^*_{v,t_k} - d(t_k)$
    else
      for all k' s.t. $|T(s)| \geq k' > |T(s)| - k$ **do**
        $C_R(k') = \langle R(k, 1), R(k', l - 1) \rangle$
        $C_L(k') = \max\{L(k, 1), \omega^*_{v,t_k} + \omega^*_{t_k,k'} - \omega^*_{v,k'} + L(k', l - 1)\}$
      $j = \arg\min_j \{C_L(j)\}$
      if $C_L(j) \leq 0$ **then**
        $R(k, l) \leftarrow C_R(j)$
        $L(k, l) \leftarrow C_L(j)$
return R

Theorem 4.7. *`MonotonicLongestRoute` provides an approximation with ratio* $\Omega(\frac{1}{|T(s)|})$.

Proof sketch. The worst case for the approximation ratio of our algorithm occurs when the covering route including all the targets exists and each covering route returned by our heuristic algorithm visits only one target. In that case, the optimal expected utility of $\mathcal{D}$ is 1. Our algorithm, in the worst case in which $\pi(t) = 1$ for every target t, returns an approximation ratio $\Omega(\frac{1}{|T(s)|})$. It is straightforward to see that, in other cases, the approximation ratio is larger. □

4.3.3 Branch-and-bound Algorithms

The dynamic programming algorithm presented in the previous section essentially implements a breadth-first search. In some specific situations, depth-first search could outperform breadth-first search, e.g., when penetration times are relaxed and good heuristics lead a depth-first search to find in a brief time the maximal covering route, avoiding to scan an exponential number of routes as the breadth-first search would do. In this section, we adopt the branch-and-bound approach to design both an exact algorithm and an approximation algorithm. In particular, in Section 4.3.3 we describe our exact algorithm, while in Section 4.3.3 we present the approximation one.

Exact Algorithm

Our branch-and-bound algorithm (see Algorithm 7) is a tree-search based algorithm working on the space of the covering routes and returning a set of covering routes R. It works as follows.

Initial step. We exploit two global set variables, CL_{min} and CL_{max} initially set to empty (Steps 1-2 of Algorithm 7). These variables contain *closed* covering routes, namely covering routes which cannot be further expanded without violating the penetration time of at least one target during the visit. CL_{max} contains the covering routes returned by the algorithm, while CL_{min} is used for pruning as discussed below. Given a starting vertex v and a signal s, for each target $t \in T(s)$ such that $\omega^*_{v,t} \leq d(t)$ we generate a covering route $r = \langle v, t \rangle$ with $r(0) = v$ and $r(1) = t$ (Steps 5 of `Branch-and-Bound`). Thus, $\mathcal{D}$ has at least one covering route per target that can be covered in time from v. Notice that if, for some t, such minimal route does not exist, then target t cannot be covered (even the shortest path from the starting vertex v cannot guarantee capture). This does not guarantee that $\mathcal{A}$ will attack t with full probability since, depending on the values π, $\mathcal{A}$ could find more profitable to randomize over a different set of targets. The meaning of parameter ρ (used in Line 5 of `Branch-and-Bound`) is described below.

Algorithm 7 `Branch-and-Bound`(v, s, ρ)

1: $CL_{max} \leftarrow \emptyset$
2: $CL_{min} \leftarrow \emptyset$
3: **for all** $t \in T(s)$ **do**
4: **if** $\omega^*_{v,t} \leq d(t)$ **then**
5: Tree-Search($\lceil \rho \cdot |T(s)| \rceil, \langle v, t \rangle$)
6: **return** CL_{max}

Route expansions. The subsequent steps essentially evolve on each branch according to a depth-first search with backtracking limited by ρ. The choice of ρ directly influences the behavior of the algorithm and consequently its complexity. Each node in the search tree represents a route r built so far starting from an initial route $\langle v, t \rangle$. At each iteration, route r is expanded by inserting a new target at a particular position. We denote with $r^+(q, p)$ the route obtained by inserting target q after the p-th target in r. Notice that every expansion of r will preserve the relative order with which targets already present in r will be visited. The collection of all the feasible expansions r^+s (i.e., the ones that are covering routes) is denoted by R^+ and it is ordered according to a heuristic that we describe below. `Expand`, described below, is used to generate R^+ (Step 1 of `Tree-Search`). In each open branch (i.e.,

$R^+ \neq \emptyset$), if the depth of the node in the tree is smaller or equal to $\lceil \rho \cdot |T(s)| \rceil$ then backtracking is disabled (Steps 7-10 of `Tree-Search`), while, if the depth is larger than such value, is enabled (Steps 5-6 of `Tree-Search`). This is equivalent to fix the relative order of the first (at most) $\lceil \rho \cdot |T(s)| \rceil$ inserted targets in the current route. In this case, with $\rho = 0$ we do not rely on the heuristics at all, full backtracking is enabled, the tree is fully expanded and the returned R is complete, i.e., it contains all the non-dominated covering routes. Route r is repeatedly expanded in a greedy fashion until no insertion is possible. As a result, Algorithm 8 generates at most $|T(s)|$ covering routes.

Algorithm 8 `Tree-Search`(k, r)

1: $R^+ = \{r^{(1)}, r^{(2)}, \ldots\} \leftarrow$ Expand(r)
2: **if** $R^+ = \emptyset$ **then**
3: Close(r)
4: **else**
5: **if** $k > 0$ **then**
6: Tree-Search$(k-1, r^{(1)})$
7: **else**
8: **for all** $r^+ \in R^+$ **do**
9: Tree-Search$(0, r^+)$
10: Close(r^+)

Pruning. `Close` is in charge of updating CL_{min} and CL_{max} each time a route r cannot be expanded and, consequently, the associated branch must be closed. We call CL_{min} the *minimal* set of closed routes. This means that a closed route r belongs to CL_{min} only if CL_{min} does not already contain another $r' \subseteq r$. Steps 1-4 of `Close` implement such condition: first, in Steps 2-3 any route r' such that $r' \supseteq r$ is removed from CL_{min}, then route r is inserted in CL_{min}. Routes in CL_{min} are used by Algorithm 10 in Steps 2 and 6 for pruning during the search. More precisely, a route r is not expanded with a target q at position p if there exists a route $r' \in CL_{min}$ such that $r' \subseteq r^+(q, p)$. This pruning rule is safe since by definition if $r' \in CL_{min}$, then all the possible expansions of r' are unfeasible and if $r' \subseteq r$ then r can be obtained by expanding from r'. This pruning mechanism explains why once a route r is closed is always inserted in CL_{min} without checking the insertion against the presence in CL_{min} of a route r'' such that $r'' \subseteq r$. Indeed, if such route r'' would be included in CL_{min} we would not be in the position of closing r, having r being pruned before by `Expand`.

We use CL_{max} to maintain a set of the generated *maximal* closed routes. This means that a closed route r is inserted here only if CL_{max} does not

already contain another r' such that $r' \supseteq r$. This set keeps track of closed routes with maximum number of targets. `Close` maintains this set by inserting a closed route r in Step 12 only if no route $r' \supseteq r$ is already present in CL_{max}. Once the whole algorithm terminates, CL_{max} contains the final solution.

Algorithm 9 `Close`(r)

1: **for all** $r' \in CL_{min}$ **do**
2: **if** $r \subseteq r'$ **then**
3: $CL_{min} = CL_{min} \setminus \{r'\}$
4: $CL_{min} = CL_{min} \cup \{r\}$
5: **for all** $r' \in CL_{max}$ **do**
6: **if** $r \subseteq r'$ **then**
7: **return**
8: $CL_{max} = CL_{max} \cup \{r\}$

Heuristic function. A key component of this algorithm is the heuristic function that drives the search. The heuristic function is defined as $h_r : \{T(s) \setminus T(r)\} \times \{1 \dots |T(r)|\} \rightarrow \mathbb{Z}$, where $h_r(t', p)$ evaluates the cost of expanding r by inserting target t' after the p-th target of r. The basic idea, inspired by [101], is to adopt a conservative approach, trying to preserve feasibility. Given a route r, let us define the *possible forward shift* of r as the minimum temporal margin in r between the arrival at a target t and $d(t)$:

$$PFS(r) = \min_{t \in T(r)}(d(t) - A_r(t))$$

The *extra mileage* $e_r(t', p)$ for inserting target t' after position p is the additional traveling cost to be paid:

$$e_r(t', p) = (A_r(r(t')) + \omega^*_{r(p),t'} + \omega^*_{t',r(p+1)}) - A_r(r(p+1))$$

The *advance time* that such insertion gets with respect to $d(t')$ is defined as:

$$a_r(t', p) = d(t') - (A_r(r(p)) + \omega^*_{r(p),t'})$$

Finally, $h_r(t', p)$ is defined as:

$$h_r(t', p) = \min\{a_r(t', p); (PFS(r) - e_r(t', p))\}$$

We partition the set $T(s)$ in two sets T_{tight} and T_{large} where $t \in T_{tight}$ if $d(t) < \delta \cdot \omega^*_{v,t}$ and $t \in T_{large}$ otherwise ($\delta \in \mathbb{R}$ is a parameter). The previous inequality is a non-binding choice we made to discriminate targets with a tight penetration time from those with a large one. Initially, we insert all the tight targets and only subsequently we insert the non-tight targets. We use the two sets according to the following rules (see Algorithm 10):

Algorithm 10 `Expand`(r)

if $T_{tight} \not\subseteq T(r)$ **then**
  for all $q \in T_{tight} \setminus T(r)$ **do**

$$P_q = \{ p_q^{(1)}, \dots p_q^{(b)} \} \text{ s.t. } \forall i \in \{1, \dots, b\}, \begin{cases} h_r(q, p_q^{(i)}) \geq h_r(q, p_q^{(i+1)}) \\ r^+(q, p_q^{(i)}) \text{ is a covering route} \\ \nexists v' \in CL_{min} : r' \subseteq r^+(q, p_q^{(i)}) \end{cases}$$

$$Q = \{q^{(1)}, q^{(2)}, \dots, q^{(c)}\} \text{ s.t. } \forall i \in \{1, \dots, c\}, h_r(q^{(i)}, p_{q^{(i)}}^{(1)}) \geq h_r(q^{(i+1)}, p_{q^{(i+1)}}^{(1)})$$

$$R^+ = \{r^{(1)}, r^{(2)}, \dots r^{(k)}\} \text{ where } \begin{cases} r^{(1)} & = & r^+(q^{(1)}, p_{q^{(1)}}^{(1)}) \\ \cdots & = & \cdots \\ r^{(k)} & = & r^+(q^{(c)}, p_{q^{(c)}}^{(b)}) \end{cases}$$

if $T_{large} \not\subseteq T(r)$ **then**
  for all $u \in T_{large} \setminus T(r)$ **do**

$$Q_p = \{ q_p^{(1)}, \dots q_p^{(b)} \} \text{ s.t. } \forall i \in \{1, \dots, b\}, \begin{cases} h_r(q_p^{(i)}, p) \geq h_r(q_p^{(i+1)}, p) \\ r^+(q_p^{(i)}, p) \text{ is a covering route} \\ \nexists r' \in CL_{min} : r' \subseteq r^+(q_p^{(i)}, p) \end{cases}$$

$$P = \{p^{(1)}, p^{(2)}, \dots, p^{(c)}\} \text{ s.t. } \forall i \in \{1, \dots, c\}, h_r(q^{(1)}, p_{q^{(1)}}^{(i)}) \geq h_r(q^{(1)}, p_{q^{(1)}}^{(i+1)})$$

$$R^+ = R^+ \cup \{r^{(k+1)}, r^{(k+2)}, \dots r^{(K)}\} \text{ where } \begin{cases} r^{(k+1)} & = & r^+(q_p^{(1)}, p^{(1)}) \\ \cdots & = & \cdots \\ r^{(K)} & = & r^+(q_p^{(b)}, p^{(c)}) \end{cases}$$

return R^+

- the insertion of a target belonging to T_{tight} is always preferred to the insertion of a target belonging to T_{large}, independently of the insertion position;
- insertions of $t \in T_{tight}$ are ranked according to h considering first the insertion position and then the target;
- insertions of $t \in T_{large}$ are ranked according to h considering first the target and then the insertion position.

The rationale behind this rule is that targets with a tight penetration time should be inserted first and at their best positions. On the other hand, targets with a large penetration time can be covered later. Therefore, in this last case, it is less important which target to cover than when to cover it.

Theorem 4.8. `Branch-and-Bound` *with* $\rho = 0$ *is an exact algorithm and has an exponential computational complexity since it builds a full tree of covering routes with worst-case size* $O(|T(s)|^{|T(s)|})$.

Approximation Algorithm

Since ρ determines the *completeness degree* of the generated tree, we can exploit `Branch-and-Bound` tuning ρ to obtain an approximation algorithm that is faster w.r.t. the exact one.

In fact, when $\rho < 1$ completeness is not guaranteed in favor of a less computational effort. In this case, the only guarantees that can be provided for each covering route $r \in CL_{max}$, once the algorithm terminates are:

- no other $r' \in CL_{max}$ dominates r;
- no other $r' \notin CL_{max}$ such that $r \subseteq r'$ dominates r. Notice this does not prevent the existence of a route r'' not returned by the algorithm that visits targets $T(r)$ in a different order and that dominates r.

When ρ is chosen as $\frac{k}{|T(s)|}$ (where $k \in \mathbb{N}$ is a parameter), the complexity of generating covering routes becomes polynomial in the size of the input. We can state the following theorem, whose proof is analogous to that one of Theorem 4.7.

Theorem 4.9. *`Tree-Search` with $\rho = \frac{k}{|T(s)|}$ provides an approximation with ratio $\Omega(\frac{1}{|T(s)|})$ and runs in $O(|T(s)|^3)$ given that heuristic h_r can be computed in $O(|T(s)|^2)$.*

4.3.4 Solving SRG-v

Now we can formulate the problem of computing the optimal signal response strategy for $\mathcal{D}$. Let us denote with $\sigma^{\mathcal{D}}_{v,s}(r)$ the probability with which $\mathcal{D}$ plays route r under signal s and with $R_{v,s}$ the set of all the routes available to $\mathcal{D}$ generated by some algorithm. We introduce function $U_{\mathcal{A}}(r,t)$, representing the utility function of $\mathcal{A}$ and defined as follows:

$$U_{\mathcal{A}}(r,t) = \begin{cases} \pi(t) & \text{if } t \notin r \\ 0 & \text{otherwise} \end{cases}.$$

The best $\mathcal{D}$'s strategy (i.e., the maxmin strategy) can be found by solving the following linear mathematical programming problem:

$$\begin{aligned} & & \min \quad & g_v & \text{s.t.} \\ \sum_{s \in S(t)} p(s \mid t) \sum_{r \in R_{v,s}} & \sigma^{\mathcal{D}}_{v,s}(r) U_{\mathcal{A}}(r,t) \leq g_v & & \forall t \in T \\ & \sum_{r \in R_{v,s}} \sigma^{\mathcal{D}}_{v,s}(r) = 1 & & \forall s \in S \\ & \sigma^{\mathcal{D}}_{v,s}(r) \geq 0 & & \forall r \in R_{v,s}, s \in S \end{aligned}$$

The size of the mathematical program is composed of $|T|+|S|$ constraints (excluded ≥ 0 constraints) and $O(|V||S| \max_{v,s}\{|R_{v,s}|\})$ variables. This shows that the hardness is due only to $\max_{v,s}\{|R_{v,s}|\}$, which, in its turn, depends only on $|T(s)|$. We provide the following remark.

Remark 3. *We observe that the discretization of the environment as a graph is as accurate as the number of vertices is large, corresponding to reduce the size of the areas associated with each vertex, as well as to reduce the temporal interval associated with each turn of the game. Our algorithms show that increasing the accuracy of the model in terms of number of vertices requires polynomial time.*

4.4 Patrolling Game

In this section, we focus on the PG. Specifically, in Section 4.4.1 we state our main result showing that patrolling is not necessary when an alarm system is present, in Section 4.4.2 we propose the algorithm to deal with the PG, in Section 4.4.3 we summarize the complexity results about Questions 1-4.

4.4.1 Stand Still

We focus on the problem of finding the best patrolling strategy given that we know the best signal response strategy for each vertex v in which $\mathcal{D}$ can place. Given the current vertex of $\mathcal{D}$ and the sequence of the last, say n, vertices visited by $\mathcal{D}$ (where n is a tradeoff between effectiveness of the solution and computational effort), a patrolling strategy is usually defined as a randomization over the next adjacent vertices [22]. We define $v^* = \arg\min_{v \in V}\{g_v\}$, where g_v is the value returned by the optimization problem described in Section 4.3.4, as the vertex that guarantees the maximum expected utility to $\mathcal{D}$ over all the SRG-vs. We show that the maxmin equilibrium strategy in PG prescribes that $\mathcal{D}$ places at v^*, waits for a signal, and responds to it.

Theorem 4.10. *Without false positives and missed detections, if* $\forall t \in T$ *we have that* $|S(t)| \geq 1$*, then any patrolling strategy is dominated by the placement in* v^*.

Proof. Any patrolling strategy different from the placement in v^* should necessarily visit a vertex $v' \neq v^*$. Since the alarm system is not affected by missed detections, every attack will raise a signal which, in turn, will raise a response yielding an utility of g_x where x is the current position of $\mathcal{D}$ at the moment of the attack. Since $\mathcal{A}$ can observe the current position of $\mathcal{D}$

before attacking, $x = \arg\max_{v \in P}\{g_v\}$ where P is the set of the vertices patrolled by $\mathcal{D}$. Obviously, for any $P \supseteq \{v^*\}$ we would have that $g_x \geq g_{v^*}$ and therefore placing at v^* and waiting for a signal is the best strategy for $\mathcal{D}$. □

The rationale is based upon the fact that, without false positives and missed detections, the signal response strategy solely depends on the current vertex occupied by $\mathcal{D}$ and on the generated signal (not depending then on previously visited vertices.) So, if the patrolling strategy of $\mathcal{D}$ prescribes to patrol a set of vertices, say V', then, since $\mathcal{A}$ can observe the position of $\mathcal{D}$, the best strategy of $\mathcal{A}$ is to wait for $\mathcal{D}$ being in $v' = \arg\max_{v \in V'}\{g_v\}$ and then to attack. Thus, by definition of g_{v^*}, if $\mathcal{D}$ leaves v^* to patrol additional vertices the expected utility it receives is no larger than that it receives from staying in v^*.

The validity of Theorem 4.10 goes beyond the model studied in this work. Indeed, it can be easily shown how such results keep valid when having multiple resource controlled by $\mathcal{D}$. Also, a deeper analysis of Theorem 4.10 can show that its scope does include cases where missed detections are present up to a non-negligible extent. For such cases, placement-based strategies keep being optimal even in the case when the alarm systems fails in detecting an attack. We encode the occurrence of this robustness property in the following proposition, which we shall prove by a series of examples.

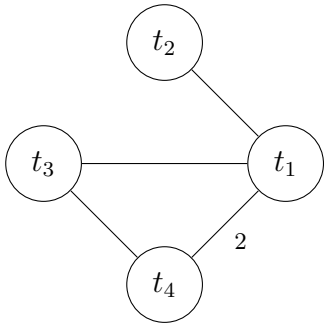
Proposition 4.11. *There exist Patrolling Games where staying in a vertex, waiting for a signal, and responding to it is the optimal patrolling strategy for $\mathcal{D}$ even with a missed detection rate $\alpha = 0.5$.*

Proof. The expected utility for $\mathcal{D}$ given by the placement in v^* is $(1-\alpha)(1-g_{v^*})$, where $(1-\alpha)$ is the probability with which the alarm system correctly generates a signal upon an attack and $(1-g_{v^*})$ denotes $\mathcal{D}$'s payoff when placed in v^*. A non-placement-based patrolling strategy will prescribe, by definition, to move between at least two vertices. From this simple consideration, we observe that an upper bound to the expected utility of any non-placement strategy is entailed by the case where $\mathcal{D}$ alternately patrols vertices v^* and v_2^*, where v_2^* is the second best vertex in which $\mathcal{D}$ can statically place. Such scenario gives us an upper bound over the expected utility of non-placement strategies, namely $1 - g_{v_2^*}$. It follows that a sufficient condition for the placement in v^* being optimal is given by the following inequality:

$$(1-\alpha)(1-g_{v^*}) > (1-g_{v_2^*}). \tag{4.1}$$

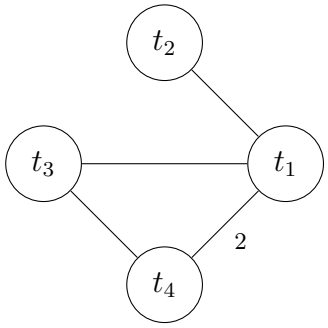
To prove Proposition 4.11, it then suffices to provide a Patrolling Game

instance where Equation 4.1 holds under some non-null missed detection rate α. In Figure 4.4(a) and Figure 4.4(b), we report two of such examples. The depicted settings have unitary edges except where explicitly indicated. For both, without missed detections, the best patrolling strategy is a placement $v^* = 4$. When allowing missed detections, in Figure 4.4(a) it holds that $g_{v^*} = 0$ and $g_{v_2^*} = 0.75$, where $v^* = 4$ and $v_2^* = 1$. Thus, by Equation 4.1, placement $v^* = 4$ is the optimal strategy for $\alpha \leq 0.25$. Under the same reasoning scheme, in Figure 4.4(b) we have that $g_{v^*} = 0$ and $g_{v_2^*} = 0.5$, making the placement $v^* = 4$ optimal for any $\alpha \leq 0.5$. □



t	$\pi(t)$	$d(t)$	$p(s_1 \mid t)$
t_1	0.5	1	1.0
t_2	0.5	3	1.0
t_3	0.5	2	1.0
t_4	0.5	2	1.0

(a) Equation 4.1 holds for $\alpha \leq 0.25$.



t	$\pi(t)$	$d(t)$	$p(s_1 \mid t)$
t_1	1.0	1	1.0
t_2	1.0	3	1.0
t_3	1.0	2	1.0
t_4	1.0	2	1.0

(b) Equation 4.1 holds for $\alpha \leq 0.5$.

Figure 4.4: *Two examples proving Proposition 4.11.*

It is reasonable to expect that a similar result holds also for the case with false positives. However, dealing with false positives is much more intricate than handling false negative and requires new models. For example, $\mathcal{D}$ could respond to an alarm signal only with a given probability and with the remaining probability could stay in the current vertex. For this reason, we leave the treatment of false positives and a more accurate treatment of false negatives to future works.

4.4.2 Computing the Best Placement

Under the absence of false positives and missed detections, Theorem 4.10 simplifies the computation of the patrolling strategy by reducing it to the problem of finding v^*. To such aim, we must solve a SRG-v for each possible starting vertex v and select the one with the maximum expected utility for $\mathcal{D}$. Algorithm 11 depicts the solving algorithm. Function `SolveSRG`(v) returns the optimal value $1 - g_{v^*}$. The complexity is linear in $|V|$, once g_v has been calculated for every v.

Algorithm 11 `BestPlacement`(G, s)

1: $U(v) \leftarrow 0$ for every $v \in V$
2: **for all** $v \in V$ **do**
3: $\quad U(v) \leftarrow$ `SolveSRG`(v)
4: **return** $\max(U)$

Since all the vertices are possible starting points, we should face this hard problem (see Theorem 4.1) $|V|$ times, computing, for each signal, the covering routes from all the vertices. To avoid this issue, we ask whether there exists an algorithm that in the worst case allows us to consider a number of iterations such that solving the problem for a given starting vertex v could help us finding the solution for another starting vertex v'. In other words, considering a specific set of targets, we wonder whether a solution for SET with starting vertex v can be used to derive, in polynomial time, a solution to COV-SET for another starting vertex v'. This would allow us to solve an exponential-time problem only once instead of solving it for each vertex of the graph. To answer this question, we resort to hardness results for reoptimization, also called *locally modified* problems [31]. We show that, even if we know all the covering routes from a starting vertex, once we changed the starting vertex selecting an adjacent one, finding the covering routes from the new starting vertex is hard.

Definition 4.10 (LM-COV-ROUTE)**.** *A locally modified covering route (*LM-*COV-ROUTE) problem is defined as follows:*
INSTANCE: graph $G = (V, E)$, a set of targets T with penetration times d, two starting vertices v_1 and v_2 that are adjacent, and a covering route r_1 with $r_1(0) = v_1$ such that $T(r_1) = T$.
QUESTION: is there r_2 with $r_2(0) = v_2$ and $T(r_2) = T$?

Theorem 4.12. LM-*COV-ROUTE is* NP*-complete.*

This shows that iteratively applying Algorithm 5 to SRG-v for each starting vertex v and then choosing the vertex with the highest utility is the best we can do in the worst case.

4.4.3 Summary of Results

We summarize our computational results about Questions 1-4 in Table 4.1, including also results about the resolution of the PG. We use '?' for the problems remained open.

Question \ Topology	Tree	Arbitrary
Question 1	FNP-hard	APX-hard
Question 2	NP-hard	NP-hard
Question 3	NP-hard	NP-hard
Question 4	NP-hard	NP-hard
Question 2 reoptimization	?	NP-hard

Table 4.1: *Computational complexity of discussed questions.*

4.5 Experimental Evaluation

We implemented our algorithms in MATLAB and we used a 2.33GHz LINUX machine to run our experiments. For a better analysis, we provide two different experimental evaluations. In Section 4.5.1, we apply our algorithms to worst-case instances, in order to evaluate the worst-case performance of the algorithms and to investigate experimentally the gap between our APX-hardness result and the theoretical guarantees of our approximation algorithms. In Section 4.5.2, we apply our algorithms to a specific realistic instance we mentioned at the beginning of the chapter, Expo 2015.

4.5.1 Worst-case Instances Analysis

We evaluate the scalability of Algorithm 5 and the quality of the solution returned by our approximation algorithms for a set of instances of SRG-v. We do not include results on the evaluation of the algorithm to solve completely a PG, given that it trivially requires asymptotically $|V|$ times the effort required by the resolution of a single instance of SRG-v. In the next section we describe our experimental setting, in Section 4.5.1 we provide a quantitative analysis of the exact algorithms while in Section 4.5.1 we evaluate the quality of our approximations.

Setting

We build hard instances for our problem from instances of HAMILTONIAN-PATH (HP). More precisely, these worst-case instances can be easily reduced from any instance of (HP) showing how they admit a covering route for all the targets if and only if the corresponding instance of HP admits an Hamiltonian path. They are defined as follows:

- all the vertices are targets,
- edge costs are set to 1,
- there is only one signal,
- penetration times are set to $|T| - 1$,
- values are drawn from $(0, 1]$ with uniform probability for all the targets,
- the number of edges is drawn from a normal distribution with mean ϵ, said *edge density* and defined as $\epsilon = |E| / \frac{|T|(|T|-1)}{2}$, and
- starting vertex v is drawn among the targets of T with uniform probability.

We explore two parameter dimensions: the number of targets $|T|$ and the value of edge density ϵ. In particular, we use the following values:

$$|T| \in \{6, 8, 10, 12, 14, 16, 20, 25, 30, 35, 40, 45, 50\},$$
$$\epsilon \in \{0.05, 0.10, 0.25, 0.50, 0.75, 1.00\}.$$

For each combination of values of $|T|$ and ϵ, we randomly generate 100 instances with the constraint that, if $\epsilon \frac{|T|^2}{2} < |T|$, we introduce additional edges in order to assure the graph connectivity. The suitability of our worst-case analysis is corroborated by the results obtained with a realistic setting (see Section 4.5.2) which present hard subproblems characterized by the features listed above.

Exact Algorithms Scalability

We report in Figure 4.5 the compute time (averaged over 100 SRG-v instances) required by our exact dynamic programming algorithm (Algorithm 5), with the annotation of dominated covering sets and the generation of the routes, as $|T|$ and ϵ vary. It can be observed that the compute times are exponential in $|T|$, the curves being lines in a semilogarithmic plot, and the value of ϵ determines the slope of the line. Notice that with $\epsilon \in \{0.05, 0.10, 0.25\}$

the number of edges is almost the same when $|T| \leq 16$ due to the constraint of connectivity of the graph, leading thus to the same compute times. Beyond 16 targets, the compute times of our exact dynamic programming algorithm are excessively long (with only $\epsilon = 0.25$, the compute time when $|T| = 20$ is lower than10^4 seconds). Interestingly, the compute time monotonically decreases as ϵ decreases. This is thanks to the fact that the number of covering sets generated by Algorithm 5 dramatically reduces as ϵ reduces.

We do not report any plot of the compute times of our exact branch-and-bound algorithm, since it requires more than 10^4 seconds when $|T| > 8$ even with $\epsilon = 0.25$, resulting thus non-applicable in practice. This is because the branch-and-bound algorithm has a complexity $O(|T|^{|T|})$, while the dynamic programming algorithm has a complexity $O(2^{|T|})$.

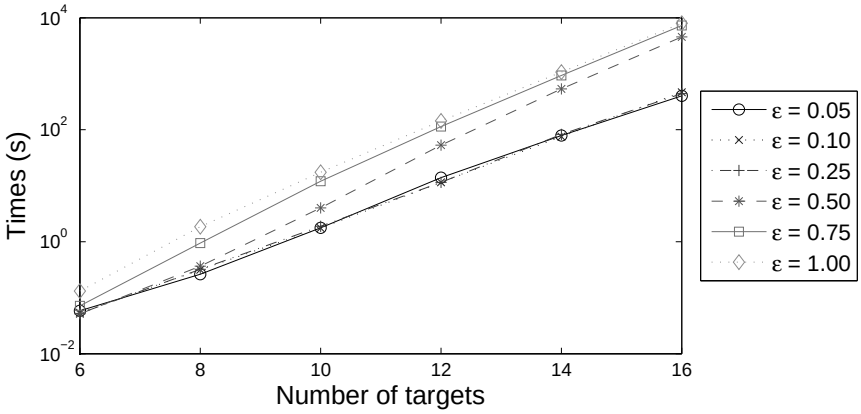


Figure 4.5: *Compute times in seconds of our exact dynamic programming algorithm, `DP-ComputeCovSets`, with the annotation of dominated covering sets and the generation of the routes, as $|T|$ and ϵ vary.*

In Figure 4.6, we report the boxplots of the results depicted in Figure 4.5. They show that the variance of the compute times drastically reduces as ϵ increases. This is because the number of edges increases as ϵ increases and so the number of covering sets increases approaching $2^{|T|}$. On the other hand, with small values of ϵ, the number of covering sets of different instances can be extremely different.

Figure 4.7 shows the impact of discarding dominated actions from the game when $\epsilon = 0.25$. It depicts the trend of some performance ratios for different metrics. We shall call $\mathcal{G}$ the complete game including all $\mathcal{D}$'s dominated actions and $\mathcal{G}_R$ the reduced game; CCS will denote the full version of

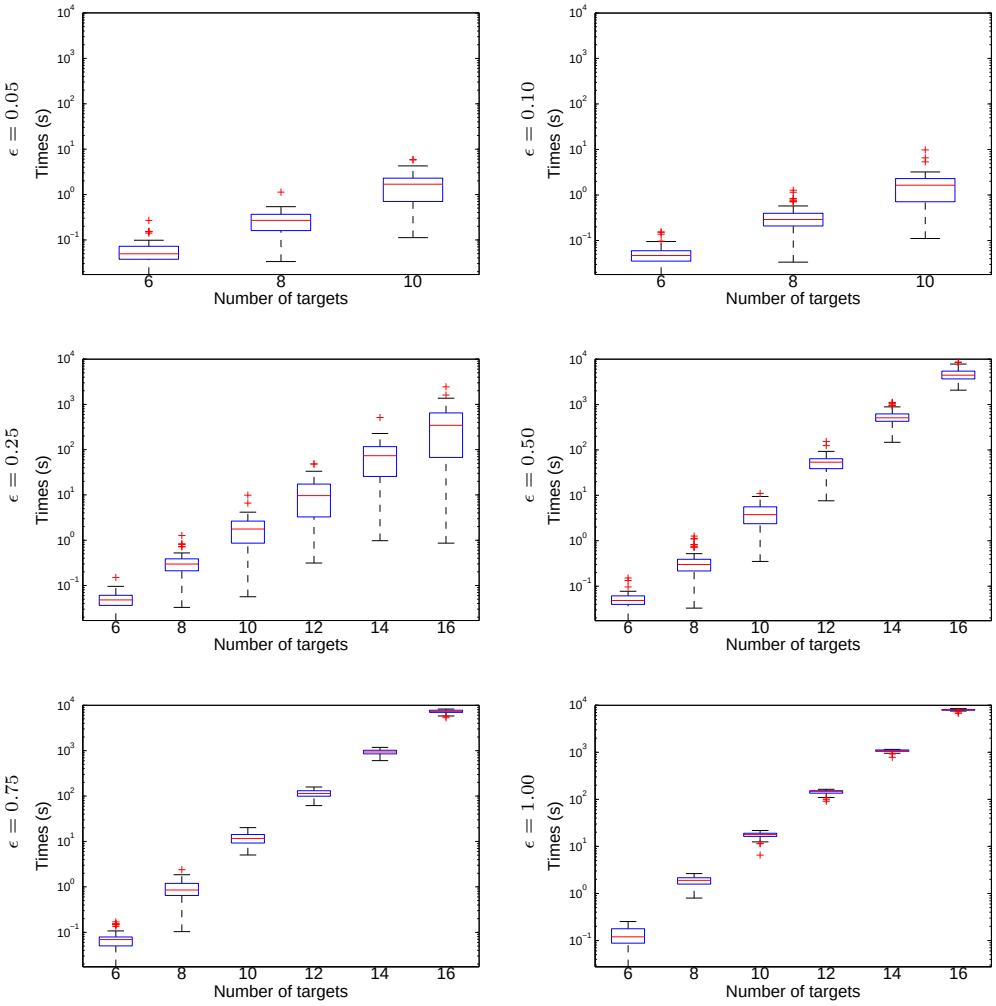


Figure 4.6: *Boxplots of compute times required by our exact dynamic programming algorithm.*

Algorithm 5 and LP will denote the linear program to solve SRG-v. Each instance is solved for a random starting vertex v; we report average ratios for 100 instances. "n. covsets" is the ratio between the number of covering sets in $\mathcal{G}_R$ and in $\mathcal{G}$. As it can be seen, non-dominated actions constitute a small percentage, decreasing with the number of targets. This result indicates that the structure of the problem exhibits a non-negligible degree of redundancy. LP times (iterations) report the ratio between $\mathcal{G}_R$ and $\mathcal{G}$ for the time (iterations) required to solve the maxmin linear program. A relative gain directly proportional to the percentage of dominated covering sets is observable (LP has less variables and constraints). A similar trend is not visible when considering the same ratio for the total time, that is LP+CCS. Indeed, the time needed by CCS largely exceed LP's and removal of dominated actions determines a polynomial additional cost, which can be seen in the slightly increasing trend of the curve. The relative gap between LP and CCS compute times can be assessed by looking at the LP/CCS curve: when more targets are considered the time taken by LP is negligible w.r.t. CCS's. This shows that removing dominated actions is useful, allowing a small improvement in the average case, and assuring an exponential improvement in the worst case.

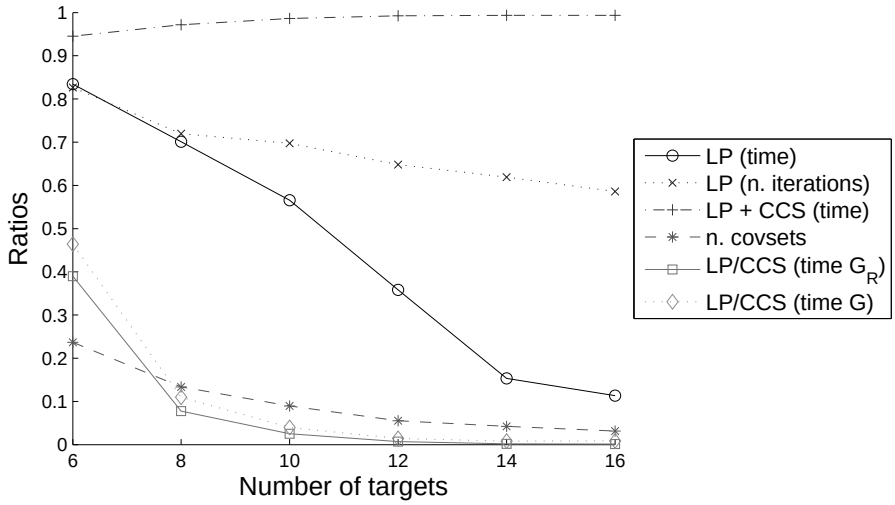


Figure 4.7: *Ratios evaluating dominances with $\epsilon = 0.25$ as $|T|$ varies.*

Figure 4.8 shows the game value for $\mathcal{D}$, $1 - g_v$, as $|T|$ and ϵ vary (averaged over 100 instances). It can be observed that the game value is almost

constant as $|T|$ varies for $\epsilon \in \{0.05, 0.10, 0.25\}$ and it is about 0.87. This is because all these instances have a similar number of edges, very close to the minimum number necessary for having connected graphs. With a larger number of edges, the game value increases. Interestingly, fixed a value of ϵ, there is a threshold of $|T|$ such that beyond the threshold the game value increases as $|T|$ increases. This suggests that the minimum game value is obtained for connected graphs with the minimum number of edges.

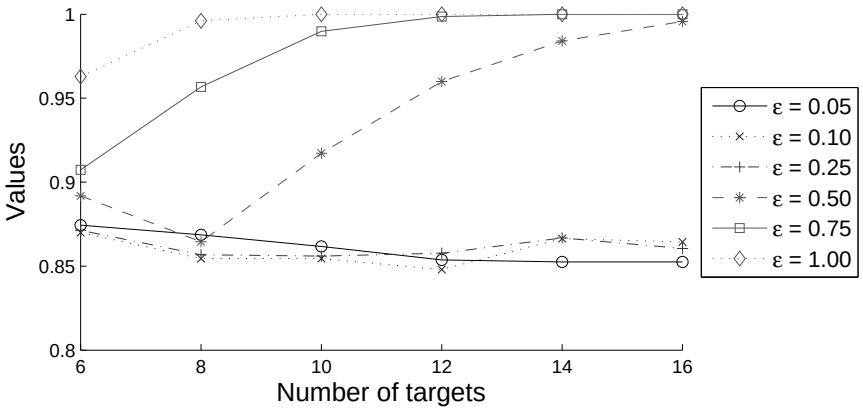


Figure 4.8: *Optimal game values as $|T|$ and ϵ vary.*

In Table 4.2, we report compute times with multiple signals, where the targets covered by a signal and the probability that a target triggers a signal are randomly chosen according to a uniform distribution. Values are averages over 100 random instances and give insights on the computation effort along the considered dimensions. The results show that the problem is computationally challenging even for a small number of targets and signals.

m \ $\lvert T(s)\rvert$	5	10	15
2	-	17.83	510.61
3	-	33.00	769.30
4	0.55	35.35	1066.76
5	0.72	52.43	1373.32

Table 4.2: *Compute times (in seconds) for multi-signal instances.*

Approximation Algorithms

We evaluate the empirical approximation ratios obtained with our approximation algorithms as $(1-\hat{g}_v)/(1-g_v)$, where g_v is the expected utility of $\mathcal{A}$ at the equilibrium considering all the covering sets and $\hat{g}_v$ is the expected utility of $\mathcal{A}$ at the equilibrium when covering sets are generated by our heuristic algorithm. We execute our approximation dynamic programming algorithm with a different number, say RandRes, of randomly generated orders from $\{10, 20, 30, 40, 50\}$, in addition to the 3 heuristics discussed in Section 4.3.2. We executed our approximation branch and bound algorithm with constant values of ρ from $\{0.25, 0.50, 0.75, 1.00\}$ (we recall that with $\rho = 1.00$ backtracking is completely disabled).

Figure 4.9 and Figure 4.10 report the empirical approximation ratios (averaged over 100 instances) obtained with our approximation algorithms for different values of $|T| \in \{6, 8, 10, 12, 14, 16\}$, i.e., the instances for which we know the optimal game value, and $\epsilon \in \{0.05, 0.10, 0.25, 0.50, 0.75, 1.00\}$. We remark that the ratios obtained with the approximation branch-and-bound algorithm for some values of ρ are omitted. This is because the compute time needed by the algorithm is over 10^4 seconds. The algorithm always terminates by the deadline for only $\rho \in \{0.75, 1.00\}$.

We focus on the ratios obtained with the dynamic programming algorithm. Given a value of ϵ, as $|T|$ increases, the ratio decreases up to a given threshold of $|T|$ and then it is a constant. The threshold increases as ϵ decreases, while the constant decreases as ϵ decreases. The value of the constant is high for every ϵ, being larger than 0.8. Although the ratios increase as RandRes increases, it is worth noting that the increase is not very significant, being of the order of 0.05 between 10 RandRes and 50 RandRes. We focus on the ratios obtained with the branch-and-bound algorithm. Given a value of ϵ, as $|T|$ increases, the ratio decreases up to a given threshold of $|T|$ and then it increases approaching a ratio of 1. The threshold increases as ϵ decreases, while the minimum ratio decreases as ϵ decreases. Interestingly, ratios with $\rho = 1.00$ are very close to ratios with $\rho = 0.75$, showing that performing even significant backtracking around the solution found with $\rho = 1.00$ does not lead to a significant improvement of the solution. The solution can be effectively improved only with $\rho = 0.25$, but it is not affordable due to the excessive required compute time. This shows that the heuristic performs very well. Comparing the ratios of the two algorithms, it can be observed that the approximation dynamic programming algorithm performs better than the approximation branch-and-bound algorithm although this last one turned out to be slightly better in a limited number of cases (see

Figure 4.11). While the dynamic programming one always provides a ratio larger than 0.8, the branch-and-bound one provides for combinations of $|T|$ and ϵ ratios lower than 0.4.

