UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

ADVERSARIAL DETECTION GAMES IN NETWORK SECURITY APPLICATIONS WITH IMPERFECT AND INCOMPLETE INFORMATION

TESIS DOCTORAL

JUAN PARRAS MORAL

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

2020

DEPARTAMENTO DE SISTEMAS, SEÑALES Y RADIOCOMUNICACIONES

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

ADVERSARIAL DETECTION GAMES IN NETWORK SECURITY APPLICATIONS WITH IMPERFECT AND INCOMPLETE INFORMATION

Autor:

Juan Parras Moral

Máster en Ingeniería de Telecomunicación

Director:

Santiago Zazo Bello

Doctor en Ingeniería de Telecomunicación

Catedrático de Universidad del Dpto. de Señales, Sistemas y Radiocomunicaciones

Universidad Politécnica de Madrid

2020

TESIS DOCTORAL

ADVERSARIAL DETECTION GAMES IN NETWORK SECURITY APPLICATIONS WITH IMPERFECT AND INCOMPLETE INFORMATION

AUTOR: Juan Parras Moral

DIRECTOR: Santiago Zazo Bello

Tribunal nombrado por el Magfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid, el día X de X de 2020.

PRESIDENTE:

SECRETARIO:

VOCAL:

VOCAL:

VOCAL:

SUPLENTE:

SUPLENTE:

Realizado el acto de defensa y lectura de la Tesis el día X de X de 2020.

En la E.T.S de Ingenieros de Telecomunicación.

Calificación:

EL PRESIDENTE:

EL SECRETARIO:

LOS VOCALES:

Abstract

This Ph.D. thesis deals with security problems in Wireless Sensor Networks. As the number of devices interconnected grows, the amount of threats and vulnerabilities also increases. Namely, in this thesis, we focus on two family of attacks: the backoff attack, which affects to the multiple access to a shared wireless channel, and the spectrum sensing data falsification attack, which arises in networks which try to make a decision about the state of a spectrum channel cooperatively.

First, we use game theory tools to model the backoff attacks. We start by introducing two different algorithms that can be used to learn in discounted repeated games. Then, we motivate the importance of the backoff attack by showing analytically its effects on the network resources, which are not shared evenly as the attacking sensors receive a larger part of the network throughput. Afterwards, we show that the backoff attack can be modeled, under certain assumptions, using game theory tools, namely, static and repeated games, and provide analytical solutions and also algorithms to learn these solutions.

A problem that arises for the defense mechanism is that it is possible that the agent is able to adapt to it. We then explore what happens if the agent knows the defense mechanism and acts in such a way that it is able to exploit the defense mechanism without being discovered. As we show, this is a significant threat to both attacks studied in this work, as the agent is able to successfully exploit the defense mechanism: in order to alleviate this attack, we propose a novel detection framework that is successful against such attack.

However, we can even develop attack strategies that do not need the agent to know the defense mechanism: by means of reinforcement learning tools, it is able even to exploit a possibly unknown mechanism simply by interacting with it. Hence, these attack strategies are a significant threat against current defense mechanisms. We finally develop a defense mechanism against such intelligent attackers, based on inverse reinforcement learning tools, which is able to successfully mitigate the attack effects.

Resumen

Esta tesis trata con problemas de seguridad en redes de sensores inalámbricas. Debido a que el número de dispositivos interconectados crece, la cantidad de amenazas y vulnerabilidades también lo hace. Concretamente, en esta tesis nos centramos en dos familias de ataques: los ataques de backoff, que afectan al acceso múltiple a un canal inalámbrico compartido, y el ataque de falsificación de datos de sensado de espectro, que surge en redes que tratan de tomar una decisión respecto al estado del canal de manera cooperativa.

En primer lugar, usamos herramientas de teoría de juegos para modelar el ataque de backoff. Comenzamos introduciendo dos algoritmos diferentes que se pueden usar para aprender juegos repetidos con descuento. Luego, desarrollamos la importancia del ataque de backoff al mostrar de forma analítica sus efectos sobre los recursos de la red, ya que provoca que estos no sean distribuidos de manera uniforme, debido a que los sensores atacantes obtienen una porción mayor del ancho de banda de la red. Después, mostramos que el ataque de backoff puede ser modelado, bajo ciertas asunciones, usando herramientas de teoría de juegos; concretamente, juegos estáticos y repetidos, y proporcionamos soluciones analíticas y algoritmos que aprenden estas soluciones.

Un problema para el mecanismo de defensa es que es posible que el agente se pueda adaptar. Así que pasamos a explorar qué ocurre si el agente conoce el mecanismo de defensa y actúa de tal manera que es capaz de atacarlo sin ser descubierto. Como mostramos, esta es una amenaza significativa para ambos ataques estudiados en este trabajo, ya que el agente es capaz de tener éxito burlando al sistema de defensa. Para mitigar los efectos de este ataque, proponemos un modelo novedoso de detección que tiene éxito frente a estos ataques.

Sin embargo, podemos incluso desarrollar estrategias de ataque que no requieren que el agente conozca el mecanismo de defensa: usando herramientas de aprendizaje por refuerzo, el atacante es capaz de atacar un mecanismo de defensa posiblemente desconocido simplemente interactuando con él. De modo que estas estrategias son una amenaza significativa contra los sistemas de defensa actuales. Finalmente, desarrollamos un mecanismo de defensa contra estos ataques inteligentes, basado en aprendizaje por refuerzo inverso, que es capaz de mitigar con éxito los efectos del ataque.

Isaac Newton dijo que si llegamos lejos, es porque avanzamos a hombros de gigantes. Creo que esta frase es aplicable, no solamente al avance de la ciencia en general, sino al progreso de cada persona en particular, debido a que vivimos en sociedad y nos influimos mutuamente. De modo que esta tesis no sólo es el producto de mi esfuerzo, sino también el producto del esfuerzo de otras personas que han desempeñado un papel fundamental en mi trayecto hasta el día de hoy.

Me acuerdo con especial cariño de muchos de mis profesores, desde parvulario hasta la misma Universidad, que han invertido horas de esfuerzo y dedicación en formarme como alumno. Por mencionar algunos, está Andrés Galindo, quien vio mi curiosidad insaciable durante el colegio y me ayudó a satisfacerla a través de los libros. En el instituto, recuerdo a profesores como Antonio Pulgar, Juan Sánchez o Don Pedro, cuyas clases de Matemáticas y Física definitivamente me enamoraron de ambas materias. Y ya en la Universidad, una mención especial se merece Pedro Vera, con quien me inicié en el camino de la investigación hace ocho años y aquí sigo hasta el día de hoy.

Una vez empecé a trabajar en la UPM, cabe destacar el papel que han tenido mis compañeros de laboratorio durante estos años: Javier, Sergio, los dos Jorges, Carlos, Dani, David, Ignacio y Belén, compañeros de trabajo, de fatigas y de alegrías. Asimismo, no puedo olvidarme de los compañeros de la Universidad de Lincoln, especialmente de Geri, Max y Riccardo. Y por supuesto, la persona que más horas, trabajo e ilusión le ha puesto a esta tesis: mi tutor Santiago Zazo, sin cuya confianza, dedicación y consejo, este trabajo no hubiera llegado a existir.

Es evidente que hay muchas más personas que han tenido también un papel hasta llegar a este punto: compañeros de piso, de clase y de prácticas, como Jenni, Dani o Pedro; la gente de Torredelcampo y de Buempa, y también la gente de la Resi, con quienes tantos momentos juntos hemos vivido; y otros muchos que me dejo por el camino. A todos vosotros: simplemente gracias por vuestro trabajo y dedicación.

Finalmente, están aquellos que me han estado apoyando en todo momento: mi familia. A papá, mamá, Eli y Lidia: gracias por vuestro apoyo, las risas y los múltiples sacrificios que hacen, no sólo que esta tesis haya sido posible, sino que sea quien soy. Gracias también a Eunice: tú has sido un apoyo y acompañamiento fundamental durante este trayecto. Y por supuesto, como decía Johann Sebastian Bach, *Soli Deo Gloria*.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

AS	Attacking Sensor
BF	Bayes Factor
CA	Communicate & Agree
CDF	Cumulative Distribution Function
CE	Correlated Equilibrium
CR	Cognitive Radio
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSS	Cooperative Spectrum Sensing
DLA	Deep RL Attacker
DNN	Deep Neural Network
DP	Dynamic Programming
DQN	Deep Q-Networks
DRQN	Deep Recurrent Q-Networks
EWSZOT	Enhanced Weighted Sequential Zero/One Test
FC	Fusion Center
FIM	Fisher Information Matrix
FNN	Feedforward Neural Network
GAIL	Generative Adversarial Imitation Learning
GAN	Generative Adversarial Network
GS	Good Sensor
GT	Game Theory

HT Hypothesis Test

IRL Inverse Reinforcement Learning

LEWIS LEarn WIth Security

LSTM Long-Short Term Memory

MAC Medium Access Control

MAL Multi-Agent Learning

MDP Markov Decision Process

MEP Maximum Entropy Principle

MME Mean-based Mean Embedding

MSE Mean Squared Error

NE Nash Equilibrium

NNME Neural Network Mean Embedding

NN Neural Network

OCSVM One Class SVM

PE Policy Evaluation

PG Policy Gradient

PI Policy Iteration

POMDP Partially Observable Markov Decision Process

POSG Partially Observable Stochastic Game

PRNG Pseudo-Random Number Generator

RG Repeated Game

RL Reinforcement Learning

RM Regret Matching

RNN Recurrent Neural Network

SG Stochastic Game

SOO Stochastic Optimistic Optimization

SPE Subgame Perfect Equilibrium

SPRT Sequential Probability Ratio Test

SSDF Spectrum Sensing Data Falsification

SVM Supporting Vector Machine

TRPO Trust Region Policy Optimization

UNR Unforgiving Nash Reversion

VI Value Iteration

WSN Wireless Sensor Network

Chapter 1

Introduction

1.1 Motivation

Our world has assisted in the last years to a spectacular development and evolution of the telecommunication and networking technologies. We are assisting to an unprecedented increase of the services offered through these networks, as well as to a massive growth in the number of devices connected. This development affects to all the society, as not only businesses are obtaining new services, but also the individuals. One concept that is frequently used to refer to this tendency in the growth of interconnected devices is the term Internet of Things, which denotes the idea of connecting as many devices as possible to the Internet network. The irruption of new application and devices causes that the number of interconnected devices does not stop growing.

Thus, it is not a surprise that a lot of research effort is spent in different aspects related to the Internet of Things [133], such as network protocols [166], [254], [248], efficient implementation architectures [29], [165] or concrete applications, such as industry related ones [22] or smart cities [11]. These works show that a key concept related to the Internet of Things are Wireless Sensor Networks (WSNs), which are wireless networks of low capabilities devices, known as sensors, which are designed for a specific task. The authors of [191] propose five main types of WSNs, namely, terrestrial, underground, underwater, mobile and multimedia, and mention that these WSNs find applications in many different areas, such as health, smart cities, smart grid, intelligent transportation systems, farming, remote monitoring, security and surveillance of a certain area, animal tracking or disaster management. WSNs are continuously growing, and their applications, as well as their deployment, is expected to keep on growing in the future.

However, this massive growth and evolution in WSNs has also meant that new vulnerabilities and attacks against WSN mechanisms arise. Note that WSNs can be the target of many attacks due to the limited capabilities of the sensors [72], [257]. Hence, it is not a surprise that security is one of the most active areas of research in the field of WSN, as many recent works show: [253], [208], [244], [5], [72], [132], [199], [220]. Security related topics become of utmost importance given the fact that both the number of devices interconnected and the set of WSN applications are not expected to stop growing. However, in spite of the great effort spent in this research, most of the security solutions included in communication protocols and standards used in WSN are still at a proof-of-concept level according to [220].

A powerful mathematical framework that can be used to model and address many security issues that arise in WSN is Game Theory, which is the branch of mathematics which specializes in studying the conflict among different agents. This theory can be considered mature, with many reference works such as [76], [19], [146]

or [150]. We note that the idea of applying game theory tools to security problems is not new, as there are many works on this topic, such as [8], [193], [148], [135], [68] and [145], to mention some. However, in many cases, game theory based approaches have a limited impact, because realistic, complex models easily become computationally intractable.

We also note that a field that has experienced a significant growth in the last few years is the field of Deep Learning. As the availability of big amounts of data and the evolution of the computational power available to researchers has grown, the number of advances and applications in the field has experienced a significant increase [81]. The problem of WSN security also takes advantage of these recent advances in Deep Learning [202], [244]. A family of algorithms of special interest to us is the one known as Reinforcement Learning, which is inspired by biology and tries to make an agent interacting with a dynamical system learn the optimal sequence of actions, that is, the sequence of actions that provides it the highest reward. A key milestone in the field was the work in [154], in which the researchers were able to train a computer to play a set of Atari games achieving better performance than a human; after this seminal work, many others have come that have significantly expanded the field. We note that Reinforcement Learning tools have been used in security settings, such as routing, data latency, path determination, duty cycle management, QoS provisioning or resource management [9].

The present thesis lies at the intersection of the three mentioned fields, as we address security problems that arise in WSN using Game Theory and Deep Learning tools. Namely, we model security situations in which the attacker and/or the defense mechanism make decisions sequentially, and these decisions have an impact on whether the attack is successful or not. We start modeling our security attack problems in WSN using game theory tools, which provide analytical solutions in controlled environments. However, as we increase the complexity of the security setup, we note that game theory tools are too expensive computationally, and hence, we turn to Deep Reinforcement Learning tools in order to obtain tractable solutions to our security problems.

1.2 Thesis overview

Let us now proceed to provide an overview of the main topics studied in the thesis. In this Chapter 1, we provide a brief motivation and introduction to the topics that will be covered in the rest of the thesis. Chapter 2 is devoted to present the mathematical framework in which our thesis is based. The main topics presented are related to the control theory and game theory fields. Control theory tries to obtain the sequence of actions that an agent has to follow in order to optimize a certain outcome when the agent interacts with a dynamical system. This framework is a generalization of optimization theory, as the role of time is key in order to obtain optimal control sequences. There are many possible control cases, depending on whether the agent has a perfect observation of the state of the dynamical system or only a partial observation, and depending on whether the agent knows how the system evolves with time or not. Game theory generalizes control theory when there are several agents that interact among them: in this case, note that the actions of each agent affect the rest of the agents, and hence, each agent solves a control problem coupled to the rest of agents. These frameworks are presented and discussed in depth in Chapter 2.

Chapter 3 is devoted to study a concrete setup, known as repeated games, which are of interest as a first approach to WSN security, as they allow taking into account the effect of time. We note here that, when optimizing a sequence of actions, we may consider that the outcomes obtained by the agent need to be discounted, that is, present outcomes matter more than future ones. This makes sense in WSN setups because of their volatility, but surprisingly it is a case which has not been thoroughly addressed in current literature.

Hence, in Chapter 3 we start by studying two important effects that discounting introduces in these situations, and then present two algorithms specifically designed to obtain solutions in such environments.

In Chapters 4-7, we apply the tools developed in Chapter 2 and 3 to two concrete WSN attack situations. The first one, known as backoff attack, arises in a CSMA/CA multiple access situation in which several sensors try to communicate with a central node without colliding. A widely used mechanism that can be used in this situation is the backoff mechanism, by which the sensors defer their transmission a certain time so that the collision probability is minimized. However, an attacker may ignore the backoff mechanism in order to get an advantage over the rest of sensors: this situation is known as backoff attack and is the central problem that we address in this thesis. There are several possible variations of the backoff problem, but in this work we address three, depending on two assumptions. The first assumption is that the defense mechanism is able to instantaneously detect when an agent deviates from a previously negotiated strategy, and the second is that the defense mechanism is able to instantaneously detect any deviation from the backoff procedure. Both assumptions are considered in Chapter 4; in Chapter 5 we show what happens when the first assumption is dropped, and finally, Chapters 6 and 7 deal with the case in which none of these assumptions hold. Note that we refer indistinctly to this case as backoff or CSMA/CA attack.

A second WSN security problem that we include consists in using a WSN in order to detect whether a spectral channel is free or it is being used to transmit. This situation is known as Cooperative Spectrum Sensing (CSS). Here, an attacker may send false channel reports in order to mislead the decision about the channel state: this attack is known as Spectrum Sensing Data Falsification (SSDF) attack. It is possible that each sensor sends as a report a binary variable which indicates whether it senses the channel free or not: this case is known as hard fusion and is studied in Chapter 5 and 6. It is also possible that each sensor sends the energy level they measure: in this case, the dimensionality of the problem grows as the report now is a continuous variable. This case is known as soft fusion, and is addressed in Chapter 6. We refer to these problems indistinctly as CSS or SSDF attacks.

In Chapter 4, we study the backoff attack using the two assumptions explained before. We first present analytical results on the impact that a backoff attack has on the network distribution of resources, and we conclude that it causes that some sensors have access to more resources than others. In order to overcome this situation, we use Game Theory tools to model this scenario. We start by modeling the CSMA/CA game using static game theory, and then use dynamic game theory tools, namely, repeated games. We provide analytical solutions for the two player cases, i.e., for the case in which we have a single attacker and the defense mechanism, and also provide algorithms to solve the games when there are more than two players, where we use the two algorithms developed in Chapter 3.

An important assumption in the backoff attack of the Chapter 4 is that the defense mechanism is able to instantaneously detect when an attacker deviates from a previously negotiated strategy, which in game theory language is known as mixed action observability. This assumption needs not hold in real environments, so we drop it in Chapter 5. We model this new situation using detection theory tools, both including and excluding prior information, and we assume that the agent is able to perfectly observe the state of the defense mechanism and has complete information about this defense mechanism, i.e., it knows which defense mechanism is being used. This has very important consequences, as we are able to derive an optimal attack strategy against the defense mechanism. To counter this attack strategy, we develop a novel detection tool that successfully detects the attack. The advances of this Chapter introduce significant changes in the CSMA/CA setup used in the Chapter 4, which we assess empirically. We also show that the attack strategy developed in this Chapter can be used to exploit a hard fusion defense mechanism in a CSS setup.

Chapter	CSMA/CA	CSS	Player	Information	Observation (A/S)	Behavior
4	Yes	No	Attack Defense	Complete Complete	Mixed / - Mixed / -	Static Static
5	Yes	Yes	Attack Defense	Complete Incomplete	- / State Realization / -	Dynamic Static
6	Yes	Yes	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / -	Dynamic Static
7	Yes	No	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / Observation	Dynamic Dynamic

Table 1.1 Table comparing the different setups used in Chapters 4-7. CSMA/CA. i.e., the backoff attack, and CSS, i.e., the SSDF attack, denote whether each of these setups is used in the Chapter. Information denotes whether each player knows the target of the other player (Complete) or not (Incomplete). Observation refers to what each agent observes with respect to the actions / states of the other players: regarding actions, they observe the mixed actions or the actions realizations, and regarding states, they observe the state or an observation of the rest of players: this is related to having perfect or imperfect information. Behavior refers to whether the player adapts its behavior with time or not.

However, the attacker in Chapter 5 had complete information of the defense mechanism, which may not hold in real environments. Hence, in Chapter 6 we drop that assumption: now the agent does not have complete information about the defense mechanism. Moreover, the agent does not perfectly observe the state of the defense mechanism, but it has a partial observation of it. In this situation, we develop an attack strategy that is able to successfully exploit an unknown defense mechanism simply by interacting with it. We test our ideas in three different setups: a hard and a soft fusion CSS environments, and also in a backoff attack, in which we drop the second assumption, that is, that the defense mechanism is not able to instantaneously detect which sensors are not following the prescribed backoff mechanism. Actually, our proposed attacker is a real threat against current WSN defense mechanisms: we show that it is able to coordinate several attackers with a partial observation of an unknown defense mechanism and successfully exploit it.

It is important noting that Chapters 5 and 6 allowed the attacker to have a dynamic behavior, while the defense mechanism was considered static, that is, it did not change its behavior with time. Thus, they were asymmetric situations between the attackers and the defense mechanism, in which they have different capabilities and also they do not have complete information about the other players. In Chapter 7, we break that asymmetry as we introduce an intelligent defense mechanism that is able to face the intelligent attacker presented in Chapter 6, whose performance is successfully tested on the same backoff attack setup of the Chapter 6.

Chapter 8, finally, draws some conclusions of the thesis and also discuss several future research lines that could arise from this work.

In Table 1.1, we include a summary on the different setups that we study in Chapters 4-7, in order to facilitate the ideas of the reader. This table is repeated along the text for the sake of clarity, so that the reader is oriented regarding the concrete setup under study in each of the Chapters of the thesis.

1.2.1 Publications associated to the thesis

Most of this thesis has already been published in 5 international journals and one international conference, and we note that the rest of the thesis is currently under review for publication. We now include a list that summarizes these publications, ordered by Chapter:

- In Chapter 3, we present two algorithms specifically designed for learning repeated games using a discounted scheme. The CA algorithm, which negotiates equilibria in these situations in a fully distributed way, has been published as: Parras, J., and Zazo, S, *A distributed algorithm to obtain repeated games equilibria with discounting*, Applied Mathematics and Computation, [180], whose journal metrics for 2018 are: IF: 3.092, Rank Q1 (94.685 in Mathematics: applied). The other algorithm, LEWIS, which is designed for learning such games in an online fashion, is currently under review as: Parras, J., & Zazo, S, *Learning to play discounted repeated games with worst case bounded payoff*, Journal of Machine Learning Research, whose journal metrics for 2018 are: IF: 4.091, Rank Q1 (80.224 in Computer science: artificial intelligence).

- In Chapter 4, we deeply study the backoff attack effects and model it using static and repeated game theory tools. The effects of the attack, as well as the static game solutions, have been published as: Parras, J., & Zazo, S., *Wireless Networks under a Backoff Attack: A Game Theoretical Perspective*, Sensors, [175], whose journal metrics for 2018 are: IF: 3.031, Rank Q1 (76.23 in Instruments and Instrumentation). Also, the repeated game solutions presented in this Chapter have been published as: Parras, J., & Zazo, S., *Repeated game analysis of a CSMA/CA network under a backoff attack*, Sensors, [177], whose journal metrics for 2018 are: IF: 3.031, Rank Q1 (76.23 in Instruments and Instrumentation).

- In Chapter 5, we present an optimal attack against a sequential test and a novel defense mechanism that can successfully detect such attack: these results are published as: Parras, J., & Zazo, S., *Using one class SVM to counter intelligent attacks against an SPRT defense mechanism*, Ad-hoc networks, [179], whose journal metrics for 2018 are: IF: 3.490, Rank Q1 (77.097 in Computer science: information systems). We also include an efficient sequential test that can incorporate prior information, and which is shown to be fast and accurate; it is published as: Parras, J., & Zazo, S., *Sequential Bayes factor testing: a new framework for decision fusion*, in the 20th International workshop on Signal processing advances in wireless communications (SPAWC), [178].