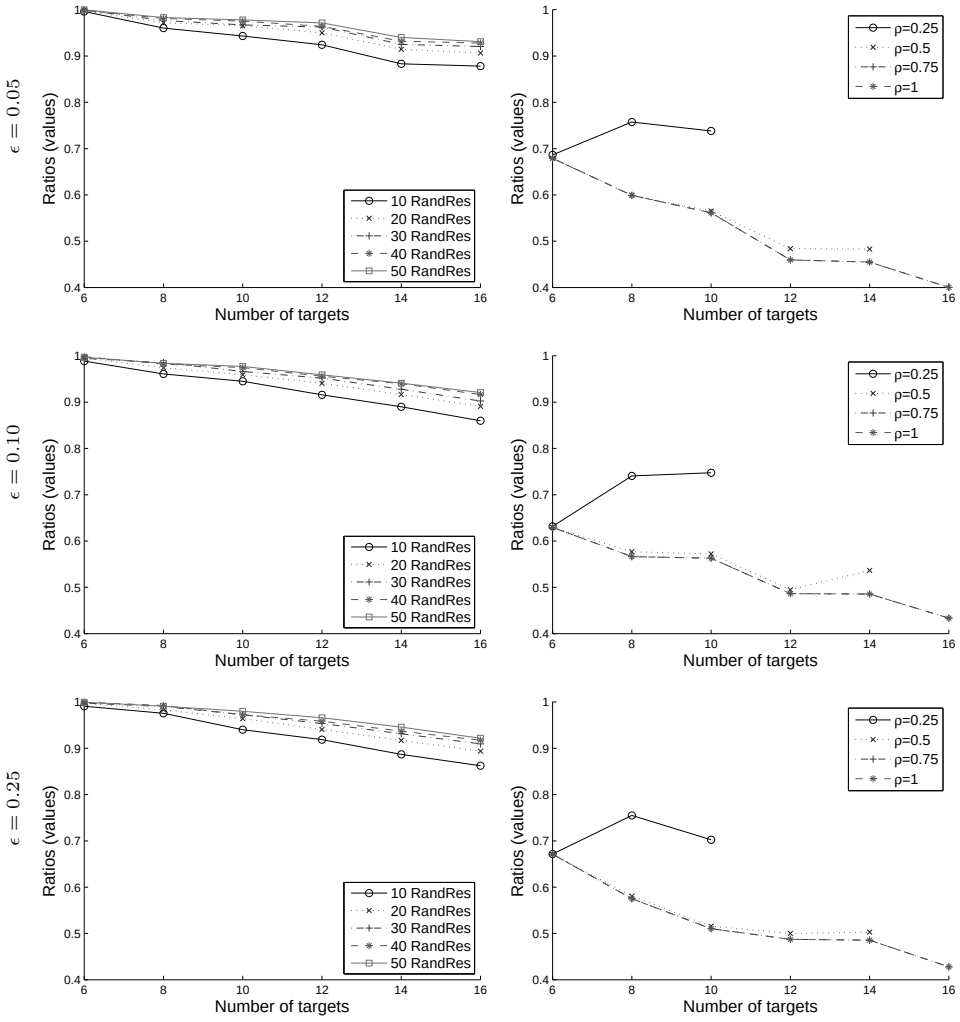


Figure 4.9: *Approximation ratios as* $|T|$ *varies,* $\epsilon \in \{0.05, 0.10, 0.25\}$.

Figure 4.11 reports game values obtained with the approximation dynamic programming algorithm for every value of RandRes with the approximation branch-and-bound algorithm when $|T| \in \{20, 25, 30, 35, 40, 45, 50\}$ only for $\rho = 1.00$. Indeed, with $\rho = 0.75$ the compute time is exces-

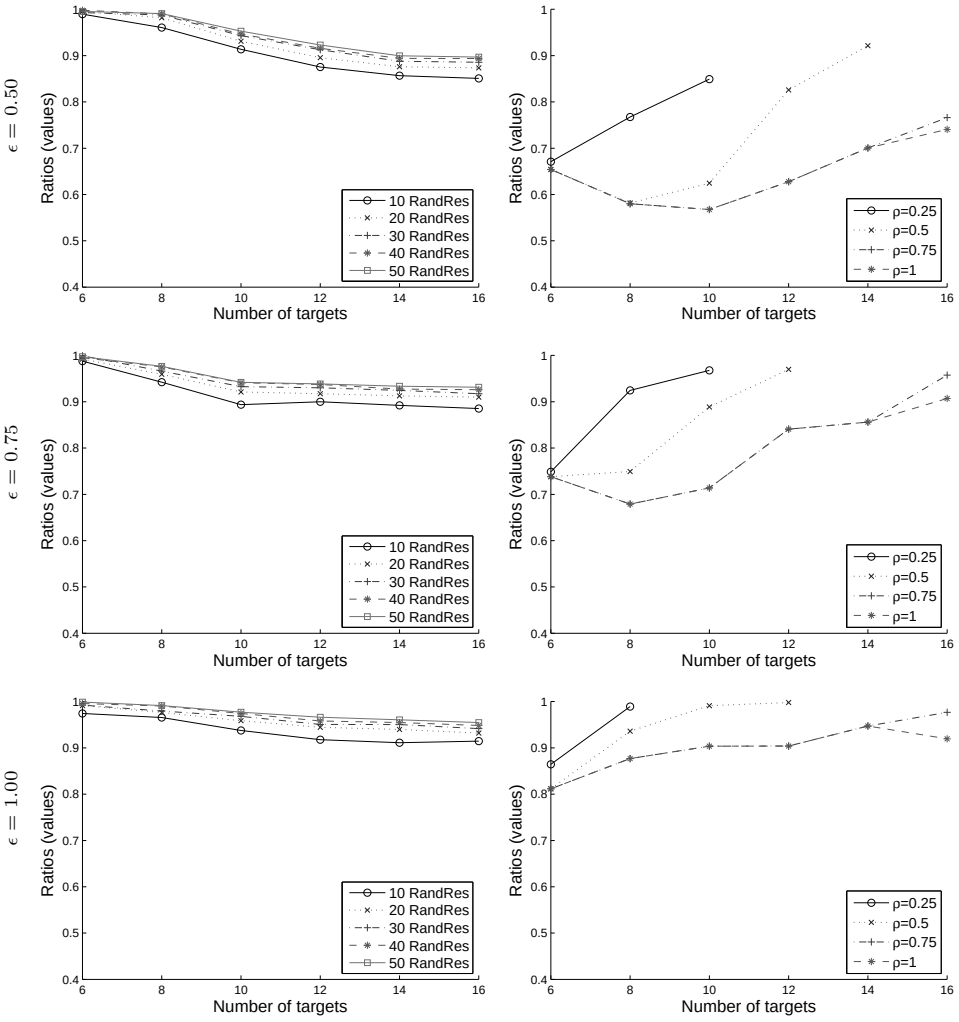


Figure 4.10: *Approximation ratios as* $|T|$ *varies,* $\epsilon \in \{0.50, 0.75, 1.00\}$.

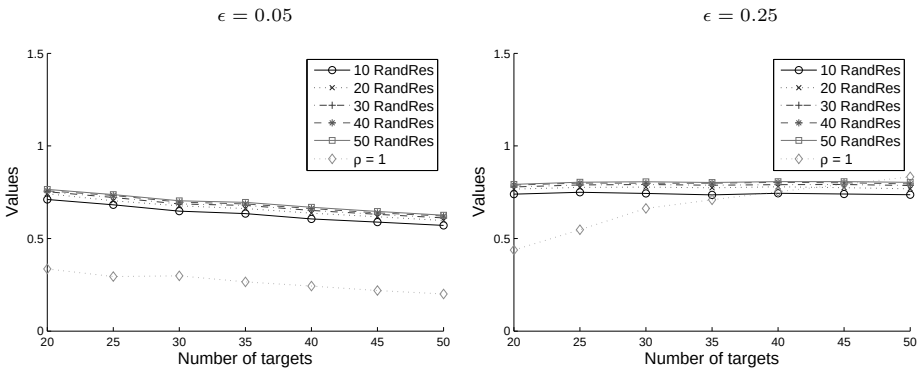


Figure 4.11: *Game values as $|T|$ varies.*

sive and, as shown above, the purely heuristic solution cannot be significantly improved for $\epsilon \geq 0.75$. We report experimental results only for $\epsilon \in \{0.05, 0.25\}$. We notice that for these instances we do not have the optimal game value. However, since the optimal game value cannot be larger than 1 by construction of the instances, the game value obtained with our approximation algorithms represents a lower bound to the actual approximation ratio. It can be observed that, given a value of ϵ, the ratios obtained with the dynamic programming algorithm are essentially constant as $|T|$ increases and this constant reduced as ϵ reduces. Surprisingly, after a certain value of $|T|$, the game values obtained with the branch and bound algorithm are higher than those obtained with the dynamic programming algorithm. This is because, fixed a value of ϵ, as $|T|$ increases, the problem becomes easier and the heuristic used by the branch and bound algorithm performs well finding the best covering routes. This shows that there is not an algorithm outperforming the other for every combination of parameters $|T|$ and ϵ. Furthermore, the above result shows that the worst cases for the approximation algorithms are those in which $\epsilon = O(\frac{1}{|T|})$, corresponding to instances in which the number of edges per vertex is a constant in $|T|$. It is not clear from our experimental analysis whether increasing $|T|$ with $\epsilon = \frac{\nu}{|T|}$ for some $\nu > 1$ the game value approaches to 0 or to a strictly positive value. However, our approximation algorithms provide a very good approximation even with a large number of targets and a small value of ϵ.

Figure 4.12 reports the compute times required by the approximation dynamic programming algorithms. As it can be seen, the required time slightly increases when adopting a larger number of randomly generated orders with respect to the baseline with $\rho = 1.00$.

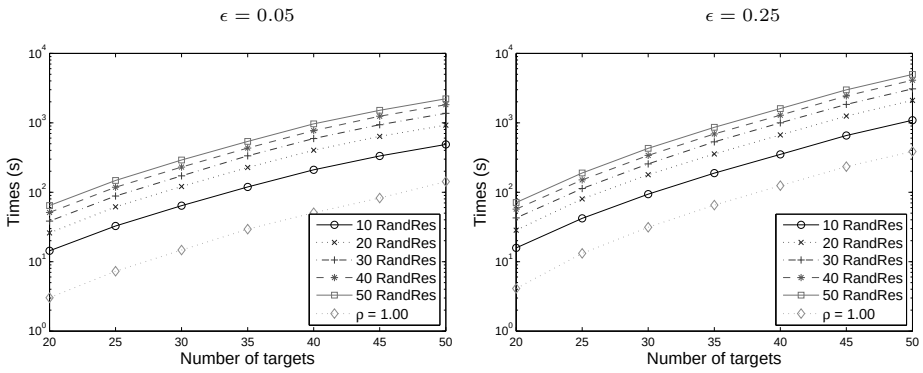


Figure 4.12: *Compute times as $|T|$ varies.*

4.5.2 Real Case Study

In Section 4 we presented a motivating scenario describing the fair site of Expo 2015. We now formalize an instance for our problem inspired by such scenario to derive important insights on how our techniques operate in real cases. Expo 2015 has been a large setting which, as we anticipated, underwent a remarkable deployment of security resources with about 700 guards operating on the field. This experiment addresses a worst-case scenario where we stress our route-generation algorithms by assuming that each guard has to protect the whole environment. The resolution of this problem is necessary even when admitting multiple defending resources for which no environment partition is pre-assigned.

Figure 4.13 shows the map of the Expo 2015 site together with its graph representation. We manually build a discretized version of the site map by exploiting publicly available knowledge of the event[8]. We identify ≈ 170 sensible locations which correspond to an equal number of targets in our graph. More specifically, we identify ≈ 130 targets located at the entrances of each pavilion and in the surroundings of those areas which could be of interest for a high number of visitors. Some targets (≈ 35) are located over the main roads, being these critical mainly due to their high crowd. Such roads also define our set of edges which resulted in a density of ≈ 0.02. Figure 4.13 reports a graphical representation of chosen deadlines $d(\cdot)$ and values $\pi(\cdot)$, respectively. To determine such values in a reasonable way we apply a number of simple rules of thumb. First, to ease our task, we discretize the spaces of possible deadlines and values in four different levels. To assign a value to a target, we estimate the interest (in terms of popularity

[8]Detailed information can be found at `http://www.expo2015.org/`.

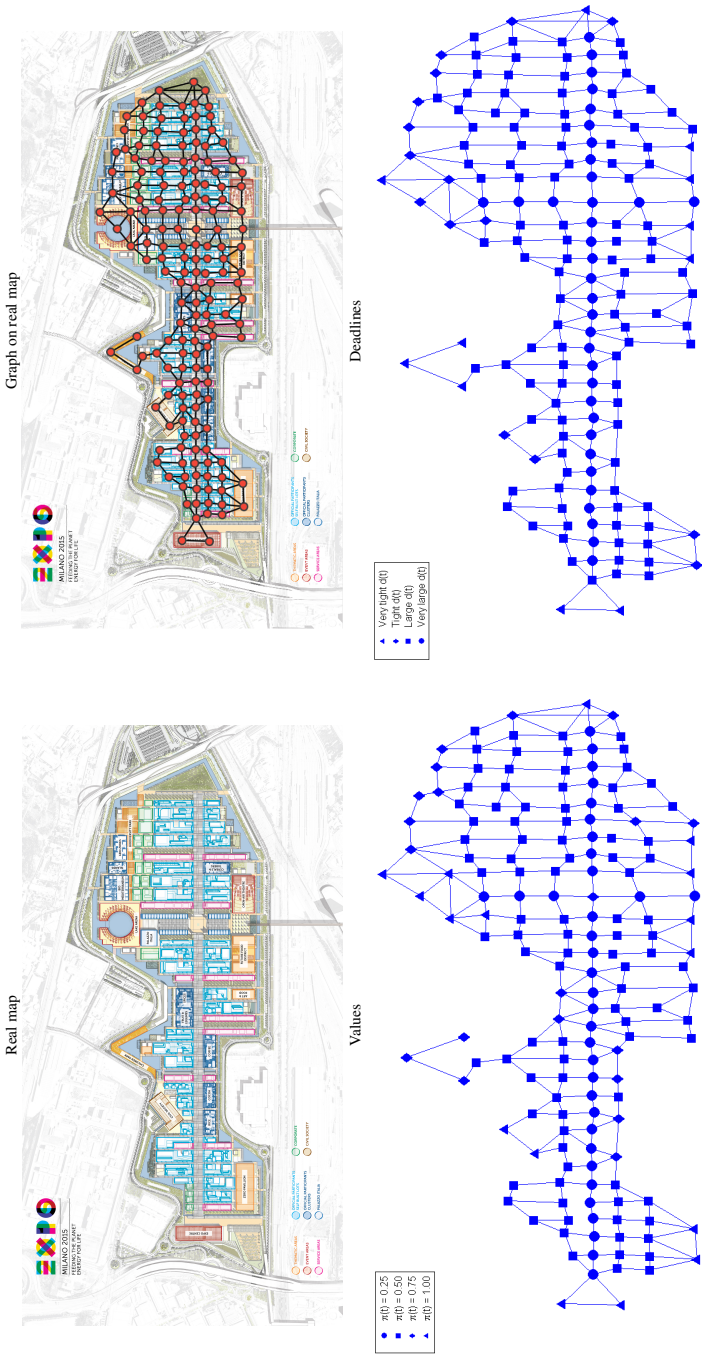


Figure 4.13: *Expo 2015 instance.*

and expected crowd) of the corresponding area in the fair site. The higher the interest, the higher the value (actual values are reported in the figure). To assign deadlines we estimate the time an attacker should spend to escape from the attacked target after some malicious activity is started (for example, blending into the crowd without leaving any trace). In particular, we estimate a smaller escape time for those locations lying near the external border of the fair site while for locations that are more central we estimated a larger time. The smaller the escape time, the tighter the deadline for that target. Actual values are extracted from a normal distribution where $\sigma^2 = 1$ and μ is set according to the chosen level. The maximum distance between any two target locations is about 1.5Km which we assume can be be covered in about 7.5 minutes (we imagined a crowded scenario). Given such reference scenario, our means span from 5 minutes (very tight) to 7.5 minutes (very large). To derive our alarm system model we assume to have a number of panoramic cameras deployed in the environment at locations we manually choose in order to cover the whole environment and to guarantee a wide area of view for each camera (i.e., trying to keep, in general, more than one target under each camera's view). To map our set of cameras over the alarm system model, we adopt this convention: each group of cameras sharing an independent partial view of a target t is associated to a signal $s \in S(t)$; if target t is covered by k signals then each signal is generated with probability $1/k$ once t is attacked. Obviously, a deeper knowledge of the security systems deployed on the site can enable specific methods to set the parameters of our model. This is why we encourage involving agencies in charge of security when dealing with such task.

We first show a qualitative evaluation of our method. Figure 4.14 depicts the best placement for the Defender (the circle in the figure) and the attacked targets (the squares in the figure, these are the actions played by the Attacker with non-null probability at the equilibrium). As intuition would suggest, the best location from where any signal response should start is a central one w.r.t. the whole fair site. Our simulations show that the optimal patrolling strategy coincides with such fixed placement even under false negatives rates of at least ≈ 0.3. Notice that such false negatives value can be considered unrealistically pessimistic for alarm systems deployed in structured environment like the one we are dealing with. Attacked targets correspond to areas, which exhibit rather high interest and small escape time. Figure 4.15 reports an example of signal response strategy for a given starting vertex (the small circle in the figure) and a given signal (whose covered targets are depicted with the large circle in the figure). The table lists the computed covering sets and the probabilities with which the Defender plays the corresponding

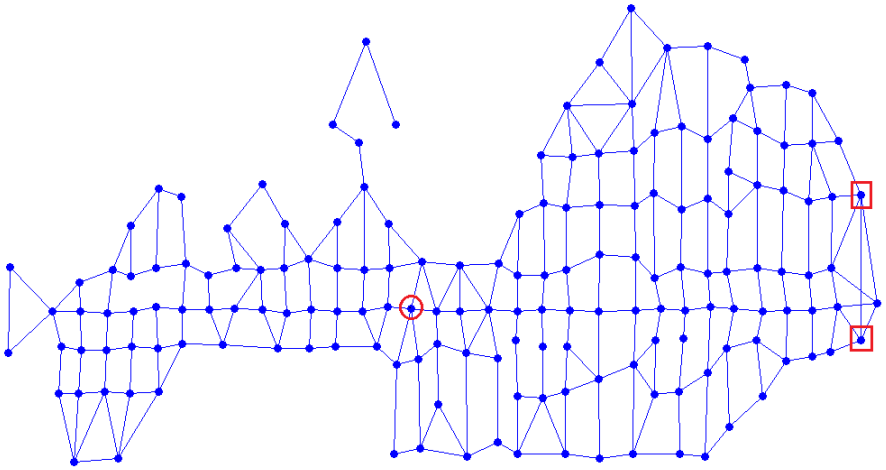
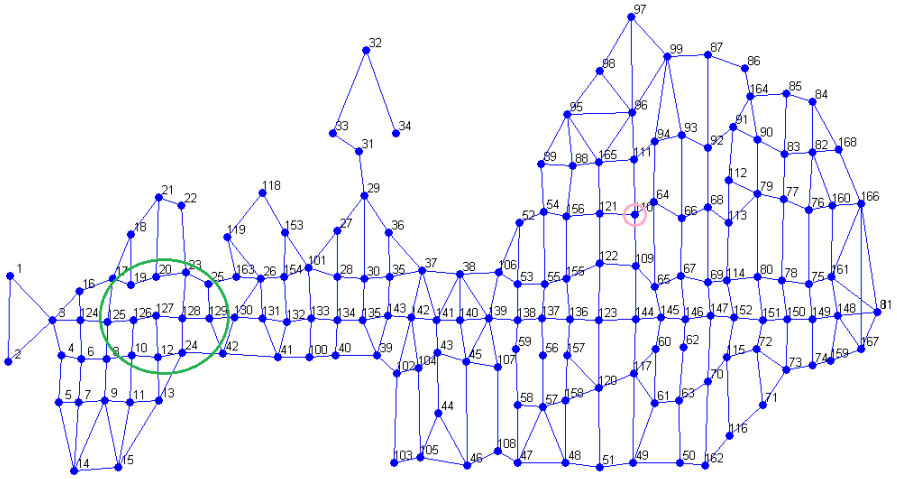


Figure 4.14: *Expo best placement and attack locations.*

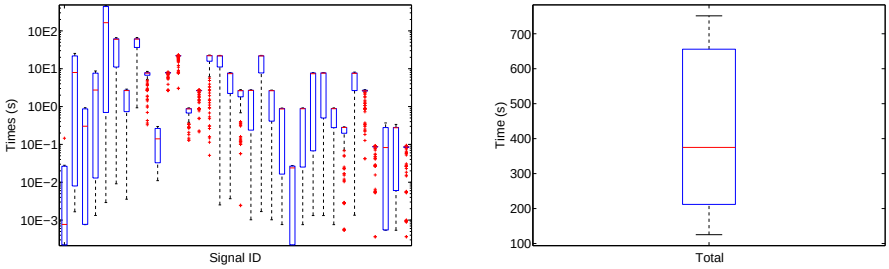
covering routes.

Boxplots of Figure 4.16(a) provide some quantitative insights on the computational effort we measured in solving such realistic instance. Given a signal, we report the statistical distribution of the time required by Algorithm 5 to compute covering routes from each possible start vertex. In general, we observe a high variance in each boxplot. Indeed, once fixed a signal s in our realistic instance, it is easy to identify starting vertices from which computing covering routes is likely to be very easy or, instead, much harder. For the easy case, consider a starting vertex lying very much far away from the group of targets covered by s. In such case, Algorithm 5 will soon stop iterating through covering set cardinalities being not able to further generate feasible sets. Such feature is induced by the large distance of the starting vertex from the targets covered by s together with the low edge density and the spatial locality shared among targets covered by the same signal (these last two are, indeed, features that frequently recur in realistic scenarios). For the harder case, just consider a situation in which the distance of the starting vertex from the targets covered by s is such that a large number of covering routes is available. An example of this kind can be inspected in Figure 4.15. Interestingly, a similar high variance trend is not always observed when depicting the statistical distribution of the compute time per starting vertex. The boxplot of Figure 4.16(b) shows an example of the distribution of compute time from a given starting vertex (the sample here is the composed by compute times associated to the resolution from that starting vertex with respect to each signal). Such distribution (characterized by a lower variance



Covering set	Probability
{19, 20, 23, 25, 125, 126, 127, 128}	0.0194
{10, 12, 23, 24, 25, 126, 127, 128}	0.0231
{10, 12, 24, 25, 126, 127, 128, 129}	0.0333
{12, 23, 24, 25, 126, 127, 128, 129}	0.0494
{10, 12, 23, 24, 25, 125, 126, 128}	0.0344
{10, 12, 24, 25, 125, 126, 128, 129}	0.0488
{10, 12, 25, 125, 126, 127, 128, 129}	0.0493
{12, 23, 24, 25, 125, 126, 127, 128}	0.0502
{12, 24, 25, 125, 126, 127, 128, 129}	0.0692
{19, 20, 23, 25, 125, 126, 129}	0.0492
{19, 20, 23, 125, 126, 128, 129}	0.0492
{20, 23, 25, 126, 127, 128, 129}	0.0657
{10, 23, 25, 125, 126, 127, 128}	0.0662
{19, 20, 23, 25, 125, 128, 129}	0.0412
{23, 25, 125, 126, 127, 128, 129}	0.1146
{20, 23, 24, 25, 127, 128}	0.0645
{10, 12, 24, 125, 126, 127}	0.0877
{20, 23, 24, 25, 128, 129}	0.0846

Figure 4.15: *Expo example of response strategies to signal.*



(a) Time boxplots by signal

(b) Time boxplots by node

Figure 4.16: *Time boxplots for Expo instance.*

with respect to what can be observed in Figure 4.15) suggests that there can be starting vertices posing easy resolution and, similarly, others posing only hard ones.

CHAPTER 5

Introducing Missed Detections

By Silence, the discretion of a man is known: and a fool, keeping Silence, seemeth to be wise.

— Pythagoras

In the present chapter, we deal with uncertainties towards a more general model, deeply analyzing security games in which the alarm system is both characterized by detection uncertainty and spatial imperfection, tackling the challenge of designing tractable algorithms for real-life scenarios.

5.1 Original Contributions

We formalize a game model where a *Defender* controls a single patrolling unit in the environment (for the sake of presentation, we do not distinguish between the agent and its resource). Solving this game is APX-hard even without false negatives and PSPACE-hard even with no alarm signals. We prove that, in absence of signals, placement-based strategies may be arbitrarily worse than strategies where the Defender moves. This entails that, to solve the game, we must compute strategies in which the Defender both

patrols over some targets and responds to alarm signals, introducing innovative and more involved techniques w.r.t. the case without false negatives. We formulate the problem of searching for the best Defender's strategy as a mixed-integer-non-linear mathematical program whose resolution can be done in practice only by means of local-search techniques based on non-linear mathematical programming. A strategy determines an action given an history of previous Defender's locations up to the current one. The length of such histories plays a key role. If arbitrary length histories are used, the size of the mathematical program explodes. Differently, if the problem is restricted on current-location (unitary-length) histories, the quality of the solution can be very low. Indeed, we prove that current-location strategies may be arbitrarily worse than those conditioning on an arbitrary number of previous locations. Motivated by the hardness of the problem, we study its properties in the attempt to design efficient algorithms. We propose an *anytime* approach based on a set of *oracles* in which we consider deterministic and randomized patrolling strategies in absence of alarm signals. We design an algorithm searching for the optimal subset of targets over which the Defender moves. For each candidate subset, we invoke an oracle to check for the existence of a deterministic or randomized patrolling strategy. If such strategy exists, we use another oracle to determine the best response to alarm signals. We propose three search heuristics. Two of them are *static*, enumerating the subsets of targets at time zero and keeping such an order during all the search. One is *dynamic*, changing the order on the basis of the information got by the evaluation of the previous subsets. Finally, we provide an experimental evaluation to assess the quality and the scalability of our method.

5.1.1 Chapter Structure

In Section 5.2, we introduce our game model. In Section 5.3, we study, from a theoretical perspective, the problem of finding the strategy of the Defender for responding to an alarm signal when there is a positive missed detection rate. Then, Section 5.4 shows the approach we adopted to solve the problem and, finally, Section 5.5 shows the experimental evaluations we performed on the algorithms we proposed.

5.2 Problem Formulation

We study a turn-based constant-sum extensive-form game with infinite horizon and imperfect information between an *Attacker* $\mathcal{A}$ and a *Defender* $\mathcal{D}$. The environment is formally described by an undirected connected graph

$G = (V, E)$, with vertices representing the different areas and edges modeling the connections among them. Each edge $(i, j) \in E$ requires one turn to be traversed. We denote with $\omega^*_{i,j}$ the temporal cost (in turns) of the shortest path between any i and $j \in V$. $\mathcal{D}$ controls a single patrolling resource on the graph that, for simplicity, we identify with $\mathcal{D}$ itself. The subset $T \subseteq V$ represents locations of interest, called *targets*, for both $\mathcal{A}$ and $\mathcal{D}$. Each target $t \in T$ has a value $\pi(t) \in (0, 1]$ and a penetration time $d(t) \in \mathbb{N}$ quantifying the number of turns needed to successfully complete an attack over t. Finally, a *spatially uncertain alarm system* is available to $\mathcal{D}$: such system raises an *alarm signal* once a target t is under attack. It is modeled as a pair (S, p), where $S = \{s_0, s_1, \cdots, s_m\}, m \geq 1$, is the set of possible signals that can be triggered while $p(s \mid t)$ specifies the probability of generating a signal s given that target t has been attacked. With s_0 we denote the *null signal*, corresponding to the absence of alarms, while s_i $(i > 0)$ denotes an alarm signal caused by some attack. For simplicity, we assume that $p(s_0 \mid t) = \alpha$ for every target t.

At each turn of the game, $\mathcal{A}$ and $\mathcal{D}$ play simultaneously. If $\mathcal{A}$ has not attacked in the previous turns, it observes the position of $\mathcal{D}$ in the graph and decides whether to attack a target[1] or to wait for a turn. In the same turn, $\mathcal{D}$ observes whether no signal or s_i has been generated and decides the next vertex to patrol among those adjacent to the current one. Once $\mathcal{A}$ attacked target t, she cannot take any other action for the next $d(t)$ turns during which can be captured if $\mathcal{D}$ patrols t. In the opposite case the attack is successful. The game is constant-sum (then equivalent to a zero-sum game): if $\mathcal{A}$ is captured, $\mathcal{D}$ receives a utility of 1 and $\mathcal{A}$ receives 0, while, if an attack over t has success, $\mathcal{D}$ receives $1 - \pi(t)$ and $\mathcal{A}$ receives $\pi(t)$. Finally, if $\mathcal{A}$ waits forever, $\mathcal{D}$ receives 1 and $\mathcal{A}$ receives 0. The appropriate solution concept is the leader-follower equilibrium. The game being constant sum, the best leader's strategy is its maxmin/minmax strategy. We represent the strategy of $\mathcal{A}$ with the tuple $\sigma^{\mathcal{D}} = (\sigma^{\mathcal{D}}_p, \sigma^{\mathcal{D}}_a)$. The components $\sigma^{\mathcal{D}}_p$ and $\sigma^{\mathcal{D}}_a$ denote the patrolling strategy adopted by $\mathcal{D}$ during the two possible states of the game. The first one, $\sigma^{\mathcal{D}}_p$, is the strategy adopted when no alarm signal has been generated or, formally, when signal s_0 is received. In general, we refer to it as *patrolling strategy* and we use more specific terms to identify the special forms it can have. For example, we say that it is a *placement* when it prescribes to occupy some vertex and never move. The second one, $\sigma^{\mathcal{D}}_a$, is the strategy adopted when some alarm signal s_i $(i > 0)$ has been generated. We refer to it as *signal response strategy*.

[1] As is customary, we assume that $\mathcal{A}$ can instantly reach the attacked target. This assumption can be relaxed as shown in [23].

5.3 Problem Analysis

We start by providing a theoretical analysis that motivates our algorithmic approach for solving the game. The results we present here will exploit a special type of patrolling strategy that we call *covering cycle* and that will be central for the contributions of our work.

Definition 5.1 (Covering cycle for a subset of targets)**.** *Given a subset of targets $T' \subseteq T$, a covering cycle on them is the shortest cyclic walk on graph G such that, when repeatedly followed, two subsequent visits to any $t \in T'$ always have a temporal delay not greater than $d(t)$.*

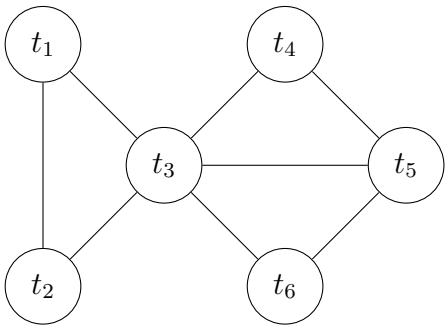
Let us start by studying the optimality of placement-based strategies or, for short, *placements*, prescribing that, in absence of alarm signals, $\mathcal{D}$ should stay in a vertex and wait. A signal response strategy is then undertaken when a signal is triggered. In settings where $\alpha = 0$, the optimal patrolling strategy is indeed a placement. However, this no longer holds with a non-null missed detection rate, even with very small values. That is, the Defender may have an incentive of patrolling the environment even when no alarm signal has been received.

Proposition 5.1. *There exist instances with $\alpha > 0$ where the best placement in a vertex v is not optimal.*

Proof. Consider a generic instance admitting a covering cycle over the whole set of targets. If such a cycle exists, then the Defender plays it and gets a utility of 1 (notice that this holds for any value of α). If $\alpha > 0$, any placement-based strategy will fail to capture attacks in the occurrence of a false negative. For $\mathcal{D}$, this will clearly result in an expected utility smaller than 1. Interestingly, one can question whether placement-based strategies are not optimal only when such a covering cycle exists. Figure 5.1 shows an instance where patrolling the graph yields a strictly larger expected utility that adopting the best placement. In the figure, all the nodes are targets with different values and same penetration time, all the edges are unitary, all the targets are covered by one single signal, and $\alpha = 0.1$. It can be verified that if $\mathcal{D}$ patrols along the cycle $t_5, t_6, t_3, t_1, t_3, t_5$, it gets a utility of 0.9447. The best placement is t_3, providing a utility of 0.9217. Hence, even for small values of α, a patrolling strategy can be better than the best placement. □

We can strengthen the above result in the following way.

Proposition 5.2. *There exist instances with $\alpha > 0$ in which the best placement is arbitrarily worse than the best patrolling strategy.*



t	$\pi(t)$	$d(t)$	$p(s_1 \mid t)$
t_1	0.628	5	1.0
t_2	0.012	5	1.0
t_3	0.934	5	1.0
t_4	0.553	5	1.0
t_5	0.783	5	1.0
t_6	0.685	5	1.0

Figure 5.1: *A game instance where the best-placement is not optimal.*

Proof. Consider an instance where $\alpha = 1$ and each target t has $\pi(t) = 1$. This is a degenerate setting where attacks are never detected by the alarm system. If $\mathcal{D}$ adopts a placement-based strategy on a target t, then $\mathcal{A}$ can win on any other target providing the former with a utility of 0. On the other side, if there exists a covering cycle over the whole set of targets, $\mathcal{D}$ obtains 1. Thus, placement-based strategies provide an approximation factor of 0. □

The above results motivate the main challenge of this work. When false negatives are present, placements cannot provide acceptable approximations of the optimal strategy. We then need to define a resolution method for the more general class of patrolling strategies. In terms of computational complexity, we already know that the problem of finding $\mathcal{D}$'s maxmin strategy is APX-hard from the analysis of the case in which $\alpha = 0$. Now we can prove that the problem is hard also in terms of spatial complexity.

Theorem 5.3. *The problem of deciding whether $\mathcal{D}$ can get a utility of at least k is* PSPACE*-hard.*

Proof. Let us consider the case in which $\alpha = 1$. Then, searching for a strategy that gives a utility equal to 1 is at least as hard as searching for a covering cycle that allows the Defender to reach each target within its deadline. This problem, as showed in [65] is PSPACE-hard. Thus, our problem is PSPACE-hard. □

The game resolution, namely finding the maxmin strategy, can be formulated as follows. We define strategy $\sigma_p^{\mathcal{D}} : H^l \times V \to [0, 1]$, which returns the probability to move to an adjacent vertex v given a history of previously visited vertices $h \in H^l$ where H^l is the space of graph walks with maximum length upper bounded by l. We call Σ^l the space of patrolling strategies conditioning over H^l. We define *route* $r \in R$ as a sequence of potentially non-adjacent vertices such that $r(0) \in V$ and $r(i) \in T$ with

$i > 0$. A route is an abstraction over $\mathcal{D}$'s action space, playing a route r, means to follow a graph walk starting in $r(0)$ and covering shortest paths between $r(i)$ and $r(i+i)$. We say that a route r is *covering* if the first arrival at each visited target is within $d(t)$ time steps from its start. We define $\sigma_a^{\mathcal{D}}$ as a function $\sigma_a^{\mathcal{D}} : S \times R \to [0,1]$, returning the probability to play route $r \in R$ once alarm signal $s \in S$ has been raised. The actions available to $\mathcal{A}$ are attack-when(t,h), meaning that $\mathcal{A}$ attacks target t after observing the Defender covering a walk in H^l. We call Γ^l the space of $\mathcal{A}$'s actions that condition over walks in H^l. For the sake of simplicity, we report the mathematical programming formulation with $l = 1$.

$$\min \; u \tag{5.1}$$

$$u \geq \mathbb{E}[U(\text{attack-when}(t,v))] - 1 + I_v \qquad \forall t \in T, v \in V \tag{5.2}$$

$$I_v \geq P_{\text{steady}}(v, \sigma_p^{\mathcal{D}}) \qquad \forall v \in V \tag{5.3}$$

$$I_v \in \{0,1\} \qquad \forall v \in V \tag{5.4}$$

$$\sigma_a^{\mathcal{D}} \text{ is well defined in } \Sigma^1 \tag{5.5}$$

$$\sigma_p^{\mathcal{D}} \text{ is well defined in } \Gamma^1 \tag{5.6}$$

where $P_{\text{steady}}(v, \sigma_p^{\mathcal{D}})$ is the steady state probability that $\mathcal{D}$ is at v given strategy $\sigma_p^{\mathcal{D}}$ and

$$\mathbb{E}[U(\text{attack-when}(t,v))] = \Bigg[\alpha\pi(t)\Big(1 - P_{\text{capture}}(t, v, \sigma_p^{\mathcal{D}})\Big) + \sum_{s \in S \setminus \{s_0\}} p(s \mid t)\pi(t)\Big(1 - \sum_{r : r(0)=v, t \in r} \sigma_a^{\mathcal{D}}(r,s)\Big)\Bigg] \tag{5.7}$$

$P_{\text{capture}}(t, v, \sigma_p^{\mathcal{D}})$ is the probability that t is visited by $d(t)$ turns starting from v given $\sigma_p^{\mathcal{D}}$ (recall that $\alpha = p(s_0|t)$). Notice that I_v is equal to 0 only if $P_{\text{steady}}(v, \sigma_p^{\mathcal{D}}) = 0$ and it is necessary in the formulation to enable/disable Constraints (5.12)—the rationale is that, if a vertex $\overline{v}$ is never visited, all the actions attack-when$(t, \overline{v})$ must be disabled. The above mathematical program is mixed integer, due to I_v, non-linear non-convex, due to $P_{\text{steady}}(v, \sigma_p^{\mathcal{D}})$ and $P_{\text{capture}}(t, v, \sigma_p^{\mathcal{D}})$. With $l > 1$, the program has the above structure except that the number of variables and constraints grows exponentially w.r.t. l.

Many issues are related to formulation (5.1)–(5.6). Specifically, its size explodes for larger values of l, making the use of strategies conditioned over

the current location of $\mathcal{D}$, *de facto*, the only affordable solution. Unfortunately, such restriction comes with no bounded loss of optimality.

Proposition 5.4. *Optimal patrolling strategies in Σ^1 are arbitrarily worse than the optimal patrolling strategies.*

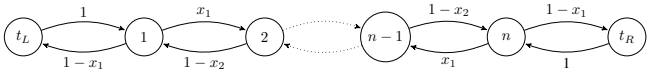


Figure 5.2: *A game instance based on a symmetric line topology.*

Proof sketch. We sketch the general idea of the proof, for the full technical details please refer to Appendix A. Consider a line graph as the one represented in Figure 5.2. The leftmost and rightmost vertices are the only targets, that is $T = \{t_L, t_R\}$. They are connected by n internal vertices, their values are $\pi(t_L) = \pi(t_R) = 1$, and their penetration times are $d(t_L) = d(t_R) = 2(n+1)$. Arcs are used to depict a patrolling strategy in Σ^1 where labels denote the movement probabilities, e.g., in Figure 5.2 $\sigma_p^{\mathcal{D}}(1,2) = x_1$. In the game played on such graph, $\mathcal{A}$ will always be indifferent between two non-dominated actions: attack-when(t_L, t_R) and attack-when(t_R, t_L). As a consequence, the capture probability $P_{capture}$ in the two configurations must be the same at the optimum, i.e., $\mathcal{D}$'s probability of visiting t_L starting from τ_R in at most $2(n+1)$ turns and *vice versa* must equal each other. Due to this observation and to the symmetry of the graph, a set of equality constraints over the movement probabilities are induced. Such constraints are summarized by the arc labels in Figure 5.2.

We first focus on the optimal patrolling strategy in Σ^1. In Appendix A, we show how to derive a formula to compute a value UB which has the following properties. First, it is an upper bound over $P_{capture}$ at the optimum. Second, if for any i it holds that $x_i = \frac{1}{2}$ then UB is maximum and goes to zero as n goes to infinity. We can therefore conclude that $P_{capture}$ goes to zero as the number of internal vertices increases. For longer line graphs, the optimal patrolling strategy in Σ^1 decreases in expected utility.

As a second step, we point out how the maximum utility of 1 can always be achieved, independently from n, with a strategy in Σ^2 defined as a covering cycle prescribing a repeated back-and-forth walk between τ_L and τ_R. Thus, the larger n the wider the optimality gap between $l = 1$ and $l = 2$ optimal strategies. □

Problem (5.1)–(5.6) can be solved exactly only by means of global optimization tools, e.g., [42]. However, these tools can deal only with extremely small programs (approximately 30 variables) and therefore they cannot be

applied even to toy instances. When non-linear mathematical programming tools [26] are adopted, two drawbacks arise. First, integer variables cannot be effectively handled. Second, only locally optimal solutions can be guaranteed. Finally, differently from the case without false negatives, Problem (5.1)–(5.6) cannot be separated into two independent programs, one searching for the best $\sigma_p^{\mathcal{D}}$ and another for the best $\sigma_a^{\mathcal{D}}$: in fact, doing so would inevitably provide sub-optimal patrolling and signal response strategies.

5.4 Resolution Approach

Tackling the resolution of Problem (5.1)–(5.6) in its original formulation would be, as discussed above, prohibitive in terms of computational effort. It would require to limit the space of strategies by fixing l, incurring in potentially large losses of utility. Therefore, instead of directly seeking the optimal solution we design an approximation method working in anytime fashion. Since no approximation guarantees can be given, the challenge resides in computing good solutions within a reasonable time. We deal with it by exploiting some formal results induced by optimal strategies. To show them, we start from the notion of support graph.

Definition 5.2 (Support graph of a patrolling strategy)**.** *The support graph of a patrolling strategy $\sigma_p^{\mathcal{D}}$ is a pair $G^{\sigma_p^{\mathcal{D}}} = (V^{\sigma_p^{\mathcal{D}}}, E^{\sigma_p^{\mathcal{D}}})$, where $V^{\sigma_p^{\mathcal{D}}} = \{v \in V \mid P_{steady}(v, \sigma_p^{\mathcal{D}}) > 0\}$ is called support, and $E^{\sigma_p^{\mathcal{D}}} = \{(i,j) \in E \mid \exists h_i\ \sigma_p^{\mathcal{D}}(h_i, j) > 0\}$ in which h_i is a walk ending in i.*

In general, the support of the optimal patrolling strategy may be a strict subset of V. Indeed, the alarm system enables the protection of targets only via signal responses. We can leverage this in building a simple enumeration Multi-Non-Linear Program scheme, where each sub-problem is obtained from (5.1)–(5.6) by fixing I_v under a given support (removing the integer variables from the problem). Despite this would obviously require to enumerate $O(2^{|V|})$ subproblems, working in the space of supports gives the possibility of defining pruning rules to speedup the enumeration. First, we show that not all the supports need to be enumerated.

Theorem 5.5. *The support of the optimal patrolling strategy will either be a singleton or will contain at least two targets.*

Proof. We shall prove that every patrolling strategy inducing a support with less than two targets is weakly dominated by a static placement strategy, that is a patrolling strategy with a singleton support. Let us denote with $\sigma = (\sigma_p^{\mathcal{D}}, \sigma_a^{\mathcal{D}})$ a strategy for $\mathcal{D}$ and with $V^{\sigma_p^{\mathcal{D}}}$ its associated support.

Case 1: $V^{\sigma_p^{\mathcal{D}}}$ *contains no targets.* In this case, $\mathcal{A}$'s expected utilities for any attack do not depend on $\sigma_p^{\mathcal{D}}$ but only on the support $V^{\sigma_p^{\mathcal{D}}}$. To see this, let us consider any action attack-when(t, v). If no signal is raised (due to a missed detection) there is no chance for $\mathcal{D}$ to reach t from v via $\sigma_p^{\mathcal{D}}$ since, by hypothesis, $P_{\text{steady}}(t, \sigma_p^{\mathcal{D}}) = 0$. With probability α then, any attack will be successful independently of $\sigma_p^{\mathcal{D}}$. When instead a signal is raised, $\mathcal{A}$'s expected utility will depend on $\mathcal{D}$'s response from vertex v as prescribed by $\sigma_a^{\mathcal{D}}$. In this scenario, $\mathcal{A}$ will play action attack-when(t^*, v^*) which maximizes the expectation of $U(\cdot)$ under $\sigma_a^{\mathcal{D}}$. It is easy to see that, since $\mathcal{A}$'s expected utilities do not depend on $\sigma_p^{\mathcal{D}}$, any $\overline{\sigma} = (\overline{\sigma}_p^{\mathcal{D}}, \sigma_a^{\mathcal{D}})$ where $V^{\overline{\sigma}_p^{\mathcal{D}}} \subseteq V^{\sigma_p^{\mathcal{D}}}$ cannot provide a worse expected utility to $\mathcal{D}$. This holds because either $v^* \in V^{\overline{\sigma}_p^{\mathcal{D}}}$ and the game equilibrium keeps the same or $v^* \notin V^{\overline{\sigma}_p^{\mathcal{D}}}$ and $\mathcal{A}$ is forced to play an action different from attack-when(t^*, v^*) which, by hypothesis, cannot be strictly better.