- In Chapter 6, we present a thorough mathematical study of a hard fusion CSS problem and compare several methods to obtain attack strategies against it; this work was published as: Parras, J., & Zazo, S., *Learning attack mechanisms in Wireless Sensor Networks using Markov Decision Processes*, Expert Systems with Applications, [176], whose journal metrics for 2018 are: IF: 4.292, Rank Q1 (82.33 in Computer science: artificial intelligence, 81.70 in Engineering, electrical and electronic). The rest of the Chapter 6, which includes studying the case in which there are several agents with partial observation in the soft fusion CSS problem and the backoff attack, is currently under review as: Parras, J., Hüttenrauch, M., Zazo, S., & Neumann, G., *Deep reinforcement learning for attacking wireless sensor networks*, ACM Transactions on Intelligent Systems and Technology, whose journal metrics for 2018 are: IF: 2.861, Rank Q2 (63.534 in Computer science: artificial intelligence, 66.774 in Computer science: information systems).

- Finally, Chapter 7 is also currently under review for publication as: Parras, J., & Zazo, S., *Inverse Reinforcement Learning: a new framework to mitigate an intelligent backoff attack*, IEEE Transactions on control of Network Systems, whose journal metrics for 2018 are: IF: 4.802, Rank Q1 (81.45 in Automation and Control systems, 92.58 in Computer Science: Information Systems).

Chapter 2

Mathematical background

2.1 Introduction

This thesis deals with security problems that arise in WSNs when sensors make decisions. The existing communication protocols clearly define which actions a sensor has to choose so that the network functions properly. However, from a security point of view, we cannot assume that all sensors will follow these procedures, as there might be sensors which intentionally deviate from the prescribed actions in order to take advantage of the network. We denote such intentional deviations from the actions prescribed by the communication protocols as attacks, and the sensors that deviate are Attacking Sensors (ASs), in contrast to the Good Sensors (GSs) which are those that follow the prescribed actions. Hence, in a WSN there might be GSs and ASs, and we seek to study the impact that the actions chosen by the ASs have on the whole network.

By the own nature of WSN protocols, we need to take into account the time. The actions are taken in stages, and current actions affect the future performance of the network. Moreover, we focus only in discrete time, as most problems in WSN can be studied under this model. As we will see, the history of actions, which is the set of past actions, plays a central role in every defense mechanism, and thus, the main objective of the ASs is not optimizing actions in isolation, but obtaining a sequence of actions that is optimal under some criterion, as maximizing a certain reward. As we will see, this has a deep impact on the mathematical framework needed to study the interactions between ASs and the network: ASs are studied using the framework of control, which is a generalization of the optimization framework which takes into account the effect of time. While optimization outcomes a vector of optimal values for the optimization variables, control outcomes a policy, a law that defines which is the optimal action for each time step. The entities that take actions in the control procedure are known as agents: in our case, agents are either the ASs or the defense mechanism of the WSN.

Thus, we consider that WSN are dynamical systems. In each time step, a dynamical system is defined by a state which contains the information needed to optimize the actions. When an agent takes an action, the dynamical system transients to a different state with a certain probability. We present the case in which the agent has access to the state in Section 2.2. However, there are cases in which the agent only has access to a noisy or partial observation of the state: this case is presented in Section 2.3.

Control theory is used to optimize the sequence of actions of a single agent. However, this needs not be the case in our WSN environment, as there might be more than a single AS. Note that having several ASs increases the possible attack policies, as ASs may act coordinately in order to take an advantage of the WSN. A

mathematical model proposed for this situation, in which we have several agents with a common objective, is the Swarm model, which we present in Section 2.4.

Of course, it may happen that the network has a defense mechanism that tries to detect and apply countermeasures to the actions taken by the ASs. In this case, note that we have two kinds of agents: the ASs and the defense mechanism, each of them having a different set of actions and different objectives. Furthermore, the outcome that each agent obtains depends not only on its own actions, but also on the actions of the rest of agents. Thus, this situation gives rise to coupled control problems between the agents, and it is studied by making use of game theory tools, which we present in Section 2.5.

2.2 Markov Decision Processes

In order to model our dynamical system, we choose to employ the Markov Decision Process (MDP) framework, as it is a flexible, well-studied and widely used model to describe such systems [25], [26], [218]. In this Section, we introduce this model and several ways to solve it, when the transition function between states is both known and unknown. We also introduce the inverse problem, in which we try to obtain the reward function that an agent is optimizing when we are given the policy of the agent.

2.2.1 Markov Decision Process

A Markov Decision Process (MDP) is defined as follows [25], [218]:

Definition 1 (Markov Decision Process)**.** *An MDP is a 5-tuple $\langle S, A, P, R, \gamma \rangle$ where:*

- *S is the state set, containing all the possible states $s \in S$ of the dynamical system.*
- *A is the action set, containing all the possible action vectors $a \in A$ that the agent can use to interact with the dynamical system.*
- *$P : S \times S \times A \to [0,1]$ is the transition probability function in case that the states are discrete and $P : S \times S \times A \to [0, +\infty)$ in case that the states are continuous, where $P(s^{n+1}|s^n, a^n)$ denotes the probability of transitioning to state s^n given that the agent is in state s^n and takes action a^n. Note that unless explicitly indicated, we assume discrete states. The superscript indicates the time step, where n indicates the current time step and $n+1$ the next time step, $n \in \{0, 1, 2, 3, ..., N-1\}$. We consider that P is stationary, that is, it does not depend on n.*
- *$R : S \times A \to \mathbb{R}$ is the reward function, where $r(s^n, a^n)$ denotes the reward that the agent receives when it is in state s^n and takes action a^n. In case that $n = N$, where N is the terminal time step, s^N is a terminal state of the system, there are no more actions to be taken, and $r(s^N)$ is the terminal state reward. We assume that R is bounded and stationary.*
- *$\gamma \in (0,1)$ is a discount factor, used to obtain the total reward for the agent.*

In general, MDPs can be of finite or infinite horizon, depending on whether the final time N is finite of infinite.

In a general dynamical system, the probability of transitioning from one state to another depends on the previous history, that is, the whole set of past states and actions. Note that the history set for time index n is:

$$\mathscr{H}^n \equiv \prod_{j=0}^{n} A^j \times S^j, \tag{2.1}$$

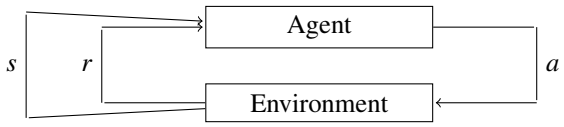


Fig. 2.1 MDP basic interaction scheme.

where the history set increases with the time index n. Thus, in general, the probability transition function would be $P(s^{n+1}|s^n,a^n,s^{n-1},a^{n-1},...,s^0,a^0)$. The key idea behind an MDP is the Markovian property: the probability of transitioning to state s^{n+1} by playing action a^n depends exclusively on the current state s^n and is independent of previous states. This assumption is actually satisfied by many dynamical systems and introduces a significant simplification in the model, as we do not need to deal with the history space, which is considerably larger than the $S \times A$ space.

The solution for an MDP is a policy $\pi : S \times N \to A$, where $\pi(s^n,n)$ is a probability distribution over A denoting the probability that the agent plays action $a^n \in A$ where it is in state s^n in time step n. Note that we consider that the policies are Markovian, and it is important noting that this kind of policies can replicate any other policy that would take into account the whole history [26, Section 1.1.4]. This is another reason to use this kind of policies: they are not only simpler to obtain, but also allow exactly replicating more complex policies.

An MDP has a cyclic behavior illustrated by Figure 2.1. There is a single agent that interacts in time steps $0,1,2,...,n-1,n,n+1,...,N-1$ with the dynamical system. At each time step n, the system is in a certain state s^n, which the agent knows. Then, following a certain policy $\pi(s^n,n)$, the agent chooses an action a^n and plays it, which causes the system to transition to state s^{n+1} and the agent receives a reward $r(s^n,a^n)$. When $n = N$, the interaction ends: a final state s^N is returned, as well as a final reward $r(s^N)$.

Depending on the context, there are several variations of the reward function with respect to the MDP model presented above:

- In an MDP, the agent is in state s^n, takes action a^n and receives both the reward $r(s^n,a^n)$ and the next state s^{n+1}. One possibility is considering that the reward depends on s^n and a^n explicitly, and hence, the reward is a mapping $R : S \times A \to \mathbb{R}$, as in the model we have presented. Another possibility is considering that the reward depends on s^{n+1} and hence, the reward is a mapping $R : S \to \mathbb{R}$. Both models are equivalent, as s^{n+1} depends on s^n and a^n through the transition probability function $P(s^{n+1}|s^n,a^n)$.

- The reward function is to be maximized and is frequent in Artificial Intelligence contexts. However, in control contexts, it is frequent to deal with a cost function, which is to be minimized by the agent. Again, both models are equivalent: we can transform a reward into a cost function (and the other way around) simply by multiplying by -1.

As we indicated at the beginning of the Chapter, in this work we only deal with discrete time, as all the problems that we study in the incoming sections are of the discrete time kind. However, we note that most of the theoretical developments in this Section have also counterparts in the continuous time case: a thorough presentation of such results is in [73] and [25, Ch.3], which we do not detail as they are out of the scope of this work.

2.2.2 Solving a finite horizon MDP

Let us focus in the case in which N is finite, which is known as finite horizon case. Given an initial state s^0 and a given policy $\pi(s^n,n)$, it is possible to define the expected reward of the policy π starting at state s^0, $J_\pi(s^0)$, as:

$$J_\pi(s^0) = \mathbb{E}_{\pi,P}\left[r(s^N) + \sum_{n=0}^{N-1} r\left(s^n, \pi(s^n,n)\right)\right], \tag{2.2}$$

where $\mathbb{E}$ denotes the mathematical expectation, which in this case is taken over the random variables $s^{n+1} \sim P(s^{n+1}|x^n, \pi(s^n,n))$ and $\pi(s^n,n)$, which we recall, is a probability distribution over the action set. Hence, note that the expected reward is strongly affected by the policy chosen, and indeed, the optimal policy $\pi^*(s^n,n)$ is the one that maximizes the total reward:

$$J_{\pi^*}(s^0) = J^*(s^0) = \max_{\pi\in\Pi} J_\pi(s^0), \tag{2.3}$$

where Π is the set of admissible policies, that is, valid distributions over actions $a \in A$, and $J^*(s^0)$ is the optimal expected reward, which we remark, depends on the initial state s^0.

A standard tool to obtain the optimal policy for an MDP is the technique known as Dynamic Programming (DP), which is owed to Bellman and that states the following [25, Proposition 1.3.1]:

Lemma 1. *For every initial state s^0, the optimal expected reward $J^*(s^0)$ equals the $J^0(s^0)$ obtained using the following backwards recursion from time step $N-1$ to time step 0:*

$$\begin{aligned} J^N(s^N) &= r(s^N) \\ J^n(s^n) &= \max_{\pi\in\Pi} \mathbb{E}_\pi\left[r\left(s^n,a^n\right) + \sum_{s^{n+1}\in S} P(s^{n+1}|s^n,a^n)J^{n+1}\right]. \end{aligned} \tag{2.4}$$

The optimal policy $\pi(s^n,n)$ can be obtained as:

$$\begin{aligned} J^N(s^N) &= r(s^N) \\ \pi^*(s^n,n) &= \arg\max_{\pi\in\Pi} \mathbb{E}_\pi\left[r\left(s^n,a^n\right) + \sum_{s^{n+1}\in S} P(s^{n+1}|s^n,a^n)J^{n+1}\right]. \end{aligned} \tag{2.5}$$

As shown by the previous Lemma, DP algorithm proceeds backwards by relying on the fact that, when optimizing for time step n, we have previously optimized for time steps $n+1$ to N. Thus, we cannot improve J^{n+1} as it has already been optimized in the previous iteration, and hence, we have to find the optimal action for the current time step n. This is known as the Principle of Optimality, and according to Bertsekas, "the principle of optimality suggests that an optimal policy can be constructed in piecemeal fashion, first constructing an optimal policy for the "tail subproblem" involving the last two stages, and continuing in this manner until an optimal policy for the entire problem is constructed" [25, p. 19]. Note that the DP algorithm shown in Lemma 1 suffers from the so called "curse of dimensionality": the algorithm scales badly with large states and action spaces, and long horizons, as the memory and computation complexity depends on these parameters. Hence, it is no surprise that in practice, infinite horizon iterative methods, which scale better, are generally used.

The results in this Section have been established considering that the probability transition function was stochastic. However, it is possible that $P(s^{n+1}|s^n,a^n)$ is deterministic, i.e., the probability transition function assigns all the probability to a single state s^{n+1} for each (s^n,a^n) pair. In this case, the optimal policy $\pi^*(s^n,n)$

may be replaced for a sequence of actions of length N: $a^0, a^1, ..., a^{N-1}$, as the state trajectory is perfectly predictable given s^0 [25, Chapter 3]. This case is sometimes known as Open-Loop: for each initial state, the agent needs to compute an optimal action sequence and apply it, without the need to observe the state $s^n, \quad n > 0$, as this state is perfectly predictable. A widely used tool for such problems is the Minimum Principle, derived by Pontryagin [73]. In contrast with the Open-Loop situation, we do not assume in this work that the transition function is deterministic. Thus, the agent does need to observe the state s^n in order to achieve optimality: this solution implies the use of a certain policy $\pi(s^n, n)$ which depends on s^n. This solution, frequently known as Closed-Loop or feedback solution, can be computed using the Dynamic Programming algorithm just explained and will be used in the rest of this work. Note that using a Closed-Loop solution does not provide a loss of generality, as in the case of having a deterministic transition function, the policy obtained by DP is optimal.

2.2.3 Solving an infinite horizon MDP

In an infinite horizon problem, we have that $N = \infty$, that is, that the number of time steps are infinite. In these cases, the expected reward is defined as:

$$\lim_{N\to\infty} \mathbb{E}_{\pi,P}\left[\sum_{n=0}^{N-1} \gamma^n r(s^n, \pi(s^n, a^n))\right], \tag{2.6}$$

where we note these important differences between (2.2) and (2.6):

- In the infinite horizon problem there is no final reward associated with a final state, as there is no such a final state due to having an infinite horizon.
- In the infinite horizon problem, there is a discount factor $\gamma \in (0, 1)$, which is used to weight how important are future rewards. As $\gamma < 1$, future rewards matter less than the current reward from the optimization perspective. Note that we need $\gamma < 1$ and R bounded in order to guarantee that the expected reward (2.6) exists, as it is the sum of infinite terms geometrically weighted by γ^n. In the finite horizon problem, note that $\gamma = 1$, which means that all rewards matter the same from the optimization perspective. It is also possible to work with an average reward concept in the infinite horizon setting, but we only use it in Chapter 3, where we introduce it. A detailed introduction to the average reward case is given in [26].
- The policy in the infinite horizon problem now is stationary: note that it does not depend on time. Intuitively, this is due to the fact that, each time that we are in a certain state s, there is still infinite time steps to come, and hence, from the optimization perspective, the actual value of the time step is indifferent to the agent. Note that this is not the case in the finite horizon case, as being closer to the final time step N may have an influence on the optimal policy.

We here introduce some notation regarding the policy. $\pi(a|s)$ is the vector of dimension A which contains the probability distribution over the action space given the state s; that is, each entry of the vector contains the probability of choosing each action in the state s. $\pi(s,a)$ is a scalar which contains the probability of choosing action a in state s: note that $\pi(a|s)$ is formed by stacking the $\pi(s,a)$ values for all possible actions in the state s. And finally, we reserve $\pi(s)$ to the mapping used when the policy is deterministic, where $\pi(s)$ returns the action a prescribed by the deterministic policy π for the state s.

Discounted value functions

In the infinite horizon case, it is frequent working with discounted rewards, as we have indicated. In order to avoid confusions, we preserve J for cumulative rewards without discounting and introduce a new notation for the discounted case in this Section. We can define the discounted cumulative reward from time n onward, G^n, as:

$$G^n = r^n + \gamma r^{n+1} + \gamma^2 r^{n+2} ... = \sum_{i=0}^{\infty} \gamma^i r^{n+i}, \tag{2.7}$$

where we use the shorthand $r^n = r(s^n, a^n)$. It is common to define the value function $V_\pi(s)$ as a mapping $V_\pi : S \to \mathbb{R}$ that represents the expected return over all possible trajectories when the agent starts is in a certain state s and follows the policy π as follows:

$$V_\pi(s) = \mathbb{E}_{\pi,P}\left[G^n | s^n = s, a^{n+k} \sim \pi\right], \quad k = 0, 1, 2, 3, ..., \infty, \tag{2.8}$$

where a very important property of the value function is that it can be expressed recursively as follows:

$$\begin{aligned} V_\pi(s) =& \mathbb{E}_{\pi,P}\left[G^n | s^n = s, a^{n+k} \sim \pi\right] \\ =& \mathbb{E}_{\pi,P}\left[r^n + \gamma r^{n+1} + \gamma^2 r^{n+2} ... | s^n = s, a^{n+k} \sim \pi\right] \\ =& \mathbb{E}_{\pi,P}\left[r^n + \gamma G^{n+1} | s^n = s, a^{n+k} \sim \pi\right] \\ =& \mathbb{E}_{\pi,P}\left[r^n + \gamma V_\pi\left(s^{n+1}\right) | s^n = s, a^{n+k} \sim \pi\right], \quad k = 0, 1, 2, 3, ..., \infty. \end{aligned} \tag{2.9}$$

Similarly, we can define the state-action value function $Q_\pi(s,a)$ as a mapping $Q : S \times A \to \mathbb{R}$ that represents the expected return when the agent is in state s, takes action a and then follows policy π as:

$$Q_\pi(s,a) = \mathbb{E}_{\pi,P}\left[G^n | s^n = s, a^n = a, a^{n+k} \sim \pi\right], \quad k = 1, 2, 3, ..., \infty, \tag{2.10}$$

where again, we can express the state-action value function recursively as:

$$\begin{aligned} Q_\pi(s,a) =& \mathbb{E}_{\pi,P}\left[G^n | s^n = s, a^n = a, a^{n+k} \sim \pi\right] \\ =& \mathbb{E}_{\pi,P}\left[r^n + \gamma r^{n+1} + \gamma^2 r^{n+2} ... | s^n = s, a^n = a, a^{n+k} \sim \pi\right] \\ =& \mathbb{E}_{\pi,P}\left[r^n + \gamma G^{n+1} | s^n = s, a^n = a, a^{n+k} \sim \pi\right] \\ =& \mathbb{E}_{\pi,P}\left[r^n + \gamma Q_\pi(s^{n+1}, a^{n+1}) | s^n = s, a^n = a, a^{n+k} \sim \pi\right], \quad k = 1, 2, 3, ..., \infty. \end{aligned} \tag{2.11}$$

Both functions can be related as follows:

$$\begin{aligned} V_\pi(s) &= \sum_{a \in A} \pi(s,a) Q_\pi(s,a) \\ Q_\pi(s,a) &= r(s,a) + \gamma \sum_{s^{n+1}} P(s^{n+1} | s^n = s, a^n = a) V_\pi\left(s^{n+1}\right), \end{aligned} \tag{2.12}$$

and this formulation is very interesting because it is possible to combine both expressions to obtain the following linear equation:

$$V_\pi(s) = \sum_{a \in A} \pi(s,a) \left[r(s,a) + \gamma \sum_{s^{n+1}} P(s^{n+1}|s^n = s, a^n = a) V_\pi\left(s^{n+1}\right), \right] \tag{2.13}$$

which can be expressed in matrix form as:

$$v_\pi = r_\pi + \gamma P_\pi v_\pi, \tag{2.14}$$

where each element is defined as follows, where $|S|$ denotes the cardinality of the state set:

- v_π is a vector of dimension $|S|$, formed by stacking the value function values as follows:

$$v_\pi = \left(V_\pi(s)\right)_{s \in S}. \tag{2.15}$$

- r_π is a vector of dimension $|S|$, formed by stacking the reward values induced by the policy π as follows:

$$r_\pi = \left(\sum_{a \in A} \pi(s,a) r(s,a) \right)_{s \in S}. \tag{2.16}$$

- P_π is a matrix of dimension $|S| \times |S|$, formed by stacking the transition probabilities induced by the policy π as follows:

$$P_\pi = \left(\sum_{a \in A} \pi(s,a) P(s^{n+1}|s^n = s, a^n = a) \right)_{s, s^{n+1} \in S}. \tag{2.17}$$

  Note that P_π is stochastic, as all its rows are probability vectors, i.e., the row elements sum 1 and all the matrix entries are greater than or equal to 0.

The Bellman operators

The matrix form of the value function expression (2.14) is of special interest in order to obtain a Dynamic Programming algorithm for the infinite horizon case. Note that we cannot apply the results obtained in Lemma 1, as $N \to \infty$ and we do not have well-defined terminal states in the infinite horizon case. First, let us define the next two mappings:

$$\begin{aligned} T_\pi\left(V_\pi(s)\right) &= \sum_{a \in A} \pi(s,a) \left[r(s,a) + \gamma \sum_{s^{n+1}} P(s^{n+1}|s^n = s, a^n = a) V_\pi\left(s^{n+1}\right) \right] \\ T\left(V_{\pi^*}(s)\right) &= \max_{a \in A} \left[r(s,a) + \gamma \sum_{s^{n+1}} P(s^{n+1}|s^n = s, a^n = a) V_{\pi^*}\left(s^{n+1}\right) \right], \end{aligned} \tag{2.18}$$

where T_π is the Bellman operator for the policy π and T is the optimal Bellman operator. Note that in the optimal Bellman operator, we are evaluating for the optimal policy, i.e., the one that provides the optimal value. A very important result is collected by the next Lemma:

Lemma 2. *Both Bellman operators introduced in* (2.18) *are contraction mappings with respect to the infinity norm, for* $0 < \gamma < 1$.

Proof. A mapping $F : X \to X$ is a contraction mapping is the following conditions is fulfilled:

$$||F(x_1) - F(x_2)|| \leq c||x_1 - x_2||, \quad 0 < c < 1, \quad \forall x_1, x_2 \in X, \tag{2.19}$$

where $||\quad||$ is a metric defined on the space X. Also, the infinity norm, or max-norm, is defined as $||v||_\infty = \max |v|$ for a vector v.

The proof for both Bellman operators introduced in (2.18) is similar, so we only include the Bellman operator case. First, as the probability transition matrix P_π is stochastic, the following is always satisfied:

$$||P_\pi v_\pi||_\infty \leq ||v_\pi||_\infty. \tag{2.20}$$

Thus, using the operator definition (2.18) and the matrix form given by (2.14), we have that:

$$\begin{aligned} ||T_\pi(v_1) - T_\pi(v_2)||_\infty &= ||(r_\pi + \gamma P_\pi v_1) - (r_\pi + \gamma P_\pi v_2)||_\infty \\ &= ||\gamma P_\pi (v_1 - v_2)||_\infty \\ &\leq ||\gamma P_\pi||v_1 - v_2||_\infty||_\infty \\ &\leq \gamma||v_1 - v_2||_\infty. \end{aligned} \tag{2.21}$$

□

The importance of the fact that Bellman operators are contraction mappings is that they converge to a unique fixed point using the Banach Fixed Point Theorem. The main results of applying this Theorem to the Bellman operators are in the following Theorem [26].