Case 2: $V^{\sigma_p^{\mathcal{D}}}$ *contains target* $\hat{t}$. What derived in the previous case can be refined by stating that moving to a support $V^{\overline{\sigma}_p^{\mathcal{D}}} \subseteq V^{\sigma_p^{\mathcal{D}}}$ does not change utilities for attacks to targets not belonging to $V^{\sigma_p^{\mathcal{D}}}$ (same previous reasoning). Therefore in this case, when replacing σ with $\overline{\sigma}$, attacking $\hat{t}$ can change its expected utility and, in principle, become more profitable. However, it is immediate to see that if $\overline{\sigma}_p^{\mathcal{D}}$ prescribes to statically place in $\hat{t}$ then $\overline{\sigma}$ cannot be worse than σ. Indeed, the expected utility of attacking outside S keeps the same (from Case 1) while attacking $\hat{t}$ yields the minimum utility since $\mathcal{A}$ will always be captured. As a consequence, a static placement in $\hat{t}$ cannot be worse than any patrolling strategy on a support S that does not contain targets other than $\hat{t}$. Finally, notice that the same conclusions cannot be taken when S contains two or more targets since a static placement can obviously guarantee capture on a single target at most. □

From Theorem 5.5 we know that we can save the time needed by enumerating all the supports with less than two targets by enumerating the $O(|V|)$ singleton ones.

Let us focus our attention to the other supports to be enumerated, namely those containing at least two targets. Consider a subset of targets T' with $|T'| \geq 2$. We drive our attention to support $V' \supseteq T'$, defined as the one among all supports covering only targets T' that provides $\mathcal{D}$ with the lowest (best) expected utility as per Problem (5.1)–(5.6). Our aim is to derive some information about the non-target vertices composing V'. The following result shows how optimality induces a structural property, which can be exploited to derive some insights on supports with two or more targets.

Theorem 5.6. *The support graph of the optimal patrolling strategy does not*

contain non-target terminal vertices (vertices with degree 1).

Proof. Let us suppose to have a $\sigma = (\sigma_p^{\mathcal{D}}, \sigma_a^{\mathcal{D}})$ whose support graph G^{σ} contains targets $T' \subseteq T$ and one or more non-target terminal vertices. Let us call v_i any of those terminals inside the support graph. Consider then any $\overline{\sigma} = (\overline{\sigma}_p^{\mathcal{D}}, \sigma_a^{\mathcal{D}})$ where $V^{\overline{\sigma}_p^{\mathcal{D}}} = V^{\sigma_p^{\mathcal{D}}} \setminus \{v_i\}$. It is easy to show that $\overline{\sigma}$ weakly dominates σ. From Theorem 5.5 (Case 1) we have that the expected utility any action attack-when(t, v) where $v \in V^{\overline{\sigma}_p^{\mathcal{D}}}$ and $t \in T \setminus T'$ does not change (recall that the expected utility of attacking outside the support does only depend on $\sigma_a^{\mathcal{D}}$ which here is kept invariant). For actions attack-when(t, v) where $v \in V^{\overline{\sigma}_p^{\mathcal{D}}}$ and $t \in T'$ the expected utility cannot increase for the following two reasons. Recall that, from (5.7), the expected utility is defined as the sum of a patrolling-dependent component (first) and a response-dependent one (second). First, the response-dependent component does not change for the same reason of before, i.e., $\sigma_a^{\mathcal{D}}$ is kept invariant. Second, the patrolling-dependent component (the one that could provide capture under a missed detection) cannot increase since it can be shown that there always exists a $\overline{\sigma}_p^{\mathcal{D}}$ weakly dominating $\sigma_p^{\mathcal{D}}$. Indeed, from [22] we have that occupying a non-target terminal vertex is a dominated action for $\mathcal{D}$. The intuition is that the probability of timely reaching any target starting from v_i cannot be higher than when starting from the unique node adjacent to v_i. Finally, notice that the theorem trivially holds for singleton supports, since by definition they contain vertices with degree 0. □

The previous results convey the idea that if the optimal support does not encode a static placement, then it will be composed by two or more targets (Theorem 5.5) and, for each pair of them, by a number of (not necessarily shortest) acyclic paths of non-target vertices connecting them (Theorem 5.6). While guessing which vertices will be contained in the optimal support can be a very difficult task, guessing the non-target vertices connecting a given subset of targets in the support is relatively feasible by means of heuristics (as we shall show). So, if $|T|$ is the number of targets, our approach is to perform a $O(|V| - |T| - 1 + 2^{|T|}) \approx O(2^{|T|})$ search, substantially enumerating subsets of targets instead of subset of vertices in $O(2^{|V|})$ (recall that, by definition, $|V| \geq |T|$). For each subset of targets, our approach is to guess the remaining vertices composing the support by querying a *patrolling strategy oracle*. This oracle tries to compute a patrolling strategy over the subgraph iduced by a given $T' \subseteq T$. If a strategy is returned, then the support graph is determined. (Such graph will contain all the vertices visited with non-null probability by the strategy returned by the oracle.) The overall method exploits an additional oracle and can be schematized as follows (see Figure 5.3

for a graphical representation):

1. select a subset of targets T';
2. query a patrolling strategy oracle returning $\sigma_p^{\mathcal{D}}$ if found or answering $\perp$ otherwise;
3. if $\sigma_p^{\mathcal{D}}$ was found, query a Signal Response Oracle (SRO) returning the optimal $\sigma_a^{\mathcal{D}}$ (as per Problem (5.1)–(5.6)) assuming $\sigma_p^{\mathcal{D}}$'s support;
4. store $(\sigma_p^{\mathcal{D}}, \sigma_a^{\mathcal{D}})$ and its value in a list;
5. repeat from Step 1 until a timeout expires and, in that case, select the strategy yielding the minimum (best for $\mathcal{D}$) value among those stored.

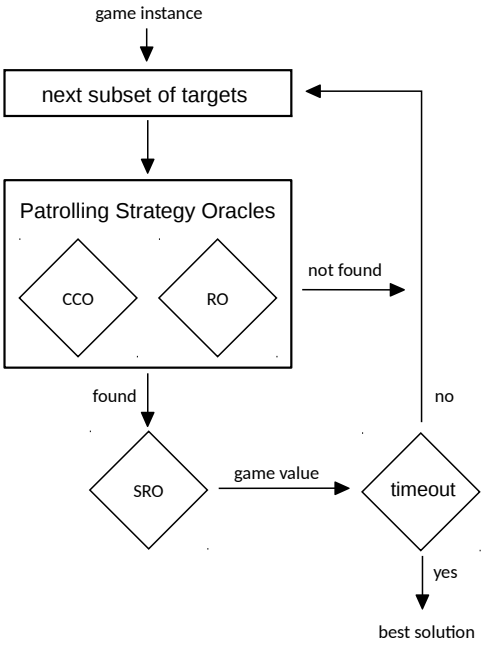


Figure 5.3: *Algorithmic approach.*

5.4.1 Patrolling Strategy Oracles

A patrolling strategy oracle can be modeled as a function $O : 2^T \to \{\sigma_p^{\mathcal{D}}, \perp\}$ which, given a subset of targets T' tries to compute a patrolling strategy over it. The oracle can also provide a null answer, in case such strategy cannot be found. We present two oracles. The first and most important one works

by searching a covering cycle over T' and is called Covering Cycle Oracle (CCO). The second one tries to compute the optimal strategy in Σ^1 and is called Random Oracle (RO).

Covering Cycle Oracle (CCO)

This oracle, tries to compute a covering cycle protecting a given subset of targets T'. If found, no better patrolling strategy can be given for any support containing the same targets since it always guarantees capture on them, even in presence of missed detections. Moreover, by applying Definition 5.1, we implicitly make the assumption that in the optimal support targets are connected along shortest paths. Despite not optimal in general, this is a reasonable assumption especially considering that in many realistic settings is the only non-dominated way to move between targets. The CCO must solve a PSPACE-hard problem (Theorem 5.3), so no exact efficient method can be designed. We resort to the approach of [22], where an algorithm for such problem is provided with correctness guarantees. The algorithm limits the solution length to $k|V|$ (k is a parameter) and applies a backtracking search. The algorithm is still exponential in the worst case (even for a fixed k), but empirically it demonstrated acceptable performance on realistic settings, even with large numbers of targets. The algorithm only indirectly poses a limit on l by setting a maximum solution length, i.e., if a solution is returned it would be such that $l < k|V|$. The CCO is our leading oracle: when a covering cycle is found it is always the preferred choice for the considered subset of targets.

Random Oracle (RO)

The Random oracle (RO) computes the optimal patrolling strategy in Σ^1 over a subgraph $G_{T'} \subseteq G$, obtained by connecting targets from T' with arbitrarily chosen shortest paths. This problem is the equivalent to the computation of the optimal randomized patrolling strategy as done in [22] and can be done with non-linear programming techniques. The RO is meant to work as a backup and is run in parallel to the CCO (see Figure 5.3). When the CCO terminates with no answer or reaches a timeout, we check if RO was able to terminate and, in the positive case, we fetch the patrolling strategy it computed and continue with our search. Notice that the term "randomized" is used in general sense. Indeed, the returned strategy can be deterministic (prescribing with full probability the next vertex to patrol given the current one) and, in very special cases, can even be a covering cycle (however, in such case the CCO is very likely to find it first). Finally, notice that given the

$\sigma_p^{\mathcal{D}}$ returned by the oracle, its support does not necessarily coincide with all vertices in $G_{T'}$ since it might be a strict subset of it.

5.4.2 Signal Response Oracle (SRO)

The SRO takes as input a patrolling strategy $\sigma_p^{\mathcal{D}}$ (returned by one of the above oracles) and computes the optimal signal response from every vertex in the support $V^{\sigma_p^{\mathcal{D}}}$ induced by that strategy. The objective is indeed to find out the optimal way to respond from any of the vertices that could be the current location of $\mathcal{D}$ upon the reception of a signal. In practice, the SRO amounts to the resolution of problem (5.1)–(5.6). Since $\sigma_p^{\mathcal{D}}$ and $V^{\sigma_p^{\mathcal{D}}}$ are now given, such problem scales down to an LP where variables $\sigma_p^{\mathcal{D}}$ and I_v have been removed. The result can be formalized in this way:

$$\min \; u \tag{5.8}$$

$$u \geq \mathbb{E}[U(\text{attack-when}(t, v))] \qquad \forall t \in T, v \in V^{\sigma_p^{\mathcal{D}}} \tag{5.9}$$

$$\sigma_a^{\mathcal{D}} \text{ is well defined} \tag{5.10}$$

The value of $\mathbb{E}[U(\text{attack-when}(t, v))]$ is computed by instantiating Constraint 5.7 to one of two cases depending on the target t. In the first case, the target t belongs to the support $V^{\sigma_p^{\mathcal{D}}}$. If $\sigma_p^{\mathcal{D}}$ is a covering cycle over its support, then $P_{\text{capture}}(t, v, \sigma_p^{\mathcal{D}}) = 1$ (a direct implication of Definition 5.1). Intuitively, if in absence of any signal $\mathcal{D}$ follows a covering cycle and t is part of it, then upon a missed detection t will keep being protected. If $\sigma_p^{\mathcal{D}}$ is not a covering cycle (and, as a consequence, a strategy in Σ^1) $P_{\text{capture}}(t, v, \sigma_p^{\mathcal{D}})$ will be equal to the probability that $\mathcal{D}$ reaches t within $d(t)$ turns. This value is a non-linear function of $\sigma_p^{\mathcal{D}}$ returned, alongside the strategy, by our implementation of the RO.

If $\sigma_p^{\mathcal{D}}$ is a covering cycle, then each target in the support will be fully protected upon a missed detection. This is immediately verified by the fact that Definition 5.1 implies the capture probability P_{capture} (see Constraint 5.7) equal to 1 for any target visited by the covering cycle. In this case, $\mathbb{E}[U(\text{attack-when}(t, v))]$ is computed in the following way. If $t \in V^{\sigma_p^{\mathcal{D}}}$, we can rewrite Constraint (5.7) by considering that the capture probability when attacking such target is always equal to 1 in case of a false negative, that is the utility is given by the following value:

$$\alpha\Big(1 - \pi(t)\Big) + \sum_{s \in S \setminus \{s_0\}} p(s \mid t)\Big(1 - \pi(t) \sum_{r : r(0) = v, t \in r} \sigma_a^{\mathcal{D}}(v, r, s)\Big).$$

In the second case $t \notin V^{\sigma_p^{\mathcal{D}}}$. Here a missed detection will always ensure a successful attack by yielding a null capture probability. Intuitively, this happens because upon a missed detection any vertex outside the patrolling strategy's support will never be visited. Constraint (5.7) becomes then as follows:

$$\sum_{s \in S \setminus \{s_0\}} p(s \mid t) \Big(1 - \pi(t) \sum_{r : r(0)=v, t \in r} \sigma_a^{\mathcal{D}}(v, r, s)\Big)$$

We solve this problem separately for each vertex in the support since we work under the assumption that $\sigma_p^{\mathcal{D}}$ is fixed and equal to the covering cycle C. That is, by searching for the optimal response we account for the fact that each target in the support of $\sigma_p^{\mathcal{D}}$ will be fully protected upon a missed detection. This is immediately verified by the fact that Definition 5.1 implies the capture probability P_{capture} (see Constraint 5.7) equal to 1 for any target visited by the covering cycle C. The problem we obtain from (5.1)–(5.6) reduces to a LP formulation when we fix $\sigma_p^{\mathcal{D}}$ to a covering cycle C (see Definition 5.1), whose support is denoted by S_C. Since under such assumption both the patrolling strategy and the support are given, variables $\sigma_p^{\mathcal{D}}$ and I_v can be removed by rewriting the problem in this form:

$$\min \; u \tag{5.11}$$

$$u \geq \mathbb{E}[U(\text{attack-when}(t, v))] \qquad \forall t \in T, v \in S_C \tag{5.12}$$

$$\sigma_a^{\mathcal{D}} \text{ is well defined} \tag{5.13}$$

Thus, we are able to find the optimal response strategy $\sigma_a^{\mathcal{D}}$ efficiently.

Our implementation of SRO is based on the exact and approximate algorithms presented in Chapter 4 to construct $\mathcal{D}$'s actions space. In particular, we adopt two algorithms:

- `DP-ComputeCovSets`: an exact method based on dynamic programming that runs in $O(2^{|T|})$;
- `MonotonicLongestRoute`: an approximate method based on dynamic programming and l random restarts that runs in $O(l|T|^3)$.

Once each response strategy has been computed, the overall value of the strategy is simply given by the response that provides $\mathcal{A}$ with the highest expected utility.

5.4.3 Target Selection Heuristics

As we explained in previous sections, by showing some structural properties of the optimal support, we are able to devise an algorithmic approach that requires implementation of two oracles (CCO and SRO) to map the process of computing the optimal strategy to a search in the power set of targets. The problem of refining such oracles or to devise better implementations remains open and lies outside the scope of this chapter (and thesis), whose aim is primarily the presentation of our analysis, approach, and experimental validation. However, we underline that our modular scheme can, in principle, integrate any implementation for such oracles leaving it open to possible future improvements. Moving to an enumeration on the power set of targets, which usually are less than the total number of vertices, we can obtain some computational advantages by means of a reduction of the search space. However, we are still dealing with an exponentially large size and the design of good heuristics to drive the search can be of critical importance. In this section, we describe the three heuristic methods we propose to effectively drive our enumeration of subsets of targets. The first one follows a dynamic approach in the sense that the ranking of possible solutions to explore changes as new subsets are explored. The last two, on the other side, follow a static approach, meaning that the order of preference between the candidate solutions is fully determined by the problem instance and does not vary during the search process.

Dynamic Method based on Attacker Responses (H1)

The first mechanism to drive the search leverages the following rationale: if we searched subset T' and we obtained strategy σ', then constructing a T'', obtained by including in T' targets that provide $\mathcal{A}$ with large expected utilities under σ' and searching it, might result in a better strategy σ''. With this method, we progressively include in the support targets that are the most strategically appealing to the Attacker. (Notice that to assess how much a target can be appealing we cannot simply consider static parameters like its value or its penetration time, but we need to solve for a game-theoretic equilibrium. The approach is inspired to the *double-oracle method* commonly used in Operations Research and Security Games.)

Specifically, our method first searches among the singleton supports the one that represents the best static placement. Then, new targets are added to this base support by considering $\mathcal{A}$'s actions ranked according to their expected utility, from the best response to other non-optimal responses. The algorithm is essentially a greedy search. Formally:

1. compute the optimal σ prescribing a static placement w^* and set $U = \{w^*\}$;
2. sort targets not included in U according to the best expected utility value $\mathcal{A}$ can get by attacking them, calling $T^{(i)}$ the set of targets that, if attacked, will yield the i-th largest of such expected utilities;
3. for $i = 1, 2, \ldots$ enumerate subsets Q_i, where $Q_i \subseteq U \cup T^{(i)}$ and $Q_i \not\subseteq U$; for each of them, compute σ by exploiting the two oracles (Figure 5.3);
4. repeat this procedure from Step 2 for each computed σ by setting $U = Q_i$.

This enumeration scheme is complete since, in the end, the optimal subset will be evaluated.

Static Method based on Targets Values (H2)

This method evaluates supports by considering the cumulative value of targets belonging to it. Thus, we adopt a static parameter (the value) to estimate the likelihood of having a target in the support. The rationale is that most valuable targets are more likely to be attacked. Thus, providing them with some extra protection given by the patrolling strategy can be a good choice. In practice, the method tries to search first the supports with an high cumulative value and with few targets. Formally:

1. rank targets according their value $\pi(\cdot)$, denote with $T^{(i)}$ the group of targets with the i-th highest value, set $U = T^{(1)}$ and $u = 1$;
2. enumerate subsets Q, where $Q \subseteq U$ and, if $u > 1$, $Q \not\subseteq \cup_{i \in \{1,\ldots,u-1\}} U^i$; for each of them, compute σ by exploiting the two oracles (Figure 5.3);
3. set $U = U \cup U^{(u+1)}$, $u = u + 1$ and repeat from Step 2.

Static Method based on Targets Distances (H3)

The last method tries to privilege the subset of targets where the shortest distance between any pair of included ones in not above some given threshold. Here are the steps of the distance-based method:

1. set a threshold value $\tau = 1$;
2. enumerate all the subsets of targets T' such that for any $t_1, t_2 \in T'$ it holds that $d(t_1, t_2) \leq \tau$;
3. repeat from Step 2 after setting $\tau = \tau + 1$.

5.5 Experimental Evaluation

5.5.1 Experimental Setting

We generate the worst-case instances for our problem from the instances of HAMILTONIAN PATH, with graphs where all the nodes are targets, edges are unitary, and, for each target, $\pi(t) \in (0, 1]$ and $d(t) = |T| - 1$. There is one single signal covering all the targets. We shall denote with ϵ the instance edge density defined as $\epsilon = |E|/\frac{|T|(|T|-1)}{2}$. Setting a timeout (in our case one hour) implicitly poses a limit on the maximum size (number of targets) of instances whose resolution can be completed. We implemented our algorithms in Matlab and we run them on a UNIX computer with 2.33GHz CPU and 16GB RAM.

5.5.2 CCO Tuning

One of the critical components of our resolution approach is the covering cycle oracle (CCO). As we explained above, we need to upper bound with $k|V|$ the solution length (the number of vertex visits made by the cycle) to trade off completeness for a lower computational effort (recall that the problem is PSPACE-hard). Small values for k result in faster executions of the CCO due to the shrinkage induced on the solution space, but they could discard effective solutions. The experiment summarized in Fig. 5.4 shows this tradeoff with $k \in \{0.5, 1, 2\}$. For each value of k we report the number of covering cycles with at most $k|V|$ visits we are able to find in one hour (we enumerate subsets of vertices for increasing cardinalities checking for the existence of covering cycles with no more than $k|V|$ visits).

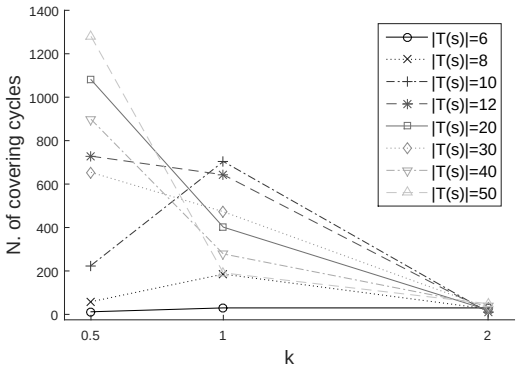


Figure 5.4: *Number of covering cycles found in one hour with solution length upper bounded to $k|V|$.*

The trends suggest $k = 1$ as acceptable bounding level when $|T| \leq 10$ and $k = 0.5$ when $|T| > 10$. Indeed, when k is larger, extremely few cycles are returned since CCO spends too long time in searching for a single a covering cycle and therefore not exploring a sufficient number of cycles. Interestingly, with $k = 2$ the number of returned covering cycles is close to 0.

5.5.3 Optimal SRO

We set k as discussed above and we start by assessing the applicability extent of the exact SRO (`DP-ComputeCovSets` algorithm). We do this by evaluating, at the same time, the impact of the false negative rate α on the optimal game value. In Figure 5.5, we report results obtained within a one hour timeout averaged over 50 instances. The graphs suggest how such limit is 10 targets when employing the exact version of SRO in the game resolution process. Due to the time limit, one hour, such approach never terminates for instances composed of 12 targets. Furthermore, as intuition suggests, for increasing values of α the game's optimal value has a negative trend (approximately) linear in α. Section 5.5.3 depicts this behavior for instances with $\epsilon = 0.25$ (the same can be observed for higher ϵ). Also, it can be noticed how with more targets, the game value decreases (about exponentially in $|T|$), confirming that more targets correspond to more difficult environments to protect. Section 5.5.3 shows the same trends w.r.t. $|T|$. We applied this approach also to realistic instances, built taking $\pi(t)$ from a uniform random distribution on the inteval $(0, 1]$ while $d(t)$ and $\omega^*_{i,j}$ are extracted from a uniform random distribution on the interval $(0, \delta]$, where δ is the diameter of the graph. As for the hard instances, $T = V$ and there is one single signal s that covers each target. Regarding these instances, within the time limit the algorithm solves graphs with up to 12 targets.

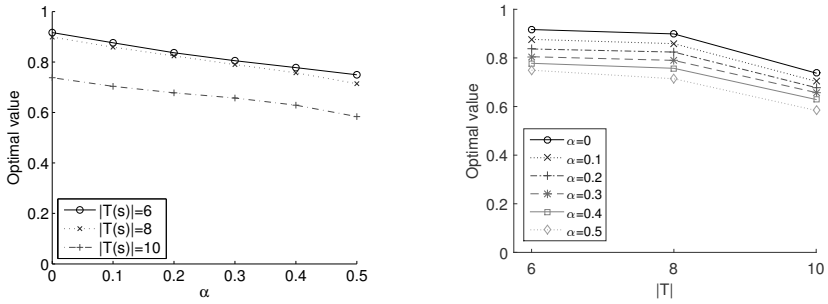


Figure 5.5: *Optimal game value with* `DP-ComputeCovSets` *SRO.*

5.5.4 Approximate SRO

Despite several common single defender reality scenario can fit in the limit of 10 targets (e.g., when targets can be associated with large sub-areas), some others are characterized by a higher number of targets (e.g., when attackers can act on a fine discretization of the environment). This is what justified our choice to introduce an approximate method, moving the focus of the problem from scalability to solution's quality. As reported above, our SRO implementation based on the `MonotonicLongestRoute` algorithm has a worst-case complexity of $O(|T|^3)$. This allowed us to process, within the one hour timeout, instances with up to 50 vertices (possibly scaling further for larger timeouts). The main questions here are two. First, we would like to assess the gap (in terms of utility) between the strategy we compute with our method and the best placement. Second, we want to compare the different heuristic methods we proposed in order to identify which one is the more suitable.

Figure 5.12.(a)–(b) depict a representative example of what we observed during our simulations with $l = 10$ random restarts. Here we report results averaged over 50 instances each with 20 targets for two density values ($\epsilon = 0.25$ can be thought as representative of indoor cluttered environment while $\epsilon = 0.75$ could correspond to outdoor, highly interconnected, areas). Among the three heuristic methods, H1 (dynamic) is the only one yielding a competitive gap within the time limit w.r.t. the best placement strategy (up to about $+10\%$ with $\epsilon = 0.25$ and $+35\%$ with $\epsilon = 0.75$). H3 (distance) provided marginal improvements while H2 (value) very rarely outperformed the best placement. Notice how, as α gets close to 0, all the heuristics tend to yield the best placement value. Such results suggest that H1 (including the strategic component in ranking targets) is right direction to obtain better results in reasonable time. Surprisingly, the heuristic based on the targets' values (H2) performed rather poorly despite often used in reality as an evaluation principle. Due to the previous analysis, we made further experiments on H1, reported in Fig. 5.12 (c)–(f). Specifically, Fig. 5.12 (c)–(d) depicts the results with $l = 1$ as the number of targets vary, while Fig. 5.12.(e)–(f) do the same with $l = 10$, showing that with $|T| \geq 40$ all the heuristics do not improve the quality of the best placement (requiring thus a time longer than one hour). On the other side, it can be seen that $l = 10$ are useful in one hour with $|T| \geq 20$, allowing one to find better responses to the alarms, but they degrade the quality solution when $|T| > 20$ since the SRO takes too long time.

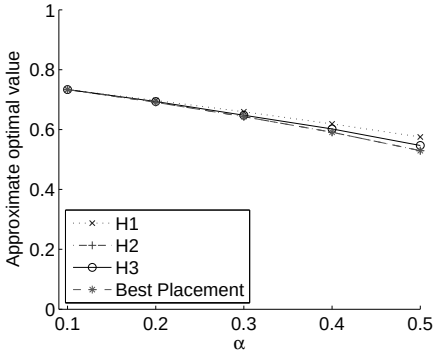


Figure 5.6: *Game values reported w.r.t. alpha,* $\epsilon = 0.25$*,* 10 *random restart*

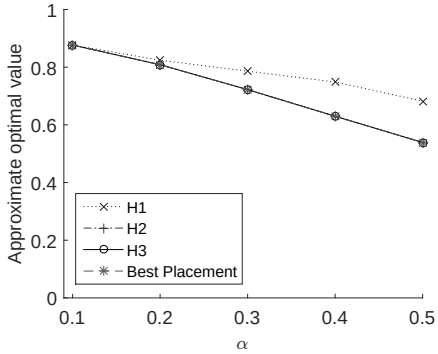


Figure 5.7: *Game values reported w.r.t. alpha,* $\epsilon = 0.25$*,* 10 *random restart*

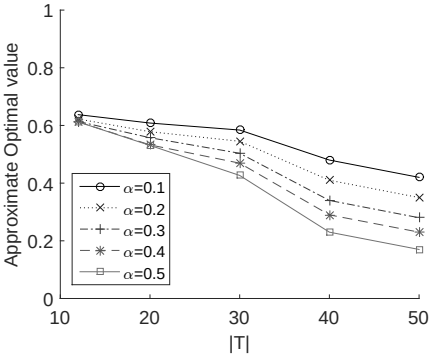


Figure 5.8: *Game values reported w.r.t.* $|T|$*,* $\epsilon = 0.25$*,* 1 *random restart*

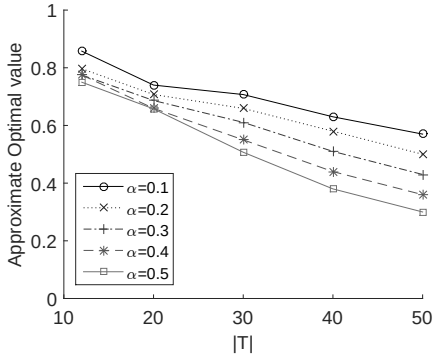


Figure 5.9: *Game values reported w.r.t.* $|T|$*,* $\epsilon = 0.75$*,* 1 *random restart*

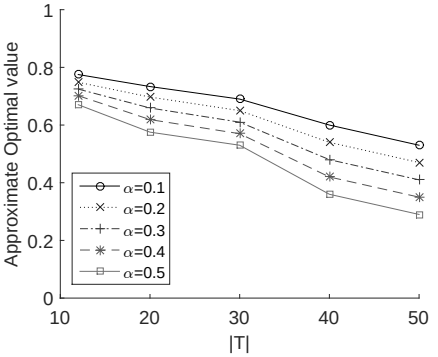


Figure 5.10: *Game values reported w.r.t.* $|T|$*,* $\epsilon = 0.25$*,* 10 *random restart*

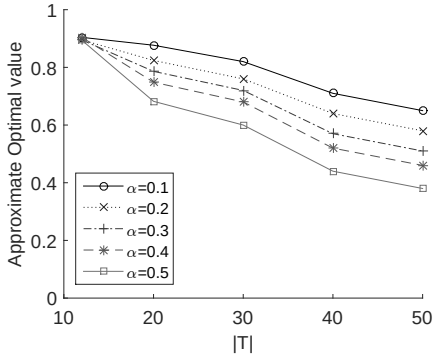


Figure 5.11: *Game values reported w.r.t.* $|T|$*,* $\epsilon = 0.75$*,* 10 *random restart*

Figure 5.12: *Approximate game value with different heuristics.*

Part III

A Coordinated Defense and Multiple Attacks

CHAPTER 6

Multiple Defensive Resources

For the strength of the Pack is the Wolf, and the strength of the Wolf is the Pack.

— Rudyard Kipling

In the previous chapters we have studied how to compute the best strategy to respond to alarm signals for a single mobile defensive resource. Here, we provide a solution for the multi-resource case, addressing the challenge of designing algorithms to coordinate a scaling-up number of defensive resources.

6.1 Original Contributions

In this chapter, we focus on settings with a spatially uncertain alarm system and multiple defensive resources. The challenge is designing algorithms able to scale up with the number of resources. The problem of finding the best Defender's strategy when a number of resources are given is FNP-hard from the case with a single resource. We show that even the problem of finding the minimum number of resources assuring non-null protection to every target, which is of high relevance in practice due to resource costs

and to the need for assuring a minimum level of protection to each target, is log-APX-complete on arbitrary graphs, while it is in FP in tree and cycle graphs, usually representing docks and borders respectively. Then, we study the problem of finding the best strategy to respond to any alarm signal once an allocation of resources in the environment is given, according to different *degrees of coordination* among the resources, each described by an adversarial team game with different forms of strategies (*correlated* or *mixed*). To the best of our knowledge, finding equilibria in adversarial team games is a largely unexplored problem and it may be crucial when the goal is to coordinate different (e.g., defensive) resources in a strategic situation. We provide exact and approximation algorithms and show they perform very well empirically, scaling up to more than 100 targets and 15 resources both in correlated and mixed strategies with a negligible optimality gap.

6.1.1 Chapter Structure

First, in Section 6.2 we report the main feature of the model we already presented in Chapter 4, introducing the presence of multiple resources that can be coordinated by the Defender, up to a certain level. Then, Section 6.3 defines the minimum level of protection we should guarantee to each target, i.e., all the targets, once an alarm signal is raised, should be potentially reached by a patroller within their deadlines. Once this number of resources has been defined, in Section 6.4 we study their patrolling strategies, according to three different degrees of coordination. We summarize our approach, putting together all the contributions in Section 6.5, and finally present an experimental evaluation in Section 6.6, showing the importance of exploiting coordination among defensive resources when such feature is available.

6.2 Game Model with Multiple Defensive Resources

Our security game is a generalization of the one presented in Chapter 4, obtained allowing the Defender to control an arbitrary number of resources, denoted by m, instead of just a single one. We summarize its main features. A *patrolling setting* is a graph $G = (V, E)$ representing an environment where areas that can be attacked are given by n target vertices, denoted as $T \subseteq V$. A target $t \in T$ has a value $\pi(t) \in (0, 1]$ and requires $d(t) \in \mathbb{N}^+$ time units (*penetration time*) to be compromised. Edges in E have unitary cost, while $\omega^*_{i,j}$ is the smallest traveling cost between vertices i and j. An *alarm system* (S, p) generates a signal $s \in S$ if target t is attacked with probability $p(s \mid t)$. S, p and any generated signal s is common knowledge. We call $T(s) = \{t \in T \mid p(s \mid t) > 0\}$ the *support of a signal* s and

$S(t) = \{s \in S \mid p(s \mid t) > 0\}$ the *support of a target* t. Given the novelty of the setting, we start from the basic case where $p(\emptyset \mid t) = p(s \mid \emptyset) = 0$ for any $t \in T$ and $s \in S$, i.e., we assume no false positives or missed detections. The results we derive under such assumption are functional for addressing the general case and can be effectively applied in situations where a small false negatives rate is present. We also assume that $|S(t)| \geq 1$ for any $t \in T$.

A 2-player security game takes place between an Attacker $\mathcal{A}$ and a Defender $\mathcal{D}$. In this game, $\mathcal{A}$ seeks to gain value by compromising some target while $\mathcal{D}$ can control m mobile resources by specifying a movement strategy for them. At any turn of the game, $\mathcal{A}$ and $\mathcal{D}$ play simultaneously: $\mathcal{A}$ observes the position of the m resources and decides whether to attack a target (we assume that $\mathcal{A}$ can instantly reach the attacked target) or to wait. On the opposite side, $\mathcal{D}$ has no information about $\mathcal{A}$ but, if an attack is present, it observes the alarm signal and decides where to move the m resources along the graph. If $\mathcal{A}$ attacks target t and a resource traverses t in any of the next $d(t)$ turns, then $\mathcal{D}$ and $\mathcal{A}$ receive payoffs of 1 and 0, respectively. Otherwise, they receive $1 - \pi(t)$ and $\pi(t)$.

Given a resource j and a signal s, an arbitrary finite length l sequence $r_{s,i} = (r(0), r(1), \ldots, r(l))$ where $r(0)$ is any vertex and $r(i)$ is any target in $T(s)$ is a *route* for resource j under signal s. A route can be instantiated to a graph walk for resource j starting from $r(0)$ and traveling any shortest path between $r(i)$ and $r(i+1)$. For any $i \in \{1, \ldots, l\}$, call $A(r(i)) = \sum_{h=0}^{i-1} \omega^*_{r(h),r(h+1)}$ (the traveling cost required by such walk for reaching $r(i)$). We say that $r_{s,j}$ is a *covering route* if $\forall i \in \{1, \ldots, l\}$, we have that $A(r(i)) \leq d(r(i))$. Any other target t not appearing in the route is not visited or visited after $d(t)$ time units from the start of the route. Covering routes are abstract representations for $\mathcal{D}$'s pure strategies when a signal s is raised. When a resource j follows a covering route, all and only the targets appearing in such route are protected (notice that this representation is without loss of information). A *joint* covering route is an m-tuple $r_s = \langle r_{s,1}, r_{s,2}, \ldots, r_{s,m} \rangle$. To simplify the notation, we will omit signal s when clear from the context.

The resolution approach for $m = 1$ is given in Chapter 4 where it is shown how with no false positives and missed detections, the best strategy is to place the resource on a vertex v and, when a signal s is received execute a *signal response strategy* which, in general, is defined as a mixed strategy over all the covering routes starting from v. Denoted with g_v $\mathcal{D}$'s expected utility when responding to signals from v, then the best strategy is to place the resource on the vertex maximizing such value. This best placement keeps being the optimal strategy even when small missed detec-

tion rates are present. For the case of multiple resources, we can derive an analogous result.

Theorem 6.1. *Without false positives and missed detections, if* $\forall t \in T$ *it holds that* $|S(t)| \geq 1$*, then the optimal strategy prescribes to place each resource on some vertex and execute a signal response from there.*

Proof sketch. Call P^* the set of m vertices specifying the optimal placement of each resource. Denote with g_P^* the value, in terms of expected utility for $\mathcal{A}$, of the resulting signal response game, that is the strategic game where $\mathcal{A}$ has to choose a target to attack and $\mathcal{D}$ must choose a joint covering route where each resource starts from the associated vertex in P^*. Consider any patrolling strategy that, in absence of signals, prescribes to visit at least one vertex $v' \notin P^*$. Since the alarm system is not affected by missed detections, an attack will always raise a signal which, in turn, will raise a response yielding a utility of g_X, where X is the set of vertices corresponding to the current positions of the resources at the moment of the attack. Since $\mathcal{A}$ can observe the position of the resources before attacking, $X = \arg\max_{P \in \Pi}\{g_P\}$, where Π is the space of possible joint locations of the resources given the patrolling strategy. Since we assumed that P^* is optimal, for any $X \in \Pi$ we would have that $g_X \geq g_P^*$. Thus, in absence of signals, $\mathcal{D}$ has no strict incentive to move the resources away from P^*. □

This result allows us to work under the same problem decomposition operated in the single-resource case which rewrites the game as two independent sub-games. The first is the *Signal Response Game* where $\mathcal{D}$ has to determine how to respond when receiving any signal from any given joint location resource. As we will show, solving such game amounts to finding a strategy that, upon reception of a signal, draws and executes a covering route for any resource deployed in the environment. The second sub-game is the *Patrolling Game* where the joint placement for the m resources must be determined. Such placement must encode, for each resource, its starting position and must be selected taking into account the expected utility $\mathcal{D}$ gets when playing the associated Signal Response Games (where the starting positions for the resources are specified by such joint placement). The literature results for the single-resource case show how, despite the game being constant-sum, its resolution is FNP-hard even in tree graphs, its difficulty mainly being in generating the covering routes under signal s and starting from v. An approximation algorithm for such difficult subproblem is given in Chapter 4.

The multi-resource setting clearly inherits this hardness profile, posing the major need for algorithms capable of providing acceptable solutions in

affordable time and showing scalability w.r.t. the number of resources in the game. The presence of multiple resources also poses the problem of determining their minimum number in order to guarantee non-null protection to every target. Last but not least, coordination becomes an issue that, as we will show, has a critical role during signal response. We start from the minimum number of resources problem and then we deal with the game resolution under different coordination schemes.

6.3 Minimizing the Number of Resources

Ideally, the best number of defensive resources depends on both the level of protection that can be achieved and on the costs of the resources. Realistic scenarios often pose the requirement that, for each target $t \in T$, there is at least a resource in a vertex v such that $\omega^*_{v,t} \leq d(t)$, i.e., it can cover t by $d(t)$, thus stopping a potential ongoing attack. Indeed, the authority in charge of the security system's deployment usually requires some protection guarantees over all the targets, even if it forces a non-optimal placement. In other words, leaving a target completely uncovered is generally excluded *a priori*. This entails the existence of, and the need of computing, a minimum number of resources necessary for the protection of any environment. An upper bound can be defined too as the number of resources such that, for any signal s, there is a response strategy to s covering all the targets in $T(s)$ thus assuring the Defender a utility of 1. We deal with this problem by introducing the concept of covering placement and subsequently finding the minimum covering placement.

Definition 6.1 (Covering placement)**.** *A covering placement is a set* $P = \{p_1, \ldots, p_m\}$ *where* $p_i \in V$, $p_i \neq p_j$ *if* $i \neq j$ *and for any* $t \in T$ *it holds that* $\omega^*_{p_i,t} \leq d(t)$ *for some* $p_i \in P$.

6.3.1 Arbitrary Instances

With arbitrary graphs, we state the following results.

Theorem 6.2. *Computing the minimum covering placement is log-APX-hard even in the basic case where all the vertices are targets with penetration times equal to 1.*

Proof. Log-APX-hardness of our problem directly follows from the optimization version of DOMINATING-SET [48] that is known to be log-APX-complete. DOMINATING-SET is defined as follows.

Definition 6.2 (DOMINATING-SET problem)**.** *The optimization version of the DOMINATING-SET problem is defined as:*

- *INPUT: a graph* $\overline{G} = (\overline{V}, \overline{E})$*;*
- *OUTPUT:* $\overline{V}' \subseteq \overline{V}$ *such that* $|\overline{V}'|$ *is minimized under the constraint that for all* $v \in \overline{V} \setminus \overline{V}'$ *there is at least a* $v' \in \overline{V}'$ *such that* $(v, v') \in \overline{E}$*.*

The mapping between the two problems is:

- $V = \overline{V}$;
- $E = \overline{E}$;
- $T = V$;
- $d(t) = 1$ for every $t \in T$;
- $\pi(t) = 1$ any for every $t \in T$;
- $S = \{s_1\}$ with $p(s_1 \mid t) = 1$ for every $t \in T$.

The objective function m of the problem of minimizing the covering placement equals the objective function $|\overline{V}|$ of DOMINATING-SET. Thus, any α-approximation of the minimum covering placement is an α-approximation of DOMINATING-SET. □

Theorem 6.3. *Computing the minimum covering placement is in log-*APX*.*

Proof. The membership to log-APX follows from the fact that every instance of our problem can be directly formulated as an optimization version of a SET-COVER instance and that SET-COVER is in log-APX [38]. Let us notice that our problem is more general than DOMINATING-SET, justifying the need for considering SET-COVER, which is defined as follows.

Definition 6.3 (SET-COVER problem)**.** *The optimization version of the SET-COVER problem is defined as:*

- *INPUT: universe set* U*, family* $\mathcal{L} = \{L \subseteq U\}$*;*
- *OUTPUT: a cover* $\mathcal{C} = \{L \in \mathcal{L}\}$ *of universe set* U *such that* $|\mathcal{C}|$ *is minimum.*

The mapping between the two problems is:

- $U = T$;
- $\mathcal{L} = \{L_v : v \in V\}$ where $L_v = \{t : t \in T \text{ and } \omega^*_{v,t} \leq d(t)\}$.

Notice that $\omega^*_{v,t} \leq d(t)$ can be computed in polynomial time and therefore the mapping can be performed in polynomial time. The objective function $|\mathcal{C}|$ of SET-COVER equals the objective function m of the problem of minimizing the covering placement. Therefore any α-approximation of SET-COVER is an α-approximation of the minimum covering placement. □

The proof of Theorem 6.3 shows that we can find a solution to our problem by formulating it as a SET-COVER and then by using algorithms for this latter problem: Integer Linear Programming (ILP) for finding the exact solution and greedy or local-search algorithms to find an approximate solution, see [38] or [83], respectively.

6.3.2 Special Instances: Tree and Cycle Graphs

Interestingly, with particular instances that are rather common in real-world applications the problem is optimally solvable in polynomial time. Let us start from the following lemma.

Algorithm 12 `TreePlacements`(v)

1: $res_v \leftarrow 0$ for every $v \in V$
2: **if** v is a leaf **then**
3: **return** $(\infty, d(v) - 1)$
4: **else**
5: **for all** $v' \in Succ(v)$ **do**
6: $\big(Cov(v'), UnCov(v')\big) \leftarrow$ `TreePlacements`$\big(v'\big)$
7: **if** $\min_{v'}\{UnCov(v'), d(v)\} - \min_{v'} Cov(v') \geq 0$ **then**
8: **return** $(\min_{v'} Cov(v') + 1, \infty)$
9: **else if** $\min_{v'}\{UnCov(v'), d(v)\} - 1 \geq 0$ **then**
10: **return** $(\infty, \min_{v'}\{UnCov(v'), d(v)\} - 1)$
11: **else**
12: $res_v \leftarrow 1$
13: **return** $(1, \infty)$

Lemma 6.4. *A minimum covering placement in a tree rooted in $\hat{v}$ can be computed in polynomial time with* `TreePlacements`*.*

`TreePlacements` (Algorithm 12) works recursively taking as input a vertex $v \in V$. To ease its description, let us assume that $T = V$. The case including non-target vertices only amounts to minor modifications. Binary variables res_v are initially set to 0 and their assertion corresponds to place a resource on $v \in V$. With $Succ(v) \subseteq V$ we denote all the immediate

successors of v on the path leaving the root. The idea is to recursively allocate resources by processing the graph in a bottom-up fashion, from its leaves to the root. Let us consider a function call for a generic vertex v. By recursively invoking `TreePlacements`(v') for each $v' \in Succ(v)$ we get, for each successor, a *coverage profile* defined with a pair of values $(Cov(v'), UnCov(v'))$. They encode the following conditions under the currently built resource placement. If variables res_v of the subtree rooted in v' constitute a covering placement for the whole subtree, the coverage profile is such that $Cov(v') = k < \infty$ where k is the distance between v' and the closest resource on such subtree and $UnCov(v') = \infty$. Otherwise, the coverage profile is such that $Cov(v') = \infty$ and $UnCov(v') = k < \infty$ where k is the distance from v' by which we need to place a resource to have a covering placement for the subtree rooted in v'.

We start from an empty placement and derive coverage profiles recursively from the base case in which v is a leaf (Line 2). Since $d(v) \geq 1$ and costs are unitary, a resource on a leaf is never necessary: we can always cover it from any ancestor whose distance from v is $\leq d(v) - 1$. Hence the coverage profile for the base case is $(\infty, d(v) - 1)$ (Line 3).

Let us consider the recursive step in which v is a non-leaf vertex. From Line 7 we have all the coverage profiles of each one of the immediate successors of v. We must return the coverage profile of v and decide if a resource must be placed on it. We have three cases.

Case 1 (Lines 7-8): the subtree rooted at v is covered by the resources we placed in such subtree. Hence we do not need any resource in v. Moreover, the ancestor of v will be at distance $\min_{v'} Cov(v') + 1$, where $v' \in Succ(v)$, from the closest of such resources.

Case 2 (Lines 9-10): the subtree rooted at v is *not* covered by the resources we placed in such subtree. We can achieve coverage by placing a resource in v or in any ancestor whose distance from v should not exceed $\min_{v'}\{UnCov(v'), d(v)\} - 1$, where $v' \in Succ(v)$. Since we are trying to minimize the number of resources, there is always interest in postponing a resource allocation in our bottom-up processing of the tree. Therefore, we do not allocate a resource in v and we return the coverage profile naturally resulting from the above considerations (Line 10).

Case 3 (Lines 12-13): no resource in any ancestor of v can complete the coverage for the subtree rooted at v. We are forced to place a resource in v which makes the associated coverage profile equal to $(1, \infty)$ (Line 13).

`TreePlacements` can be adopted also to solve cycle graphs by extracting the n linear subgraphs spanning all the n targets, solving each of them with `TreePlacements`, and then selecting the solution with the least

number of resources. We summarize the above positive results in the following theorem.

Theorem 6.5. *The problem of finding the minimum covering placement in trees and cycle graphs is in* FP.

6.4 Signal Response

We focus on computing a signal response strategy for m resources given a joint placement $P \subseteq V^m$. Notice that we do not assume it to be exactly the minimum one but only to satisfy the lower bound discussed in the previous section. W.l.o.g., we assume that only one signal s is present and that $T(s) = T$, omitting s in our formulas (the general case comes by refining notation).

Any resource i will always move along a covering route. We denote by R_i the set of covering routes for resource i starting from p_i (we will omit the dependency on p_i since P is always fixed in the scope of a signal response game). Covering routes can be computed exactly or approximately by means of the methods proposed in Chapter 4. The presence of multiple resources poses the problem of their coordination [25]. We consider three different coordination schemes for which we define three *Signal Response Oracles* (SROs). Each oracle works on a different adversarial team game and returns the signal response strategy from a given joint placement P under the corresponding scheme. Strategies for $\mathcal{D}$ and $\mathcal{A}$ are denoted as $\sigma^{\mathcal{D}}$ and $\sigma^{\mathcal{A}}$, respectively. If not defined differently, $\sigma^{\mathcal{A}} : T \to [0, 1]$ gives the probability $\sigma^{\mathcal{A}}(t)$ of attacking t.

6.4.1 Full Coordination SRO (FC-SRO)

We assume that $\mathcal{D}$ acts as central planner and executor of the signal response strategy defined on the joint moves of the resources. Equivalently, the defensive resources are players of a constant-sum adversarial team game in which they play correlated strategies. This scheme captures scenarios where resources are connected to a central control unit from which orders are issued (e.g., police patrols equipped with radio transceivers). Formally, we have $\sigma^{\mathcal{D}} : R \to [0, 1]$, $R = \times_{i=1}^{m} R_i$ and $\sigma^{\mathcal{D}}(r)$ is the probability of playing joint covering route r. We define $I(r, t)$ as a function returning 1 if and only if target t is protected by r (with notation overload, we allow r to be both a joint and a single-resource covering route) and 0 otherwise. For any $r \in R$ and $t \in T$, (r, t) is a game outcome where $\mathcal{A}$ and $\mathcal{D}$ receive $U_{\mathcal{A}}(r, t) = (1 - I(r, t))\pi(t)$ and $U_{\mathcal{D}}(r, t) = 1 - U_{\mathcal{A}}(r, t)$, respectively. The game can be solved computing the maxmin strategy by linear programming.

However, each R_i can have exponential cardinality and its computation entails the resolution of an NP-hard problem. Even adopting approximation methods and working with incomplete sets of covering routes, the set of joint covering routes is exponential in the number of resources. Nevertheless, we observe that in our case there is always a maxmin equilibrium where $\mathcal{D}$ plays at most $|T|$ routes with strictly positive probability as proved in [107], showing that working with the complete set of all the joint covering routes is unnecessary.

For this reason, we devise a row-generation approach that iteratively generates rows in the game matrix (corresponding to joint covering routes). The row generation routine first generates, for each resource i, a set of covering routes R_i. Then, it considers a joint covering route r', sets $R = \{r'\}$ and finds the $\mathcal{A}$'s minmax strategy in the corresponding constant-sum game using R as the action space for $\mathcal{D}$. Subsequently, given the $\mathcal{A}$'s strategy, the following ILP is solved to find the $\mathcal{D}$'s best response among all the joint covering routes without their explicit enumeration:

$$
\begin{aligned}
\max \quad & 1 - \sum_{t \in T} \sigma^{\mathcal{A}}(t)\pi(t)(1 - y_t) \quad \text{s.t.} \\
& \sum_{i=1}^{m} \sum_{r \in R_i} I(r,t) x_{ir} - y_t \geq 0 \quad \forall t \in T \\
& \sum_{r \in R_i} x_{ir} = 1 \quad \forall i \in \{1 \dots m\} \\
& y_t \in \{0,1\} \quad \forall t \in T \\
& x_{ir} \in \{0,1\} \quad \forall i \in \{1 \dots m\}, r \in R_i
\end{aligned}
$$

where x_{ir} is a binary variable taking value of 1 when route $r \in R_i$ is selected for resource i, y_t is a binary variable taking value of 1 when target t is protected by the set of selected covering routes (y_t can be safely relaxed on $[0,1]$, turning the ILP into a MILP), and $\sigma^{\mathcal{A}}$ is $\mathcal{A}$'s minmax strategy in the previously solved game. From the solution of this problem we obtain a joint covering route $\hat{r}$ where resource i plays route r if $x_{ir} = 1$. Clearly, $\hat{r}$ is a best response for $\mathcal{D}$ in the incumbent game equilibrium. If $\hat{r} \notin R$, then it is included and the game is solved again, otherwise the set of actions played by $\mathcal{D}$ at the equilibrium is found. However, finding the best response cannot be done in polynomial time unless P = NP, as a consequence of the following theorem.

Theorem 6.6. *Given m resources, a set of single-resource covering routes*

R_i for each resource i, and a subset of targets T', deciding if there is at least a joint route covering all the targets in T' is NP-hard.

Proof. NP-hardness of our problem follows from a direct reduction from the decision version of SET-COVER (see Definition 6.3). The mapping between the two problems is:

- $m = k$;
- $V = U \cup \{v\}$ where v is the starting vertex;
- graph G is fully connected;
- $T' = U$;
- for any $L \in \mathcal{L}$, we have a route r covering all the targets $t \in L$;
- $R_i = \mathcal{L}$ for every resource i;
- $d(t) = |T'|$ for every $t \in T$;
- $\pi(t) = 1$ for every $t \in T$;
- $S = \{s_1\}$ with $p(s_1 \mid t) = 1$ for every $t \in T$.

There is at least a joint route covering all the targets T' if and only if there is at least a cover $\mathcal{C}$ with $|\mathcal{C}| \leq k$. □

Nevertheless, the problem admits a polynomial-time algorithm with constant approximation, returning a joint covering route providing an expected utility of al least $0.63 \cdot OPT_{BR}$ where OPT_{BR} is the expected utility of the best response.

Theorem 6.7. *Given the attacker's strategy, m resources and a set of single-resource covering routes R_i for each resource i, the problem of maximizing the defender's utility admits a polynomial-time algorithm with approximation guarantee of $1 - \frac{1}{e} \simeq 0.63$ where e is the Euler's constant.*

Proof. We show this by exploiting a well-known result from submodular optimization of set functions. We start from some basic definitions.

Definition 6.4 (Submodular set function)**.** *A set function $f : 2^U \to \mathbb{R}$ is* submodular *if, for every $A, B \subseteq U$, it holds that $f(A \cap B) + f(A \cup B) \leq f(A) + f(B)$.*

Definition 6.5 (Monotone set function)**.** *A set function $f : 2^U \to \mathbb{R}$ is monotone if, for every $A \subseteq B \subseteq U$, it holds that $f(A) \leq f(B)$.*

Definition 6.6 (Matroid)**.** *A* matroid *is a pair* (U, I) *such that* U *is a finite set and* $I \subseteq 2^U$ *is a collection of subsets of* U *satisfying the following two properties:*

1. $\forall B \in I, \forall A \subset B \Rightarrow A \in I$;
2. $\forall A, B \in I, |A| < |B| \Rightarrow \exists e \in B \backslash A$ *such that* $A \cup \{e\} \in I$.

Definition 6.7 (Partition matroid)**.** *A partition matroid* (U, I) *is a matroid where* U *is partitioned into disjoint sets* $U_1, U_2, \ldots, U_l$ *with associated integers* $k_1, k_2, \ldots, k_l$*, and sets* I *are defined as* $I = \{X \subseteq U : |X \cap U_i| \leq k_i, \forall i = 1, \ldots, l\}$.

Definition 6.8 (MON-SUBMODULAR-PART-MAT problem)**.** *Given a monotone submodular set function* $f : 2^U \rightarrow \mathbb{R}_+$ *and a partition matroid* (U, I)*, call MON-SUBMODULAR-PART-MAT the problem defined as defined as* $\max_{H \in I} f(H)$.

The objective function (to be maximized) is: $1 - \sum_{t \in T} \sigma^A(t)\pi(t)\,(1 - y_t)$ where σ^A is the strategy of A, $\pi(t)$ is the value of target t and y_t is a binary variable which has value 1 if and only if target t is covered. This is equivalent to maximizing the value of covered targets, defined as $\sum_{t \in T} \sigma^A(t)\pi(t)y_t$.

Let $U = \bigcup_{i=1}^{m} R_i$ be the ground set of the union of all possible covering routes. We consider the function $f : 2^U \rightarrow \mathbb{R}_+$ defined as $f(r) = \sum_{t \in T} \sigma^A(t)\pi(t)y_t$, where r is a joint covering route $r = \langle r_1, \ldots, r_m \rangle$ and y_t is a binary variable taking value 1 if and only if target t is covered by at least one (single-resource) covering route r_i.

We show that function f is monotone submodular.

- f is monotone since all its discrete derivatives are nonnegative, i.e., for every $X \subseteq U$ and $e \in U$ it holds that $\Delta(e|X) \geq 0$.
- The submodularity property is satisfied since $X \subseteq Y \subseteq U$ implies that Y covers at least all the targets covered by X and thus the marginal utility of adding an element $e \in U \backslash Y$ is surely bigger when e is added to X.

We define the partition matroid (U, I) as follows:

- the ground set U is defined as $U = \bigcup_{i=1}^{m} \bigcup_{r \in R_i} \langle i, r \rangle$, describing the set of couples composed of player and covering route;
- the ground set U is partitioned as $U_1, \ldots, U_m$ where $\forall i \in \{1, \ldots, m\}$, $U_i = \bigcup_{r \in R_i} \langle i, r \rangle$;

- the sets I are defined as $I = \{X \subseteq U : |X \cap U_i| \leq 1, \forall i = 1, \ldots, m\}$.

With this formulation, our problem can be seen as a MON-SUBMODULAR-PART-MAT with function f over the partition matroid (U, I). There is a polynomial-time approximation algorithm based on non-oblivious local search [51] with an approximation guarantee of $1 - \frac{1}{e} \simeq 0.63$. □

6.4.2 Partial Coordination SRO (PC-SRO)

Partial coordination models those situations where $\mathcal{D}$ can act as central planner but cannot communicate with each resource to prescribe a joint action. Equivalently, the defensive resources are players of a constant-sum adversarial team game in which they play mixed strategies [17]. Real scenarios falling in this scope can be characterized by resources for which a communication with the control unit is not available (e.g., patrolling units operating in stealth mode). Formally, we define a $(m+1)$-player game where resource players $\mathcal{D}_1, \ldots, \mathcal{D}_m$ compete together against $\mathcal{A}$. Each resource strategy is defined as $\sigma_i^{\mathcal{D}} : R_i \to [0, 1]$, where $\sigma_i^{\mathcal{D}}(r_i)$ denotes the probability of having resource i following covering route $r_i \in R_i$. A game outcome is again defined with (r, t)-r is a joint covering route-where each $\mathcal{D}_i$ receives the same utility $U_{\mathcal{D}}$ as previously defined.

The proper solution concept of this game is the team maxmin equilibrium [121] whose equilibrium constraints are non-linear non-convex. The minmax/maxmin value with 3 or more players may be irrational and not exactly computable even when the payoffs can assume only three different integer values [63]. Approximating the minmax value of 2 players against 1 player within an additive error of $\frac{1}{3z^2}$ where z is the number of actions of each player is NP-hard even with payoffs $\{0, 1\}$ [33]. The work proposed in [63] provides a quasi-polynomial algorithm with complexity $O(z^{l\frac{\log(z)}{\epsilon^2}})$ approximating with additive error ϵ the minmax value with l players with payoffs in $[0, 1]$. The algorithm is not applicable in our case even for toy instances and non-negligible ϵ (e.g., with 10 actions, 2 resources, and $\epsilon = 0.5$, the number of needed iterations is of the order of 10^{18}). Another crucial issue is that the size of $\mathcal{D}$'s payoff matrix increases exponentially in the number of resources. However, by exploiting the structure of the problem, we can provide a Non-Linear Program (NLP) whose size is linear in the number of resources compressing exponentially the size of the game (notice that such a compression is not possible for FC-SRO):

$$\min_{\sigma_1^{\mathcal{D}} \dots \sigma_m^{\mathcal{D}}} v \quad \text{s.t.}$$

$$v - \pi(t) \prod_{i=1}^{m} \left(1 - \sum_{r \in R_i} I(r,t)\sigma_i^{\mathcal{D}}(r)\right) \geq 0 \quad \forall t \in T$$

$$\sum_{r \in R_i} \sigma_i^{\mathcal{D}}(r) = 1 \quad \forall i \in \{1 \dots m\}$$

$$\sigma_i^{\mathcal{D}}(r) \geq 0 \quad \forall i \in \{1 \dots m\}, r \in R_i$$

Our objective is the minimization of v, i.e., the expected utility of $\mathcal{A}$, w.r.t. the strategies played by the resources of $\mathcal{D}$. We observe that such strategies are implemented independently by the resources.

Constraints (2) are the core of the formulation. Let $I : R \times T \to \{0, 1\}$ be the following indicator function:

$$I(r,t) = \begin{cases} 1 & \text{if } t \in r \\ 0 & \text{otherwise} \end{cases}.$$

For each target t, we require that v is greater or equal than the protection probability the Defender can ensure to t. We compute such protection probability multiplying the value $\pi(t)$ of target t by the product of the non-covering probabilities of each resource i of the Defender. Specifically, given a resource i and a target t, such non-covering probability is expressed as $1 - \sum_{r \in R_i} I(r,t)\sigma_i^{\mathcal{D}}(r)$, where $\sum_{r \in R_i} I(r,t)\sigma_i^{\mathcal{D}}(r)$ is the covering probability of t given by i when executing routes r with probability $\sigma_i^{\mathcal{D}}(r)$. We observe that these constraints provide an exponential compression of the size of the game such that the resulting game is linear in the number of resources.

Constraints (3), (4) require that the pure strategies of each resource are feasible. In other words, for each resource i, we require that each action r is played with non-negative probability $\sigma_i^{\mathcal{D}}(r)$ and the sum of such probabilities is equal to 1. We solve the program by means of global-optimization techniques, which are largely unexplored in the equilibrium computation field, able to find the global maximum within a given accuracy and, in the case the posed time limit is expired, able to return the quality of the best solution found w.r.t. the tightest upper bound found. Although global optimization may require exponential time, we will show that in our problem it provides satisfactory performances.

6.4.3 No Coordination SRO (NC-SRO)

When no coordination is allowed among resources, we are ruling out not only strategy correlation but also joint planning. In this scenario, resources are unaware of each other and act as single players against $\mathcal{A}$. Defensive resources are completely non-coordinated and $\mathcal{A}$ can observe the strategy of each resource and play at best by taking into account all of them. This case is modeled with m independent single resource signal response games as defined in Chapter 4, where the game associated with resource i has p_i as starting vertex and is played on the restricted set of targets $T_i = \{t \in T \mid \omega_{p_i,t} \leq d(t)\}$. For each game i, the maxmin strategy is computed taking R_i as the action space of $\mathcal{D}_i$. Thus, given the m resource allocation strategies obtained in this way, we can compute the game value as $1 - \max_{t \in T} \prod_{i=1}^{m} \left(1 - \sum_{r_i \in R_i} \sigma_i^{\mathcal{D}}(r_i) I(r,t)\right) \pi(t)$.

6.5 Overall resolution approach

The resolution approach we use is sketched in Fig. 6.1. We remark that such approach can be adopted when the Defender controls *any* number of resources, which are at least the ones required for the minimum covering placement. For the sake of simplicity, here we consider the case with the minimum number of resources as defined in Sec. 6.3 (we will keep this assumption also for the experimental results shown in Sec. 4.5). We start by tackling the problem of finding the minimum-size resource placement by solving the associated SET-COVER formulation (or our polynomial algorithm with trees and cycles). If the problem cannot be solved exactly in a short time, we adopt the greedy approximation algorithm of [38] and then apply a simple local search [83] to improve the greedy solution. This gives us a number of resources m assuring a (sub)minimal covering placement. As previously stated, in absence of false negatives and false positives, the best strategy when no signal is raised is to statically place the m resources in the best covering placement in terms of expected utility maximization. To deal with this, we resort again to a simple variation of the local search procedure to enumerate covering placements of exactly m resources in an anytime fashion. For each considered covering placement, we compute sets R_i for each resource i as mentioned in the previous section and then we run the signal response oracles we introduced.

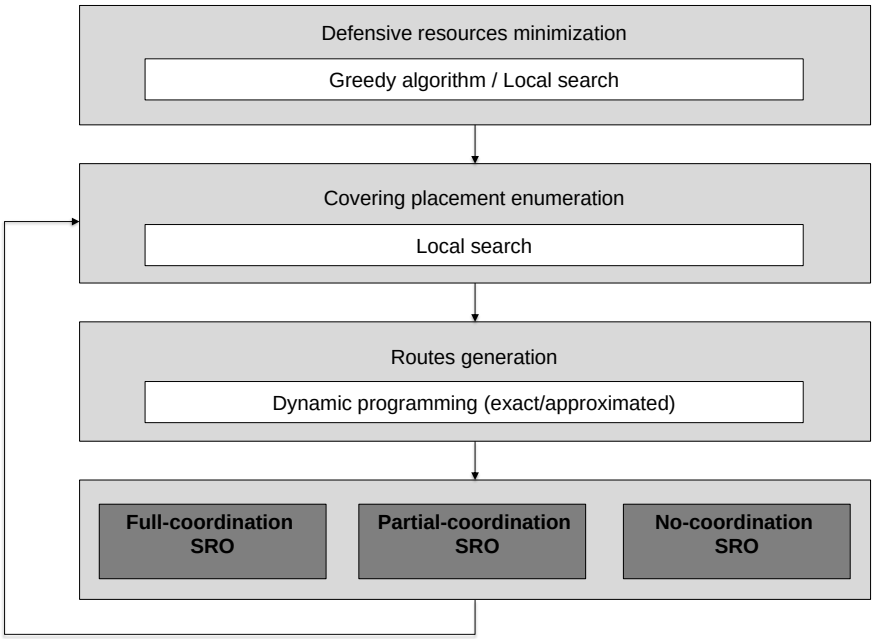


Figure 6.1: *Overview of the resolution approach.*

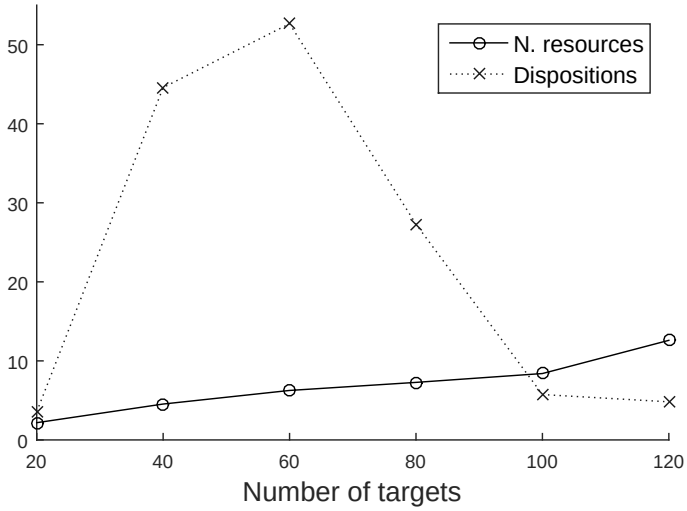
6.6 Experimental Evaluations

To evaluate our algorithms, we implemented a random instance generator that leverages some domain knowledge we gathered by discussing with domain experts from the Italian National Police. The patrolling instances could represent realistic urban environments, such as streets, squares or districts, capturing scenarios with police patrolling units placed in police stations. In the graphs, all the vertices are targets, $|T| \in \{20, 40, 60, 80, 100, 120\}$, edges have unitary costs, the average indegree of each vertex is 3, and $d(t)$ is: $d(t) = 3$ for $|T| \in \{20, 40\}$, $d(t) = 4$ for $|T| \in \{60, 80\}$ and $d(t) = 5$ for $|T| \in \{100, 120\}$; $\pi(t)$ is drawn uniformly from $(0, 1]$. There is one single signal covering all the targets, corresponding to the worst case for the computation of the routes (the problem of scaling up with exponential signals is still open). In our experiments, oracles were run with a deadline of 60 minutes. All the numerical results are averaged on 50 instances for each $|T|$. For the instances employed here, we used the exact method of Chapter 4 to generate the covering routes, requiring a compute time comparable to the approximation algorithm. We use GUROBI to solve exactly LPs and (M)ILPs, BARON to solve exactly NLPs, and Python for the algorithms. We run experiments on a UNIX computer with 2.3GHz CPU and 128GB RAM.

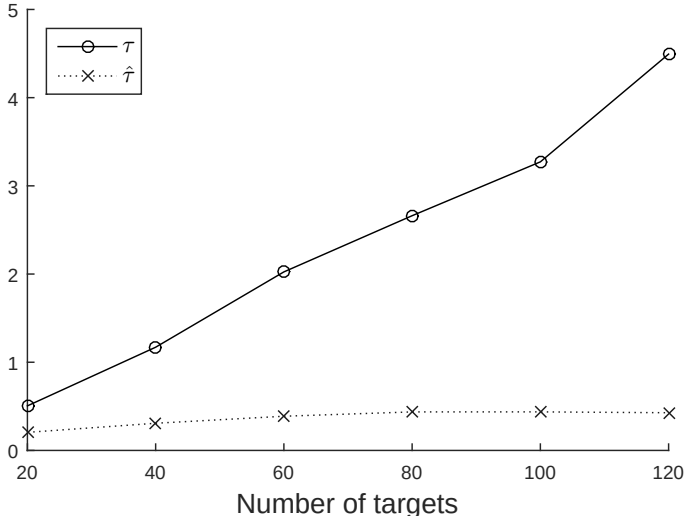
Resources and allocations We initially evaluate the performance of the algorithms to find the best covering placement. The ILP-based method finding the optimal solution scales up to instances with 500 vertices when a time limit of 1 hour is used. The greedy algorithm returning an approximate solution achieves an average optimality gap $< 5\%$ with up to 500 targets requiring a negligible compute time, suggesting that it can provide good suboptimal results even for settings with more than 500 targets. In the following analysis, we use the ILP-based method.

First, we focus on the characteristics exhibited by our experimental settings in terms of number of resources needed for the covering placement and the degree of overlap over the targets covered by the resources, see Fig. 6.2. We observe that the average number of used resources is linear in the number of targets (see Fig. 6.2(a)), which is reasonable in real-world scenarios. To quantify the overlap degree induced by a given covering placement $P = \{p_1, p_2, \ldots, p_m\}$, we define two complementary indicators. We recall that T_i is the subset of targets that can be covered (reached by their $d(t)$) from vertex p_i. Then the following quantity counts the number of *extra coverings* induced by P: $\eta = \sum_{i=1}^{m} |T_i| - |T|$. Notice that $0 \leq \eta \leq (|T| - m)(m-1)$ reaches its maximum value when each resource covers a different subset of $|T| - m + 1$ targets. We denote with $\tau = \frac{\eta}{|T|}$ the average overlap per target and $\hat{\tau} = \frac{\eta}{(|T|-m)(m-1)}$ the normalized overlap. Fig. 6.2(b) shows how these two indicators vary with the number of targets. The index τ grows linearly in the number of targets due to the linear growth of the number of resources while $\hat{\tau}$ has a slower growth since for any resource i the number of covered targets $|T_i|$ is usually not as big as $|T|$.

Now we focus on the covering placement enumeration phase and we evaluate the number of covering placements that our algorithm is able to consider within a 60 minutes deadline. Referring again to Fig. 6.1, notice that each additionally considered placement requires to compute the covering routes sets and to run one of the three SROs. To find an upper bound over the number of covering placements we use the NC-SRO since it is the oracle requiring the minimum compute time. The number of generated covering placements is reported in Fig. 6.2(a). The curve grows reaching its maximum for $|T| = 60$ and then decreases. Indeed, when the size of the problem is small, the algorithm terminates before the deadline, returning few placements. On the other side, when the size of the problem is large, due to the time limit and the large amount of time required by the computation of the covering routes, the time to compute new placements lowers significantly and thus the curve decreases. We remark that we are able to solve large instances up to 120 targets.



(a) Resources and dispositions



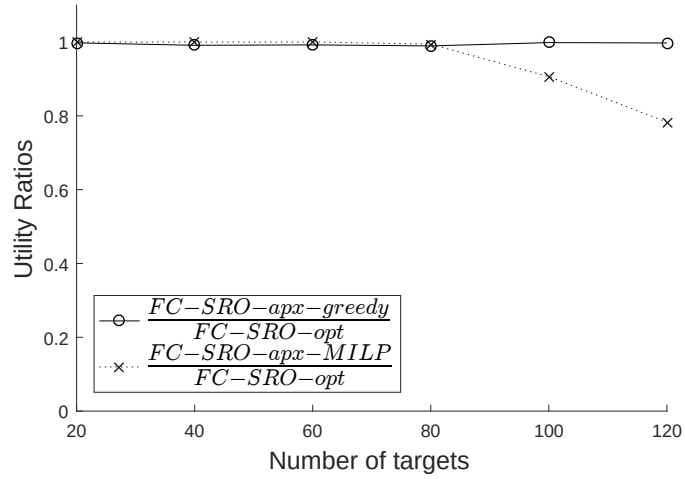
(b) Overlap degree

Figure 6.2: *Analysis of the instances before an attack.*

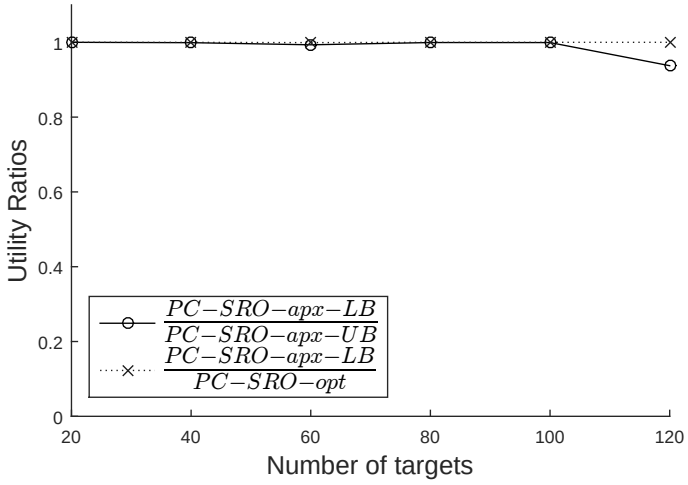
SROs quality and performance We compare the performance of the three SROs, both in terms of utility and compute time. We call FC-SRO-opt the FC oracle using MILP for the row generation, FC-SRO-apx-MILP the same with a time limit of 1 hour, FC-SRO-apx-greedy the FC oracle using the approximation algorithm for the row generation with a time limit of 1 hour. Furthermore, we call PC-SRO-opt the PC oracle, PC-SRO-apx-LB the lower bound returned by the PC oracle with a time limit of 1 hour and PC-SRO-apx-UB the upper bound returned by the PC oracle with a time limit of 1 hour. In Fig. 6.3(a), we report, as the number of targets varies, the average ratio between the utilities returned by FC-SRO-apx-greedy and FC-SRO-opt and the average ratio between the utilities returned by FC-SRO-apx-MILP and FC-SRO-opt. FC-SRO-apx-MILP returns the optimal solution when $|T| \leq 60$. Interestingly, FC-SRO-apx-greedy provides solutions very close to the optimal ones (with an efficiency always larger than 99%) and dramatically outperforms FC-SRO-apx-MILP with 80 targets or more. In Fig. 6.3(b), we report, as the number of targets varies, the average ratio between the utilities returned by PC-SRO-apx-LB and PC-SRO-opt and the average ratio between the utilities returned by PC-SRO-apx-LB and PC-SRO-apx-UB. PC-SRO-apx-LB returns the optimal solution when $|T| \leq 40$. Also in this case, the approximation algorithm (i.e., PC-SRO-apx-LB) provides performance very close to the optimal solution (with an efficiency always larger than 99%).

Fig. 6.4(a) shows the comparison between the different oracles. Partial coordination introduces negligible inefficiency in patrolling games (this does not hold in generic games). Surprisingly, PC-SRO-apx-LB is better than FC-SRO-apx-greedy for 100 targets and therefore it should be considered as approximation oracle for full coordination. This results is due to the fact that FC-SRO-apx-greedy cannot finish its execution within the timeout we set. No coordination is extremely inefficient w.r.t. full and partial coordination. Fig. 6.4(b) shows the time ratio between the different oracles. Interestingly, the full coordination oracles are much faster than the partial coordination oracles when $|T| \leq 60$. Beyond 60 targets, FC-SRO-apx-greedy/MILP and PC-SRO-apx-LB stop at the time limit of 1 hour and so their compute time is the same.

Utility trend in time Finally, we focus on the evolution of the utility in time, after different placements have been enumerated and evaluated. Fig. 6.5 shows two instances with $|T| = 60$ where we compare the performances when using the three different oracles with a 20 minutes time limit: we use FC-SRO-apx-greedy for full coordination and PC-SRO-apx-LB for partial coordination. While NC-SRO is quite constant even though several

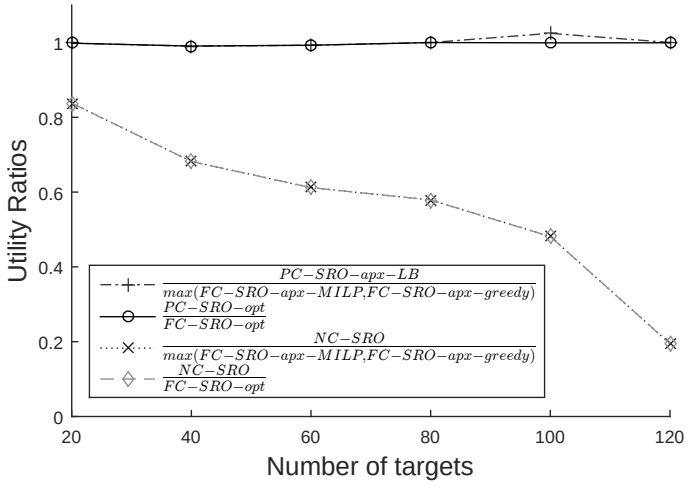


(a) FC-SRO

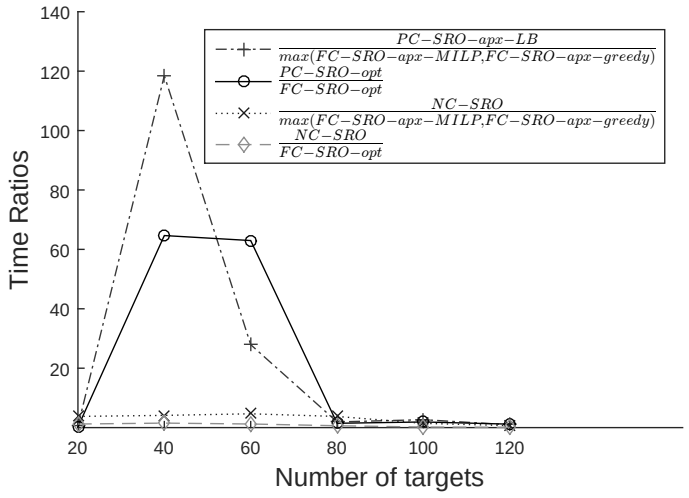


(b) PC-SRO

Figure 6.3: *FC-SRO and PC-SRO evaluation.*



(a) Utility ratios



(b) Time ratios

Figure 6.4: *Utility and time ratios of the three SROs.*

placements are evaluated, FC-SRO and PC-SRO utilities increase, with the former always preceding the latter, being faster in finding good solutions.

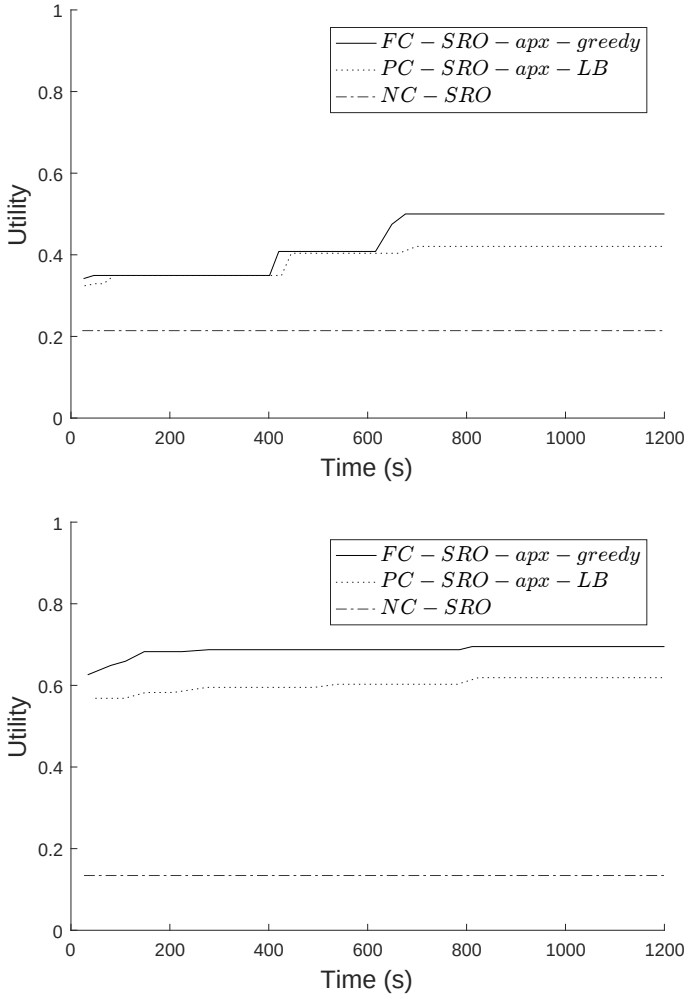


Figure 6.5: *Utility trend in time.*

CHAPTER 7

Multiple Attacking Resources

But of all the Pack of two hundred fighting dholes, whose boast was that all jungles were their Jungle, and that no living thing could stand before them, not one returned to the Dekkan to carry that word.

— Rudyard Kipling

In this chapter, we study, to the best of our knowledge, the first security game in which an Attacker can observe the movements of a patroller on an *arbitrary* graph and perform sequential attacks exploiting multiple resources. Differently w.r.t. security games customarily studied in the literature, here the game is in extensive-form and each player can play multiple times. The Defender is supported by an alarm system and, even when every target is able to raise a target-specific signal, the treatment is challenging. We show that our problem is NP-hard and provide a polynomial-time algorithm finding the optimal strategies on the equilibrium path when the number of Attacker's resources is a fixed parameter. Then, we study the robustness of the Defender's strategy when she makes a wrong guess about the number of Attacker's resources and propose two *online* algorithms, analyzing their *competitive* factors.

The situation we aim at studying is in principle simple. We have an environment modeled as an arbitrary graph and the Defender has a mobile resource that can move along the graph to protect valuable vertices, called *targets*. Every target is perfectly alarmed, raising a specific alarm signal when attacked. On the other side, there is an Attacker with multiple resources that can observe the actions undertaken by the Defender and use her resources to attack the targets adopting any sequential strategy. For instance, the Attacker could attack some targets to deceive the Defender from the most interesting ones and subsequently attack them (this is exactly what happened at the terroristic attacks in London 2005 and Paris 2015). Although the model can be refined along a number of dimensions (e.g., uncertainty over the alarm signals, multiple defensive resources), even the study of the basic version is not straightforward and completely unexplored in the literature so far. To the best of our knowledge, sequential attacks are studied only in patrolling games with very restricted environments [111] and in cybersecurity games where no graph constraints are present [132].

7.1 Original Contributions

We show that, when we restrict the Attacker to use all her resources simultaneously, there is no algorithm that requires polynomial time in the number of Attacker's resources unless $\mathsf{P} = \mathsf{NP}$. Conversely, when the number of resources is a fixed parameter, the problem admits a polynomial time algorithm. Furthermore, fixing the number of attacking resources, the unrestricted model in which the Attacker can perform different sequential attacks admits a non-trivial polynomial time algorithm based on dynamic programming, able to find the strategies *on the equilibrium path* (instead, *off the equilibrium path*, the game may have infinite horizon). The computation of the equilibrium strategies requires the common knowledge about the number of Attacker's resources. This information—as well as a Bayesian prior required in Bayesian games—is unlikely to be common. For this reason, we follow a different approach that we believe to be more suitable in practice, but largely unexplored in Game Theory. We assume that the Defender makes a guess about the number of resources, say k', and plays her best strategy accordingly, and we evaluate the worst-case inefficiency of this strategy when the Attacker has $k \neq k'$ resources. We show that the inefficiency can be arbitrary even when the guess is a wrong estimate—both over and under—for just a single resource. This suggests the use of *online* algorithms that do not need any guess. We provide a tight upper bound over the *competitive factor* when non-stochastic online algorithms are used and we show that the factor

can be improved by resorting to randomization.

7.1.1 Chapter Structure

Section 7.2 introduces the new model, based on the one presented in Chapter 4, reporting its main features and introducing the presence of multiple attacking resources. Then, we focus on the Attacker, who may perform multiple attacks: Section 7.3 studies what happen if she can perform simultaneous attacks while Section 7.4 generalizes such cases, enabling the Attacker to carry on different sequential attacks. Finally, in Section 7.5, we investigate scenarios in which the Defender does not know *a priori* the number of resources available to the Attacker, proving that the loss can be arbitrary.

7.2 Model with Multiple Attacking Resources

Our game is modeled on the basis of the model introduced in Chapter 4, but we allow the Defender to exploit information gained from a *perfect* alarm system and the Attacker to control an arbitrary number k of homogeneous resources, which can be employed simultaneously or sequentially. The environment to patrol is modeled as a graph $G = (V, E)$, where the vertices represent different areas of the environment and the edges represent the connections among the areas. All the edges require one turn to be traversed and we denote with $\omega^*_{i,j}$ the smallest traveling cost in turns between vertices i and j. We define the set of targets $T \subseteq V$ as the set of valuable nodes, characterized by a value $\pi(t) \in (0, 1]$ and a penetration time $d(t) \in \mathbb{N}^+$ to be compromised. A perfect *alarm system* generates a signal s_i from a set of signals S *if* and *only if* target t_i is under attack. S and any generated signal s_i are common knowledge. Since each signal corresponds exactly to one target and *vice versa*, we can safely refer to the signals triggered by the alarm system directly by the targets under attack.

A 2-player security game is played by an Attacker $\mathcal{A}$ and a Defender $\mathcal{D}$. In this game, $\mathcal{A}$ seeks to gain value by compromising some targets employing k resources while $\mathcal{D}$ controls one single patroller by specifying a movement strategy for it. The game can be formulated as an extensive-form infinite-horizon zero-sum game, with $\mathcal{A}$ and $\mathcal{D}$ playing alternatively. Each turn is constituted by one action for the Attacker and the subsequent action for the Defender. At each turn τ, $\mathcal{A}$ may decide to *wait* or to *attack*, with an attack being characterized by the pair $(\tau, \texttt{attacked}(\tau))$ where τ is the turn at which the attack begins[1] and $\texttt{attacked}(\tau) \subseteq T$ is the *the support of the*

[1] We assume $\mathcal{A}$ can instantly reach the attacked target. This can be relaxed as shown in [23].

attack, i.e., the set of the attacked targets. Once $\mathcal{A}$ has employed a resource to make an attack, $\mathcal{A}$ cannot remove such resource from the target until the attack is not concluded. Moreover, each resource can be employed just once by $\mathcal{A}$. On the other side, $\mathcal{D}$ observes the signals triggered by the alarm system (if any) and decides whether to keep its resource in the same area or to move it along the graph. We assume the Defender places her patrolling resource in the environment before the first attack is performed. The choice of such placement is part of the solution to the problem. The utilities of $\mathcal{D}$ and $\mathcal{A}$ are $\left(-\sum_{i=1}^{k} \gamma_i \, \pi(t_i), \sum_{i=1}^{k} \gamma_i \, \pi(t_i)\right)$, where $\gamma_i = 0$ if $\mathcal{A}$ attacks target t_i at τ and the patroller traverses t_i by $d(t)$ turns after τ, catching the attacking resource, otherwise, if $\mathcal{A}$ completes the attack on target t_i without being detected, $\gamma_i = 1$. Once a resource of $\mathcal{A}$ attacking target t has been detected by $\mathcal{D}$, the resource is discarded from the game and, in principle, $\mathcal{A}$ can attack target t again in future by means of another resource (if any available). Similarly, a target can be successfully compromised only once, after that it is considered as a vertex without any value. Finally, if $\mathcal{D}$ protects the environment from all the attacks, her utility is zero, corresponding to the maximum utility she can get. Otherwise, for each target successfully compromised, $\mathcal{D}$ loses the value of such target.