Theorem 1. *The following results hold for the Bellman operator T_π for a policy π:*

- *There exists a unique solution v_π for the fixed point equation $T_\pi(v) = v$.*
- *For any $v^0 \in \mathbb{R}^{|\mathbb{S}|}$, the sequence generated by $v^{k+1} = T_\pi\left(v^k\right)$ converges to v_π as $k \to \infty$.*

And the following results hold for the optimal Bellman operator T:

- *There exists a unique solution v_{π^*} for the fixed point equation $T(v) = v$.*
- *For any $v^0 \in \mathbb{R}^{|\mathbb{S}|}$, the sequence generated by $v^{k+1} = T\left(v^k\right)$ converges to v_{π^*} as $k \to \infty$.*

Thus, this suggest that it is possible to obtain the optimal value function, and hence, the optimal policy, by recursively applying the Bellman operators. Furthermore, it is possible to define equivalent Bellman operators for the Q function with analogous properties, which we have not included for the sake of clarity. We now turn our attention to Value Iteration (VI) and Policy Iteration (PI), two DP algorithms for the infinite horizon case, which are derived from the convergence properties of the Bellman operators shown in Theorem 1.

Policy Iteration

One of the first implications of the convergence properties of the Bellman operators in Theorem 1 is that we can recursively use the Bellman operator T_π in order to evaluate the value function induced by a certain policy π. This procedure is known as Policy Evaluation (PE) and consists in randomly initializing v_π and repeatedly applying the Bellman operator, until a convergence criterion is met, such as the maximum difference between value functions in two consecutive iterations is below a threshold. The procedure is summarized in Algorithm 1.

Algorithm 1 Policy Evaluation (PE) procedure. A usual convergence criterion is that the maximum difference between the v vector in two consecutive iterations is below a threshold.

Input: S, A, π, R, P, γ
1: Initialize v_π arbitrarily
2: **while** Convergence criterion is not met **do**
3: **for** $s \in S$ **do**
4: $v_\pi(s) = \sum_{a \in A} \pi(s,a) \left[r(s,a) + \gamma \sum_{s^{n+1}} P(s^{n+1}|s^n = s, a^n = a) v_\pi \left(s^{n+1} \right) \right]$
Output: v_π

Algorithm 2 Policy iteration (PI) algorithm.

Input: S, A, R, P, γ
1: Initialize π^0 arbitrarily
2: Set $k = 0$
3: **while** Convergence criterion is not met **do**
4: Obtain v_{π^k} using the Policy Evaluation procedure (Algorithm 1).
5: Improve the policy: $\pi^{k+1}(s) = \arg\max_{a \in A} r(s,a) + \gamma \sum_{s^{n+1}} P(s^{n+1}|s^n = s, a^n = a) v_{\pi^k} \left(s^{n+1} \right)$
6: Set $k = k + 1$
Output: $v_{\pi^{k-1}}$, π^k

Policy Iteration algorithm (PI) uses the convergence properties of the Bellman operator T_π to obtain an optimal policy π^*. It starts with π^0, a randomly initialized policy, and in each iteration k of the algorithm, the policy π^k is evaluated using the Policy Evaluation procedure already described in Algorithm 1 in order to obtain V_{π^k}, and then, the policy is improved using:

$$\pi^{k+1}(s) = \arg\max_{a \in A} r(s,a) + \gamma \sum_{s^{n+1}} P(s^{n+1}|s^n = s, a^n = a) V_{\pi^k} \left(s^{n+1} \right). \tag{2.22}$$

This procedure converges to the optimal policy π^* as $k \to \infty$ [215]. The procedure is summarized in Algorithm 2.

Value iteration

Value Iteration algorithm (VI) uses the convergence properties of the optimal Bellman operator T to obtain an optimal policy π^*. Instead of alternating between evaluating a value function and optimizing the policy as in PI algorithm, VI repeatedly applies the optimal Bellman operator to obtain the optimal value function, V_{π^*}, and once that this value has been approximated, it is used to obtain the optimal policy as:

$$\pi^*(s) = \arg\max_{a \in A} r(s,a) + \gamma \sum_{s^{n+1}} P(s^{n+1}|s^n = s, a^n = a) V_{\pi^*} \left(s^{n+1} \right), \tag{2.23}$$

and again, this procedure converges to the optimal policy π^* as $k \to \infty$ [215]. The procedure is summarized in Algorithm 3. Note that PI requires a double loop: the inner loop firsts solves the Bellman equation for a given policy, and then optimizes the policy, and this procedure is repeated several times. On the other hand, VI has a single loop which is used to approximate the optimal value function, and then a single policy is obtained at the end of the computation. Depending on the dimensions of the state and action spaces, the computational resources and time required by each algorithm may differ, although both converge to the optimal solutions of the MDP.

Algorithm 3 Value iteration (VI) algorithm.

Input: S, A, R, P, γ

Initialize v^0 arbitrarily
Set $k = 0$
while Convergence criterion is not met **do**
  for $s \in S$ **do**
    $v^{k+1}(s) = \max_{a \in A} \left[r(s,a) + \gamma \sum_{s^{n+1}} P(s^{n+1}|s^n = s, a^n = a) v^k \left(s^{n+1}\right) \right]$
  Set $k = k+1$
Obtain the optimal policy: $\pi(s) = \arg\max_{a \in A} r(s,a) + \gamma \sum_{s^{n+1}} P(s^{n+1}|s^n = s, a^n = a) v^k \left(s^{n+1}\right)$

Output: v^k, π

As noted by Bertsekas [25, Chapter 7], the infinite horizon problem never holds in practice. Yet it is a reasonable approximation for problems with many time steps. Also, note that we can model a finite horizon problem under the infinite horizon perspective by simply adding an absorbing state to the state space of an infinite horizon MDP: an absorbing state is a state such that the probability to transition to itself is 1, which means that the agent cannot leave that state, regardless of its actions. Also, the policies in the infinite horizon case are stationary, and hence, simpler than in the finite horizon case, and PI and VI iterative methods scale better than the method shown in Lemma 1 for large state and action spaces. All these reasons make infinite horizon MDPs very popular in current applications, and also justify that we use infinite horizon models in the rest of this work.

2.2.4 Model-free methods

So far, we have assumed that the transition probability function P was known and available to the agent for the optimization procedure. However, this needs not be the case in real life problems, as the function P may be unknown or hard to obtain. In such cases, an MDP can still be approximately solved using the set of techniques known as Reinforcement Learning (RL). RL methods are biologically inspired and their basic intuition is that it is possible to learn how to act optimally in a dynamical system as the one shown in Figure 2.1 by interacting with the system using trial and error. That is, the agent interacts with the dynamical system and stores sets of tuples containing information about each interaction, that is, $\left(s^n, a^n, r^n, s^{n+1}\right)$. This information is then used by the agent to optimize a policy π that maximizes the expected discounted reward. A complete introduction to the field is given in [215].

Q-learning

We now introduce Q-learning, a well-known RL algorithm [215]. The main idea of Q-learning is to store estimates of the Q function (2.10) and update them as the agent interacts with the system. For simplicity, we now assume that $|S|$ and $|A|$ are discrete, and hence, the Q values can be stored in a table of dimension $S \times A$. Q-learning algorithm starts by initializing the Q function to any value, for instance, $Q(s^n, a^n) = 0, \quad \forall s^n \in S, \quad \forall a^n \in A$. Then, the agent starts to repeatedly interact with the dynamical system: it observes its current state s^n and takes action a^n following an ε-greedy policy:

$$\pi_{\varepsilon-greedy}(s^n) = \begin{cases} a^* = \arg\max_{a^n \in A} Q(s^n, a^n) & \text{with probability} \quad 1-\varepsilon \\ Random(a^n \in A \setminus a^*) & \text{with probability} \quad \varepsilon \end{cases}, \tag{2.24}$$

Algorithm 4 Q-learning algorithm. A typical convergence criterion is the maximum number of iterations.

Input: S, A, α, ε
Initialize $Q_\pi(s^n, a^n) = 0, \quad \forall a^n \in A, \quad \forall s^n \in S$
while Convergence criterion is not met **do**
  Initialize $n = 0$
  Set initial state s^0
  while State s^n is not final **do**
    Obtain action a^n using ε-greedy policy (2.24)
    Take action a^n and obtain s^{n+1} and r^n
    Update $Q_\pi(s^n, a^n)$ using (2.25)
    Set $n = n + 1$
for $s^n \in S$ **do**
  $\pi(s^n) = \arg\max_{a \in A} Q_\pi(s^n, a)$
Output: π

where the idea in (2.24) is that the agent chooses to take the action a^* that maximizes $Q(s^n, a^n)$ with probability $1 - \varepsilon$ and with probability ε, it takes a random action except a^*. Observe that the values of ε will regulate the exploration-exploitation trade-off: high values of ε will cause that the agent explores the rewards that different actions give it, whereas low values of ε will cause the agent to exploit the action that gives it the highest payoff. Note that a high exploration is desirable in order not to get stuck in poor maximum, while a high exploitation provides larger rewards to the agent. In practice, it is common starting with a high ε to explore often, and diminish its value as the training progresses. Even though other policies than ε-greedy are valid for the Q-learning algorithm, we use it for its simplicity. Note that $0 \leq \varepsilon \leq 1$.

When the agent takes action a^n, the environment transitions to a new state s^{n+1} and it returns an immediate reward r^n to the agent. This allows updating the Q-function value for state s^n and action a^n as follows:

$$Q_\pi(s^n, a^n) = Q_\pi(s^n, a^n) + \alpha \left(r^n + \gamma \max_a Q_\pi(s^{n+1}, a) - Q_\pi(s^n, a^n) \right), \tag{2.25}$$

where $\alpha \in [0, 1]$ is a parameter that controls the learning rate: low values of α means a low Q update, but high values of α introduce a high variance on Q values. Q-learning algorithm is summarized in Algorithm 4. Note that there are similarities between Q-learning (Algorithm 4) and Value Iteration (Algorithm 3) due to the fact that Q-learning is a model-free implementation of VI, which relies on similar ideas: we try to approximate the optimal value function, and then, we obtain an approximate optimal policy. Q-learning converges in the limit to the optimal Q-function under some mild convergence conditions [215], [234].

Linear approximations

The previous Q-learning algorithm was derived under the assumption that the action and state spaces were discrete, which allowed the Q-values to be stored in a table. However, as the number of states and/or actions grows, the memory required to store the table might be too expensive. Also, note that having a discrete state space may be too restrictive in real life dynamical systems. A solution to these problems is to use function approximations for the value functions. A possible approach is using linear approximations, in which the value function is approximated linearly using M basis functions ϕ as:

$$Q_\pi(s, a) = \sum_{m=1}^{M} \alpha_m \phi_m(s, a), \tag{2.26}$$

where α is a vector of parameters with M entries and ϕ_m are M basis functions used to approximate the value function. The algorithms that we have seen in the previous Sections can be adapted to deal with these linear approximations [26, Chapter 6], in order to obtain the α vector that best approximates the value function. The main advantage of using these approximations is that they allow dealing with many discrete states or even continuous states, and at the same time, control the problem complexity, as the optimization complexity depends on M instead of on $|S|$. However, the main problem with this approach is that the quality of the approximation is directly related to the approximation capabilities of the basis functions: if the actual value function is contained in the subspace spanned by the basis functions, then the approximation error will be small, but unluckily, this is not often the case. Also, good approximation capabilities require carefully handcrafted basis functions. Due to these problems, linear approximations are rarely used today. In the next Section, we introduce neural networks as universal function approximators, and afterwards, we present two RL algorithms that use their approximation capabilities.