The game being in perfect information, the suitable solution concept is the Subgame Perfect Equilibrium (SPE) [90], which requires a strategy profile to be a Nash Equilibrium for each subgame of the game tree. In the following sections, first we study the restricted case in which $\mathcal{A}$ deploys all the resources simultaneously. Notice that this restriction induces the game to be finite and thus the SPE can be computed before the game is played. Subsequently, we study the unrestricted case. Remarkably, in that case, the game tree may be arbitrarily large—$\mathcal{A}$ may wait indefinitely before making an attack—and therefore there is no way to find a SPE before the execution of the game. Nevertheless, we show that, once k is fixed, there is a polynomial time algorithm to find the equilibrium path of the SPE. Thus, for small values of k, $\mathcal{D}$ can compute the equilibrium path before the play and apply it and, if $\mathcal{A}$ behaves irrationally not following the equilibrium path, $\mathcal{D}$ can compute on-the-flight the equilibrium path of the subgame she is actually playing.

7.3 Facing Simultaneous Attacks

Initially, we study the restricted case in which $\mathcal{A}$ attacks employing all the k resources simultaneously. Tackling such problem is functional to solve

the general case with sequential attacks. Trivially, since $\mathcal{A}$ does not pay any cost to use the resources, she will use all of them. When the attack takes place, k signals will be raised and the Defender must compute a path along the graph to protect the corresponding targets. W.l.o.g., we assume that the attack begins at turn $\tau = 0$. We recall that a *route* is a sequence $r = (r(0), r(1), \ldots, r(h))$ of arbitrary finite length h, where $r(0)$ is any vertex of G and $r(i)$ is any target in $\mathsf{attacked}(0)$, and it said to be *covering*, denoted as r^c, if $\forall i \in \{1, \ldots, h\}$, we have that $A(r(i)) \leq d(r(i))$.

A route can be instantiated to a graph walk starting from $r(0)$ and traveling any shortest path between $r(i)$ and $r(i+1)$. For any $i \in \{1, \ldots, h\}$, call $A(r(i)) = \sum_{l=0}^{i-1} \omega^*_{r(l),r(l+1)}$ (i.e., the time required by the walk to go from $r(0)$ to $r(i)$). Any other target t not appearing in the route is not visited or visited after $d(t)$ turns from the start of the route. Covering routes are abstract representations for $\mathcal{D}$ pure strategies when signals are raised. When $\mathcal{D}$ plays a covering route, all and only the targets appearing in such route are protected.

The resolution approach for $k = 1$ is easy, being a sub-case of the problem studied in Chapter 4. More precisely, the best strategy for the patroller is to stay on a vertex v, wait for a signal s associated with a target t and, when raised, move towards t along the shortest path connecting v to t. The problem can be solved in polynomial time in $|V|$.

To deal with multiple attacking resources, we first have to figure out the space of the actions available to $\mathcal{A}$ and $\mathcal{D}$. The Attacker can attack any subset of k targets, i.e., her actions are all the possible combinations of k targets w.r.t. $|T|$ targets, namely, $\binom{|T|}{k} \approx |T|^k$. On the other side, the Defender must compute the covering routes for the $\mathsf{attacked}(0)$ targets under attack. As it can be seen, the space of the actions is exponential both on the side of $\mathcal{D}$ and of $\mathcal{A}$. Thus, it is natural to wonder whether we have to enumerate all of them or if we can find a compact way to express them. On the Attacker side, we can adopt marginal strategies, i.e., $\mathcal{A}$ plays directly on the targets, and then, exploiting the Birkhoff-von Neumann theorem [27], as done in [75], we map the correlated strategy back to a feasible mixed strategy. Thus, her space of actions can be exponentially compressed. Conversely, nothing can be done for the Defender, being the computation of a covering route a NP-hard problem also in our novel setting and therefore there is no algorithm running in polynomial time in k, unless $\mathsf{P} = \mathsf{NP}$.

Definition 7.1 (Simultaneous-Attack Problem (SA-v))**.**
INSTANCE: an instance of our problem with the patroller in a given vertex v, with $\mathsf{attacked}(0)$ *targets attacked simultaneously by $\mathcal{A}$;*

QUESTION: does $\mathsf{attacked}(0)$ *admit any covering route* r^c*?*

Theorem 7.1. *SA-*v *is* NP*-hard.*

Proof. We provide a reduction from COV-SET, which is proved being NP-hard in Chapter 4.

Definition 4.7 (COV-SET problem)**.** *The COV-SET problem is defined as:*
INSTANCE: an instance of SRG-v *with a target set* T*;*
QUESTION: is T *a covering set? (Equivalently, does* T *admit any covering route* r*?)*

We map an instance of COV-SET to an instance of SA-v by constructing $\mathsf{attacked}(0) = T' = \{t_1, t_2, \ldots, t_h\}$ and associating to each $t_i \in \mathsf{attacked}(0)$ a unique signal s_i. Now, if SA-v admits a covering route for $\mathsf{attacked}(0)$, this means all the targets in $\mathsf{attacked}(0)$ can be covered within their deadlines and, by construction, also T' admits a covering route. The *vice versa* can be proved following similar steps: if T' admits a covering route, there is also a covering route for $\mathsf{attacked}(0)$. □

Finally, observe that when the starting vertex v and $\mathsf{attacked}(0)$ are fixed, the problem can be solved in $O(\,2^k\,k^5\,)$, where k is the size of $\mathsf{attacked}(0)$, while in general the complexity of the problem is $O(\,|V|\,2^k\,k^5\,\binom{|T|}{k}\,) \approx O(\,|V|^{k+1}\,2^k\,k^5\,)$. This follows from the algorithm `DP-ComputeCovSets`, hereafter called $\mathtt{SolveSRG}(v, T')$, proposed in Chapter 4, where v is the starting vertex and T' the set of attacked targets, whose complexity in general is $O(2^{|T'|}\,|T'|^5\,)$. Furthermore, the following result holds.

Theorem 7.2. *SA-*v *problem is* NP*-hard on tree graphs.*

The proof follows from the reduction of Theorem 4.1, which exploits instances with a single signal, and can be thus directly applied to our case.

7.4 Facing Sequential Attacks

The first question we pose is whether $\mathcal{A}$ can gain strictly more from sequential attacks w.r.t. simultaneous attacks. The answer is positive, as stated in the next proposition.

Proposition 7.3. *There exist patrolling games that can provide a strictly higher utility for the Attacker when she performs multiple sequential attacks to the targets rather than a single attack to multiple targets.*

Proof. The proof is given by example. Consider the following graph, where $d(t_1) = d(t_2) = 4$, edges are unitary and $k = 2$.

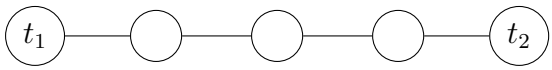


Figure 7.1: *Linear graph employed to prove Proposition 7.3.*

If $\mathcal{A}$ can perform only simultaneous attacks to the two targets, $\mathcal{D}$ will stay in t_1 (or t_2), able to reach t_2 (or t_1) within its deadline. Hence, $\mathcal{D}$ will protect both targets. Conversely, if $\mathcal{A}$ can attack the two targets sequentially, it can be observed that, no matter her position, the Defender can only protect one of the targets, losing the other. For instance, if $\mathcal{D}$ stays in t_1 (or t_2), $\mathcal{A}$ will attack t_2 (or t_1). Subsequently, if $\mathcal{D}$ does not move to cover t_2 (or t_1), then the target under attack is lost. Otherwise, if $\mathcal{D}$ moves to cover t_2 (or t_1), then $\mathcal{A}$ will attack t_1 (or t_2) immediately after the first move of $\mathcal{D}$ and thus $\mathcal{D}$ cannot cover both targets. □

For the sake of clarity, we first study the case in which $k = 2$, and subsequently we discuss the extension of the result to the case in which k is arbitrary.

7.4.1 Addressing $k = 2$

We can formalize the game with $k = 2$ as the following extensive-form game. At the root, the Defender chooses a vertex to place her resource. Then, the Attacker selects her action: she may attack two targets with a simultaneous attack, generating $|T|(|T| - 1)$ nodes, or attack one target or wait, thus generating $|T| + 1$ nodes. In the former case, i.e., both attacking resources have been deployed, we observe that each node corresponds to a game that can be solved invoking `SolveSRG`, after the deadline of the first attacked target has been reduced according to the level of the tree that has been reached. In the latter case, namely $\mathcal{A}$ attacking one target or waiting, $\mathcal{D}$ has at most $|V|$ actions, corresponding to the vertices adjacent to v_s. For each of these actions, we have again $|T|(|T| - 1)$ and $|T| + 1$ nodes. The construction of the tree is performed iterating this process. Thus, the number of generated nodes at turn τ is $\sum_{l=1}^{\tau} |T|(|T| - 1)|V|^l$. In principle, the game tree is infinite, making hard computing the SPE while computing the equilibrium path is polynomial in the size of the problem. In order to find such path, we provide the following algorithm, `PathFinder`.

Let the first attack be performed against target t' with the patroller in v_s. Our goal is to figure out if we can cover t' within its deadline and, if so, what route should be followed. `PathFinder` builds the paths from v_s to t' incrementally, assuming that the second attack may be performed at each step towards t' and solving this problem invoking the `AttackPrediction` al-

Algorithm 13 `PathFinder` (v_s, t')

1: $\forall i,j \in V \times V, M(i,j) = (r_{si}, r^c_{ij}, u_{ij}) \leftarrow (\langle\cdot\rangle, \langle\cdot\rangle, 0)$
2: $M(s,1) = (\langle v_s\rangle, \langle\cdot\rangle, 0)$
3: **for all** $j, i \in V \times V$ s.t. $M(i,j) \neq (\langle\cdot\rangle, \langle\cdot\rangle, 0)$ **do**
4: $\quad (r^c_{min}, u_{min}) \leftarrow$ `AttackPrediction` (v_i, t', j)
5: $\quad M(i,j) \leftarrow (\cdot, r^c_{min}, \min(u_{min}, u_{ij}))$
6: $\quad$ **for all** $(v_i, v_{adj}) \in E$ **do**
7: $\quad\quad$ **if** $u_{ij} \leq u_{adj,j+1}$ **then**
8: $\quad\quad\quad M(adj, j+1) \leftarrow (\langle r_{si}, v_{adj}\rangle, \cdot, u_{ij})$
9: $u^* \leftarrow \max(u_{1|V|}, u_{|V||V|})$
10: **return** $(r^*, r^{c,*}, u^*) \leftarrow M(i,j) | M(i,j) = (\cdot, \cdot, u^*)$

Algorithm 14 `AttackPrediction` (v, t', j)

1: $d(t') \leftarrow d(t') - j$
2: **for all** $t \in T, t \neq t'$ **do**
3: $\quad (R, U) \leftarrow$ `SolveSRG`$(v, \{t\} \cup \{t'\})$
4: **return** $(r^c_{min}, u_{min}) \leftarrow (\arg\min_i U_i, \min_i U_i)$

gorithm. `PathFinder` uses a $|V| \times |V|$ matrix M, where each element $M(i,j) = (r_{si}, r^c_{ij}, u_{ij})$ consists of the route r_{si} from v_s to v_i, the best covering route r^c_{ij} from vertex v_i at time instant j to cover the targets under attack and the utility u_{ij} for $\mathcal{D}$ associated to such route. The rows represent the vertices of the graph while the columns represent the time steps. Each cell is set to $(\langle\cdot\rangle, \langle\cdot\rangle, 0)$ except for the one corresponding to the starting point, $M(v_s, 1)$, which contains the route constituted only by v_s (Alg. 13, Lines 1-2). M is filled column by column, as time passes by (Alg. 13, Lines 3-8). `PathFinder` processes elements of M corresponding to visited vertices. Let us focus on a cell of the matrix, $M(i,j)$. `AttackPrediction` is called on v_i (Alg. 13, Line 4): first, it reduces the deadline of t' according to the turn we are considering, then it calls `SolveSRG` to obtain the set of covering routes R that should be followed if the second attack would happen against $t \neq t'$ with $\mathcal{D}$ in v_i at time instant j and the utilities U associated to the routes (Alg. 14, Lines 2-3). After performing such computation, since $\mathcal{A}$ will carry on her second attack in our worst-case scenario, the algorithm saves the covering route and the value corresponding to the worst attack in r^c_{min}, u_{min}, respectively, (Alg. 14, Line 4) and return them back. The values of the current cell are updated, taking r^c_{min} and the minimum value between u_{min} and the previous one contained in the cell, namely u_{ij} (Alg. 13, Line 5). The values of the cells corresponding to vertices v_{adj} adjacent to v_i are also updated: if the minimum value between the previous one contained

in $M(adj, j+1)$ and the updated value of u_{ij} is the latter, the route from v_s to v_i to which v_{adj} is appended and u_{ij} are saved (Alg. 13, Lines 6-8). Finally, the algorithm returns the element of M containing the highest utility for the Defender (Alg. 13, Line 10), i.e., either the best path to reach t' or standing still in v_s.

Theorem 7.4. `PathFinder` *is optimal.*

Proof sketch. We report just a sketch of the proof, based on the following lemmata.

Lemma 7.5. *Let the first attack be performed against target t' with the patroller in v_s. Then, in absence of the second attack, the Defender will never traverse twice the same vertex along her path from v_s to t'.*

Proof. Let p be the path from v_s to t' followed by $\mathcal{D}$ when no other attack is occurring. Being the graph unitary, the patroller will reach t' after p time units. If $\mathcal{D}$ should traverse a vertex twice, she would reach t' in more than p time units, without getting any increase in terms of utility. Thus, $\mathcal{D}$ will follow p. □

Lemma 7.6. *Let p_1, p_2 be two paths that visit the same node v at time instant τ and u_1, u_2 the utilities obtained considering the worst-attack that can be perfomed when traveling along such paths. If $u_1 \geq u_2$, then traveling along p_1 dominates traveling along p_2.*

Proof. After visiting v, both paths will have the same expected utility associated to the next steps towards t'. Thus, we can compare p_1, p_2 w.r.t. their utilities u'_1, u'_2 in reaching v. Since $u_1 \geq u_2$, then $u'_1 \geq u'_2$ and so traveling along p_1 dominates traveling along p_2. □

Lemma 7.7. *Let the first attack being performed against target t' with the patroller in v_s. Then, the Attacker will perform the second attack while the patroller is moving from v_s to t'.*

Proof. We consider the second attack occurring when $\mathcal{D}$ is going from v_s to t along p and on her way back, from t' to v_s, following path p' (following a path different from p on the way back is a dominated action). If the Attacker can complete the attack while $\mathcal{D}$ is moving along p', then the same attack can be completed also when $\mathcal{D}$ is moving along p. But if the Attacker cannot complete the attack while $\mathcal{D}$ is going back to v_s, then $\mathcal{A}$ might be able to complete the attack when $\mathcal{D}$ is traveling along p according to the patrolling policy. □

Lemma 7.8. *At the equilibrium, the size of the game tree is finite.*

Proof. From Lemma 7.5, we know that the patroller will never visit twice the same node and so the path of the Defender cannot be longer than $|V|$. Moreover, Lemma 7.7 tell us that the second attack will be performed while the patroller is traveling to cover the first target. Thus, the depth of the game tree is limited by $|V|$. □
From the above lemmas, it follows the optimality of `PathFinder`. This concludes the proof sketch. □

Before actually running `PathFinder`, we compute the shortest paths among all the vertices, with a cost of $O(|V^3|)$, by means of the Floyd-Warshall algorithm [43]. We build a matrix M, whose size is $|V|^2$, with the cost of filling each $M(i, j)$ dominated by the calls to `AttackPrediction`, whose cost is $|V|$ (invoking `SolveSRG` has a cost that is constant in k since it is a fixed parameter). The complexity of invoking `PathFinder` is then $O(|V|^3)$. We call `PathFinder` for each $t' \in T$, returning the best routes with the associated utilities. At the end of these executions, we know which are the best responses for the patroller from v_s when any target is attacked. We repeat this procedure for each vertex $v \in V$, reaching a total complexity of $O(|V|^5)$. After this, the Defender knows, for each $v_i \in V$, the utility $u^*_{v_i}$ of placing the patrolling resource in v_i and how to best respond from there to each possible attack. The best starting placement for the patroller is then $v^* = \arg\max(u^*_{v_1}, u^*_{v_2}, \ldots, u^*_{v_{|V|}})$.

Thus, once k is fixed, the computation of the equilibrium path can be done in polynomial time. Moreover, if the Attacker is irrational, i.e., playing off the equilibrium path, `PathFinder` can still be used to find the best response in polynomial time.

7.4.2 Extending to Arbitrary k

The extension of `PathFinder` to an arbitrary number of resources is very involved, we provide the rationale behind it.

First, we analyze what happens if $\mathcal{A}$ performs her first attack employing $k - 1$ resources. We introduce two additional features w.r.t. `PathFinder`. First, we add a third dimension to M, say l, considering all the combinations with repetitions of the $k-1$ targets under attack (a target that has already been attacked but not successfully compromised can be attacked again). We move along l according to the targets covered by the patroller or whose deadline is expired. Second, since a target can be visited even after its deadline, each element of M contains also the set of covered targets.

We follow steps similar to `PathFinder`, filling M column by column and for increasing l, calling an extended version of `AttackPrediction`

that takes as input the subset of targets T' under attack. According to targets whose deadline is expired and the ones covered by the covering route, we fill the corresponding cell of M and update the cell of the adjacent vertices, adopting the same rationale employed by `PathFinder`. Once M has been completely processed, we select the last matrix along the l dimension and apply the same operations we performed at the end of `PathFinder` (Alg. 13, Lines 9-10), returning the element of M associated with the best utility for the Defender.

Here, the size of M is $|V| \cdot (k-1)|V| \cdot \binom{|T|+k-1}{k}$ and this time calling `AttackPrediction` has a cost equal to $O(2^k k^5)$ due to the call to `SolveSRG`. So, the cost of invoking the extended version of `PathFinder` is $O((|V|+k)^k 2^k k^6 |V|^2)$. We run this algorithm for all subsets of $k-1$ targets, namely $\binom{|T|}{k-1}$, to know which are the best responses for the patroller from v_s when any subset of $k-1$ targets is attacked. We then repeat this procedure for each $v \in V$ and select the vertex with the highest utility as the starting placement for the patroller, reaching a total complexity of $O((|V|+k)^k |V|^{k+2} 2^k k^6)$ to solve our problem.

Now let us consider the general case in which $\mathcal{A}$ employs $k-k'$ resources for the first attack. This means we must solve all the problems with $k-k'+1$ possible targets under attack and for each of these, all the problems with $k-k'+2$. We proceed recursively until we reach problems with $k-1$ targets under attack: we solve these problems as described above and propagate back the solutions, solving step by step all the problems until we reach the original one.

To compute the solution for the general problem, we have to solve a very large number of problem, namely $\prod_{h=1}^{k'-1} \binom{|T|}{k-k'+h} \approx O(|V|^{k^2})$, each with the exponential complexity showed above.

7.5 In Absence of Common Knowledge

Till now we assumed that the Defender knew *a priori* the number of resources the Attacker could employ to perform her attack. In real-life scenarios, however, it is very unlikely that such information is available for the Defender to be used. A possible approach to face this issue is employing a probability distribution on the number of resources available to the Attacker as required in Bayesian games. Unfortunately, in this case, the problem is even computationally harder and the assumption that such a prior is common knowledge is even more unlikely. Thus, we study the scenario in which knowledge about the number of attacker's resources is not common.

7.5.1 Robustness to a Wrong Guess

Initially, we study the scenario where $\mathcal{D}$ makes a guess about the number k' of resources available to $\mathcal{A}$, being the number of such resources actually equal to k. We study the loss in the worst-case for the Defender of playing her best strategy against k' resources when the guess is wrong, assuming the Attacker to be rational and knowing the strategy of the Defender, i.e., her guess k'. We denote with $\sigma^*_{\mathcal{D},k}$ and with $v^*_{\mathcal{D},k}$, respectively, the optimal strategy of the patroller and the value of the equilibrium when the number of resources available to $\mathcal{A}$ is k and the guess is correct. Similarly, $v_{\mathcal{D},k,k'}$ is the value obtained by the Defender playing strategy $\sigma^*_{\mathcal{D},k,k'}$ when the Attacker has k resources but the guess is k', and $\mathcal{A}$ plays her best response to $\sigma^*_{\mathcal{D},k,k'}$. Before tackling the problem, in order to have a correct definition for the loss, we have to normalize the values of the targets onto $[0, 1]$. Specifically, given an instance where $\pi(t)$ are the initial values for the targets, we change their values follows: $\pi'(t) = \frac{\pi(t)}{\sum_{t_i \in T_{topk}} \pi(t_i)}$, where T_{topk} is obtained re-labeling the targets in T in descending order w.r.t. $\pi(t)$ and selecting the first k. This way, the Defender gets a utility equal to 1 if no targets are conquered by the Attacker while she gets 0 if all the top k targets are conquered. In other words, a utility of 0 means that there is not other outcome that is worse for the Defender. We define the relative loss of a strategy based on a wrong guess w.r.t. the best strategy based on the correct guess by resorting to the tools used in online algorithms, where the performance of an algorithm is compared w.r.t. the performance of the clairvoyant algorithm, which *a priori* knows all the information. In particular, we resort to the concept of *competitive factor* [50].

Definition 7.2 (Competitive factor)**.** *The competitive factor* Γ *of an algorithm is given by the worst-case ratio* $\frac{v}{v^*}$*, where* v *is the value given by the algorithm and* v^* *is the maximum value given by a clairvoyant algorithm.*

We recall that an algorithm is said *competitive* when $\Gamma > 0$. In our case, the competitive factor when the Defender makes a guess k' while the Attacker has k resources is equal to $\Gamma = \frac{v_{\mathcal{D},k,k'}}{v^*_{\mathcal{D},k}}$, once excluded all the instances in which $v^*_{\mathcal{D},k} = 0$. We state the following.

Theorem 7.9. *If the Defender makes a wrong guess w.r.t. the actual number of resources available to the Attacker, then* $\Gamma = 0$*, independently by the value of the guess.*

Proof. We consider separately the case in which the Defender is *underestimating* the number of the attacker's resources from the case in which she is

overestimating it.

Consider the case in which the Defender is underestimating the number of resources available to the Attacker. Let us consider a graph composed of a clique of size $k - k' + 1$, with unitary edges, $k - k'$ targets, say t_c, with $\pi(t_c) = 1, d(t_c) = k - k'$, and the non-target vertex v of the clique connected to k' other targets, t_o, with $\pi(t_o) = \epsilon, d(t_o) = k$, through edges with weight equal to k. If $k' = k$, starting in v, $\mathcal{D}$ will lose at most $(k-1)\epsilon$ since she will wait for attacks to targets t_c and, if they happen, the patroller will be able to cover all of them. According to our normalization, $v^*_{\mathcal{D},k} = 1$. On the other side, if $k' < k$, $\mathcal{A}$ will employ k' resources on targets t_o and $k - k'$ to attack t_c. From v, $\mathcal{D}$ will be able to save only a target t_o, so $v_{\mathcal{D},k,k'} = \epsilon$. Since $\Gamma = \epsilon$, $\Gamma \xrightarrow{\epsilon \to 0} 0$.

Consider the case in which the Defender is overestimating the number of resources available to the Attacker. Let us consider a *star* graph with unitary edges and k targets connected to a central node v. The values of the targets are $\pi(t) = 1 - \epsilon$ except for one target, t_{max}, whose value is $\pi(t_{max}) = 1$. From v, if $\mathcal{D}$ could employ the clairvoyant algorithm, she would always be able to save one target, thus $v^*_{\mathcal{D},k} = 1 - \frac{(k-1)(1-\epsilon)}{1+(k-1)(1-\epsilon)} = \frac{1}{1+(k-1)(1-\epsilon)}$. On the other side, since $\mathcal{A}$ employs $k < k'$ resources, the Defender will wait for the last $k - k'$ targets to be attacked since t_{max} could be among them. In this case, $v_{\mathcal{D},k,k'} = 1 - \frac{k(1-\epsilon)}{1+(k-1)(1-\epsilon)} = \frac{\epsilon}{1+(k-1)(1-\epsilon)}$. Thus, $\Gamma \xrightarrow{\epsilon \to 0} 0$. □

The above result shows that playing a strategy that is optimal for a given guess is not robust in practice, since it is sufficient that the guess is wrong just by one resource to have an arbitrary loss w.r.t. the optimal clairvoyant solution.

7.5.2 Online Algorithms

We have seen that if the guess of the Defender on the number of resources available to the Attacker is wrong, underestimating and, more surprisingly, overestimating such number leads to an arbitrarily small value for $\mathcal{D}$. Now, we ask whether there exists a competitive online algorithm that is independent by the actual number of resources of the Attacker, relying only on the observed attacks.

Theorem 7.10. *There is no deterministic online competitive algorithm with a competitive factor better than* $\frac{1}{k-1}$*, where* k *is the actual number of resources available to the Attacker.*

Proof. Let us consider the following instance. A vertex v is connected to $\max\{1, k-1\}$ targets, with deadlines equal to $\max\{1, 2k-1\}$ through uni-

tary edges, and to t_f with an edge whose cost is $2k$, where $d(t_f) = 2k$. The value of all the targets is $\pi(t) = 1$.

We split the proof considering first $k = 1$ and then $k > 1$. If $k = 1$, with the patroller in v, the optimal strategy for $\mathcal{D}$ is to protect the first target under attack, obtaining a value of 1. Any online algorithm that does not prescribe to cover the first target under attack will have a competitive factor equal to 0 since the Defender will take 0. If $k > 1$, the optimal strategy for the Defender is to cover the first target under attack if and only if such target is not t_f. This way, in the optimal case, the Defender will take a utility of $\frac{k-1}{k}$, protecting all the targets except for one. Conversely, any strategy prescribing to defend t_f when attacked as the first target would lead to a competitive factor of $\frac{1}{k-1}$. Thus, the best competitive factor is $\frac{1}{k-1}$. □

It is now worth asking whether we can achieve a better result employing randomization.

Theorem 7.11. *Let Γ_d be the best competitive factor of a deterministic online algorithm. There exists a randomized online algorithm with competitive factor Γ_r such that $\Gamma_r > \Gamma_d$ and, asymptotically, $\Gamma_r \to \Gamma_d$.*

Proof. We prove the theorem only in the worst case for deterministic online algorithms. This allows us to show the improvement one can obtain by means of randomization in the worst case.

Let us consider a graph composed of a clique with unitary edges of $k - h + 1$ nodes, $k - h$ of which are targets, say t_c, with $d(t_c) = k - h$, and the non-target vertex v of the clique connected to h other targets, t_o, with $d(t_o) = k$, through edges with weight equal to k. $\pi(t_c) = \pi(t_o) = 1$.

We already know (see Thereom 7.10) that a deterministic approach reaches a competitive factor $\Gamma_d = \frac{1}{k-1}$. We propose the following randomized algorithm: let v be the starting point for $\mathcal{D}$ and, whenever an attack occurs, she has probability $\frac{1}{2}$ of protecting the target under attack while she stands still in v with probability $\frac{1}{2}$. Since we want to compute Γ_r, we normalize the values of the targets dividing them by k. If there are multiple sequential attacks on targets t_o, the utility of protecting them is equal to $\frac{1}{k}\left(\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^h}\right) = \frac{1}{2k}\sum_{i=0}^{h}\left(\frac{1}{2}\right)^i$ while standing still and covering attacks against t_c gives a utility equal to $\left(\frac{1}{2}\right)^h \frac{k-h}{k}$. The clairvoyant algorithm achieves a utility of $\frac{k-h}{k}$. Thus, $\Gamma_r = \frac{\frac{1}{2k}\sum_{i=0}^{h}\left(\frac{1}{2}\right)^i + \left(\frac{1}{2}\right)^h \frac{k-h}{k}}{\frac{k-h}{k}} = \frac{1}{k-h}\left(1 - \left(\frac{1}{2}\right)^{h+1}\right) + \left(\frac{1}{2}\right)^h$. Among all the instances, we want to find the worst, so, given k, we want to minimize Γ_r w.r.t. h. In the following table, we report the values of Γ_r and Γ_d achieved for different k, taking the h minimizing Γ_r.

k	3	4	5	6	7	8	9	10	100
Γ_r	0.87	0.69	0.54	0.44	0.36	0.30	0.26	0.22	0.01
Γ_d	0.50	0.33	0.25	0.20	0.17	0.14	0.12	0.11	0.01

Table 7.1: *Values of the competitive factors of a randomized and a deterministic online algorithm, respectively.*

As it can be seen, for small values, a very simple randomized algorithm can ensure a competitive factor that is twice better than the one achieved by a deterministic algorithm. Moreover, $\Gamma_r \xrightarrow{k\to\infty} \Gamma_d$. □

Notice that, beside the definition we have provided, we can actually express the competitive factor as an *additive factor*, according to an ϵ-NE fashion.

Definition 7.3 (Additive competitive factor)**.** *The additive competitive factor Γ' of an algorithm is given by the worst-case difference $v - v*$, where v is the value given by the algorithm and v^* is the maximum value given by a clairvoyant algorithm.*

We observe that Γ' has been defined such that, if the guess is wrong, it results negative, meaning the Defender is incurring in some loss. In this case, adopting a similar approach, it can proved that the loss strictly depends on the difference between the guess k' of the Defender and the actual number of resources available to the Attacker, i.e., k. Specifically:

$$\Gamma' = \begin{cases} -(k-k') + \epsilon, & k' < k \\ k' - k(1-\epsilon), & k < k' < 2k \\ -k(1-\epsilon), & k' \geq 2k. \end{cases}$$

Part IV

Facing the Unknown

CHAPTER 8

Learning and Exploiting the Attacker's Profile

My momma always said, "Life is like a box of chocolates. You never know what you're gonna get.

— Forrest Gump

In this chapter, we study a security game in which the Defender faces a rationally bounded adversary, whose behavior is not known *a priori*, being one among a set of possible behavioral profiles.

The literature provides a number of models of bounded rationality [89, 9]. Probably, the most elegant one is the Quantal Response (QR) [80], which fixes the probability distribution over the non-optimal actions of an agent on the basis of their optimality gap.

The crucial issue is that all the works on bounded rationality make an assumption about the specific behavior of the opponent and this assumption could be never met in real-world applications. In that case, such an assumption may lead to an arbitrarily loss for the Defender. For instance, a terrorist could decide either to attack a target that is not patrolled, since she is sure to not be caught, or a target not so valuable itself, but that would cause a huge

panic reaction in the population (e.g., this is what happened in November 2015 in Paris attacks at the Bataclan theatre). The same challenge may be faced by a company that aims at planning the production of a product and has to decide when and how it is convenient to enter the market when another company is already the Defender in such a market—this is the well-known *Stackelberg duopoly* [119]. Whenever the assumption of perfect rationality is not met, each agent may in principle exploit her opponent's strategy. This is a "natural" *online identification* problem: in fact, the Defender aims at identifying the Attacker's behavioral profile to exploit at best the potential non-rationality of the opponent, while minimizing the regret due to the initial lack of information.

Differently from the existing literature, we study the original single-agent-learning problem in which the behavior of the Attacker is one among a set of possible behavioral profiles—e.g., the rational one (i.e., best response), a rationally bounded one based on the QR, a stochastic strategy—and the Defender does not initially know it, but she can learn it by exploiting the opponent's behavior at best. The question we pose is *whether* and *how* the Defender can learn the behavioral profile of an Attacker in Security Games and so, our goal is to design online learning techniques capable to identify the behavior of the Attacker while minimizing the regret due to the initial lack of information. We propose a set of algorithms based on sequential learning techniques [35] that are able to infer the behavior of the Attacker the Defender is playing against exploiting the repeated interactions between the two players.

8.1 Original Contributions

We define a novel scenario in which a Defender plays against a Attacker whose behavior is unknown but it belongs to a set of known profiles. We show that state-of-the-art bandit and expert algorithms—suitable for our problem—suffer from a linear and logarithmic regret, respectively, in the length of the time horizon. Thus, we introduce two novel approaches to deal with our problem, bridging together game-theoretical techniques and online learning tools. In the first approach, the Defender has a *belief* about the Attacker and updates it during the game. We name the algorithm `Follow the Belief` (`FB`) and we provide a finite-time analysis showing that the regret of the algorithm is constant in the length of the time horizon. In the second approach, namely `Follow the Regret` (`FR`), the learning policy is driven directly by the estimated expected regret and is based on a backward induction procedure. Finally, we provide a thorough experimental

evaluation in concrete security settings, comparing our algorithms with the main ones available in the state of the art of the online learning field and showing that our approaches provide a remarkable improvement in terms of expected pseudo-regret minimization.

8.1.1 Chapter Structure

The rest of the chapter is organized as follows. Section 8.2 formally introduces and define the problem we are going to tackle. Then, in Section 8.3, we present the type of Attacker's profiles we considered, i.e., either stochastic or strategy aware. Section 8.4 shows how to tackle the problem: we propose two algorithms based on different approaches and we provide a regret analysis. Finally, in Section 8.5, we experimentally evaluate the pseudo-regret of the algorithms in concrete security games, showing that our algorithms outperform the online learning algorithms available in the state of the art.

8.2 Problem Formulation

We now introduce our problem, still formulated as a security game but whose resolution will require different techniques than the ones adopted in a customary game-theoretic setting. For the sake of clarity and presentation, in this chapter, we slightly change the notation adopted previously.

Let us consider a 2-player normal-form repeated game $\mathcal{G}_N$ defined over a finite number of rounds $N \in \mathbb{N}$, where a Defender $\mathcal{D}$ and an Attacker $\mathcal{A}$ play against each other in some environment with some valuable targets $\mathcal{M} = \{1, \dots, M\}$, characterized by values $\mathbf{v} = (v_1, \dots, v_M)^T, v_m \in (0, 1]$. The goal of the Defender $\mathcal{D}$ is to protect such targets while the Attacker $\mathcal{A}$ aims at compromising them. The space of actions of $\mathcal{D}$ and $\mathcal{A}$ is given by the set of targets such that $\mathcal{D}$ chooses the target to protect, while $\mathcal{A}$ chooses the target to attack. The course of the game is represented in 8.1. Specifically, at each round $n \in \{1, \dots, N\}$, the Defender $\mathcal{D}$ announces the strategy she commits to $\boldsymbol{\sigma}_{D,n} \in \Delta_M$ (Line 1), where Δ_M denotes the M-dimensional simplex, while $\mathcal{A}$ observes such a commitment (Line 2).

Then, they concurrently play their action over the target space (Line 3), i.e., the Defender plays actions $i_{D,n} \in \mathcal{M}$ according to $\boldsymbol{\sigma}_{D,n}$ while $\mathcal{A}$, the Attacker, plays $i_{A,n} \in \mathcal{M}$ according to some Attacker model $\boldsymbol{\sigma}_A(\boldsymbol{\sigma}_{D,n}) \in \Delta_M$. The game is zero-sum: if $\mathcal{D}$ and $\mathcal{A}$ choose the same target at round n, they both get a utility equal to 0, conversely, if $\mathcal{A}$ attacks the i-th target while $\mathcal{D}$ decides to protect the j-th one, $\mathcal{A}$ gets v_i and $\mathcal{D}$ gets $-v_i$ since she

for each $n \in \{1, \dots, N\}$

1. $\mathcal{D}$ publicly commits to a strategy $\boldsymbol{\sigma}_{D,n}$
2. $\mathcal{A}$ observes the strategy $\mathcal{D}$ committed to
3. $\mathcal{D}$ and $\mathcal{A}$ play $i_{D,n}$ and $i_{A,n}$, respectively
4. $\mathcal{D}$ incurs in loss l_n according to 8.1

Figure 8.1: *Defender-Attacker interaction*

lost the target. More concisely, the Defender incurs in the *loss* (Line 4):

$$l_n := v_{i_{A,n}} \mathcal{I}\{i_{A,n} \neq i_{D,n}\}, \tag{8.1}$$

not suffering from any loss if both players select the same target.[1] Hereafter, we assume that the Defender is able to compute the best response strategy $\boldsymbol{\sigma}_D^*(A) \in \Delta_M$ if she is given the Attacker model she is playing against. Similarly, we denote with $\boldsymbol{\sigma}_A^*(\boldsymbol{\sigma}_D) \in \Delta_M$ the best response $\mathcal{A}$ plays against strategy $\boldsymbol{\sigma}_D$ of $\mathcal{D}$. According to such assumption, we can compute the expected loss of $\mathcal{D}$ against a generic Attacker $\mathcal{A}$ as:

$$L(A) := \sum_{m \in \mathcal{M}} \sigma_A(\boldsymbol{\sigma}_D^*(A))_m \, v_m \, (1 - \sigma_D^*(A)_m), \tag{8.2}$$

where $\sigma_\cdot(\cdot)_m$ is the probability associated with target m by the strategy.

The problem we study in this work is defined as follows.

Definition 8.1 (Attacker's Behavior Identification in Security Games)**.** *The Attacker's Behavior Identification in Security Games (ABI-SG) problem is a tuple* $(\mathcal{G}_N, \mathcal{A}, A_{k^*})$*, where* $\mathcal{G}_N$ *is a 2-player normal-form repeated game and* $\mathcal{A} = \{A_1, \dots, A_K\}$ *is a set of possible Attacker behavioral profiles, with* $A_{k^*} \in \mathcal{A}$ *denoting the actual profile of the Attacker in* $\mathcal{G}_N$*, unknown to the Defender* $\mathcal{D}$ *.*

In this work, we cast the ABI-SG as a sequential decision learning problem, where, at each round n, the Defender aims at selecting her best response to the Attacker in order to identify the actual Attacker profile $A_{k^*} \in \mathcal{A}$ while minimizing the loss suffered during the learning process.

Definition 8.2 (Policy)**.** *A* policy $\mathfrak{U}$ *is an algorithm able to provide at each round* n *a strategy profile* $\boldsymbol{\sigma}_{D,n}$ *for the Defender* $\mathcal{D}$ *. Formally:*

$$\mathfrak{U}(h_n) := \boldsymbol{\sigma}_{D,n},$$

[1] Hereafter, we denote with $\mathcal{I}\{E\}$ the indicator function of a generic event E.

where h_n is the history collected so far, i.e., all the strategies declared by the Defender $\{\boldsymbol{\sigma}_{D,1}, \dots, \boldsymbol{\sigma}_{D,n-1}\}$, the actions played by the two players $\{i_{D,1}, i_{A,1}, \dots, i_{D,n-1}, i_{A,n-1}\}$ in the past rounds and the corresponding losses $\{l_1, \dots, l_{n-1}\}$.

We evaluate the performance of a given policy $\mathfrak{U}$ over a finite-time horizon of N rounds by means of the expected cumulative *pseudo-regret*, defined as:

$$R_N(\mathfrak{U}) = \mathbb{E}\left[\sum_{n=1}^{N} l_n\right] - L^* N,$$

where $L^* := L(A_{k^*})$ is the expected loss incurred by the Defender if she plays the best response to the actual Attacker A_{k^*}, l_n is the loss incurred by using the policy $\mathfrak{U}$ at round n and the expectation $\mathbb{E}[\cdot]$ is taken w.r.t. the stochasticity of the Attacker strategy, the Defender policy and the policy $\mathfrak{U}$. The goal of a generic policy $\mathfrak{U}$ is to minimize the pseudo-regret $R_N(\mathfrak{U})$ incurred while learning the true Attacker's profile.

8.3 Analyzed Attacker Profiles

In this section, we describe the different Attacker profiles we study in this work and formalize the definition of the Attacker strategy $\boldsymbol{\sigma}_{A_{k^*}}(\cdot)$ for two sets of Attackers, grouped depending on their ability to change their behavior w.r.t. the strategy $\mathcal{D}$ commits to. Specifically, on one side, we take into account stochastic Attackers, which disregard the strategy of $\mathcal{D}$, on the other, we focus on strategy-aware Attackers, able to modify their strategies depending on the Defender announced strategy $\boldsymbol{\sigma}_{D,n}$.

8.3.1 Stochastic Attacker

The first class of Attackers is the *Stochastic* (Sto) one, where the attacking player does not take into account the strategy $\boldsymbol{\sigma}_{D,n}$ announced by the Defender $\mathcal{D}$ and thus has a fixed probability over time to attack the available targets. This class of Attackers models opponents focused on specific targets and whose preferences are not influenced by the Defender behavior. At round n, a stochastic Attacker Sto plays according to the strategy:

$$\boldsymbol{\sigma}_{Sto}(\boldsymbol{\sigma}) = \mathbf{p}(Sto) \quad \forall \boldsymbol{\sigma} \in \Delta_M,$$

where $\mathbf{p}(Sto) \in \Delta_M$ is a probability distribution over the targets, which is known to $\mathcal{D}$. In this case, the Defender best response $\boldsymbol{\sigma}^*_D(\boldsymbol{\sigma}_{Sto})$ is defined

as:

$$\sigma_D^*(Sto)_m = \begin{cases} 1 & \text{if } m = \arg\max_{i \in \mathcal{M}} \{v_i\, p(Sto)_i\} \\ 0 & \text{otherwise} \end{cases}.$$

8.3.2 Strategy Aware Attacker

The second class of Attackers we examine consists of strategy aware Attackers, corresponding to agents able to modify their strategy depending on the strategy of the Defender $\mathcal{D}$. In particular, we study *Stackelberg* (Sta) Attackers [119], who are able to exploit the information provided by strategy profile declared by the Defender $\mathcal{D}$ and optimally respond to it, and SUQR Attackers [89], having bounded rationality and being capable to partially exploit the information provided by the Defender, disregarding heavily patrolled targets.

Stackelberg Attacker Given a strategy profile declaration $\boldsymbol{\sigma}_{D,n}$, a Stackelberg Attacker Sta responds with:

$$\boldsymbol{\sigma}_{Sta}(\boldsymbol{\sigma}) = \arg \max_{\boldsymbol{\sigma}' \in \Delta_M} \sum_{m \in \mathcal{M}} \sigma'_m\, v_m\, (1 - \sigma_m)$$

and the Defender best-responds to this Attacker is:

$$\boldsymbol{\sigma}_D^*(Sta) = \arg \min_{\boldsymbol{\sigma}' \in \Delta_M} \max_{\boldsymbol{\sigma} \in \Delta_M} \sum_{m \in \mathcal{M}} \sigma'_m\, v_m\, (1 - \sigma_m),$$

as reported in [39], where it is proved that, for 2-player zero-sum games, the optimal mixed strategy for the Defender to commit to is equivalent to computing the minmax strategy, i.e., to minimize the maximum expected utility that the opponent can obtain.

SUQR Attacker The SUQR Attacker responds to the commitment $\boldsymbol{\sigma}_{D,n}$ as:

$$\sigma_{SUQR}(\boldsymbol{\sigma})_m = \frac{\exp\{-\alpha\sigma_m + \beta v_m + \gamma\}}{\sum_{h=1}^{M} \exp\{-\alpha\sigma_h + \beta v_h + \gamma\}},$$

where $\alpha \in \mathbb{R}^+$, $\beta, \gamma \in \mathbb{R}$ are parameters known to the Defender, characterizing the Attacker and depending on the underlying application. In this case, we do not have a closed form for the best response, but we can compute the minmax solution to the problem following the steps taken in [128]. We will refer to $\boldsymbol{\sigma}_D^*(SUQR)$ as the best response to an Attacker with a SUQR profile.

8.4 Identifying the Attacker

Initially, we describe how the state-of-the-art techniques can be adapted to address the ABI-SG problem. Direct approaches are provided by MAB [35] and expert [37] algorithms, where arms/experts represent the different Attacker behavioral profiles in $\mathcal{A}$. These are general-purpose techniques not exploiting the structure of the problem we are tackling. Summarily, MAB algorithms do not use the expert feedback to learn the Attacker behavior, while expert algorithms do not differentiate among feedbacks received after the Defender committed to different strategies. We show below the regret obtained when these algorithms are used in a ABI-SG problem.

When using MAB algorithms, we are able to directly apply the derivation of an upper bound over the pseudo-regret available in the literature to our problem. We can state the following result for the case of UCB1 [11].

Theorem 8.1 (UCB1 Pseudo-regret upper bound)**.** *Let us consider an instance of the ABI-SG problem and apply the UCB1 algorithm, where each possible behavioral profile $A_k \in \mathcal{A}$ is an arm which receives reward $-l_n$ if played. Then, we incur in the following pseudo-regret:*

$$R_N(\mathfrak{U}) \leq 8 \sum_{k \neq k^*} \frac{\ln N}{(\Delta L_k)} + \left(1 + \frac{\pi^2}{3}\right) \sum_{k \neq k^*} \Delta L_k,$$

where $\Delta L_k = \sum_{m=1}^{M} \sigma_{A_{k^*}}(\boldsymbol{\sigma}_D^*(A_k))_m \, v_m \, (1 - \sigma_D^*(A_k)_m) - L^*$ is the expected regret of playing the best response to Attacker A_k when the real Attacker is A_{k^*}.

When using an expert algorithm, for instance FPL [37], we could exploit an (expert) feedback over all arms since we can compute the expected loss also for the Attacker profiles that have not been played at turn n. Nevertheless, if the Attacker is strategy aware and we adopt an expert feedback, $\mathcal{D}$ incurs in a linear regret. We formally state this result in the following theorem.

Theorem 8.2. *Let us consider an instance of the ABI-SG problem and apply the* FPL *algorithm, where each possible profile A_k is an expert and receives, at round n, an expert reward equal to minus the loss she would have incurred observing $i_{A_{k^*},n}$ by playing the best response to the Attacker A_k. Then, there always exists an Attacker set $\mathcal{A}$ s.t. the Defender $\mathcal{D}$ incurs in an expected pseudo-regret of:*

$$R_N(\mathfrak{U}) \propto \Delta L_k \, N.$$

Proof. Let us analyze the ABI-SG problem in which the Attacker profile set is $\mathcal{A} = \{Sta, Sto\}$, the true Attacker $A_{k^*} = Sta$ and we use the Follow the Defender algorithm [37]. Assume that the best response $\boldsymbol{\sigma}^*_D(Sto)$ to the stochastic Attacker Sto corresponds to the pure strategy played by the Stackelberg Attacker at the equilibrium, i.e, $\boldsymbol{\sigma}^*_{Sta}(\boldsymbol{\sigma}^*_D(Sta)) = \boldsymbol{\sigma}^*_D(Sto)$. Assume the chosen target by the two strategies has value $v_{\hat{m}}$ in target $\hat{m}$, maximum value $v_{\bar{m}}$ in target $\bar{m}$ and that the stochastic Attacker has strategy $\mathbf{p}$ s.t.:

$$p_m = \begin{cases} \alpha & \text{if } m = \hat{m} \\ 1-\alpha & \text{if } m = \bar{m} \\ 0 & \text{otherwise} \end{cases},$$

where $\alpha = \frac{v_{\bar{m}} - L(Sta)}{v_{\bar{m}}}$ and $\alpha v_{\bar{m}} > (1-\alpha) v_{\bar{m}}$. In this case, the Defender might commit to two different strategies:

- if the Defender $\mathcal{D}$ declares its best response to the Stackerlberg Attacker $\boldsymbol{\sigma}^*_D(Sta)$ for the turn, it would provide zero loss as feedback for the stochastic Attacker expert and loss equal to $-L(Sta)$ to the Stackelberg one;
- if the Defender $\mathcal{D}$ selects the best response to the stochastic Attacker $\boldsymbol{\sigma}^*_D(Sto)$, she would gain loss equal to $-(1-\alpha)v_{\bar{m}} = -L(Sta)$ for the stochastic Attacker expert and $-L(Sta)$ for the Stackelberg one. Thus, in this case the two types would receive the same feedback.

Summarizing, we have that the Stackelberg Attacker expert always incurs in a loss greater or equal to the one of the stochastic one, even if the real Attacker is Stackelberg. Thus, with a probability grater than 0.5 we are incurring in a loss of ΔL_k for the entire horizon, with a total regret proportional to $\Delta L_k N$. Even by resorting to randomization, thus even adopting the `FPL` we would have a probability of at least $0.5 - \varepsilon$ (being ε the probability with which the `FPL` chooses a suboptimal option) to select the wrong option, thus also the `FPL` algorithm would incur in a linear regret over the time horizon. □

The above results show that MAB techniques provide, in the general case, better guarantees than expert algorithms, assuring a worst-case pseudo-regret of $O(\ln N)$ vs. $O(N)$.

In the following, we propose two different techniques that effectively exploit the information both on stochastic and strategy aware Attackers, providing better guarantees over the worst-case pseudo-regret. The first algorithm, `Follow the Belief` (`FB`), conducts the learning process taking

into account the belief of the learner about the different behavioral profiles. The second method, `Follow the Regret` (`FR`), is based on a value iteration algorithm over the belief space that minimizes the expected regret over the next rounds.

8.4.1 Follow the Belief

The pseudo-code of `FB` is presented in 15.

Algorithm 15 `FollowTheBelief` `FB`

1: $\mathcal{P} = \mathcal{A}$
2: **for all** $A' \in \mathcal{P}$ **do**
3: $\quad b_1(A') = \frac{1}{K}$
4: **for all** $n \in \{1, \dots, N\}$ **do**
5: $\quad$ Select $A_{k_n} = \arg\max_{A' \in \mathcal{P}} b_n(A')$
6: $\quad$ Play $\boldsymbol{\sigma}^*_D(A_{k_n})$
7: $\quad$ Observe Attacker action $i_{A_{k^*},n}$
8: $\quad$ **for all** $A' \in \mathcal{P}$ **do**
9: $\quad\quad$ **if** $\sigma_{A'}(\boldsymbol{\sigma}^*_D(A_{k_n}))_{i_{A_{k^*},n}} = 0$ **then**
10: $\quad\quad\quad \mathcal{P} \leftarrow \mathcal{P} \setminus A'$
11: $\quad\quad$ **else**
12: $\quad\quad\quad$ Compute $b_{n+1}(A')$ with 8.3

At the beginning, `FollowTheBelief` initializes a set of active Attackers $\mathcal{P} = \mathcal{A}$ and a belief $b_1(A_k) = 1/K$ for all the Attacker profiles $A_k \in \mathcal{P}$ (Lines 1–3). At each round n, the algorithm selects the Attacker A_{k_n} for which the belief is the largest one (where ties are broken arbitrarily), best responds with the strategy $\boldsymbol{\sigma}^*_D(A_{k_n})$ and observes the action actually played by the Attacker $i_{A_{k^*},n}$ (Lines 4-7). After that, the belief is updated as follows:

$$b_{n+1}(A') = \frac{w_n(A')}{\sum_{A \in \mathcal{P}} w_n(A)}, \tag{8.3}$$

where $w_n(A) = b_n(A')\ \sigma_{A_{k^*}}(\boldsymbol{\sigma}^*_D(A_{k_n}))_{i_{A_{k^*},n}}$ (Lines 8-12). In other words, the algorithm updates the likelihood of the sequence of the actions for each profile in $A' \in \mathcal{P}$ according to the observed action $i_{A_{k^*},n}$ at round n (Line 12). If the realization $i_{A_{k^*},n}$ is not consistent for Attacker A' (zero likelihood), profile A' is removed from $\mathcal{P}$ (Line 10).

Let $b_{kj,t} := \mathbb{E}_{\sigma^*_D(A_j)}[B_{k,t}]$, be the expected value of the belief we get for Attacker A_k when we are best responding to A_j and the true type is $A_{k^*} \neq A_k$ and denote with $\Delta b_k := \min_{j|A_j \in \mathcal{A}} \ln(b_{k^*j,t}) - \ln(b_{kj,t})$ the minimum difference of such values. We can upper bound the regret of `FB` algorithm as stated by the following theorem (the proof is reported in Appendix A).

Theorem 8.3. *Given an instance of the ABI-SG problem s.t.* $\Delta b_k > 0$ *for each* $A_k \in \mathcal{A}$ *and applying* FB*, the Defender incurs in a pseudo-regret of:*

$$R_N(\mathfrak{U}) \leq \sum_{k=1}^{K} \frac{2(\lambda_k^2 + \lambda_{k^*}^2)\Delta L_k}{(\Delta b_k)^2},$$

where

$$\lambda_k := \max_{m \in \mathcal{M}} \max_{\boldsymbol{\sigma} \in \mathcal{S}} \ln(\sigma_{A_k}(\boldsymbol{\sigma})_m) - \min_{m \in \mathcal{M}} \min_{\boldsymbol{\sigma} \in \mathcal{S}} \ln(\sigma_{A_k}(\boldsymbol{\sigma})_m) \mathcal{I}\{\sigma_{A_k}(\boldsymbol{\sigma})_m \neq 0\}$$

is the range where the logarithm of the beliefs realizations lies (excluding realizations equal to zero, which end the exploration of a profile) and $\mathcal{S} := \cup_k \boldsymbol{\sigma}_D^*(A_k)$ *is the set of the available best response to the Attackers profile.*

Comparing the derived results, we notice that the FB algorithm presents an upper bound over the pseudo-regret that is strictly better than that of MAB algorithms, i.e., a constant regret $O(1)$ in N vs. a logarithmic one $O(\ln N)$.

8.4.2 Follow the Regret

FB adopts the belief as discriminant factor to select the strategy profile to play in the next round. Conversely, in what follows, we describe the FR algorithm which is driven by a value iteration procedure that directly minimizes the expected regret over the remaining rounds $\{n+1, \ldots, N\}$. In principle, one should perform the procedure until the last round N, but, for computational purposes, an approximate solution can be obtained by setting a maximum level of recursion $h_{\max}$ and carry on the optimization only on the rounds $\{n+1, \ldots, \min\{n + h_{\max}, N\}\}$.

Algorithm 16 FollowTheRegret FR($h_{\max}$)

for all $A_k \in \mathcal{A}$ **do**
  Initialize $b_k^{(1)} = \frac{1}{K}$
for all $n \in \{1, \ldots, N\}$ **do**
  $\hat{\mathbf{R}}$ = RE$(1, \mathbf{b}^{(n)}, h_{\max})$
  Select A_{k_n} s.t. $k_n = \arg\min_t \hat{R}_t$
  Play $\boldsymbol{\sigma}_D^*(A_{k_n})$
  Observe Attacker action $i_{A_{k^*}, n}$
  for all $A_k \in \mathcal{A}$ **do**
    Compute $b_k^{(n+1)}$ according to 8.6

The pseudo-code of the FR algorithm is presented in 16, which recursively exploits the subroutine 17. At first, the FR algorithm requires to

Algorithm 17 `Regret Estimator` RE$(h, \mathbf{b}, h_{\max})$

1: **for all** $A_k \in \mathcal{A}$ **do**
2: **for all** $(i, j) \in \mathcal{M}^2$ **do**
3: **for all** $A_t \in \mathcal{A}$ **do**
4: $\hat{b}_t \leftarrow b_t \, \sigma_{A_t}(\boldsymbol{\sigma}^*_D(A_k))_j$
5: $\hat{\mathbf{b}} \leftarrow \frac{\hat{\mathbf{b}}}{\sum_m \hat{b}_m}$
6: Compute $r_{ij,k}$ according to 8.4
7: **if** $h < h_{\max}$ **then**
8: $\tilde{\mathbf{R}}$ = RE$(h+1, \hat{\mathbf{b}}, h_{\max})$
9: $r_{ij,k} \leftarrow r_{ij,k} + \min_k \tilde{R}_k$
10: Compute $\hat{R}_k$ according to 8.5
11: Return $\hat{\mathbf{R}}$

initialize a belief vector $b_k^{(1)} = \frac{1}{K}$ for each Attacker $A_k \in \mathcal{A}$ (Line 2, Alg. 16). At each round n, the algorithm computes the estimated expected regret vector $\hat{\mathbf{R}}$ suffered by $\mathcal{D}$ if she plays the best response $\boldsymbol{\sigma}^*_D(A_k)$ to A_k for each Attacker profile $A_k \in \mathcal{A}$ (Line 4, Alg. 16), by recursively calling the `Regret Estimator` (RE) algorithm. Here, for every possible Attacker $A_k \in \mathcal{A}$ and for every pair of possible actions of the Defender and the Attacker $(i, j) \in \mathcal{M}^2$, we create a new belief vector $\hat{\mathbf{b}}$ by updating $\mathbf{b}$ according to the information the Attacker played action j (Line 4, Alg. 17). After that, we compute $r_{ij,k}$, i.e., the estimated expected loss in the case the Defender $\mathcal{D}$ plays action $i_{D,n} = i$ and the Attacker A_k plays $i_{A_k,n} = j$ averaged over the beliefs $b_n(A)$, as follows:

$$r_{ij,k} = v_j \mathcal{I}\{i \neq j\} - \sum_{t \in \{1,\dots,K\}} \hat{b}_t \, L(A_t). \tag{8.4}$$

If the maximum recursion level $h_{\max}$ has been reached, the above value corresponds to the total estimated expected regret, otherwise we recursively compute the regret by calling RE over the following rounds and sum it to the instantaneous one $r_{ij,k}$ (Line 9, Alg. 17). Finally we compute the estimated total regret of choosing a specific Attacker A_k for the next turn (Line 10, Alg. 17) as follows:

$$\begin{aligned} \hat{R}_k := & \sum_{i=1}^{M} \sum_{j=1}^{M} r_{ij,k} \, \sigma^*_D(A_k)_i \\ & \cdot \sum_{A_{k'} \in \mathcal{A}} b_{k'} \, \sigma_{A_{k'}}(\boldsymbol{\sigma}^*_D(A_k))_j, \end{aligned} \tag{8.5}$$

where the regret $r_{ij,k}$ is weighted with the probabilities that action i is se-

lected by $\mathcal{D}$ and action j is selected by $\mathcal{A}$. The Defender $\mathcal{D}$ plays, for the current round n, the best response to the Attacker A_{k_n}, providing the minimum estimated expected regret $\hat{R}_{k_n}$ (Line 6, Alg. 16) and observing action $i_{A_{k^*},n}$ undertaken by the Attacker A_{k^*}. Finally, the algorithm updates the beliefs (Line 9, Alg. 16) as follows:

$$b_k^{(n+1)} = \frac{w_{nk}}{\sum_{k' \in \{1,\dots,K\}} w_{nk'}}, \tag{8.6}$$

where $w_{nk} = b_k^{(n)} \, \sigma_{A_k}(\boldsymbol{\sigma}_D^*(A_{k_n}))_{i_{A_{k^*},n}}$.

8.4.3 Computational Complexity

In this section, we analyze the proposed algorithms from a computational perspective. FB has complexity $O(KN)$, since it performs a belief update for each of the K Attacker profiles, repeating this operation over N rounds. Thus, it results being linear both in the number of profiles and the rounds the game is played. Conversely, FR requires much more computational time. Indeed, for each Attacker profile K, we consider M actions for both players and update the expected regret over the K profiles current beliefs. This leads to a cost of $O(M^2K^2)$ for a single round and an overall computational cost of $O(M^2K^2N)$ over the problem horizon N. If we want to employ the strategy from the current round n to the end of the horizon to compute the estimated expected regret $\hat{R}_n(A_k)$ by means of a forward procedure, the computational cost required by FR is $O(M^{2(N-n)}K^{2(N-n)})$ for a round. Thus, the final computational cost required by FR is $\sum_{n=1}^{N} O(M^{2(N-n)}K^{2(N-n)}) = O\left(\frac{(MK)^{2N}-1}{(MK)^2-1}\right) \approx O(M^{2N}K^{2N})$.

8.5 Experimental Evaluation

We compare the proposed algorithms FB and FR (with $h_{\max} = 1$) with the state-of-the-art online learning approaches from the MAB [35] and expert [68] fields. In particular, We evaluate UCB1 algorithm [11], from the MAB literature, and the FPL algorithm [37], from the expert literature.

In the experiments we also analyze the case in which one of the Attacker behavioral profiles, namely U, is stochastic and her strategy is unknown to the Defender $\mathcal{D}$ (to avoid possible misunderstandings, let us notice that the stochastic behavior we describe in Section 8.3 is based on the assumption that the Defender knows the strategy). In this case, we are still able to allow the Defender to commit to a strategy that somehow minimizes the expected

loss. Indeed, we can assign:

$$\boldsymbol{\sigma}^*_{D,n}(U) = FPL(h_n),$$

where $FPL(\cdot) \in \Delta_M$ is the pure strategy prescribed by the FPL algorithm. In this case the algorithm suffers from an additional regret due to the fact that, even if it is able to correctly detect the profile, it does not know the best response $\boldsymbol{\sigma}^*_D(U)$, but it needs to learn it over time.

	Sta	Sto	$SUQR$	U	K
C_1	1	1	-	-	2
C_2	1	-	1	-	2
C_3	1	1	1	-	3
C_4	1	5	-	-	6
C_5	1	-	5	-	6
C_6	1	5	5	-	11
C_7	1	5	5	1	12

Table 8.1: *Number and type of Attacker profiles $\mathcal{A}$ used for the experiments and total number of Attacker K.*

		C_1	C_2	C_3	C_4	C_5	C_6	C_7
$M = 5$	UCB1	14.1 ± 1.9	8.6 ± 3.7	23.9 ± 5.2	45.8 ± 11.7	1.8 ± 0.4	75.8 ± 19.9	62.3 ± 12.2
	FPL	18.7 ± 35.0	11.2 ± 6.0	38.5 ± 27.2	49.8 ± 62.3	0.8 ± 0.1	68.9 ± 64.1	72.5 ± 53.3
	FB	$\mathbf{0.2 \pm 0.1}$	$\mathbf{0.2 \pm 0.2}$	$\mathbf{0.5 \pm 0.2}$	$\mathbf{0.5 \pm 0.2}$	$\mathbf{0.1 \pm 0.0}$	$\mathbf{0.7 \pm 0.2}$	$\mathbf{8.0 \pm 4.9}$
	FR	$\mathbf{0.1 \pm 0.1}$	$\mathbf{0.3 \pm 0.4}$	$\mathbf{0.4 \pm 0.3}$	$\mathbf{0.6 \pm 0.2}$	$\mathbf{0.1 \pm 0.0}$	$\mathbf{1.1 \pm 1.1}$	$\mathbf{4.8 \pm 3.3}$
$M = 10$	UCB1	16.77 ± 1.2	5.2 ± 2.8	21.2 ± 3.8	60.6 ± 8.9	4.2 ± 5.0	61.5 ± 22.5	58.9 ± 17.4
	FPL	1.1 ± 0.2	6.0 ± 3.5	12.1 ± 4.3	2.6 ± 1.0	3.2 ± 4.0	17.7 ± 16.0	$\mathbf{22.5 \pm 12.3}$
	FB	$\mathbf{0.1 \pm 0.0}$	$\mathbf{0.1 \pm 0.02}$	$\mathbf{0.3 \pm 0.2}$	$\mathbf{0.6 \pm 0.2}$	$\mathbf{0.1 \pm 0.0}$	$\mathbf{0.6 \pm 0.1}$	$\mathbf{16.1 \pm 6.9}$
	FR	$\mathbf{0.1 \pm 0.1}$	$\mathbf{0.1 \pm 0.21}$	$\mathbf{0.2 \pm 0.1}$	$\mathbf{0.4 \pm 0.2}$	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.6 \pm 0.4}$	$\mathbf{14.7 \pm 8.1}$

Table 8.2: *Expected pseudo-regret $R_N(\mathfrak{U})$ over $N = 1000$ rounds and corresponding 95% confidence intervals for different configurations (best results are in boldface).*

8.5.1 Experimental Setting

The experimental setting is as follows. We use a time horizon of $N = 1000$ rounds, with a different amount of targets $M \in \{5, 10\}$ and different profile configurations C_i, listed in Section 8.5, in which we report also the number of different stochastic, SUQR, and unknown stochastic behavioral profiles for each configuration. The configurations are ordered from the ones with smallest number of behavioral profiles ($K = 2$) to the largest one ($K = 12$). In principle, these problems should be of increasing difficulty, since the algorithms have to identify the actual behavior among a larger number of options.

The strategies of the stochastic behavioral profiles Sto are drawn from a Dirichlet distribution with $\boldsymbol{\theta} = \mathbf{1}_M$ (uniform distribution over Δ_M) and the target values $\mathbf{v}$ are uniformly sampled in $[0, 1]^M$. The parameters for the SUQR behavioral profiles are drawn from a uniform probability distribution over the intervals $\alpha \in [5, 15]$, $\beta \in [0, 1]$ and $\gamma \in [0, 1]$, whose choice is motivated by the experimental results obtained by [89]. For each combination of behavioral profiles and targets size, 10 random configurations (i.e., target values $\mathbf{v}$ and Attacker profile sets $\mathcal{A}$) are generated and the actual behavioral profile A_{k^*} is drawn from a uniform probability distribution over the given profiles set $\mathcal{A}$. For each configuration we run 100 independent experiments and we compute the average regret. We evaluate the performance in terms of expected pseudo-regret $R(\mathfrak{U})_n$ with $n \in \{1, \dots, N\}$ and computational time spent by the algorithms to execute a single run ($N = 1000$ rounds). Each component of the noise vector z in FPL is drawn from a uniform probability distribution over the interval $[0, \hat{v}K\sqrt{N}]$, where $\hat{v} = \max_{m \in M} v_m$, as described in [37], Chapter 4.

8.5.2 Experimental Results

We report in Section 8.5 the empiric pseudo-regret obtained in the experimental results. It can be observed that the algorithms we propose dramatically outperform the baselines provided by the state of the art. Furthermore, there is no strong statistical evidence that one algorithm between `FB` or `FR` outperforms the other. We recall that `FR` is more computationally demanding than `FB`, thus one might prefer `FB` for problems with many Attacker behavioral profiles, since it has comparable performance w.r.t. `FR` and is computationally more efficient. Notably, the FPL algorithm generally improves its performance when tested over larger target space $M = 10$. We think this could be induced by the fact that the specific configurations in which the FPL gets linear regret (i.e., the ones considered in Theorem 8.2) are less likely to occur when we have a larger amount of targets. Remarkably, our algorithms provide good performance also when a stochastic behavioral profile U whose strategy is unknown to the Defender is present among the possible ones.

In Figures 8.2 to 8.7 we show how the pseudo-regret $R_n(\mathfrak{U})$ evolves during the time horizon in the most challenging configurations, namely C_5, C_6 and C_7. The results in other configurations confirm the results obtained in C_5, C_6, C_7. The plots are in a semilogarithmic scale for a better comprehension. In all the presented configurations, except in C_7 with $M = 10$, there is statistical significance that the `FB` and `FR` algorithms outperform the baselines on average since the confidence intervals do not overlap after the first

≈ 50 rounds. In configuration C_7 with $M = 10$, our algorithms outperform the baselines only on average.

Finally, we analyze the computational effort required by our algorithms to solve instances over $N = 1000$ rounds and $M \in \{5, 10, 20, 40\}$ targets.[2] The average computational times are reported in Section 8.5.2. There are three observations we can make. First, we could not report the values for $M \in \{20, 40\}$ for FR since the required computational cost is too high (≥ 3600 seconds). Second, both FB and FR present the same trend w.r.t. the configurations: in fact, when the behavioral profile of the opponent can only be either Sta or Sto, both algorithms are twice more efficient than in cases in which SUQR adversaries are introduced. This is due to the fact that both Sta and SUQR models exploit the strategy the Defender commits to, making more difficult to distinguish among them. The most difficult configuration is C_7, where the presence of a stochastic unknown adversary make things even worse since the distribution must also be estimated. Finally, as expected, we notice that FB is *always* faster than FR: in fact, while they are both polynomial in the actions available to the players, i.e., the number of targets, the former is linear while the latter quadratic (since we set $h_{\max} = 1$).

		C_1	C_2	C_3	C_4	C_5	C_6	C_7
$M = 5$	FB	6	11	12	4	24	15	15
	FR	77	121	170	146	652	1029	1114
$M = 10$	FB	10	22	23	7	63	47	48
	FR	356	679	887	960	4402	7527	7292
$M = 20$	FB	33	222	138	34	485	227	229
	FR	–	–	–	–	–	–	–
$M = 40$	FB	105	2061	1412	129	2348	1634	1643
	FR	–	–	–	–	–	–	–

Table 8.3: *Computational time in seconds needed by FB and FR to solve an instance over $N = 1000$ rounds.*

[2]The computational times for the UCB1 and FPL algorithm are omitted since they are in line with the one of FB.

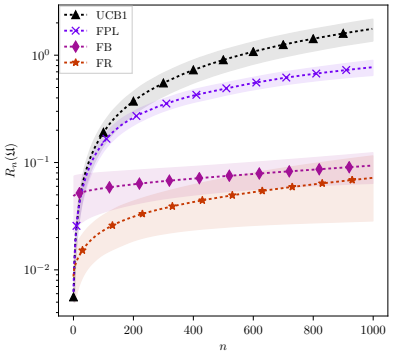


Figure 8.2: *Expected pseudo-regret for the configuration C_5 with $M = 5$ targets.*

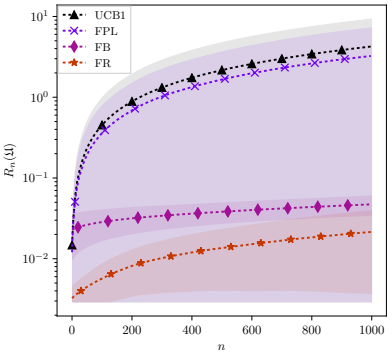


Figure 8.3: *Expected pseudo-regret for the configuration C_5 with $M = 10$ targets.*

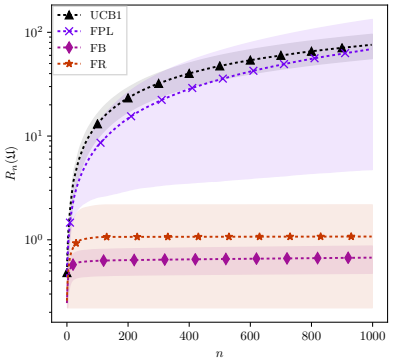


Figure 8.4: *Expected pseudo-regret for the configuration C_6 with $M = 5$ targets.*

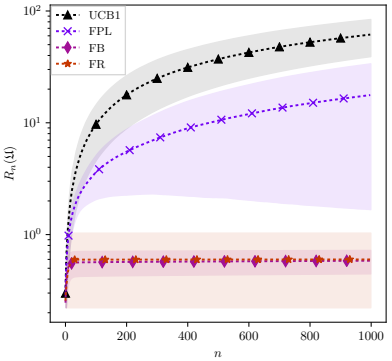


Figure 8.5: *Expected pseudo-regret for the configuration C_6 with $M = 10$ targets.*

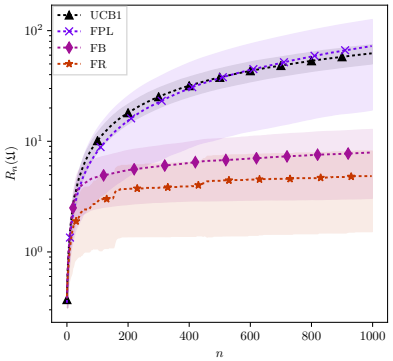


Figure 8.6: *Expected pseudo-regret for the configuration C_7 with $M = 5$ targets.*

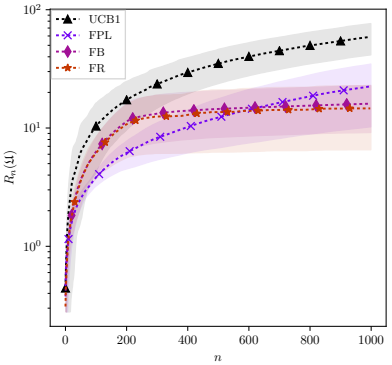


Figure 8.7: *Expected pseudo-regret for the configuration C_7 with $M = 10$ targets.*

Figure 8.8: *Expected pseudo-regret for different configurations of Attacker's profiles.*

CHAPTER 9

Conclusions and Future Research

"The stars are thin," said Grey Brother, snuffing at the dawn wind. "Where shall we lair today? For, from now, we follow new trails".

— Rudyard Kipling

Physical security is one of the most important challenges of our times. Due to the terrible events happened in the last decades, e.g. on September 11, 2001, or nowadays in several European countries, new techniques and methods are being developed to face new threats and dangers. On the other side, security means also saving lives and helping people, e.g. detecting desperate migrants that are moving across the Mediterranean Sea and rescuing them.

Algorithmic Game Theory is a fundamental tool in these settings, giving us the possibility to scientifically investigate them, modeling the interaction between law enforcers and terrorists or criminals as a mathematical problem, and designing suitable algorithms to deal with these threats.

9.1 Tackled Problems

Often, in large environments or infrastructures, a constant surveillance of every area is not affordable: to cope with this issue, a common countermeasure is the usage of cheap but wide-ranged sensors, able to detect suspicious events that occur in large areas, supporting patrollers to improve the effectiveness of their strategies. In this thesis we studied how an alarm system can effectively support patrolling strategies of mobile defensive resources. Specifically, first we asked which is the best behavior a guard can adopt if she is supported by an alarm system affected by localization uncertainty, i.e., it is able to detect the area in which the attack is occurring but not the exact place. We made a further step towards reality asking how to face this patrolling problem when the alarm system has a positive missed detection rate, namely no alarm signal is raised even though an attack is occurring. Thus, the question we posed is the following: *how would you exploit the information that something is happening in some areas, if you do not know the exact position of the attack or an alarm may not be triggered?* Then, we moved considering bigger scenarios, in which the Defender can control multiple resources or the Attacker can perform multiple attacks. The questions we wanted to answer are: *how would you coordinate multiple guards to catch the attacker? And how would you react if, once started moving towards the attack location, another alarm is triggered, telling you that multiple attacks are ongoing?* The last setting we considered is about a Defender facing an unknown opponent. The problem resulted being a learning problem, and so the question we posed is: *how would you learn the profile of an unknown Attacker to exploit it in the future?*

9.2 Achieved Results

To tackle these problems, we originally provided a security game with the presence of an alarm system able to trigger alarm signals, which carry the information about the set of targets that can be under attack and it is described by the probability of being generated when each target is attacked. We investigated two main research lines. On one side, we studied the uncertainties that may affect a real-world alarm system: spatial uncertainty, i.e., the alarm system is uncertain about the exact target under attack, and missed detections, i.e., no alarm is triggered even if an attack is occurring. On the other, we developed models and designed algorithms for scenarios in which the number of resources available to the players, either the Defender or the Attacker, increases, thus allowing a coordinated defense or multiple attacks.

Finally, we proposed two approaches to learn the profile of an unknown Attacker, so that we can exploit this information to prevent future attacks.

9.2.1 Uncertainties of the Alarm System

More specifically, at first we considered the scenario in which the Defender can control a single patroller and the alarm system is affected by spatial uncertainty. In other words, when an attack is occurring, a signal is sent to the Defender, saying that something bad is happening in an area but without specifying the exact location. We divided the problem in two phases: signal response and patrolling. For the signal response phase, we provided a complexity analysis, provided two exact algorithms and two approximation algorithms. We also showed that, without false positives and missed detections, the best patrolling strategy reduces to stay in a place, wait for a signal, and respond to it at best. Then, we introduced a significant positive missed detection rate, i.e., no alarm signal is generated even though an attack is occurring. This new scenario is in favor of the Attacker, who can exploit such flaw of the alarm system. This is why such new element puts the problem in a different perspective and requires a different approach to be solved. We deeply analyzed security games in which the alarm system is both characterized by detection uncertainty and spatial imperfection, tackling the challenge of designing tractable algorithms for real-life scenarios. In particular, we showed that standing still and waiting for a signal is no more the best response, while movements among the areas of the environment to protect should be considered. In fact, we proved that Markovian strategies are arbitrarily worse than optimal non-Markovian ones, and thus we resort to a deterministic approach.

9.2.2 A Coordinated Defense and Multiple Attacks

The other direction we investigated is about the dimension of the problem, i.e., the number of resources both the Attacker and the Defender can control. We studied how the Defender should behave if she can control multiple resources, taking into account also the level of coordination among such defensive resources, e.g., they cannot communicate because of a radio silence constraint due to the fact that they are operating on an undercover mission. The challenge is designing algorithms able to scale up with the number of resources. The problem of finding the best Defender's strategy when a number of resources are given is FNP-hard from the case with a single resource. Conversely, we would have expected that the problem of finding the minimum number of resources assuring non-null protection to every target to be

easier, since it does not require to compute the strategies. Nevertheless, we showed that this problem is log-APX-complete on arbitrary graphs, while it is in FP in tree and cycle graphs. Then, we studied the problem of finding the best strategy to respond to any alarm signal once an allocation of resources in the environment is given, according to different degrees of coordination among the resources, each described by an adversarial team game with different forms of strategies. On the other side, we investigated the opportunities the Attacker can take when she is allowed to perform multiple attacks. We showed that, when we restrict the Attacker to use all her resources simultaneously, there is no algorithm that requires polynomial time in the number of Attacker's resources unless P = NP. Conversely, when the number of resources is a fixed parameter, the problem admits a polynomial time algorithm. Furthermore, fixing the number of attacking resources, the unrestricted model in which the Attacker can perform different sequential attacks admits a non-trivial polynomial time algorithm based on dynamic programming, able to find the strategies on the equilibrium path (instead, off the equilibrium path, the game may have infinite horizon). The computation of the equilibrium strategies requires the common knowledge about the number of Attacker's resources. This information-as well as a Bayesian prior required in Bayesian games-is unlikely to be common. For this reason, we followed a different approach we believe to be more suitable in practice. We assumed that the Defender makes a guess about the number of resources and plays her best strategy accordingly, and we evaluate the worst-case inefficiency of this strategy when the Attacker has a different number of resources. We showed that the inefficiency can be arbitrary even when the guess is a wrong estimate for just a single resource. This suggests the use of online algorithms that do not need any guess. We provided a tight upper bound over the competitive factor when non-stochastic online algorithms are used and we showed that the factor can be improved by resorting to randomization.

9.2.3 Facing the Unknown

Finally, we tackled the problem of facing an unknown adversary, whose profile is just known to be in a list of possible profiles she can assume. The problem is completely different, requiring to identify the Attacker we are facing to exploit such information in the future, and be able to prevent her from performing other attacks. We defined a novel scenario in which the Defender plays against an Attacker whose behavior is unknown but it belongs to a set of known profiles. We showed that state-of-the-art bandit and

expert algorithms suffer from a linear and logarithmic regret, respectively, in the length of the time horizon. Thus, we introduced two novel approaches to deal with our problem, bridging together game-theoretical techniques and online learning tools. In the first approach, the Defender has a belief about the follower and updates it during the game and we provide a finite-time analysis showing that the regret of the algorithm is constant in the length of the time horizon. In the second approach, the learning policy is driven directly by the estimated expected regret and is based on a backward induction procedure. We provided a thorough experimental evaluation in concrete security settings, comparing our algorithms with the main algorithms available in the state of the art of the online learning field and showing that our approaches provide a remarkable improvement in terms of expected pseudo-regret minimization.

9.3 Future Research

The research developed in this thesis can be extended along different directions. First, we could enrich our model w.r.t. the uncertainties that characterize the alarm system, introducing false positives. Even though an Attacker with multiple resources could actually recreate a similar effect, performing an attack just to deceive the Defender and then attacking her main target, this is not the same as having the system affected by such an issue. In fact, the Attacker could exploit this flaw, while the Defender should decide whether it is convenient or not to move from her current position.

On the other side, it would be interesting to have a team of defending resources facing a team of attacking agents, where the two teams may have different degrees of coordination.

The final step would be the union of all the alarm uncertainties together with the employment of multiple resources on both defensive and attacking side, thus building a unique and coherent framework.

Then, it would be interesting to deploy our algorithms and techniques on the field, tackling real problems, inserting their peculiar features into our models and customize them according to the various scenarios they are embedded in.

List of Figures

List of Tables

Bibliography

[1] Micah Adler, Harald Räcke, Naveen Sivadasan, Christian Sohler, and Berthold Vöcking. Randomized Pursuit-Evasion in Graphs. *Combinatorics, Probability and Computing*, 12:225–244, 2003.

[2] Noa Agmon. On Events in Multi-robot Patrol in Adversarial Environments. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 591–598, 2010.

[3] Noa Agmon, Chien-Liang Fok, Yehuda Emaliah, Peter Stone, Christine Julien, and Sriram Vishwanath. On Coordination in Practical Multi-robot Patrol. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 650–656, 2012.

[4] Noa Agmon, Gal A Kaminka, and Sarit Kraus. Multi-robot Adversarial Patrolling: Facing a Full-knowledge Opponent. *Journal of Artificial Intelligence Research*, 42:887–916, 2011.

[5] Steve Alpern. Infiltration Games on Arbitrary Graphs. *Journal of Mathematical Analysis and Applications*, 163:286–288, 1992.

[6] Steve Alpern, Alec Morton, and Katerina Papadaki. Patrolling Games. *Operations Research*, 59(5):1246–1257, 2011.

[7] Francesco Amigoni, Nicola Basilico, and Nicola Gatti. Finding the Optimal Strategies for Robotic Patrolling with Adversaries in Topologically-represented Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 819–824, 2009.

[8] Francesco Amigoni, Nicola Basilico, Nicola Gatti, Alessandro Saporiti, and Stefano Troiani. Moving Game Theoretical Patrolling Strategies from Theory to Practice: An Usarsim Simulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 426–431, 2010.

[9] Bo An, Matthew Brown, Yevgeniy Vorobeychik, and Milind Tambe. Security Games with Surveillance Cost and Optimal Timing of Attack Execution. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 223–230, 2013.

[10] Bo An, Eric Shieh, Milind Tambe, Rong Yang, Craig Baldwin, Joseph DiRenzo, Ben Maule, and Garrett Meyer. PROTECT - A deployed game theoretic system for strategic security allocation for the United States Coast Guard. *AI Magazine*, 33(4):96, 2012.

[11] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multi-armed Bandit Problem. *Machine Learning*, 47(2-3):235–256, 2002.

[12] Robert J. Aumann. Subjectivity and Correlation in Randomized Strategies. *Journal of Mathematical Economics*, 1:67–96, 1974.

[13] Robert Axelrod. *The Evolution of Cooperation*, volume 1. Basic Books, 1984.

[14] Maria-Florina Balcan, Avrim Blum, Nika Haghtalab, and Ariel D. Procaccia. Commitment Without Regrets: Online Learning in Stackelberg Security Games. In *ACM Conference on Economics and Computation (EC)*, pages 61–78, 2015.

[15] Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Adam Meyerson. Approximation Algorithms for Deadline-TSP and Vehicle Routing with Time-windows. In *ACM Symposium on Theory of Computing (STOC)*, pages 166–174, 2004.

[16] Nicola Basilico, Andrea Celli, Giuseppe De Nittis, and Nicola Gatti. Coordinating Multiple Defensive Resources in Patrolling Games with Alarm Systems. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 678–686, 2017.

[17] Nicola Basilico, Andrea Celli, Giuseppe De Nittis, and Nicola Gatti. Team-Maxmin Equilibrium: Efficiency Bounds and Algorithms. In *Conference on Artificial Intelligence (AAAI)*, pages 356–362, 2017.

[18] Nicola Basilico, Timothy H Chung, and Stefano Carpin. Distributed Online Patrolling with Multi-agent Teams of Sentinels and Searchers. In *Distributed Autonomous Robotic Systems (DARS)*, pages 3–16. Springer, 2014.

[19] Nicola Basilico, Giuseppe De Nittis, and Nicola Gatti. A Security Game Combining Patrolling and Alarm-triggered Responses under Spatial and Detection Uncertainties. In *Conference on Artificial Intelligence (AAAI)*, pages 404–410, 2016.

[20] Nicola Basilico, Giuseppe De Nittis, and Nicola Gatti. Adversarial Patrolling with Spatially Uncertain Alarm Signals. *Artificial Intelligence*, 246:220–257, 2017.

[21] Nicola Basilico and Nicola Gatti. Automated Abstractions for Patrolling Security Games. In *Conference on Artificial Intelligence (AAAI)*, 2011.

[22] Nicola Basilico, Nicola Gatti, and Francesco Amigoni. Patrolling Security Games: Definition and Algorithms for Solving Large Instances with Single Patroller and Single Intruder. *Artificial Intelligence*, 184:78–123, 2012.

[23] Nicola Basilico, Nicola Gatti, and Thomas Rossi. Capturing Augmented Sensing Capabilities and Intrusion Delay in Patrolling-Intrusion Games. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 186–193, 2009.

[24] Nicola Basilico, Nicola Gatti, Thomas Rossi, Sofia Ceppi, and Francesco Amigoni. Extending Algorithms for Mobile Robot Patrolling in the Presence of Adversaries to More Realistic Settings. In *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI/IAT)*, volume 2, pages 557–564, 2009.

[25] Nicola Basilico, Nicola Gatti, and Federico Villa. Asynchronous Multi-robot Patrolling against Intrusions in Arbitrary Topologies. In *Conference on Artificial Intelligence (AAAI)*, 2010.

[26] Dimitri P. Bertsekas. *Nonlinear programming*. Athena Scientific Belmont, 1999.

[27] Garrett Birkhoff. Tres Observaciones sobre el Algebra Lineal. *Universidad Nacional de Tucumán. Facultad de Ciencias Exactas y Tec-*

nología. Revista. Serie A. Matemática y Física Teórica, 5:147–151, 1946.