Feedforward Neural networks

Neural networks (NNs) are function approximators that are able to approximate functions to an arbitrary degree of accuracy [105], [51]. In other words, NNs are universal function approximators. An NN is a layered structure composed by neurons [95]. A Feedforward Neural Network (FNN) is a directed architecture as seen in Figure 2.2, in which the neurons of two consecutive layers are densely connected, thus these layers are sometimes known as Dense. Each of the neurons outputs a nonlinear combination of its inputs as follows:

$$z = \varphi\left(\sum_i w_i \cdot x_i + b\right), \tag{2.27}$$

where z is the neuron output, x its input vector, w is a vector of weights, b is a scalar bias and φ is the activation function, which usually is nonlinear. Note that each neuron receives as input a vector and outputs a single, deterministic value. Some of the most popular activation functions are:

Name	$\varphi(x)$	$\varphi^{\rightarrow}(x)$
Rectified Linear Unit Activation (ReLU)	$\max(0,x)$	$[0,\infty)$
Sigmoid Activation (Sigm)	$\frac{e^x}{e^x+1}$	$(0,1)$
Hyperbolic Tangent Activation (Tanh)	$\frac{e^x-e^{-x}}{e^x+e^{-x}}$	$(-1,1)$
Linear Activation	x	$(-\infty,\infty)$

(2.28)

where $\varphi^{\rightarrow}(x)$ denotes the image set of the function $\varphi(x)$, and it is important to take into account it because the image of the output of an NN depends on the activation function of the neurons in the last layer. These activation functions are also chosen because their gradient, needed for training the NN, can be easily computed numerically. Note that we include the linear activation for completeness.

Observing the FNN in Figure 2.2, we note that there is an input layer, an output layer, and an intermediate, hidden layer. Depending on the number of hidden layers, we can difference between shallow and Deep Neural Networks (DNNs), the former having no hidden layer and the latter having at least one. Even though the universal approximation property of the neural networks has been known for years [105], [51], the lack of sufficient computational power for training these neural networks caused them not to be too successful. This radically changed as the computational power available to researchers grew: DNNs composed of many layers

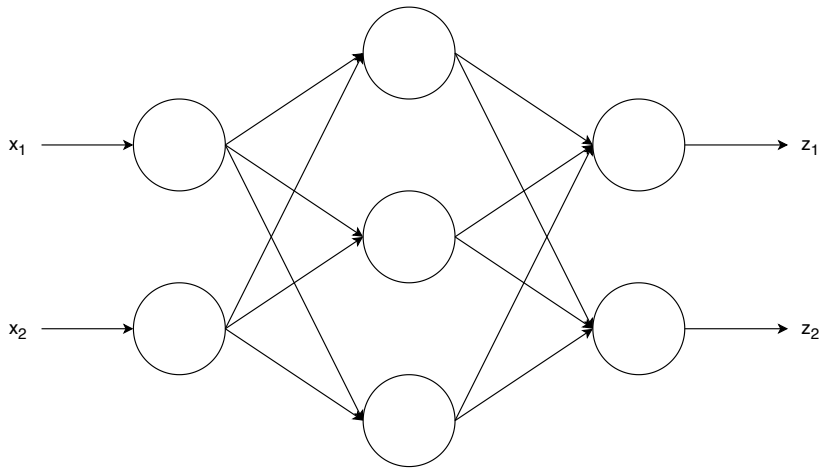


Fig. 2.2 Example of feedforward neural network. Each circle represents a neuron, which combines non-linearly its inputs following (2.27). The inputs are x_1 and x_2, and the outputs are z_1 and z_2. There is a single hidden layer, which has three neurons. Note how each of the outputs z is a nonlinear combination of the inputs x_1 and x_2.

can today be trained and many applications for DNNs began to appear [129]. Today, DNNs are used successfully for many complex tasks, such as object classification in images [258], human-like performance in Atari games [154] or indoor localization [230], to mention a few. The increasingly high computational power available even makes it feasible to use DNN in WSN deployments [181].

Training an NN means obtaining the set of weights and biases that approximates a certain function. This is usually done from data: we provide the NN with a dataset of inputs and outputs, and the weights and biases are iteratively updated in order to minimize a loss between the NN output and the desired output given by the dataset. This update is usually done by means of the backpropagation algorithm [96], which is an application of the chain rule to obtain the gradients of the outputs with respect to the parameters of the neurons, i.e., weights and biases, and then apply a first order optimization method, such as Adam [122], in order to update the weights and biases of the NN.

Deep Q-Networks

As we have seen before, a main drawback of the Q-learning algorithm presented comes when the space set is large. In order to deal with this, and also with continuous state spaces, we have already presented linear approximations, which however present questionable results, as the basis functions need to be handcrafted. A recent approach takes advantage of the advances in Deep Learning and uses NNs in order to approximate the value function. The first algorithm that followed this approach was an adaptation of Q-learning, known as Deep Q-Networks (DQN) [153], [154].

DQN approach is based on Q-learning, thus it is similar to Algorithm 4, but under DQN, the Q-function is replaced by a DNN, whose input is the current state s^n and its output is the approximated Q-value function for every action $a^n \in A$, $Q_\pi(s^n, a^n)$. Another characteristic of DQN is that it makes use of the past experience to enhance the learning process: at each time step, the experience of the agent $e^n = (s^n, a^n, r^n, s^{n+1})$ is stored in a data set E. The set E is updated as new actions are taken by the agents and the data contained in E is used to update the NN multiple times. Experience replay allows a greater data efficiency and also, it allows avoiding the correlation between consecutive samples, as the convergence of the gradient methods used to update the

Algorithm 5 Deep Q-Networks algorithm. A typical convergence criterion is the maximum number of iterations.

Input: S, A, ε
Initialize the Q-function $Q_\pi(s,a)$ DNN, with weights θ
Initialize the target network, which is equal to the Q-function $Q_\pi(s,a)$ DNN, with weights $\theta_{tg} = \theta$
Initialize replay memory $E = \{\}$
while Convergence criterion is not met **do**
  Initialize $n = 0$
  Set initial state s^0
  while State s^n is not final **do**
    Obtain action a^n using ε-greedy policy (2.24)
    Take action a^n and obtain s^{n+1} and r^n
    Store $e^n = (s^n, a^n, r^n, s^{n+1})$ in E
    for A batch of $e \in E$ randomly sampled **do**
      Obtain B using (2.29)
    Update θ using B
    Set $n = n+1$
  Update $\theta_{tg} = \theta$
for $s^n \in S$ **do**
  $\pi(s^n) = \arg\max_{a \in A} Q_\pi(s^n, a; \theta)$
Output: π

neural networks improves with uncorrelated samples [153]. Another feature of DQN consists on using target networks: at each training epoch, the Q-learning DNN is cloned and the copy is called target network. The target network is used to estimate the future Q values to update the NN during that epoch. Because target networks are updated only once per epoch, that gives better convergence results and helps to avoid oscillations [154]. We denote the parameters of the Q function DNN as θ, and the target network parameter as θ_{tg}.

Every time that the Q-function DNN has to be trained, we randomly sample a set of experiences from the replay memory E, and for each e, we build the following B value:

$$B(e) = \begin{cases} r^n & \text{for terminal} \quad s^{n+1} \\ r^n + \gamma \max_{a \in A} Q_\pi(s^n, a; \theta_{tg}) & \text{for non-terminal} \quad s^{n+1} \end{cases}, \tag{2.29}$$

where $Q_\pi(s^n, a; \theta_{tg})$ is the approximated Q-function value for the state s^n given by the target network. Thus, for each e^n vector, we train $Q_\pi(s^n, a^n; \theta)$ to minimize the difference with respect to $B(e^n)$. Note that this means that the Q-function DNN is trying to approximate the actual Q-function. Also, observe that we compute (2.29) using the target network, and use the resulting B values to update the θ weights.

An implementation of DQN can be observed in Algorithm 5. Again, we note that this Algorithm is similar to Q-learning (Algorithm 4), as DQN is a Q-learning based implementation using DNNs. The main advantages of using DNNs is that they allow using continuous states and that they significantly reduce the memory required to store the Q-function, as it is now stored as a DNN. However, note that DQN is only valid for discrete action spaces. Now, we turn our attention to a different way of using DNNs under an RL paradigm, in order to be able to deal with continuous actions.

Trust Region Policy Optimization

DQN uses a DNN to approximate the Q value function. However, it is also possible to use a DNN to approximate the policy function instead, and then compute the gradient of the total expected reward with respect to the

parameters of the DNN policy. This approach is denoted as policy gradient (PG), which englobes several popular Deep RL methods, such as Deep Deterministic Policy Gradient (DDPG) [136], Trust Region Policy Optimization (TRPO) [197] or Proximal Policy Optimization (PPO) [198]. The objective of PG methods is to solve the following problem:

$$\max_{\omega} \mathbb{E}_{\pi_\omega, P}\left[\sum_{n=0}^{\infty} \gamma^n r^n\right], \tag{2.30}$$

where ω denotes the parameters of a DNN which is used to model the policy π_ω. According to the Policy Gradient Theorem [216], [206], the gradient of (2.30) has the following form:

$$\nabla_\omega \left\{ \mathbb{E}_{\pi_\omega, P}\left[\sum_{n=0}^{\infty} \gamma^n r^n\right]\right\} = \mathbb{E}_{\pi_\omega, P}\left[\sum_{n=0}^{\infty} \gamma^n \nabla_\omega \log \pi_\omega(a^n|s^n) A_{\pi_\omega}(s^n, a^n)\right], \tag{2.31}$$

where $A_{\pi_\omega}(s^n, a^n)$ is the advantage function, used to estimate how good is action a^n when used in state s^n. Different algorithms obtain $A_{\pi_\omega}(s^n, a^n)$ in different ways, for instance, we could use $A_{\pi_\omega}(s^n, a^n) = Q_\pi(s^n, a^n)$ or $A_{\pi_\omega}(s^n, a^n) = Q_\pi(s^n, a^n) - V_\pi(s^n)$. In general, the advantage function is implemented using DNNs that estimate the value function, similarly to the procedure used by DQN.

Even though it is possible to use first order methods, following in every step the gradient direction (2.31) in order to approximate the maximum of (2.30), these methods usually have poor convergence properties as they do not take into account the curvature of the policy surface parameters. In order to alleviate this, TRPO uses a second order optimization method that does take into account information about the curvature. First, TRPO optimizes iteratively L, a lower bound of the maximization objective (2.30):

$$L = \mathbb{E}_{\pi_\omega, P}\left[\sum_{n=0}^{\infty} \gamma^n \frac{\pi_\omega(a^n|s^n)}{\pi_{old}(a^n|s^n)} A_{\pi_\omega}(s^n, a^n)\right], \tag{2.32}$$

where π_{old} refers to the value of the DNN policy in the previous iteration. This gives rise to the following optimization problem:

$$\begin{aligned} \max_{\omega} \quad & L_{\pi_{old}}(\pi_\omega) \\ s.t. \quad & \mathbb{E}_{\pi_{old}, P}\left[D_{KL}(\pi_\omega || \pi_{old})\right] \leq \delta \end{aligned}, \tag{2.33}$$

where $D_{KL}(\pi_\omega || \pi_{old})$ is the Kullback-Leibler divergence between π_ω and π_{old}, and δ is a threshold. In [197] there is a proof that (2.33) is a lower bound to (2.30), and hence, we can solve the latter by optimizing the former. Intuitively, in (2.33) we optimize the policy that maximizes the expected reward, where the maximum difference between the new policy with respect to the old one is limited by δ. This limitation is used to avoid large variations in the new policy, which may lead to poor policies: the new policy is bounded to lie within a region bounded by δ.

In order to solve (2.33), the authors in [197] propose using Natural Policy Gradient, a second order optimization method that takes into account second order information about the KL-divergence, as it estimates the Fisher Information Matrix (FIM) of the policies for the optimization. However, inverting the FIM is costly when there are many parameters, and this is the usual case with DNNs. In order to overcome this, [197] uses a Conjugate Gradient approach to efficiently solve the FIM inversion. This allows obtaining a gradient update that takes into account the policy space curvature. Finally, the policy parameters are updated using a line search procedure that ensures that the KL-divergence condition in (2.33) is satisfied.