[28] Lorenzo Bisi, Giuseppe De Nittis, Francesco Trovò, Marcello Restelli, and Nicola Gatti. Regret Minimization Algorithms for the Follower's Behaviour Identification in Leadership Games. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 282–291, 2017.

[29] Avrim Blum, Nika Haghtalab, and Ariel D. Procaccia. Lazy Defenders Are Almost Optimal against Diligent Attackers. In *Conference on Artificial Intelligence (AAAI)*, pages 573–579, 2014.

[30] Avrim Blum, Nika Haghtalab, and Ariel D. Procaccia. Learning to Play Stackelberg Security Games. Technical report, Carnegie Mellon University, Computer Science Department, 2015.

[31] Hans-Joachim Böckenhauer, Luca Forlizzi, Juraj Hromkovič, Joachim Kneis, Joachim Kupke, Guido Proietti, and Peter Widmayer. Reusing Optimal TSP Solutions for Locally Modified Input Instances. In *International Conference on Theoretical Computer Science*, pages 251–270, 2006.

[32] Hans-Joachim Böckenhauer, Juraj Hromkovic, Joachim Kneis, and Joachim Kupke. The Parameterized Approximability of TSP with Deadlines. *Theory Computing Systems*, 41(3):431–444, 2007.

[33] Christian Borgs, Jennifer Chayes, Nicole Immorlica, Adam Tauman Kalai, Vahab Mirrokni, and Christos Papadimitriou. The Myth of the Folk Theorem. *Games and Economic Behavior*, 70:34–43, 2010.

[34] Matthew Brown, Arunesh Sinha, Aaron Schlenker, and Milind Tambe. One Size Does Not Fit All: A Game-Theoretic Approach for Dynamically and Effectively Screening for Threats. In *Conference on Artificial Intelligence (AAAI)*, pages 425–431, 2016.

[35] Sébastien Bubeck, Nicolò Cesa-Bianchi, et al. Regret Analysis of Stochastic and Non-stochastic Multi-armed Bandit Problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

[36] Stefano Carpin, Mike Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. USARSim: A Robot Simulator for Research and Education. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1400–1405, 2007.

[37] Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge university press, 2006.

[38] Vasek Chvatal. A Greedy Heuristic for the Set-covering Problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[39] Vincent Conitzer and Tuomas Sandholm. Computing the Optimal Strategy to Commit to. In *ACM conference on Electronic Commerce (EC)*, pages 82–90, 2006.

[40] Vincent Conitzer and Tuomas Sandholm. AWESOME: A General Multiagent Learning Algorithm that Converges in Self-play and Learns a Best Response Against Stationary Opponents. *Machine Learning*, 67(1-2):23–43, 2007.

[41] Richard Walter Conway, William L Maxwell, and Louis W Miller. *Theory of scheduling*. Courier Corporation, 2003.

[42] William Cook, Thorsten Koch, Daniel Steffy, and Kati Wolter. An Exact Rational Mixed-integer Programming Solver. *Integer Programming and Combinatorial Optimization*, pages 104–116, 2011.

[43] Thomas H. Cormen. *Introduction to Algorithms*. MIT Press, 2009.

[44] Partha Dasgupta, Peter Hammond, and Eric S. Maskin. The Implementation of Social Choice Rules: Some Results on Incentive Compatibility. *Review of Economic Studies*, 46:185–216, 1979.

[45] Francesco Maria Delle Fave, Albert Xin Jiang, Zhengyu Yin, Chao Zhang, Milind Tambe, Sarit Kraus, and John P. Sullivan. Game-theoretic Patrolling with Dynamic Execution Uncertainty and a Case Study on a Real Transit System. *Journal of Artificial Intelligence Research*, 50:321–367, 2014.

[46] Karel Durkota, Viliam Lisỳ, Branislav Bošanskỳ, and Christopher Kiekintveld. Approximate Solutions for Attack Graph Games with Imperfect Information. In *International Conference on Decision and Game Theory for Security (GameSec)*, pages 228–249, 2015.

[47] Karel Durkota, Viliam Lisỳ, Branislav Bosanskỳ, and Christopher Kiekintveld. Optimal Network Security Hardening Using Attack Graph Games. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 526–532, 2015.

[48] Bruno Escoffier and Vangelis T. Paschos. Completeness in Approximation Classes beyond APX. *Theoretical Computer Science*, 359:369–377, 2006.

[49] Fei Fang, Peter Stone, and Milind Tambe. When Security Games Go Green: Designing Defender Strategies to Prevent Poaching and Illegal Fishing. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2589–2595, 2015.

[50] Amos Fiat. *Online Algorithms: The State of the Art*. Springer, 1998.

[51] Yuval Filmus and Justin Ward. The Power of Local Search: Maximum Coverage over a Matroid. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 14, pages 601–612, 2012.

[52] Merrill M. Flood. The Hide and Seek Game of Von Neumann. *Management Science*, 18(5-part-2):107–109, 1972.

[53] Benjamin Ford, Debarun Kar, Francesco M. Delle Fave, Rong Yang, and Milind Tambe. PAWS: Adaptive Game-theoretic Patrolling for Wildlife Protection. In *International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, pages 1641–1642, 2014.

[54] Benjamin Ford, Amulya Yadav, Amandeep Singh, Matthew Brown, Arunesh Sinha, Biplav Srivastava, Christopher Kiekintveld, Nicole Sintov, and Milind Tambe. NECTAR: Game-Theoretic Factory Inspection Scheduling and Explanation for Toxic Wastewater Abatement. In *International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, 2016.

[55] Shmuel Gal. *Search Games*. Academic Press, 1980.

[56] Jiarui Gan, Bo An, and Yevgeniy Vorobeychik. Security Games with Protection Externalities. In *Conference on Artificial Intelligence (AAAI)*, pages 914–920, 2015.

[57] Jiarui Gan, Bo An, Yevgeniy Vorobeychik, and Brian Gauch. Security Games on a Plane. In *Conference on Artificial Intelligence (AAAI)*, pages 530–536, 2017.

[58] Antonio-Javier Garcia-Sanchez, Felipe Garcia-Sanchez, and Joan Garcia-Haro. Wireless Sensor Network Deployment for Integrating Video-surveillance and Data-monitoring in Precision Agriculture over Distributed Crops. *Computers and Electronics in Agriculture*, 75(2):288–303, 2011.

[59] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[60] Shahrzad Gholami, Bryan Wilder, Matthew Brown, Dana Thomas, Nicole Sintov, and Milind Tambe. Divide to Defend: Collusive Security Games. In *International Conference on Decision and Game Theory for Security (GameSec)*, pages 272–293, 2016.

[61] Marcus Gutierrez and Christopher Kiekintveld. Adapting with Honeypot Configurations to Detect Evolving Exploits. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1565–1567, 2017.

[62] James Hannan. Approximation to Bayes Risk in Repeated Play. *Contributions to the Theory of Games*, 3:97–139, 1957.

[63] Kristoffer Arnsfelt Hansen, Thomas Dueholm Hansen, Peter Bro Miltersen, and Troels Bjerre Sørensen. Approximability and Parameterized Complexity of Minmax Values. In *Conference on Web and Internet Economics (WINE)*, volume 5385, pages 684–695, 2008.

[64] John C. Harsanyi. Games with Incomplete Information Played by 'Bayesian' Players, Parts I, II and III. *Management Science*, 14:159–182, 320–334 and 486–502, 1972.

[65] Hsi-Ming Ho and Joël Ouaknine. The Cyclic-Routing UAV Problem is PSPACE-Complete. In *International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, pages 328–342, 2015.

[66] Leonid Hurwicz. The Design of Mechanisms for Resource Allocation. *American Economic Review*, 63:1–30, 1973.

[67] Manish Jain, Bo An, and Milind Tambe. An Overview of Recent Application Trends at the AAMAS Conference: Security, Sustainability, and Safety. *AI Magazine*, 33(3):14–28, 2012.

[68] Adam Kalai and Santosh Vempala. Efficient Algorithms for Online Decision Problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

[69] Debarun Kar, Fei Fang, Francesco Delle Fave, Nicole Sintov, and Milind Tambe. A Game of Thrones: When Human Behavior Models Compete in Repeated Stackelberg Security Games. In *International*

Conference on Autonomous Agents and Multiagent Systems (AAMAS), pages 1381–1390, 2015.

[70] Debarun Kar, Thanh H Nguyen, Fei Fang, Matthew Brown, Arunesh Sinha, Milind Tambe, and Albert Xin Jiang. Trends and Applications in Stackelberg Security Games. *Handbook of Dynamic Game Theory*, pages 1–47, 2017.

[71] Richard M. Karp. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[72] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordóñez, and Milind Tambe. Computing Optimal Randomized Resource Allocations for Massive Security Games. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 689–696, 2009.

[73] Richard Klíma, Christopher Kiekintveld, and Viliam Lisỳ. Online Learning Methods for Border Patrol Resource Allocation. In *International Conference on Decision and Game Theory for Security (GameSec)*, pages 340–349, 2014.

[74] ByoungChul Ko, JunOh Park, and Jae-Yeal Nam. Spatiotemporal Bag-of-features for Early Wildfire Smoke Detection. *Image and Vision Computing*, 31(10):786–795, 2013.

[75] Dmytro Korzhyk, Vincent Conitzer, and Ronald Parr. Complexity of Computing Optimal Stackelberg Strategies in Security Resource Allocation Games. In *Conference on Artificial Intelligence (AAAI)*, 2010.

[76] Presse Louvre. 7.4 Million Visitors to the Louvre in 2016, 2017.

[77] Michael Maschler, Eilon Solan, and Shmuel Zamir. *Game Theory*. Cambridge University Press, 2013.

[78] Colin McDiarmid. On the Method of Bounded Differences. *Surveys in Combinatorics*, 141(1):148–188, 1989.

[79] Daniel L. McFadden. Quantal Choice Analysis: A Survey. In *Annals of Economic and Social Measurement*, volume 5, pages 363–390. NBER, 1976.

[80] Daniel L. McFadden. Econometric Analysis of Qualitative Response Models. *Handbook of Econometrics*, 2:1395–1457, 1984.

[81] Richard D. McKelvey and Thomas R. Palfrey. Quantal Response Equilibria for Normal Form Games. *Games and Economic Behavior*, 10(1):6–38, 1995.

[82] Enrique Munoz de Cote, Ruben Stranders, Nicola Basilico, Nicola Gatti, and Nick Jennings. Introducing Alarms in Adversarial Patrolling Games. In *International conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1275–1276, 2013.

[83] Nysret Musliu. *Local Search Algorithm for Unicost Set Covering Problem*. Springer, 2006.

[84] Roger Myerson. Optimal Auction Design. *Mathematics of Operations Research*, 6:58–73, 1981.

[85] Roger B. Myerson. *Game Theory: Analysis of Conflict*, volume 1. Harvard University Press, 1991.

[86] John F. Nash. Equilibrium Points in n-Person Games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49, 1951.

[87] John F. Nash. Non-Cooperative Games. *Annals of Mathematics*, 54:286–295, 1951.

[88] Thanh H. Nguyen, Francesco M Delle Fave, Debarun Kar, Aravind S. Lakshminarayanan, Amulya Yadav, Milind Tambe, Noa Agmon, Andrew J. Plumptre, Margaret Driciru, Fred Wanyama, et al. Making the Most of Our Regrets: Regret-based Solutions to Handle Payoff Uncertainty and Elicitation in Green Security Games. In *International Conference on Decision and Game Theory for Security (GameSec)*, pages 170–191, 2015.

[89] Thanh Hong Nguyen, Rong Yang, Amos Azaria, Sarit Kraus, and Milind Tambe. Analyzing the Effectiveness of Adversary Modeling in Security Games. In *Conference on Artificial Intelligence (AAAI)*, pages 718–724, 2013.

[90] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*, volume 1. Cambridge University Press Cambridge, 2007.

[91] Martin J. Osborne. *An Introduction to Game Theory*, volume 3. Oxford University Press New York, 2004.

[92] Katerina Papadaki, Steve Alpern, Thomas Lidbetter, and Alec Morton. Patrolling a border. *Operations Research*, 64(6):1256–1269, 2016.

[93] Christos H. Papadimitriou and Mihalis Yannakakis. The traveling Salesman Problem with Distances One and Two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[94] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. Playing Games for Security: An Efficient Exact Algorithm for Solving Bayesian Stackelberg Games. In *International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 895–902, 2008.

[95] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Deployed ARMOR Protection: The Application of a Game-theoretic Model for Security at the Los Angeles International Airport. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 125–132, 2008.

[96] James Pita, Milind Tambe, Chris Kiekintveld, Shane Cullen, and Erin Steigerwald. GUARDS: Game Theoretic Security Allocation on a National Scale. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 37–44, 2011.

[97] Yundi Qian, William B Haskell, Albert Xin Jiang, and Milind Tambe. Online Planning for Optimal Protector Strategies in Resource Conservation Games. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 733–740, 2014.

[98] Yundi Qian, Chao Zhang, Bhaskar Krishnamachari, and Milind Tambe. Restless Poachers: Handling Exploration-exploitation Tradeoffs in Security Domains. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 123–131, 2016.

[99] Herbert Robbins. Some Aspects of the Sequential Design of Experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[100] William Ruckle, Robert Fennell, Paul T. Holmes, and Charles Fennemore. Ambushing Random Walks I: Finite Models. *Operations Research*, 24(2):314–324, 1976.

[101] Martin W. P. Savelsbergh. Local Search in Routing Problems with Time Windows. *Annals of Operations Research*, 4:285–305, 1985.

[102] Thomas C. Schelling. *The Strategy of Conflict*, volume 1. Harvard University Press, 1960.

[103] Aaron Schlenker, Matthew Brown, Arunesh Sinha, Milind Tambe, and Ruta Mehta. Get Me to My GATE on Time: Efficiently Solving General-Sum Bayesian Threat Screening Games. In *European Conference on Artificial Intelligence (ECAI)*, pages 1476–1484, 2016.

[104] Aaron Schlenker, Haifeng Xu, Mina Guirguis, Chris Kiekintveld, Arunesh Sinha, Milind Tambe, Solomon Sonya, Darryl Balderas, and Noah Dunstatter. Don't Bury your Head in Warnings: A Game-Theoretic Approach for Intelligent Allocation of Cyber-security Alerts. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[105] Reinhard Selten. Spieltheoretische Behandlung eines Oligopolmodells mit Nachfragetragheit. *Zeitschrift fur die gesamte Staatswissenschaft*, 121:301–324 and 667–689, 1965.

[106] Reinhard Selten. Reexamination of the Perfectness Concept for Equilibrium Points in Extensive Games. *International Journal of Game Theory*, 4:25–55, 1975.

[107] Lloyd S. Shapley and R. N. Snow. Basic Solutions of Discrete Games. *Contributions to the Theory of Games*, 1:27–35, 1950.

[108] Eric Shieh, Manish Jain, Albert Xin Jiang, and Milind Tambe. Efficiently Solving Joint Activity Based Security Games. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 346–352, 2013.

[109] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-theoretic, and Logical Foundations*. Cambridge University Press, 2008.

[110] Marwan Simaan and Jose B. Cruz Jr. On the Stackelberg Strategy in Nonzero-Sum Games. *Journal of Optimization Theory and Applications*, 11(5):533–555, 1973.

[111] Efrat Sless, Noa Agmon, and Sarit Kraus. Multi-robot Adversarial Patrolling: Facing Coordinated Attacks. In *International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, pages 1093–1100, 2014.

[112] Zhi Sun, Pu Wang, Mehmet C Vuran, Mznah A Al-Rodhaan, Abdullah M Al-Dhelaan, and Ian F Akyildiz. BorderSense: Border Patrol through Advanced Wireless Sensor Networks. *Ad Hoc Networks*, 9(3):468–477, 2011.

[113] J. Tsai, S. Rathi, C. Kiekintveld, F. Ordóñez, and M. Tambe. IRIS - A Tool for Strategic Security Allocation in Transportation Networks. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1327–1334, 2009.

[114] Albert W. Tucker. On Jargon: The Prisoner's Dilemma. *Undergraduate Mathematics Applications Project Journal*, 1:101, 1980.

[115] Karl Tuyls and Gerhard Weiss. Multiagent Learning: Basics, Challenges, and Prospects. *AI Magazine*, 33(3):41, 2012.

[116] Amos Tversky and Daniel Kahneman. Advances in Prospect Theory: Cumulative Representation of Uncertainty. *Journal of Risk and Uncertainty*, 5:297–323, 1992.

[117] John von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[118] John von Neumann and Oskar Morgenstern. Theory of Games and Economic Behavior. *Bullettin of American Mathematical Society*, 51:498–504, 1945.

[119] Heinrich Von Stackelberg. *Marktform und Gleichgewicht*. Springer, 1934.

[120] Bernhard von Stengel. Efficient Computation of Behavior Strategies. *Games Economic Behavior*, 14:220–246, 1996.

[121] Bernhard von Stengel and Daphne Koller. Team-maxmin Equilibria. *Games and Economic Behavior*, 21:309–321, 1997.

[122] Bernhard Von Stengel and Shmuel Zamir. Leadership with Commitment to Mixed Strategies. Technical report, 2004.

[123] Yevgeniy Vorobeychik, Bo An, Milind Tambe, and Satinder P. Singh. Computing Solutions in Infinite-Horizon Discounted Adversarial Patrolling Games. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 314–322, 2014.

[124] Xinrun Wang, Qingyu Guo, and Bo An. Stop Nuclear Smuggling Through Efficient Container Inspection. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 669–677, 2017.

[125] Edmund Taylor Whittaker and George Neville Watson. *A Course of Modern Analysis*. Cambridge university press, 1996.

[126] Haifeng Xu, Long Tran-Thanh, and Nicholas R. Jennings. Playing Repeated Security Games with No Prior Knowledge. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 104–112, 2016.

[127] Rong Yang, Benjamin Ford, Milind Tambe, and Andrew Lemieux. Adaptive Resource Allocation for Wildlife Protection against Illegal Poachers. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 453–460, 2014.

[128] Rong Yang, Christopher Kiekintveld, Fernando Ordonez, Milind Tambe, and Richard John. Improving Resource Allocation Strategy Against Human Adversaries in Security Games. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 22, page 458, 2011.

[129] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless Sensor Network Survey. *Computer Networks*, 52(12):2292–2330, 2008.

[130] Yue Yin and Bo An. Efficient Resource Allocation for Protecting Coral Reef Ecosystems. In *International Conference on Artificial Intelligence (IJCAI)*, pages 531–537, 2016.

[131] Youzhi Zhang, Bo An, Long Tran-Thanh, Zhen Wang, Jiarui Gan, and Nicholas R. Jennings. Optimal Escape Interdiction on Transportation Networks. In *International Conference on Artificial Intelligence (IJCAI)*, pages 3936–3944, 2017.

[132] Mengchen Zhao, Bo An, and Christopher Kiekintveld. Optimizing Personalized Email Filtering Thresholds to Mitigate Sequential Spear

Phishing Attacks. In *Conference on Artificial Intelligence (AAAI)*, pages 658–665, 2016.

Appendices

APPENDIX $\mathcal{A}$

Proofs and Additional Results

A.1 Proof of Theorem 4.1

We first define a special class of trees we call *special 2-level stars* (S2L-STAR). See Figure A.1 for an example.

Definition A.1 (S2L-STAR)**.** *Special 2-level star graph instances (S2L-STAR) are:*

- $V = \{v_0, v_1, v_2, \ldots, v_{2n}\}$*, where* v_0 *is the starting position;*
- $T = V \setminus \{v_0\}$*, where vertices* v_i *with* $i \in \{1, \ldots, n\}$ *are called* inner targets*, while vertices* v_i *with* $i \in \{n+1, \ldots, 2n\}$ *are called* outer targets*;*
- $E = \{(v_0, v_i), (v_i, v_{n+i}) : \forall i \in \{1, \ldots, n\}\}$ *and we refer to the pair of edges* $((v_0, v_i), (v_i, v_{n+i}))$ *as the* i*-th branch;*
- *travel costs are* $c(v_0, v_i) = c(v_i, v_{n+i}) = \gamma_i$ *for every* $i \in \{1, \ldots, n\}$*, where* $\gamma_i \in \mathbb{N}^+$*;*

- *penetration times, for* $i \in \{1, \ldots, n\}$, $d(t) = \begin{cases} 6H - 3\gamma_i & t = v_i \\ 10H - 2\gamma_i, & t = v_{n+i} \end{cases}$, *where* $H = \frac{\sum_{i=1}^{n} \gamma_i}{2}$*;*
- $\pi(t) = 1$ *for every* $t \in T$.

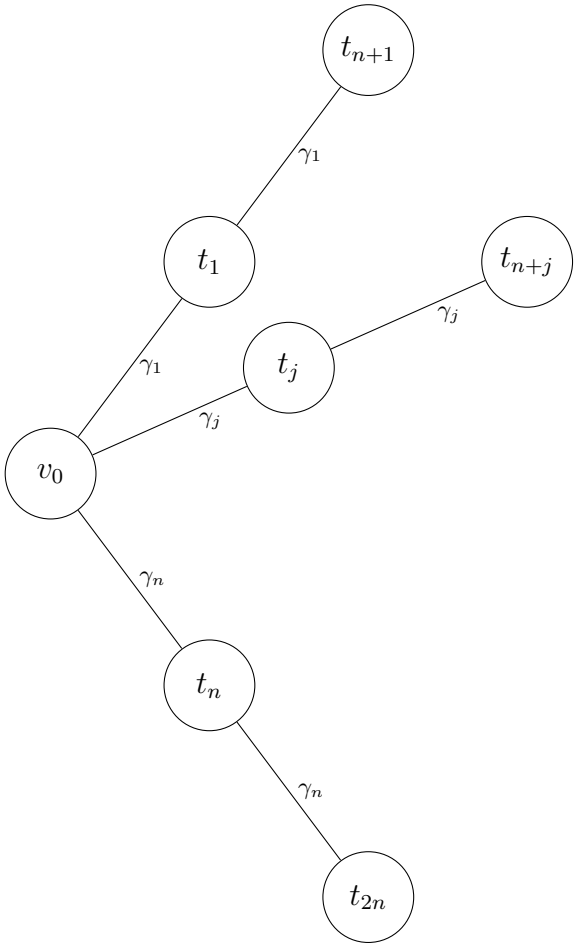


Figure A.1: *Special 2-level star graph.*

Initially, we show a property of S2L-STAR instances that we shall use below.

Lemma A.1. *If an instance of S2L-STAR admits a maximal covering route* r *that covers all the targets, then the branches can be partitioned in two sets* C_1 *and* C_2 *such that:*

- *all the branches in C_1 are visited only once while all the branches in C_2 are visited twice, and*
- $\sum_{i \in C_1} \gamma_i = \sum_{i \in C_2} \gamma_i = H$.

Proof. Initially, we observe that, in a feasible solution, the visit of a branch can be of two forms. If branch i is visited once, then $\mathcal{D}$ will visit the inner target before time $6H - 3\gamma_i$ and immediately after the outer target. C_1 denotes the set of all the branches visited according to this form. If branch i is visited twice, then $\mathcal{D}$ will visit at first the inner target before time $6H - 3\gamma_i$, coming back to v_0 immediately after, and subsequently in some time after $6H - 3\gamma_i$, but before $10H - 2\gamma_i$, $\mathcal{D}$ will visit again the inner target and immediately after the outer target. C_2 denotes the set of all the branches that are visited according to this form. All the other forms of visits (e.g., three or more visits, different order visits, and visits at different times) are useless and any route in which some branch is not neither in C_1 nor in C_2 can be modified such that all the branches are either in C_1 or in C_2 strictly decreasing the cost of the solution as follows:

- if branch i is visited only once and the visit of the inner target is after time $6H - 3\gamma_i$, then the solution is not feasible;
- if branch i is visited twice and the first visit of the inner target is after time $6H - 3\gamma_i$, then the solution is not feasible;
- if branch i is visited twice and the second visit of the inner target is before time $6H - 3\gamma_i$, then the first visit of the branch can be omitted saving $2\gamma_i$;
- if branch i is visited twice and the outer target is visited during the first visit, then the second visit of the branch can be omitted saving $\geq 2\gamma_i$;
- if branch i is visited three or more times, all the visits except the first one in which the inner target is visited and the first one in which the outer target is visited can be omitted saving $\geq 2\gamma_i$.

We assume that, if there is a maximal covering route r that covers all the targets, then the visits are such that $C_1 \cup C_2 = \{1, \ldots, n\}$ and therefore that each branch is visited either once or twice as discussed above. We show below that in S2L-STAR instances such an assumption is always true. Since r covers all the targets, we have that the following conditions are satisfied:

$$2\sum_{i\in C_2}\gamma_i + 4\sum_{i\in C_1}\gamma_i \leq 6H \tag{A.1}$$

$$6\sum_{i\in C_2}\gamma_i + 4\sum_{i\in C_1}\gamma_i \leq 10H \tag{A.2}$$

Constraint (A.1) requires that the cost of visiting entirely all the branches in C_1 and partially (only the inner target) all the branches in C_2 is not larger than the penetration times of the inner targets. Notice that this holds only when the last inner target is first-visited on a branch in C_1. We show below that such assumption is always verified. Constraint (A.2) requires that the cost of visiting entirely all the branches in C_1 and at first partially and subsequently entirely all the branches in C_2 is not larger than the penetration times of the outer targets. We can simplify the above pair of constraints as follows:

$$\underbrace{2\sum_{i\in C_2}\gamma_i + 2\sum_{i\in C_1}\gamma_i}_{4H} + 2\sum_{i\in C_1}\gamma_i \leq 6H$$

$$2\sum_{i\in C_2}\gamma_i + \underbrace{4\sum_{i\in C_2}\gamma_i + 4\sum_{i\in C_1}\gamma_i}_{8H} \leq 10H$$

obtaining:

$$\sum_{i\in C_1}\gamma_i \leq H$$

$$\sum_{i\in C_2}\gamma_i \leq H$$

since, by definition, $\sum_{i\in C_1}\gamma_i + \sum_{i\in C_2}\gamma_i = 2H$, it follows that:

$$\sum_{i\in C_1}\gamma_i = \sum_{i\in C_2}\gamma_i = H.$$

Therefore, if r covers all the targets and it is such that all the branches belong either to C_1 or to C_2, we have that r visits the last outer target exactly at its penetration time. This is because Constraints (A.1) and (A.2) hold as equalities. Thus, as shown above, in any route in which a branch is not neither in C_1 nor in C_2 we can change the visits such that all the branches are in either C_1 or C_2, strictly reducing the total cost. It follows that no

route with at least one branch that is not neither in C_1 nor in C_2 can have a total cost equal to or smaller than the penetration time of the outer targets. Similarly, from the above equality it follows that any solution where the last inner target is first-visited on a C_2 branch can be strictly improved by moving such branch to C_1 and therefore no route in which the last inner target is first-visited on a C_2 branch can have a total cost equal to or smaller than the penetration time of the outer targets. □

We can now prove Theorem 4.1.

Theorem 4.1. *k-SRG-v is* NP*-hard even when $|S| = 1$ and the graph is a tree.*[1]

Proof. We provide a reduction from PARTITION that is known to be weakly NP-hard.

Definition A.2 (PARTITION problem)**.** *The PARTITION problem is defined as:*
INSTANCE: A finite set $I = \{1, 2, \ldots, l\}$, a size $a_i \in \mathbb{N}^+$ for each $i \in I$, and a bound $B \in \mathbb{N}^+$ such that $\sum_{i \in I} a_i = 2B$;
QUESTION: Is there any subset $I' \subseteq I$ such that $\sum_{i \in I'} a_i = \sum_{i \in I \setminus I'} a_i = B$?

For the sake of clarity, we divide the proof in steps.

Reduction. We map an instance of PARTITION to an instance of k-SRG-v on S2L-STAR graphs as follows

- $S = \{s\}$,
- $n = l$ (i.e., the number of branches in S2L-STAR equals the number of elements in PARTITION);
- $\gamma_i = a_i$ for every $i \in I$;
- $H = B$,
- $k = 0$.

The rationale is that there is a feasible solution for PARTITION if and only if there is the maximal covering route that covers all the targets in a k-SRG-v on a S2L-STAR graph.

If. From Lemma A.1 we know that, if there is the maximal covering route that covers all the targets in a k-SRG-v on a S2L-STAR graph, then the branches can be partitioned in two sets C_1, C_2 such that $\sum_{i \in C_1} \gamma_i =$

[1] Rigorously speaking, we show NP-hardness for a more specific class of trees we call S2L-STARS. Details can be found in A.1.

$\sum_{i \in C_2} \gamma_i = H$. By construction $\gamma_i = a_i$ and $H = B$. So, if there is the maximal covering route that covers all the targets in a k-SRG-v on a S2L-STAR graph, then there is partition of set I in two subsets $I' = C_1$ and $I'' = C_2$ such that $\sum_{i \in C_1} \gamma_i = \sum_{i \in I'} a_i = H = B = \sum_{i \in C_2} \gamma_i = \sum_{i \in I''} a_i$.

Only if. If PARTITION admits a feasible solution, then, once assigned $I' = C_1$ and $I'' = C_2$, it is straightforward to see that the route visits all the targets by their penetration times and therefore that the route is a maximal covering route. □

As a final remark, let us notice that this result does not exclude the existence of an FPTAS, i.e., Fully Polynomial Time Approximation Scheme (in fact, PARTITION admits an FPTAS).

A.2 Proof of Theorem 4.3

Theorem 4.3. *There is no polynomial-time algorithm for MAX-COV-SET unless* P = NP.

Proof. Assume that, for simplicity, $S = \{s_1\}$ and that $T(s_1) = T$. Initially, we observe that MAX-COV-SET is in co-NP. Indeed, any covering route r such that $T(r) \supset T'$ is a NO certificate for MAX-COV-SET, placing it in co-NP. (Notice that, trivially, any covering route has length bounded by $O(|T|^2)$; also, notice that due to Theorem 2, having a covering set would not suffice given that we cannot verify in polynomial time whether it is actually covering unless P = NP.)

Let us suppose we have a polynomial-time algorithm for MAX-COV-SET, called A. Then (since P $\subseteq$ NP $\cap$ co-NP) we have a polynomial algorithm for the complement problem, i.e., deciding whether all the covering routes for T' are dominated. Let us consider the following algorithm: given an instance for COV-SET specified by graph $G = (V, E)$, a set of target T with penetration times d, and a starting vertex v:

1. assign to targets in T a lexicographic order $t_1, t_2, \ldots, t_{|T|}$;
2. for every $t \in T$, verify if $\{t\}$ is a covering set in $O(|T|)$ time by comparing $\omega^*_{v,t}$ and $d(t)$; if at least one is not a covering set, then output NO and terminate; otherwise set $\hat{T} = \{t_1\}$ and $k = 2$;
3. apply algorithm A on the following instance: graph $G = (V, E)$, target set $\{\hat{T} \cup \{t_k\}, \hat{d}\}$ (where $\hat{d}$ is d restricted to $\hat{T} \cup \{t_k\}$), start vertex v, and covering set $\hat{T}$;

4. if A's output is YES (that is, $\hat{T}$ is not maximal) then set $\hat{T} = \hat{T} \cup \{t_k\}$, $k = k + 1$ and restart from step 3; if A's output is NO and $k = |T|$ then output YES; if A's output is NO and $k < |T|$ then output NO;

Thus, the existence of A would imply the existence of a polynomial algorithm for COV-SET which (under P $\neq$ NP) would contradict Theorem 2. □

A.3 Proof of Theorem 4.4

Theorem 4.4. *The optimization version of k-SRG-v, say OPT-SRG-v, is* APX*-hard even in the restricted case in which the graph is metric, there is only one signal s, all targets $t \in T(s)$ have the same penetration time $d(t)$, there exists a maximal covering route covering all the targets.*

Proof. We produce an approximation-preserving reduction from TSP(1,2) that is known to be APX-hard [93]. For the sake of clarity, we divide the proof in steps.

TSP(1,2) instance. An instance of undirected TSP(1,2) is defined as follows:

- a set of vertices V_{TSP};
- a set of edges composed of an edge per pair of vertices;
- a symmetric matrix C_{TSP} of weights, whose values can be 1 or 2, each associated with an edge and representing the cost of the shortest path between the corresponding pair of vertices.

The goal is to find the minimum cost tour. Let us denote by $OPTSOL_{TSP}$ and OPT_{TSP} the optimal solution of TSP(1,2) and its cost, respectively. Furthermore, let us denote by $APXSOL_{TSP}$ and APX_{TSP} an approximate solution of TSP(1,2) and its cost, respectively. It is known that there is no polynomial-time approximation algorithm with $APX_{TSP}/OPT_{APX} < \alpha$ for some $\alpha > 1$, unless P = NP [93].

Reduction. We map an instance of TSP(1,2) to a specific instance of SRG-v as follows:

- there is only one signal s;
- $T(s) = V_{TSP}$;
- $w^*_{t,t'} = C_{TSP}(t, t')$, for every $t, t' \in T(s)$;
- $\pi(t) = 1$, for every $t \in T(s)$;

- $w^*_{v,t} = 1$, for every $t \in T(s)$;
- $d(t) = \begin{cases} OPT_{TSP} & \text{if } OPT_{TSP} = |V_{TSP}| \\ OPT_{TSP} - 1 & \text{if } OPT_{TSP} > |V_{TSP}| \end{cases}$, for every $t \in T(s)$.

In this reduction, we use the value of OPT_{TSP} even if there is no polynomial-time algorithm solving exactly TSP(1,2), unless $\mathsf{P} = \mathsf{NP}$. We show below that with an additional polynomial-time effort we can deal with the lack of knowledge of OPT_{TSP}.

OPT-SRG-v optimal solution. By construction of the SRG-v instance, there is a covering route starting from v and visiting all the targets $t \in T(s)$, each within its penetration time. This route has a cost of exactly $d(t)$ and it is $\langle v, t_1, \ldots, t_{|T(s)|} \rangle$, where $\langle t_1, \ldots, t_{|T(s)|}, t_1 \rangle$ corresponds to $OPTSOL_{TSP}$ with the constraint that $w^*_{t_{|T(s)|},t_1} = 2$ if $OPT_{TSP} > |V_{TSP}|$ (essentially, we transform the tour in a path by discarding one of the edges with the largest cost). Therefore, the optimal solution of SRG-v, say $OPTSOL_{SRG}$, prescribes to play the maximal route with probability one and the optimal value, say OPT_{SRG}, is 1.

OPT-SRG-v approximation. Let us call $APXSOL_{SRG}$ and APX_{SRG} an approximate solution of OPT-SRG-v and its value, respectively. We assume there is a polynomial-time approximation algorithm with $\frac{APX_{SRG}}{OPT_{SRG}} \geq \beta$ where $\beta \in (0, 1)$. Let us notice that $APXSOL_{SRG}$ prescribes to play a polynomially upper bounded number of covering routes with strictly positive probability. We introduce a lemma that characterizes such covering routes.

Lemma A.2. *The longest covering route played with strictly positive probability in $APXSOL_{SRG}$ visits at least $\beta|T(s)|$ targets.*

Proof. Assume by contradiction that the longest route visits $\beta|T(s)| - 1$ targets. The best case in terms of maximization of the value of OPT-SRG-v is, due to reasons of symmetry (all the targets have the same value), when there is a set of $|T(s)|$ covering routes of length $\beta|T(s)| - 1$ such that each target is visited exactly by $\beta|T(s)| - 1$ routes. When these routes are available, the best strategy is to randomize uniformly over the routes. The probability that a target is covered is $\beta - \frac{1}{|T(s)|}$ and therefore the value of APX_{SRG} is $\beta - \frac{1}{|T(s)|}$. This leads to a contradiction, since the algorithm would provide an approximation strictly smaller than β. □

TSP(1,2) approximation from OPT-SRG-v approximation. We use the above lemma to show that we can build a $(3-2\beta)$-approximation for TSP(1,2)

from a β-approximation of OPT-SRG-v. Given an $APXSOL_{SRG}$, we extract the longest covering route played with strictly positive probability, say $\langle v, t_1, \ldots, t_{\beta|T(s)|}\rangle$. The cost of the route is at most $d(t)$, otherwise it would not cover $\beta|T(s)|$ targets. Any tour $\langle t_1, \ldots, t_{\beta|T(s)|}, \ldots, t_{|T(s)|}, t_1\rangle$ has a cost not larger than $d(t)-1+2(1-\beta)|T(s)| = OPT_{TSP}-1+2(1-\beta)|V_{TSP}|$, in the worst case in which all edges in $\langle t_{\beta|T(s)|}, t_{\beta|T(s)|+1}, \ldots, t_{|T(s)|}, t_1\rangle$ have a cost of 2. Given that $OPT_{TSP} \geq |V_{TSP}|$, we have that such a tour has a cost not larger than $OPT_{TSP}-1+2(1-\beta)|V_{TSP}| \leq OPT_{TSP}(3-2\beta)$. Therefore, the tour is a $(3-2\beta)$-approximation for TSP(1,2). Since TSP(1,2) is not approximable in polynomial time for any approximation ratio smaller than α, we have the constraint that $3-2\beta \geq \alpha$, and therefore that $\beta \leq \frac{3-\alpha}{2}$. Since $\alpha > 1$, we have that $\frac{3-\alpha}{2} < 1$ and therefore that there is no polynomial-time approximation algorithm for OPT-SRG-v when $\beta \in (\frac{3-\alpha}{2}, 1)$, unless $\mathsf{P} = \mathsf{NP}$.

OPT_{TSP} oracle. In order to deal with the fact that we do not know OPT_{TSP}, we can execute the approximation algorithm for OPT-SRG-v using a guess over OPT_{TSP}. More precisely, we execute the approximation algorithm for every value in $\{|V_{TSP}|, \ldots, 2|V_{TSP}|\}$ and return the best approximation found for TSP(1,2). Given that $OPT_{TSP} \in \{|V_{TSP}|, \ldots, 2|V_{TSP}|\}$, there is an execution of the approximation algorithm that uses the correct guess. □

A.4 Proof of Theorem 4.12

Theorem 4.12. LM-*COV-ROUTE is* NP-*complete.*

Proof. We divide the proof in two steps, membership and hardness.

Membership. Given a YES certificate constitutes by a route, the verification is easy, requiring one to apply the route and check whether each target is visited by its deadline. It requires linear time in the number of targets.

Hardness. Let us consider the Restricted Hamiltonian Circuit problem (RHC) which is known to be NP-complete. RHC is defined as follows: given a graph $G_H = (V_H, E_H)$ and an Hamiltonian path $P = \langle h_1, \ldots, h_n\rangle$ for G_H such that $h_i \in V_H$ and $(h_1, h_n) \notin E_H$, find an Hamiltonian cycle for G_H. From such instance of RHC, following the approach of [31], we build the following instance for LM-COV-ROUTE:

- $V = V_H \cup \{v_1, v_2, v_t\}$;
- $T = V_H \cup \{v_t\}$;
- $E = E_H \cup \{(h_n, v_t), (h_i, v_s) : i \in \{1, \ldots, n\}\}$;

- $d(v_t) = n + 1$ and $d(t) = n$ for any $t \in T$ with $t \neq v_t$;
- $w_{v,v'} = \begin{cases} 1 & \text{if } v = h_n, v' = v_t \\ 1 & \text{if } v = h_i, v' = h_j, \forall i, j \in \{1, \dots n\} \\ 1 & \text{if } v = v_1, v' = h_1 \\ 2 & \text{if } v = v_1, v' = h_{n-1} \\ \geq 2 & \text{if } v = v_1, v' = h_i, \forall i \in \{1, \dots n-2, n\} \\ \geq 2 & \text{if } v = v_1, v' = v_t \\ 2 & \text{if } v = v_2, v' = h_1 \\ 1 & \text{if } v = v_2, v' = h_{n-1} \\ \geq 2 & \text{if } v = v_2, v' = h_i, \forall i \in \{1, \dots n-2, n\} \\ \geq 2 & \text{if } v = v_2, v' = v_t \end{cases}$;
- $r_1 = \langle v_1, h_1, \cdots, h_n, v_t \rangle$.

Basically, given G_H we introduce three vertices v_1, v_2, v_t, where v_1, v_2 are adjacent starting vertices and v_t is a target. We know the covering routes from v_1, and we aim at finding the covering routes from v_2. The new starting vertex (v_2) is closer to h_{n-1} than the previous one (v_1) by 1 and farther from h_1 than previous one (v_1) by 1. There is no constraint over the distances between the starting vertices and the other targets except that they are larger than or equal to 2. We report in Figure A.2 an example of the above construction. Notice that by construction, if the maximal covering route r_2 with $r_2(0) = v_2$ and $T(r_2) = T$ exists, then v_t must be the last visited target. Route r_1 is covering since $\langle h_1, \dots, h_n \rangle$ is a Hamiltonian path for G_H. We need to show that route r_2 with $r_2(0) = v_2$ and $T(r_2) = T$ exists if and only if G_H admits a Hamiltonian cycle. It can observed that, if r_2 exists, then it must be such that $r_2 = \langle v_2, h_{n-1}, \dots, h_n, v_t \rangle$ and therefore $\langle h_{n-1}, \dots, h_n \rangle$ must be a Hamiltonian path for G_H. Since we know, by r_1, that $(h_{n-1}, h_n) \in E_H$, it follows that $\langle h_{n-1}, \dots, h_n, h_{n-1} \rangle$ is a Hamiltonian cycle. □

A.5 Additional results for special topologies

In this section we show that for some special classes of graphs we can give a polynomial-time algorithm for answering Question 2.

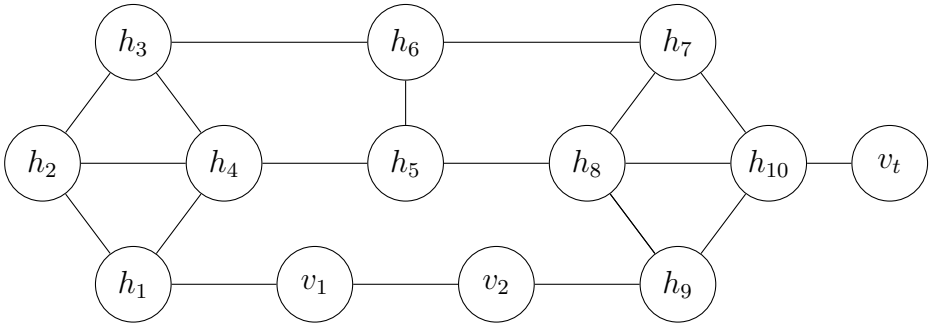
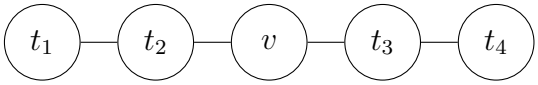


Figure A.2: *Example of construction used in the proof of Theorem 4.12: the Hamiltonian path $\langle h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, h_{10}\rangle$ on G_H is given, as well as covering route r_1 with $r_1(0) = v_1$ and $T(r_1) = T$. It can be observed that there is another Hamiltonian path for G_H, i.e., $\langle h_9, h_8, h_5, h_4, h_1, h_2, h_3, h_6, h_7, h_{10}\rangle$, allowing covering route r_2 with $r_2(0) = v_2$ and $T(r_2) = T$. Notice that, if we remove the edge (h_5, h_8), then covering route r_2 such that $r_2(0) = v_2$ and $T(r_2) = T$ does not exist.*

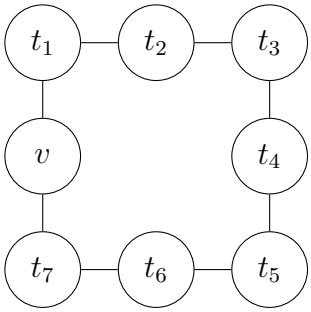
A.5.1 Line and cycle graphs

Let us first concentrate on line graphs for which an example is depicted in Figure A.3(a). Consider a modified version of Algorithm 5 where, when deciding on admissible expansions, we add to Conditions 1-3 the following one: the cost of the newly formed covering set must be smaller than or equal to the penetration time of any target not included in such set. Given $Q^k_{v,t}$, the expansion $Q^{k+1}_{v,w}$ is admissible only if for any $t \in T(s) \setminus Q^k_{v,w}$ the inequality $d(t) \geq c(Q^{k+1}_{v,w})$ holds. Such modified algorithm runs in polynomial time on linear graph instances. To see this, notice that any covering set enumerated has only two possible expansions and can be compactly represented by two vertices: a left one and a right one. All the targets between the two extremes are covered by the covering set. This makes the number of expansions performed by the algorithm $O(|T(s)|^2)$.