Algorithm 6 Trust Region Policy Optimization algorithm. A typical convergence criterion is the maximum number of iterations.

Input: δ

Initialize the policy network π_ω, with weights ω
Initialize the advantage function $A_{\pi_\omega}(s^n, a^n)$
Initialize $k = 0$
while Convergence criterion is not met **do**
    Use π_ω^k to obtain a set of trajectories E^k
    Estimate $A_{\pi_\omega^k}(s^n, a^n)$ using E^k
    Estimate the policy gradient using (2.31)
    Estimate the KL-divergence and the FIM
    Use Conjugate Gradient to estimate the next gradient step Δ, which includes the policy gradient estimation and the KL-divergence
    Perform line search to ensure that Δ satisfies the restriction in (2.33), otherwise, shrink Δ
    Update $\omega^{k+1} = \omega^k + \Delta$
    Set $k = k + 1$

Output: π^k

The whole procedure of TRPO can be observed schematically in Algorithm 6. As TRPO is computationally intensive, PPO was designed as a modification to TRPO that uses a surrogate function to alleviate the computational load [198]. In spite of this problem, TRPO has shown to be a very powerful Deep RL algorithm, which performs very well in different sets of tasks, as [64] shows. Also, it is able to deal with continuous and discrete action spaces [197], which provides a significant flexibility when it comes to real world dynamical systems. These two reasons motivate us to include TRPO in our research, as we will use it to work with continuous actions.

Finally, we note that the field of Deep RL is active with much research. There are many others algorithms proposed, such as Asynchronous Advantage Actor Critic (A3C) / Advantage Actor Critic (A2C) [152], Dueling Deep Q-Networks (DDQN) [233], Actor Critic using Kronecker-Factored Trust Region (ACKTR) [240] or Actor Critic with Experience Replay (ACER) [232], and the list expands day by day. We introduce here only two of the most important algorithms, that we will use afterwards in our problems.

2.2.5 Inverse Reinforcement Learning

Inverse reinforcement Learning (IRL), also known as Inverse Optimal Control, is the complementary situation to RL. In RL, there is an agent trying to obtain a policy that maximizes a certain total cumulative reward, while in IRL, an agent is given an optimal policy function or a set of samples from an optimal policy and tries to find the reward function that explains the given policy. Thus, IRL tries to infer the reward that best explains the behavior of an agent, usually called expert. In the seminal paper [164], the authors state two main reasons why IRL is important: first, in RL settings is often assumed that the reward function is known, which needs not be true in real life problems. Another application is apprenticeship learning: constructing an agent able to behave successfully in a certain environment by learning from an expert [1]. Hence, IRL can be used to model another agent, which is the aspect that is of interest to us. We now turn to present several IRL algorithms, from less to more complex.

Linear programming Inverse Reinforcement Learning

The simplest algorithm to solve an IRL problem is based on assuming that S and A are small, discrete spaces, and that we have access to the expert policy function π^*, which is deterministic: this means that all the transition probability is assigned to a single state. First, let us remind the matrix form of Bellman equation (2.14), which can be rearranged as follows:

$$v_\pi = r_\pi + \gamma P_\pi v_\pi = (I - \gamma P_\pi)^{-1} r_\pi. \tag{2.34}$$

By considering that $\succeq$ denotes a vector inequality, i.e., an inequality satisfied by all vector entries, it can be shown that a policy π^* is optimal if and only if [164]:

$$(P_{\pi^*} - P_\pi)(I - \gamma P_{\pi^*})^{-1} r_{\pi^*} \succeq 0, \quad \forall \pi \neq \pi^*, \tag{2.35}$$

which can be intuitively understood as follows. The first term indicates how much the transition probabilities change by using π^* with respect to any other policy π'. The second term is the value function for policy π^*. The product indicates which is the best action for each state: a policy is optimal if it chooses the best action for each state, which is what the inequality checks. In other words, (2.35) finds for a policy π^* which chooses the best action for each state. If the inequality in (2.35) is strict, then the optimal policy is unique.

In (2.35), what we are trying to obtain is the vector of rewards r_{π^*}, as we are given π^*. Note that (2.35) is an ill-posed problem, as there are infinitely many r_{π^*} vectors that satisfy (2.35), including $r_{\pi^*} = 0$. Hence, we need to restrict the possible space of r_{π^*}. In [164], the authors propose the following three ideas:

- Heavily penalize every deviation from the policy π^*. This can be done by maximizing the minimal difference between each action in the optimal policy and the next best action for each state. By defining $P_{\pi^*}(s)$ as the row of the matrix P_{π^*} corresponding to state s, where $P_\pi(s)$ represents the action prescribed by policy π in state s, the previous condition becomes:

$$\max \sum_{s \in S} \min_\pi \left[(P_{\pi^*}(s) - P_\pi(s))(I - \gamma P_{\pi^*})^{-1} r_{\pi^*}. \right] \tag{2.36}$$

- Impose an additional sparsity restriction on r_π, by minimizing a norm-1 regularizer of the reward vector. The regularizer is controlled by a parameter $\lambda \geq 0$. Note that this means that the reward obtained will have as few nonzero entries as possible, which means that the reward obtained will be as simple as possible.

- Limit the maximum value of the reward to a certain value r_{max}.

Putting these ideas together, we obtain the following Linear program:

$$\begin{aligned} \max & \sum_{s \in S} \min_\pi \left[(P_{\pi^*}(s) - P_\pi(s))(I - \gamma P_{\pi^*})^{-1} r_{\pi^*} \right] - \lambda ||r_{\pi^*}||_1 \\ s.t. \quad & (P_{\pi^*} - P_\pi)(I - \gamma P_{\pi^*})^{-1} r_{\pi^*} \succeq 0, \quad \forall \pi \neq \pi^* \\ & r_{max} \succeq |r_{\pi^*}| \end{aligned} . \tag{2.37}$$

It is possible to extend these ideas to less restricted setups. First, let us assume that we want to work with a state space S which is discrete with many states or even continuous. Similarly to (2.26), we can use a linear

approximation of the reward over a set of features mappings $\phi(s,a)$ as follows:

$$r(s,a) = \sum_{m=1}^{M} \alpha_m \phi_m(s,a), \tag{2.38}$$

where the reward function is approximated by a linear combination of M fixed feature mappings $\phi : S \times A \to \mathbb{R}$, and α is the weight vector of the linear approximation. Hence, we exchange finding a reward function $R : S \times A \to \mathbb{R}$, which would involve using calculus of variations, for finding a set of scalar weights α_m using optimization tools. Since the expectations are linear, the value function (2.8) becomes, using (2.38) and the vector form of the value function (2.14):

$$\begin{aligned} v_\pi(s^n) =& \mathbb{E}\left[\sum_{m=1}^{M} \alpha_m \phi_m(s^n, \pi(s^n)) + \gamma \sum_{m=1}^{M} \alpha_m \phi_m(s^{n+1}, \pi(s^{n+1})) + ... | \pi\right] \\ =& \sum_{m=1}^{M} \mathbb{E}\left[\alpha_m \phi_m(s^n, \pi(s^n)) + \gamma \alpha_m \phi_m(s^{n+1}, \pi(s^{n+1})) + ... | \pi\right] = \sum_{m=1}^{M} \alpha_m v_{m,\pi}(s^n) \end{aligned}, \tag{2.39}$$

where $v_{m,\pi}$ is the value function for the mth basis feature. In this case, the optimality condition can be related to the value function: a policy is optimal if the value function induced by that policy is larger or equal than the value function induced by any other policy:

$$\mathbb{E}_{\pi^*,P}\left[v_{\pi^*}(s^n)\right] \geq \mathbb{E}_{\pi,P}\left[v_\pi(s^n)\right], \quad \forall s^n \in S. \tag{2.40}$$

However, the optimality condition in (2.40) is impossible to check exhaustively on a large or infinite state space. Also, as we are approximating using a basis function, it may also happen that the optimal r_{π^*} cannot be represented in the subspace spanned by the chosen basis functions. In order to address the former, [164] proposes checking (2.40) in a finite subset $S_0 \subseteq S$. The latter is addressed by using a penalization function $p(x)$:

$$p(x) = \begin{cases} x & \text{if} \quad x \geq 0 \\ 2x & \text{if} \quad x < 0 \end{cases}. \tag{2.41}$$

The idea behind using the penalization function (2.41) is that we try to minimize the number of states for which (2.40) is not satisfied, and this allows dealing with rewards outside of the subspace spanned by the chosen basis functions. Hence, the linear program that needs to be solved for this case is:

$$\max_{\alpha_m} \quad \sum_{s^n \in S_0} \min_{\pi} \left[p\left(\mathbb{E}_{\pi^*,P}[v_{\pi^*}(s^n)] - \mathbb{E}_{\pi,P}[v_\pi(s^n)]\right)\right], \tag{2.42}$$

where we recall that v_π are functions of α_m, as shown in (2.38) and (2.39).

Finally, it is possible to define also a linear program for the case in which we do not have the expert policy function π^*, but rather, we have a set of samples trajectories following the expert policy, ζ_{π^*}, which is formed by successive states and actions obtained by interacting with the MDP following the expert policy. We assume that we also have access to a simulator of the MDP, so that we can generate trajectories ζ_π for an arbitrary policy π. We also assume that we know the initial state of the system, s^0. In this case, we can update (2.42) iteratively in order to obtain a reward function as Algorithm 7 shows [164], in which we alternate between optimizing a linear program to obtain a reward function from the samples; and in the second step, we use an RL procedure to obtain a policy that optimizes the reward function obtained previously. The linear program of the

Algorithm 7 Linear IRL using sample trajectories. A typical convergence criterion is the maximum number of iterations.

Input: An MDP simulator, s^0, ϕ_m basis functions
1: Generate a random policy π^0
2: Initialize $k = 0$
3: **while** Convergence criterion is not met **do**
4: Simulate the MDP from s^0 using the policy π^k, in order to obtain a set of trajectories ζ_{π^k}
5: Estimate v_{π^k} using (2.39) and ζ_{π^k}
6: Obtain α_m^k as the solution to (2.43)
7: Use an RL procedure to obtain an updated π^{k+1} such that maximizes the expected return over the reward function (2.38) with parameters α_m^k
8: Set $k = k+1$
Output: α_m^k

first step is:

$$\max_{\alpha_m^k} \quad \sum_{i=0}^{k} p\left(\mathbb{E}_{\pi^*,P}[v_{\pi^*}(s^n)] - \mathbb{E}_{\pi^k,P}[v_{\pi^k}(s^n)]\right), \tag{2.43}$$

where we note that (2.43) corresponds to a particularization of (2.42) for the case that we have only trajectories from the policies. Note that the value functions are estimated using (2.39), as indicated before. Also, note that we store the trajectories that correspond a previous policy, as the optimal policy should be better than any of them. The penalization function was defined in (2.41), and again, it is used to penalize the violations of the optimality conditions, which may happen due to the optimal reward vector not being contained in the subspace spanned by the basis functions used. We also note that the second step is key in the time budget needed for this approach, as in each iteration, a whole RL procedure needs to be done.

Thus, it is possible to use linear programming to solve the IRL problem for small and large or continuous state spaces, and even if we do not have access to the expert policy function, but rather to trajectories obtained with it. However, all these methods assumed that the expert policy was optimal in all circumstances, which needs not be the case in real life problems, as the expert may occasionally act suboptimally. In order to accommodate this possibility, we need to introduce the Maximum Entropy Principle (MEP), which has been key in recent IRL developments.