The soundness of the algorithm can simply be determined by noticing that the new condition we imposed on admissible expansions prevents Algorithm 5 from generating only those that would never led to the covering set including all the targets. This implies that we can answer Question 2 in polynomial time for this class of instances. Notice that this results can be easily extended to cycle graphs, see Figure A.3(b) and to tree graphs where the number of leaves is fixed.



(a) Linear graph



(b) Cycle graph

Figure A.3: *Special graphs: linear and cycle.*

Assessing the computational complexity of finding the maxmin equilibrium in linear graphs is a problem left open in this paper. While when searching for a route covering all targets we can safely consider only covering sets defined by two extremes, this is not longer true when searching for an equilibrium. The main reason is that we cannot safely discard routes traversing targets whose penetration time is expired and this does not allow one to provide a polynomially bounded representation of the covering sets. We conjecture that the problem is FNP-hard. To support our conjecture, we provide a class of instances whose non-dominated covering sets rises exponentially in $|T|$ for the values of $|T|$ such that the problem can be solved in reasonable time. More precisely, we empirically observed that the number of non-dominated covering sets of these instances is $\Omega(2^{\frac{|T|-1}{2}})$ for $|T| \leq 23$. With $|T| \geq 25$ Algorithm 1 requires too long time to solve the problem and therefore we could not evaluate the number of non-dominated covering sets (a formal proof that the number of covering sets is exponential does not seem straightforward). While this is not a hardness proof, we believe that it could be an evidence and we report such instances because they could be useful for any researcher interested in the study of linear graphs. The instances are defined as follow: given the starting vertex v, there are $(|T|-1)/2$ targets at the left of v and $(|T|-1)/2$ targets at the right of v. The cost of each edge is 1. The value of $d(t)$ is the square of the cost of the shortest path between t and v. In these instances, there are non-dominated routes continuously oscillating from the left to the right of the starting vertex v and *vice versa*.

Let us now consider a *star graph*, as shown in Figure A.4, defined as

follows.

Definition A.3 (SIMPLE-STAR)**.** *Simple star graph instances (SIMPLE-STAR) are:*

- $V = \{v_0, v_1, v_2, \ldots, v_n\}$*, where* v_0 *is the starting vertex of* $\mathcal{D}$*;*
- $T = V \setminus \{v_0\}$*;*
- $E = \{(v_0, v_i), \forall i \in \{1, \ldots, n\}\}$*;*
- *travel costs are* $c(v_0, v_i) = \gamma_i$*, where* $\gamma_i \in \mathbb{N}^+$*;*
- *penetration times* $d(t)$ *and values* $\pi(t)$ *can be any.*

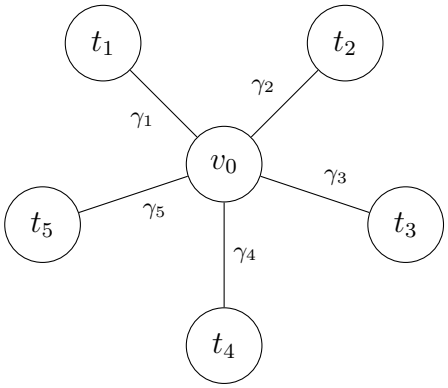


Figure A.4: *Star graph.*

We can state the following theorem, whose proof is not direct as in the case of linear and cycle graphs and thus more details are necessary.

Theorem A.3. *If the maximal covering route* r *covering all the targets exists, the Earliest Due Date algorithm returns* r *in polynomial time once applied to SIMPLE-STAR graph instances.*

Proof. The Earliest Due Date [41] (EDD) algorithm is an optimal algorithm for synchronous (i.e., without release times) aperiodic scheduling with deadlines. It executes (without preemption) the tasks in ascending order according to the deadlines, thus requiring polynomial complexity in the number of tasks. Any SIMPLE-STAR graph instance can be easily mapped to a synchronous aperiodic scheduling problem: each target t_i is an aperiodic task J_i, the computation time of J_i is equal to $2\gamma_i$, the deadline of task J_i is $d(t_i) + \gamma_i$. It is straightforward to see that, if EDD returns a feasible schedule, then there

is the maximal covering route, and, if EDD returns a non-feasible schedule, then there is not any maximal covering route. □

The above result shows that Question 2 can be answered in polynomial time.

We focus on Question 3, showing that also this question can be answered in polynomial time for the above special instances.

Theorem A.4. *Given a signal s, the best pure strategy of $\mathcal{D}$ in an SRG-v game on linear graph, cycle graph, and SIMPLE-STAR graph instances can be found in polynomial time.*

Proof. For the sake of clarity, we describe the algorithm in the simplified case in which there is only one signal s. The extension to the general case is straightforward. The algorithm works as follows:

1. apply the algorithm searching for the route covering all the targets,
2. if the maximal covering route exists, then return it,
3. else remove the target t with the smallest $\pi(t)$ from $T(s)$,
4. go to Point 1.

Essentially, the algorithm returns the subset of targets admitting a covering route minimizing the maximum value among all the non-covered targets. □

Notice that the above results can be easily extended to tree graphs where the number of leaves is fixed.

A.6 Proof of Proposition 5.4

Proposition 5.4 Optimal patrolling strategies in Σ^1 are arbitrarily worse than the optimal patrolling strategies.

Proof. Let us consider the linear graph depicted in Figure 5.2, where t_L and t_R are the only targets. The values of the targets are $\pi(t_L) = \pi(t_R) = 1$ and the deadlines are set to $d(t_L) = d(t_R) = 2(n+1)$. Being the Attacker $\mathcal{A}$ rational, it will attack node t_R once the Defender is patrolling node t_L (or vice versa). Due to this symmetry, i.e., the capture probability from t_L to t_R must be the same from t_R to t_L, the weights x_i of the edges, corresponding to the probabilities of Markovian strategy employed by $\mathcal{D}$, are unknown, but present the symmetric structure reported in the figure above. We want to compute the utility of the Defender $\mathcal{D}$, i.e., the probability that $\mathcal{D}$ will catch $\mathcal{A}$ when $\mathcal{D}$ is in node S and $\mathcal{A}$ attacks target t_R when a Markovian approach is adopted. We denote such value as P_{capture}. To show that P_{capture} goes to

zero as n goes to infinity, we compute an upper bound UB on P_{capture} and show that, in the best case, such bound goes to zero.

By definition, P_{capture} is a function of $x_1, x_2, \ldots, x_n$:

$$P_{\text{capture}} = f(x_1, x_2, \ldots, x_n)$$

We compute the upper bound as:

$$UB = f(x_1, x_2, \ldots, x_n) + g(x_1, x_2, \ldots, x_n),$$

where $g(x_1, x_2, \ldots, x_n) = f(1 - x_1, 1 - x_2, \ldots, 1 - x_n) \geq 0$.
Having defined the function $g(\cdot)$ as above, it holds that:

$$P_{\text{capture}} = f(x_1, \ldots, x_n) \leq f(x_1, \ldots, x_n) + g(x_1, \ldots, x_n) = UB \quad \text{(A.3)}$$

Now we want to maximize the upper bound. By construction, UB can be seen as composed of couples of terms, one taken from $f(\cdot)$, the other from $g(\cdot)$, such that the x_i and the $1 - x_i$ factors of the two terms constituting such couple have symmetric exponents. Let us consider the following couple, where the first term is a term of function $f(\cdot)$ while the second one is taken from function $g(\cdot)$:

$$x_1^{a_1}(1-x_1)^{b_1} \cdot \ldots \cdot x_n^{a_n}(1-x_n)^{b_n} + x_1^{b_1}(1-x_1)^{a_1} \cdot \ldots \cdot x_n^{b_n}(1-x_n)^{a_n} \quad \text{(A.4)}$$

Let us assume that $a_i \leq b_i$. Collecting shared variables, we can rewrite (A.4) as:

$$\prod_{i=1}^{n} x_i^{a_i}(1-x_i)^{a_i} \cdot \left(\prod_{i=1}^{n} x_i^{b_i - a_i} + \prod_{i=1}^{n} (1-x_i)^{b_i - a_i} \right) \quad \text{(A.5)}$$

Let us focus on Equation (A.5). It is easy to see that $\prod_{i=1}^{n} x_i^{a_i}(1-x_i)^{a_i}$ is maximized for $x_1 = \ldots = x_n = \frac{1}{2}$. We consider now $\prod_{i=1}^{n} x_i^{b_i - a_i} + \prod_{i=1}^{n}(1-x_i)^{b_i - a_i}$: also this term is maximized for $x_1 = \ldots = x_n = \frac{1}{2}$ since, when the variables are derived together, the coefficients and the exponents of such variables are all the same.

So, UB is maximized for $x_1 = \ldots = x_n = \frac{1}{2}$. Moreover, we observe that $f(\frac{1}{2}, \ldots, \frac{1}{2}) = g(\frac{1}{2}, \ldots, \frac{1}{2})$. Thus, from Equation (A.3), it follows that:

$$\begin{aligned} f(\tfrac{1}{2}, \ldots, \tfrac{1}{2}) + g(\tfrac{1}{2}, \ldots, \tfrac{1}{2}) &\geq f(x_1, \ldots, x_n) \quad \forall x_i \in [0,1], i \in \{1, \ldots, n\}, \\ 2f(\tfrac{1}{2}, \ldots, \tfrac{1}{2}) &\geq f(x_1, \ldots, x_n), \quad \forall x_i \in [0,1], i \in \{1, \ldots, n\} \end{aligned}$$

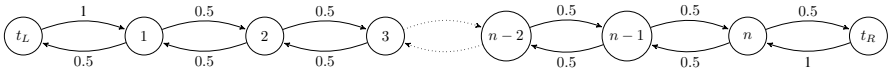


Figure A.5: *Line graph with all weigths equal to $\frac{1}{2}$ adopted for the proof of Proposition 5.4.*

The updated line graph, with $x_i = \frac{1}{2}, \forall i$, is shown in Figure A.5.

Now we want to prove that $f(\frac{1}{2}, \ldots, \frac{1}{2})$ goes to 0 as $n \to \infty$. To do this, we provide an upper bound, say $UB_{1/2}$, on $f(\frac{1}{2}, \ldots, \frac{1}{2})$ and show that such bound goes to 0 as $n \to \infty$. In fact, we enumerate all the paths that can be travelled from t_L to t_R within $d(t_R)$, i.e., all the paths that contribute to the utility of the Defender, namely $UB_{1/2}$.

Among all the paths that go from S to D, we identify the *direct path*, which is constituted only of edges that go from left to right. Thus, looking at Figure A.5, we can see that the utility associated to the direct path is equal to $\left(\frac{1}{2}\right)^n$.

All the other paths can be decomposed in the direct path, that must be necessarily travelled, and some cycles around some nodes, e.g. the ones shown in Figure A.6.

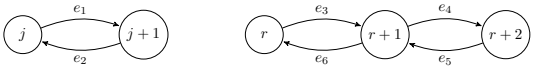


Figure A.6: *Examples of cycles in which a path can be decomposed. $\langle e_1, e_2 \rangle$ is a cycle around the node j while $\langle e_3, e_4, e_5, e_6 \rangle$ can be seen as one cycle around r and one around $r+1$.*

Now, we provide $UB_{1/2}$, enumerating all the possible cycles that can be walked from t_L to t_R, dividing them in three groups according to *how* they are travelled. Then, we show that, when n goes to infinity, the utility obtained from the Markovian approach leads to a utility equal to 0 for the Defender. It can be easily seen that the maximum number of cycles that can be done starting from S before reaching D is $\lfloor \frac{n+1}{2} \rfloor$.

Term 1. The first factor is the utility given by the direct path and cycles around t_L:

$$U_{direct,t_L} = \sum_{k=0}^{\lfloor \frac{n+1}{2} \rfloor} \left(\frac{1}{2}\right)^{n+k}$$

The direct path, obtained for $k = 0$, goes directly from t_L to t_R. The other cycles are all around t_L, with different multiplicity depending on k.

Term 2. The second term is the utility given by cycles among internal nodes:

$$U_n = \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \binom{n+k-1}{k} \left(\frac{1}{2}\right)^{n+2k} + \sum_{k=2}^{\lfloor \frac{n+1}{2} \rfloor} (n-k-1)2^{k-1} \left(\frac{1}{2}\right)^{n+2k}$$

where:

- $\sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \binom{n+k-1}{k} \left(\frac{1}{2}\right)^{n+2k}$ includes all the possible combinations (see the binomial term) around the internal nodes. The exponent of $\left(\frac{1}{2}\right)^{n+2k}$ depends on n since the direct path will be travelled with some cycles at certain points. Such cycles contributes with $2k$, where k is the number of cycles.
- $\sum_{k=2}^{\lfloor \frac{n+1}{2} \rfloor} (n+k-1)2^{k-1} \left(\frac{1}{2}\right)^{n+2k}$ includes all the paths that can be done by adjacent nodes, which allow $\mathcal{D}$ to patrol multiple egdes that go from right to left before taking the way back from t_L to t_R. The first factor counts all the possible groups of adjacent nodes that can be traversed to reach t_R within the deadline. They are $(n+k-1)$, which corresponds to all the shifts of k positions on n nodes. The second factor keeps track of all the possible cycles that can be travelled considering the fact that $\mathcal{D}$ can patrol multiple egdes that go from right to left. Specifically, we can think of the line graph as a unique segment that can be divided into k subsegments, each constituted of some internal nodes. This means that we have to compute all the possible ways in which the subsegments can be traversed considering all the possible orderings among the various subsegments. This is equivalent to compute all the possible ways in which the number k can be obtained summing up different numbers. This value is equal to 2^{k-1}. For the third factor, the considerations are similar the one reported for Term 1.

Term 3. The third term is the utility given by cycles between S and internal nodes:

$$U_{S,n} = \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{j=1}^{k-1} \binom{n+(k-j)-1}{k-j} \left(\frac{1}{2}\right)^{n+2k-j} + \\ + \sum_{k=2}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{j=2}^{k-1} (n-k)2^{k-1} \left(\frac{1}{2}\right)^{n+2k-j}$$

where:

- $\sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{j=1}^{k-1} \binom{n+(k-j)-1}{k-j} \left(\frac{1}{2}\right)^{n+2k-j}$: this expression counts the paths in which some j cycles are performed around t_L and the other $c-j$ cycles around different internal nodes. Given the number of cycles, they can be made around different internal nodes in $\binom{n+(k-j)-1}{k-j}$ ways. We introduce the sum along j since j can be done around t_L while the others around the internal nodes. From $\left(\frac{1}{2}\right)^{n+2k-j}$, we have to subtract j edges that corresponds to the j times the edge between t_L and node 1 is travelled during the j cycles around t_L.
- $\sum_{k=2}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{j=2}^{k-1} (n-k) 2^{k-1} \left(\frac{1}{2}\right)^{n+2k-j}$ includes all the paths that can be done by adjacent nodes, which allow $\mathcal{D}$ to patrol multiple egdes that go from right to left before taking the way back from t_L to t_R. The first two factors are similar to the ones of Term 2 (we have $n-k$ instead of $n-k-1$ since here the shifts include also t_L), while $\left(\frac{1}{2}\right)^{n+2k-j}$ reflects the same rationale expressed in the previous point.

Now we prove that $U_{direct,S} + U_n + U_{S,n} \to 0$ when $n \to \infty$.

Term 1.

$$\begin{aligned} U_{direct,S} &= \sum_{k=0}^{\lfloor \frac{n+1}{2} \rfloor} \left(\frac{1}{2}\right)^{n+k} = \left(\frac{1}{2}\right)^{n} \sum_{k=0}^{\lfloor \frac{n+1}{2} \rfloor} \left(\frac{1}{2}\right)^{k} \\ &= \left(\frac{1}{2}\right)^{n} \cdot \frac{1 - \left(\frac{1}{2}\right)^{\lfloor \frac{n+1}{2} \rfloor + 1}}{1 - \left(\frac{1}{2}\right)} \\ &= \left(\frac{1}{2}\right)^{(n-1)} \cdot \left(1 - \left(\frac{1}{2}\right)^{\lfloor \frac{n+1}{2} \rfloor + 1}\right) \end{aligned}$$

As $n \to \infty$, $U_{direct,S} \to 0$.

Term 2.

$$U_n = \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \binom{n+k-1}{k} \left(\frac{1}{2}\right)^{n+2k} + \sum_{k=2}^{\lfloor \frac{n+1}{2} \rfloor} (n-k-1) 2^{k-1} \left(\frac{1}{2}\right)^{n+2k} \tag{A.6}$$

We tackle the two addends separately. Let us consider the first.

$$\sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \binom{n+k-1}{k} \left(\frac{1}{2}\right)^{n+2k} \tag{A.7}$$

To show that such addend goes to 0 as $n \to \infty$, we adopt the Stirling approximation [125] to write an upper bound for (A.7). Since there are two Stirling approximations for a binomial $\binom{n}{k}$, depending on how large is n w.r.t. k and in our case k grows from 1 to $\lfloor \frac{n+1}{2} \rfloor$, we consider both approximations.

First, we approximate (A.7) considering the cases in which n is large and k is much smaller than n. We adopt this approximation:

$$\binom{n}{k} \approx \frac{\left(\frac{n}{k} - 0.5\right)^k e^k}{\sqrt{2\pi k}} \tag{A.8}$$

Plugging the approximation expression (A.8) into Equation (A.7), we get the following expression:

$$\begin{aligned}
&\sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \frac{\left(\frac{n+k-1}{k} - 0.5\right)^k e^k}{\sqrt{2\pi k}} \left(\frac{1}{2}\right)^{n+2k} \leq \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \left(\frac{n+k}{k}\right)^k e^k \left(\frac{1}{2}\right)^{n+2k} \\
&= \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \left(\frac{n+k}{k}\right)^k 2^{k \log_2 e} 2^{-n-2k} = \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \left(\frac{n+k}{k}\right)^k 2^{-n-2k+k \log_2 e} \\
&\leq \frac{n+1}{2} \left(\frac{n + \frac{n+1}{2}}{\frac{n+1}{2}}\right)^{\frac{n+1}{2}} 2^{-n-2+\log_2 e} \\
&= \frac{n+1}{2} \left(\frac{3n+1}{n+1}\right)^{\frac{n+1}{2}} 2^{-n-2+\log_2 e} \\
&\leq \frac{n+1}{2} \left(\frac{3n+3}{n+1}\right)^{\frac{n+1}{2}} 2^{-n-2+\log_2 e} = \frac{n+1}{2} 3^{\frac{n+1}{2}} 2^{-n-2+\log_2 e} \\
&= \frac{n+1}{2} 2^{\frac{n+1}{2} \log_2 3} 2^{-n-2+\log_2 e} = \frac{n+1}{2} 2^{-n-2+\log_2 e + \log_2 3 \frac{n}{2} + \frac{\log_2 3}{2}} \\
&\approx \frac{n+1}{2} 2^{-\alpha n + \beta},
\end{aligned}$$

with α, β positive constants.

As $n \to \infty$, (A.7) goes to 0.

Now, again for addend (A.7), we exploit the following Stirling approximation, to be adopted when both k and n are much larger than 1.

$$\log_2 \binom{n}{k} \sim nH\left(\frac{k}{n}\right) \tag{A.9}$$

where $H(x) = -x\log_2(x) - (1-x)\log_2(1-x)$. If we consider the approximation (A.9) and plug both sides as exponents to 2, we get:

$$\binom{n}{k} \sim 2^{nH\left(\frac{k}{n}\right)} \tag{A.10}$$

Plugging the RHS of expression (A.10) into (A.7), we get:

$$\begin{aligned}
&\sum_{k=1}^{\lfloor\frac{n+1}{2}\rfloor} 2^{(n+k-1)\left[-\frac{k}{n+k-1}\log_2\left(\frac{k}{n+k-1}\right)-\left(1-\frac{k}{n+k-1}\right)\log_2\left(1-\frac{k}{n+k-1}\right)\right]-n-2k} \\
&= \sum_{k=1}^{\lfloor\frac{n+1}{2}\rfloor} \left(2^{\left[-\frac{k}{n+k-1}\log_2\left(\frac{k}{n+k-1}\right)-\left(1-\frac{k}{n+k-1}\right)\log_2\left(1-\frac{k}{n+k-1}\right)\right]}\right)^{(n+k-1)} \cdot 2^{-n-2k} \\
&= \sum_{k=1}^{\lfloor\frac{n+1}{2}\rfloor} \left(2^{\log_2\left(\frac{k}{n+k-1}\right)^{-\frac{k}{n+k-1}}} \cdot 2^{\log_2\left(1-\frac{k}{n+k-1}\right)^{\left(1-\frac{k}{n+k-1}\right)}}\right)^{(n+k-1)} \cdot 2^{-n-2k} \\
&= \sum_{k=1}^{\lfloor\frac{n+1}{2}\rfloor} \left(\left(\frac{n+k-1}{k}\right)^{\frac{k}{n+k-1}} \cdot \left(\frac{n-1}{n+k-1}\right)^{\frac{n-1}{n+k-1}}\right)^{(n+k-1)} \cdot 2^{-n-2k} \\
&\leq \sum_{k=1}^{\lfloor\frac{n+1}{2}\rfloor} \left(\left(\frac{n+k-1}{k}\right)^{\frac{k}{n+k-1}} \cdot \left(\frac{n}{n+k}\right)^{\frac{n}{n+k}}\right)^{(n+k-1)} \cdot 2^{-n-2k} \\
&\leq \sum_{k=1}^{\lfloor\frac{n+1}{2}\rfloor} \left(\left(\frac{n+k-1}{k}\right)^{\frac{k}{n+k-1}}\right)^{(n+k-1)} \cdot 2^{-n-2k} \\
&= \sum_{k=1}^{\lfloor\frac{n+1}{2}\rfloor} \left(\frac{n+k-1}{k}\right)^{k} \cdot 2^{-n-2k} = \sum_{k=1}^{\lfloor\frac{n+1}{2}\rfloor} \left(1+\frac{n-1}{k}\right)^{k} \cdot 2^{-n-2k} \\
&\leq \frac{n+1}{2}\left(1+\frac{n-1}{\frac{n+1}{2}}\right)^{\frac{n+1}{2}} \cdot 2^{-n-2} \\
&= \frac{n+1}{2}\left(1+2\frac{n-1}{n+1}\right)^{\frac{n+1}{2}} \cdot 2^{-n-2}
\end{aligned}$$

$$= \frac{n+1}{2}(1+2)^{\frac{n+1}{2}} \cdot 2^{-n-2} = \frac{n+1}{2}3^{\frac{n+1}{2}} \cdot 2^{-n-2}$$
$$= 3^{\frac{1}{2}} \cdot 2^{-2}\frac{n+1}{2}3^{\frac{n}{2}} \cdot 2^{-n} \leq \frac{n+1}{2}2^{\log_2 3^{\frac{n}{2}}} \cdot 2^{-n}$$
$$= \frac{n+1}{2}2^{n\left(\frac{\log_2 3}{2}-1\right)} = \frac{n+1}{2}2^{-\alpha n}$$

with α positive constant.

As $n \to \infty$, (A.7) goes to 0.

Now we tackle the second addend of Equation (A.6):

$$\sum_{k=2}^{\lfloor\frac{n+1}{2}\rfloor}(n-k-1)2^{k-1}\left(\frac{1}{2}\right)^{n+2k}$$

Also in this case, we build an upper bound to this term and show it goes to 0 as $n \to \infty$.

$$\sum_{k=2}^{\lfloor\frac{n+1}{2}\rfloor}(n-k-1)2^{k-1}\left(\frac{1}{2}\right)^{n+2k} \leq \left(\frac{1}{2}\right)^{n+1}\sum_{k=2}^{\lfloor\frac{n+1}{2}\rfloor}(n-k)\left(\frac{1}{2}\right)^{k} \tag{A.11}$$

$$\leq \left(\frac{1}{2}\right)^{n+1}\frac{n+1}{2}n\left(\frac{1}{2}\right)^{2} = \left(\frac{1}{2}\right)^{n+3}\frac{n+1}{2}n \tag{A.12}$$

As $n \to \infty$, (A.11) goes to 0.

Thus, U_n goes to 0 as $n \to \infty$.

Term 3.

$$U_{S,n} = \sum_{k=1}^{\lfloor\frac{n+1}{2}\rfloor}\sum_{j=1}^{k-1}\binom{n+(k-j)-1}{k-j}\left(\frac{1}{2}\right)^{n+2k-j} + \tag{A.13}$$

$$+ \sum_{k=2}^{\lfloor\frac{n+1}{2}\rfloor}\sum_{j=2}^{k-1}(n-k)2^{k-1}\left(\frac{1}{2}\right)^{n+2k-j} \tag{A.14}$$

As done for Term 2, we tackle the two addends separately. Let us consider the first addend.

$$\sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{j=1}^{k-1} \binom{n+(k-j)-1}{k-j} \left(\frac{1}{2}\right)^{n+2k-j} \tag{A.15}$$

We denote $k-j=\alpha$. We can rewrite Equation (A.15) as:

$$\begin{aligned}
&\sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{\alpha=1}^{k-1} \binom{n+\alpha-1}{\alpha} \left(\frac{1}{2}\right)^{n+2\alpha+j} \\
&\leq \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{\alpha=1}^{k-1} \binom{n+\alpha-1}{\alpha} \left(\frac{1}{2}\right)^{n+2\alpha}
\end{aligned}$$

As done for the first addend of Term 2, we exploit the two Stirling approximations. Let us start with the first. Following passages similar to the previous ones, we get to the following expression for (A.15):

$$\sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{\alpha=1}^{k-1} \binom{n+\alpha-1}{\alpha} \left(\frac{1}{2}\right)^{n+2\alpha}$$

Again, following passages similar to the ones applied to the first addend of U_n, we get:

$$\begin{aligned}
&\sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{\alpha=1}^{k-1} \binom{n+\alpha-1}{\alpha} \left(\frac{1}{2}\right)^{n+2\alpha} \\
&= \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{\alpha=1}^{k-1} \left(\frac{n+\alpha}{\alpha}\right)^{\alpha-1} \cdot 2^{-n-2\alpha-\alpha\log_2 e} \\
&\leq \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} (k-1) \left(\frac{n+k-1}{k-1}\right)^{k-2} \cdot 2^{-n-2-\log_2 e} \\
&\leq \frac{n+1}{2} \cdot \left(\frac{n+1}{2}-1\right) \left(\frac{n+\frac{n+1}{2}}{\frac{n+1}{2}}\right)^{\frac{n+1}{2}} \cdot 2^{-n-2-\log_2 e} \\
&\leq (n+1)^2 \cdot 3^{\frac{n+1}{2}} \cdot 2^{-n-2-\log_2 e} \\
&= (n+1)^2 \cdot 2^{-\gamma n+\beta}
\end{aligned}$$

As $n \to \infty$, (A.15) goes to 0.

Now we resort to the second Stirling approximation, holding for k and n much larger than 1. (A.15) can then be rewritten as:

$$\begin{aligned}
&\sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{\alpha=1}^{k-1} \binom{n+\alpha-1}{\alpha} \left(\frac{1}{2}\right)^{n+2\alpha+j} \\
&\leq \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{\alpha=1}^{k-1} \binom{n+\alpha-1}{\alpha} \left(\frac{1}{2}\right)^{n+2\alpha} \\
&\leq \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{\alpha=1}^{k-1} \left(1+\frac{n-1}{\alpha}\right)^{\alpha} \cdot 2^{-n-2\alpha} \\
&\leq \sum_{k=1}^{\lfloor \frac{n+1}{2} \rfloor} (k-1) \left(1+\frac{n-1}{k-1}\right)^{k-1} \cdot 2^{-n-2} \\
&\leq \frac{n+1}{2} \cdot \left(\frac{n+1}{2}-1\right) \cdot 3^{\frac{n+1}{2}} \cdot 2^{-n-2} \leq (n+1)^2 \cdot 2^{-\gamma n}
\end{aligned}$$

As $n \to \infty$, (A.15) goes to 0.

Now we focus on the second addend of (A.13):

$$\sum_{k=2}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{j=2}^{k-1} (n-k) 2^{k-1} \left(\frac{1}{2}\right)^{n+2k-j} \tag{A.16}$$

Again, we compute an upper bound on such term and show it goes to 0 as $n \to \infty$.

$$\begin{aligned}
&\sum_{k=2}^{\lfloor \frac{n+1}{2} \rfloor} \sum_{j=2}^{k-1} (n-k) 2^{k-1} \left(\frac{1}{2}\right)^{n+2k-j} \leq \sum_{k=2}^{\lfloor \frac{n+1}{2} \rfloor} (n-k) 2^{k-1} \left(\frac{1}{2}\right)^{n+k+1} \\
&\leq \frac{n+1}{2} (n-2) 2^{\frac{n+1}{2}-1} \left(\frac{1}{2}\right)^{n+2+1} = \frac{n+1}{2} (n-2) \left(\frac{1}{2}\right)^{n+3-\frac{n-1}{2}} \\
&= \frac{n+1}{2} (n-2) \left(\frac{1}{2}\right)^{\frac{n+7}{2}}
\end{aligned}$$

As $n \to \infty$, (A.16) goes to 0.

Thus, $U_{S,n}$ goes to 0 as $n \to \infty$.

Since all the three terms that constitutes $U_{1/2}$ goes to 0 as $n \to \infty$, *a fortiori* this holds for $U_{1/2}$ and, consequently, also $f(\frac{1}{2}, \dots, \frac{1}{2}) \xrightarrow{n\to\infty} 0$ goes to 0. Thus, adopting a Markovian strategy, we get a utility equal to 0 if the number of nodes of the line graph approaches infinity.

We turn now to a non-Markovian approach. In the line graph, the best path to follow is going from t_L to t_R and then back from t_R to t_L. Following this path, the Defender ca reach each target within its deadline, guaranteeing itself a utility equal to 1.

Thus, the ratio between the Markovian and the non-Markovian approach is zero and so a Markovian strategy can be arbitrarily worse than a non-Markovian one. This concludes the proof. □

A.7 Proof of Theorem 8.3

Theorem 8.3. *Given an instance of the ABI-SG problem s.t.* $\Delta b_k > 0$ *for each* $A_k \in \mathcal{A}$ *and applying* FB*, the Defender incurs in a pseudo-regret of:*

$$R_N(\mathfrak{U}) \leq \sum_{k=1}^{K} \frac{2(\lambda_k^2 + \lambda_{k^*}^2)\Delta L_k}{(\Delta b_k)^2},$$

where

$$\lambda_k := \max_{m\in\mathcal{M}} \max_{\boldsymbol{\sigma}\in\mathcal{S}} \ln(\sigma_{A_k}(\boldsymbol{\sigma})_m) - \min_{m\in\mathcal{M}} \min_{\boldsymbol{\sigma}\in\mathcal{S}} \ln(\sigma_{A_k}(\boldsymbol{\sigma})_m)\mathcal{I}\{\sigma_{A_k}(\boldsymbol{\sigma})_m \neq 0\}$$

is the range where the logarithm of the beliefs realizations lies (excluding realizations equal to zero, which end the exploration of a profile) and $\mathcal{S} := \cup_k \boldsymbol{\sigma}_D^*(A_k)$ *is the set of the available best response to the Attackers profile.*

Proof. Let us analyze the regret of the FB algorithm. We get some regret if the algorithm selects a strategy profile corresponding to a type different from the real one. Thus, the regret is upper bounded by:

$$\begin{aligned} R_N(\mathfrak{U}) &= \mathbb{E}\left[\sum_{n=1}^{N} l_n\right] - L^* N \\ &= \mathbb{E}\left[\sum_{n=1}^{N} l_n - L^*\right] = \sum_{k=1}^{K} \Delta L_k \mathbb{E}[T_k(N)], \end{aligned}$$

where we recall that:

- $T_i(N) = \sum_{n=1}^{N} \mathcal{I}\{A_{k_n} = A_k\}$ is the number of times we played the best response $\boldsymbol{\sigma}_D^*(A_k)$ to attacker A_k;

- $\Delta L_k = \sum_{m=1}^{M} \sigma_A(\boldsymbol{\sigma}_D^*(A_k))_m v_m(1 - \sigma_D^*(A_k)_m) - L^*$ is the expected regret of playing the best response to attacker A_k when the real attacker is A.

Each round in which the algorithm selects a profile s.t. the best response is not equal to the one of A_{k^*} we are getting some regret.

Let us define variables $B_{k,n}$ and $B_{k^*,n}$ denoting the belief we have for the possible attacker A_k and of the real attacker A, respectively, of the action played by the real attacker A at turn n. Moreover, let $b_{kj,t} := \mathbb{E}_{\sigma_D^*(A_j)}[B_{k,t}]$ be the expected value of the belief we get for attacker A_k when we are best responding to A_j and the true type is $A_{k^*} \neq A_k$ at round t. Note that $b_{kj,t} < b_{k^*j,t}, \forall j$, since Δb_k is positive.

For each profile $A_k \neq A_{k^*}$, we have:

$$\mathbb{E}[T_k(N)] \leq \sum_{n=1}^{N} \mathbb{E}\left[\mathcal{I}\left\{\prod_{t=1}^{n} B_{k,t} \geq \prod_{t=1}^{n} B_{k^*,t}\right\}\right] \tag{A.17}$$

$$\leq \sum_{n=1}^{N} \mathbb{E}\left[\mathcal{I}\left\{\sum_{t=1}^{n} \ln(B_{k,t}) \geq \sum_{t=1}^{n} \ln(B_{k^*,t})\right\}\right] \tag{A.18}$$

$$= \sum_{n=1}^{N} \mathbb{P}\left(\frac{\sum_{t=1}^{n} \ln(B_{k,t})}{n} \geq \frac{\sum_{t=1}^{n} \ln(B_{k^*,t})}{n}\right) \tag{A.19}$$

$$= \sum_{n=1}^{N} \mathbb{P}\left(\frac{\sum_{t=1}^{n} \ln(B_{k,t})}{n} - \frac{\sum_{t=1}^{n} \ln(b_{kj_t,t})}{n} - \frac{\sum_{t=1}^{n} \ln(B_{k^*,t})}{n} + \right.$$
$$\left. + \frac{\sum_{t=1}^{n} \ln(b_{k^*j_t,t})}{n}\right) \geq \underbrace{\left(\frac{\sum_{t=1}^{n} \ln(b_{k^*j_t,t})}{n} - \frac{\sum_{t=1}^{n} \ln(b_{kj_t,t})}{n}\right)}_{\geq \Delta b_k} \tag{A.20}$$

$$\leq \sum_{n=1}^{N} \mathbb{P}\left(\frac{\sum_{t=1}^{n} \ln(B_{k,t})}{n} - \frac{\sum_{t=1}^{n} \ln(b_{kj_t,t})}{n} - \frac{\Delta b_k}{2} - \right.$$
$$\left. - \frac{\sum_{t=1}^{n} \ln(B_{k^*,t})}{n} + \frac{\sum_{t=1}^{n} \ln(b_{k^*j_t,t})}{n} - \frac{\Delta b_k}{2} \geq 0\right) \tag{A.21}$$

$$\leq \underbrace{\sum_{n=1}^{N} \mathbb{P}\left(\frac{\sum_{t=1}^{n} \ln(B_{k,t})}{n} \geq \frac{\sum_{t=1}^{n} \ln(b_{kj_t,t})}{n} + \frac{\Delta b_k}{2}\right)}_{R_1} + \\ + \underbrace{\sum_{n=1}^{N} \mathbb{P}\left(\frac{\sum_{t=1}^{n} \ln(B_{k^*,t})}{n} \leq \frac{\sum_{t=1}^{n} \ln(b_{k^*j_t,t})}{n} - \frac{\Delta b_k}{2}\right)}_{R_2}, \tag{A.22}$$

where j_t is the index of the attacker A_{j_t} we selected at round t and we defined $\Delta b_k := \min_{j|A_j \in \mathcal{A}} \ln(b_{k^*j,t}) - \ln(b_{kj,t})$, i.e., the minimum w.r.t. the best response for the available attackers of the difference between the expected value of the loglikelihood of attacker A_{k^*} and A_k if the true profile is A_{k^*}. Equation (A.19) has been obtained from Equation (A.18) since $\mathbb{E}\left[\mathcal{I}\left\{\cdot\right\}\right] = \mathbb{P}\left(\cdot\right)$ while Equation (A.20) has been computed from Equation (A.19) adding $\left(\frac{\sum_{t=1}^{n} \ln(b_{k^*j_t,t})}{n} - \frac{\sum_{t=1}^{n} \ln(b_{kj_t,t})}{n}\right)$ to both l.h.s. and r.h.s. of the inequality. We would like to point out that Δb_k does not depend on t since the distribution of $B_{k,t}$ and $B_{k^*,t}$ is the same over rounds.

Let us focus on R_1. We use the McDiarmid inequality [78] to bound the probability that the empirical estimate of the loglikelihood expected value is higher than a certain upper bound as follows:

$$\begin{aligned} R_1 &= \sum_{n=1}^{N} \mathbb{P}\left(\frac{\sum_{t=1}^{n} \ln(B_{k,t})}{n} \geq \frac{\sum_{t=1}^{n} \ln(b_{kj_t,t})}{n} + \frac{\Delta b_k}{2}\right) \\ &\leq \sum_{n=1}^{\infty} \mathbb{P}\left(\frac{\sum_{t=1}^{n} \ln(B_{k,t})}{n} \geq \frac{\sum_{t=1}^{n} \ln(b_{kj_t,t})}{n} + \frac{\Delta b_k}{2}\right) \\ &\leq \sum_{n=1}^{\infty} \exp\left\{-\frac{(\Delta b_k)^2 n}{2\lambda_k^2}\right\} \leq \frac{2\lambda_k^2}{(\Delta b_k)^2}, \end{aligned}$$

where we exploited $\sum_{x=1}^{\infty} e^{-\kappa x} \leq \frac{1}{\kappa}$. We define

$$\lambda_k := \max_{m \in \mathcal{M}} \max_{\boldsymbol{\sigma} \in \mathcal{S}} \ln(\sigma_{A_k}(\boldsymbol{\sigma})_m) - \min_{m \in \mathcal{M}} \min_{\boldsymbol{\sigma} \in \mathcal{S}} \ln(\sigma_{A_k}(\boldsymbol{\sigma})_m) \mathcal{I}\left\{\sigma_{A_k}(\boldsymbol{\sigma})_m \neq 0\right\}$$

as the range where the beliefs realizations lie (excluding realizations equal to zero which ends the exploration of a profile), where we used the fact that $\mathbb{E}[B_{k,t}] = b_k \forall k, t$ and $\mathcal{S} := \cup_k \boldsymbol{\sigma}_D^*(A_k)$ is the set of the available best response to the attackers profile.

A similar reasoning can be applied to R_2 getting an upper bound of the following form:

$$R_2 \leq \frac{2\lambda_{k^*}^2}{(\Delta b_k)^2}.$$

The regret becomes:

$$R_N(\mathfrak{U}) = \sum_{i=1}^{K} \Delta L_k \mathbb{E}[T_k(N)] \leq \sum_{i=1}^{K} \Delta L_k \left(\frac{2\lambda_k^2}{(\Delta b_k)^2} + \frac{2\lambda_{k^*}^2}{(\Delta b_k)^2} \right) \leq$$
$$\leq \sum_{i=1}^{K} \frac{2(\lambda_k^2 + \lambda_{k^*}^2)\Delta L_k}{(\Delta b_k)^2},$$

which concludes the proof. □

APPENDIX B

Notation Table

We report in Table B.1 the symbols used in Chapters 4 to 7.

Appendix B. Notation Table

	Symbol	Meaning
Basic model	$\mathcal{A}$	Attacker
	$\mathcal{D}$	Defender
	G	Graph
	V	Set of vertices
	v	Vertex
	v_i	i-th vertex
	E	Set of edges
	(v, v')	Edge
	$\omega^*_{v,v'}$	Temporal cost (in turns) of the shortest path between v and v'
	T	Set of targets
	t	Target
	t_i	i-th target
	$\pi(t)$	Value of target t
	$d(t)$	Penetration time of target t
Signals	S	Set of signals
	s	Signal
	s_0	No signals have been generated
	p	Function specifying the probability of having the system generating signal s given that target t has been attacked
	α	Missed detection rate
	$T(s)$	Targets having a positive probability of raising s if attacked
	$S(t)$	Signals having a positive probability of being raised if t is attacked
	$\perp$	No signals have been generated
	$\triangle$	No targets are under attack
Routes, strategies and utilities	H^l	Space of graph walks with maximum length upper bounded by l
	Σ^l	Space of patrolling strategies conditioning over H^l
	r	Route, i.e., sequence of potentially non-adjacent vertices
	r_i	i-th route
	$r(i)$	i-th element visited along route r
	R	Set of routes
	$U_{\mathcal{D}}(c_i, t_i)$	Defender's utility given a cycle c and a target t
	$U_{\mathcal{A}}(c_i, t_i)$	Attacker's utility given a cycle c and a target t
	$\sigma^{\mathcal{D}}$	Defender's strategy
	$\sigma^{\mathcal{D}}_p$	Defender's strategy when s_0 is generated
	$\sigma^{\mathcal{D}}_a$	Defender's strategy when some signal $s_i (i > 0)$ has been generated
	$\sigma^{\mathcal{A}}$	Attacker's strategy
	$G^{\sigma^{\mathcal{D}}_p}$	Support graph of a Defender's patrolling strategy
	$V^{\sigma^{\mathcal{D}}_p}$	Set of vertices of the support graph
	$E^{\sigma^{\mathcal{D}}_p}$	Set of edges of support graph
	g_v	Value of the game (utility of $\mathcal{A}$)
	$A(r(i))$	Time needed by $\mathcal{D}$ to visit $r(i)$ starting from $r(0)$
	$T(r)$	Set of targets covered by route r
	$c(r)$	Temporal cost (in turns) associated to r

Table B.1: *Symbols' table.*