Maximum entropy principle

The Maximum Entropy Principle (MEP) is owed to Jaynes [112]. It is frequently used when we need to fit certain data to a distribution with the largest entropy. Intuitively, this means making the less possible assumptions about the distribution. Formally, let us assume that we have a discrete variable x_i, which may take I possible values, $i = 1, 2, ..., I$, and we do not know the probabilities of each x_i, $p_i = P(x_i)$. Jaynes was trying to address the problem in which he wanted to obtain the expectation of a function $f(x)$, which was also known:

$$\mathbb{E}_x[f(x)] = \sum_{i=1}^{I} p_i f(x_i), \tag{2.44}$$

and, as p_i is a distribution, we also know that p_i belong to a simplex, i.e., $p_i \geq 0$ and $\sum_i p_i = 1$. Clearly, this is a problem in which we lack information if $I > 2$, as we only know (2.44) and that $\sum_i p_i = 1$ for I variables.

In order to address this situation, Jaynes used the concept of entropy from information theory, because it provides a unique and unambiguous criterion to work with the amount of uncertainty that a discrete distribution

has. For Jaynes, entropy and uncertainty become synonyms, and the entropy takes the following form:

$$H(p_i) = -\sum_{i=1}^{I} p_i \ln(p_i). \tag{2.45}$$

The key idea proposed by Jaynes is that, when trying to determine a discrete, unknown distribution p_i, we must maximize the entropy, i.e., maximize the uncertainty provided by our distribution. This means that we are trying to find a probability distribution that minimizes the bias while agreeing with the given data, because a high entropy means having more uncertainty about the distribution. Thus, we have the following optimization problem:

$$\begin{aligned} \max_{p_i} \quad & -\sum_{i=1}^{I} p_i \ln(p_i) \\ s.t. \quad & \sum_{i=1}^{I} p_i = 1 \\ & \sum_{i=1}^{I} p_i f(x_i) = \mathbb{E}_x[f(x)] \end{aligned}, \tag{2.46}$$

where $p_i > 0$ is forced by having a logarithm in the function to be maximized. This problem can be solved using Lagrange multipliers, where the Lagrangian is:

$$\mathscr{L}(p_i, \lambda, \mu) = -\sum_{i=1}^{I} p_i \ln(p_i) - \lambda \left[\sum_{i=1}^{I} p_i - 1 \right] - \mu \left[\sum_{i=1}^{I} p_i f(x_i) - \mathbb{E}_x[f(x)] \right]. \tag{2.47}$$

We can then obtain the gradient of (2.47) with respect to p_i and solve to find a stationary point:

$$\nabla_{p_i} \mathscr{L}(p_i, \lambda, \mu) = -\ln(p_i) - 1 - \lambda - \mu f(x_i) = 0 \rightarrow p_i = e^{-1-\lambda-\mu f(x_i)}. \tag{2.48}$$

By computing the gradient with respect to λ, we obtain the first constrain:

$$\nabla_{\lambda} \mathscr{L}(p_i, \lambda, \mu) = \sum_{i=1}^{I} p_i - 1 = 0 \rightarrow \sum_{i=1}^{I} p_i = 1, \tag{2.49}$$

and combining (2.48) with (2.49), we obtain:

$$\sum_{i=1}^{I} e^{-1-\lambda-\mu f(x_i)} = e^{-1-\lambda} \sum_{i=1}^{I} e^{-\mu f(x_i)} = 1 \rightarrow p_i = \frac{e^{-\mu f(x_i)}}{\sum_{i=1}^{I} e^{-\mu f(x_i)}}, \tag{2.50}$$

which is the Boltzmann distribution with parameter μ. Hence, the maximum entropy distribution for the case in which we only know the mean of $f(x_i)$ is the Boltzmann distribution in (2.50).

Today, the MEP is used in settings in which the information provided is different from the expected value of a function $f(x)$. We note that the Boltzmann distribution is the maximum entropy distribution for the problem (2.46), but for different problems, where other information is available, different maximum entropy distributions arise, as can be seen in [139].

Maximum entropy inverse reinforcement learning

The MEP already described allows us dealing with situations in which the expert policy cannot be guaranteed to be optimal in all situations. The incoming sections come from a control theory perspective, and in this field, it is frequent to deal with costs instead of with rewards. We remind that these two models are equivalent: a reward function $R : S \times A \to \mathbb{R}$ has always an equivalent cost function $C : S \times A \to \mathbb{R}$, where $r(s,a) = -c(s,a)$. Thus, note that the agent seeks to maximize the reward function, or alternatively, minimize a cost function, but both problems are equivalent. Let us start by assuming that the cost is a linear combination of M features:

$$c(s,a) = \sum_{m=1}^{M} \alpha_m \phi_m(s,a), \tag{2.51}$$

where we approximate in the same way as in (2.38).

The key idea now is to have a set of trajectories ζ, and define the empirical expected feature count $\tilde{\phi}$ as an empirical average over the features ϕ_m when we have K trajectories ζ of length L as follows [262]:

$$\tilde{\phi} = \frac{1}{LK} \sum_{k=1}^{K} \sum_{l=1}^{L} \sum_{m=1}^{M} \alpha_m \phi_m \left(s^l, a^l | (s^l, a^l) \in \zeta_k \right), \tag{2.52}$$

where we note that the empirical feature count $\tilde{\phi}$ gives us information about how the features are distributed in our dataset. Thus, any candidate distribution $P(\zeta)$ needs to match this expected empirical feature count, which means that:

$$\sum_{k=1}^{K} P(\zeta_k) \frac{1}{L} \sum_{l=1}^{L} \sum_{m=1}^{M} \alpha_m \phi_m \left(s^l, a^l | (s^l, a^l) \in \zeta_k \right) = \tilde{\phi}. \tag{2.53}$$

As we have no more information about our distribution $P(\zeta)$, except the fact that $\sum_i P_i(\zeta) = 1$, we can use the MEP to derive a candidate distribution for $P(\zeta)$. Note that this is the same problem that we have posed in (2.46) and solved in (2.50): thus, the candidate distribution is a Boltzmann distribution of the form [262]:

$$P_\alpha(\zeta_k) = \frac{e^{-\frac{1}{L}\sum_{l=1}^{L}\sum_{m=1}^{M} \alpha_m \phi_m \left(s^l, a^l | (s^l, a^l) \in \zeta_k\right)}}{Z(\alpha)}, \quad Z(\alpha) = \sum_{k=1}^{K} e^{-\frac{1}{L}\sum_{l=1}^{L}\sum_{m=1}^{M} \alpha_m \phi_m \left(s^l, a^l | (s^l, a^l) \in \zeta_k\right)}, \tag{2.54}$$

where $Z(\alpha)$ is the partition function. Observe that evaluating the partition function may be very hard, even unfeasible, as it has to be done over the whole space of possible trajectories. However, this model is a first step towards dealing with suboptimal expert trajectories in IRL. Intuitively, the Boltzmann distribution means that the probability associated with suboptimal trajectories decreases exponentially as their cost increases.

An important remark has to do with the dynamical nature of the problem. Note that the entropy that we are trying to maximize is the conditional entropy, which is defined as follows for a sequence of states and actions between time steps 0 and N:

$$H(a^{0:N} | s^{0:N}) = \mathbb{E}\left[-\ln \left(\prod_{n=0}^{N} P(a^n | s^{0:N}, a^{0:n-1}) \right) \right], \tag{2.55}$$

where there is a significant problem with the conditional entropy: it requires to condition on the entire trajectory of states, from 0 to N. In [262], (2.54) can be applied because the transition probability is deterministic and known, and hence, it is possible to obtain the complete state trajectory, as it is an open loop problem. However, in general, the MDP transitions are stochastic and thus, the conditional entropy is not a feasible approach. In

order to overcome the requirement of knowing the whole state sequence to compute the conditional entropy, causal entropy has been used instead [124], which is defined as follows:

$$H(a^{0:N}||s^{0:N}) = \mathbb{E}\left[-\ln\left(\prod_{n=0}^{N} P(a^n|s^{0:n}, a^{0:n-1})\right)\right], \tag{2.56}$$

where the subtle but important difference between (2.55) and (2.56) is that in causal entropy, the conditioning is done exclusively on state values already known. In [261], it is shown that the maximum causal entropy distribution for the restrictions in (2.46) remains a Boltzmann distribution. The causal maximum entropy principle can also be extended to infinite horizon settings, where the discount factor γ would appear in the causal entropy formula [30], [259].

Maximum causal entropy inverse reinforcement learning with nonlinear cost

Even though linear approximations are a good starting point when we want to deal with large or infinite state spaces, as we noted in the RL discussion, they do not tend to achieve good results in practice, as they require carefully handcrafted features basis functions to provide good approximations. We may ask whether non-linear approximations could be used, such as approximations based on NNs. And indeed, it is possible to generalize the maximum causal entropy framework to work with non-linear cost functions. Let us assume that we parameterize the cost function using a non-linear approximation whose parameter vector is θ. In this case, the probability distribution for the sample trajectories ζ follows again a Boltzmann distribution [71]:

$$P_\theta(\zeta_k) = \frac{1}{Z}\exp(-c_\theta(\zeta_k)), \tag{2.57}$$

where Z is the partition function and c_θ is the non-linear cost function parameterized by θ. An important cost to be paid by using non-linear approximations is that the optimization problem that we have to solve now are not linear programs anymore, although it is possible to use an iterative procedure which resembles the method presented for the case of using a linear approximation from sample trajectories.

Let us assume that we have a policy function, which is parameterized by a parameter vector ω. Note that the policy may be approximated using linear or non-linear approximations. The IRL problem involves obtaining a reward function that best explains the behavior of an expert. Thus, the expert policy π^* should maximize the total cumulative reward, or equivalently, minimize the total cumulative cost. This process could be done iteratively with two main steps:

- In a first step, we update ω, so that the policy π_ω both maximizes the causal entropy of a set of expert trajectories, and also, minimizes the cost c_θ. Thus, note that we use π_ω to define a policy which is similar to the expert.
- In a second step, we update θ so that the difference between the cost induced by the policy expert π^* and the cost induced by the policy π_ω is maximized. In other words, we try to find a cost function that separates as much as possible the behaviors of π^* and π_ω. We remind that a similar idea to this was behind (2.37): heavily penalizing the differences between the expert policy and a similar policy.

Thus, we alternatively optimize on θ and ω in order to solve our optimization problem in two steps. Note that, when we update one parameter vector, the other remains fixed. Mathematically, this problem can be

expressed as follows [71]:

$$\max_{\theta} \left[\left(\min_{\omega} -H(\pi_\omega) + \mathbb{E}_{\pi_\omega}[c_\theta(s,a)] \right) - \mathbb{E}_{\pi^*}[c_\theta(s,a)] \right], \tag{2.58}$$

where $\mathbb{E}_\pi[c_\theta(s,a)]$ denotes the expected cost under policy π and, hence, it is the value function using costs instead of rewards, and $H(\pi)$ is the causal entropy of policy π.

Generative Adversarial Imitation Learning

The method just described to solve (2.58) presents an important drawback, namely, that the minimization step is done by using an RL algorithm. The RL procedure is computationally expensive when having large state and action spaces. In order to address this problem, [101] proposes an efficient way to solve (2.58), named as Generative Adversarial Imitation Learning (GAIL). GAIL is an efficient IRL approach based on approximating both the policy and the cost function using DNNs. It assumes that the IRL problem (2.58) has a solution and proposes using a Generative Adversarial Network (GAN) [82] to efficiently solve the IRL problem.

A GAN is a generative model which trains two NNs in an adversarial fashion. The first NN is called generator: it takes as input a random noise and produces an output which tries to match a certain distribution of which we have only samples. The second NN is called discriminator: it takes as input a sample which may have been generated by the generator or may belong to the actual data distribution, and tries to distinguish the origin of the sample. Both NNs are trained in an adversarial fashion, that is, the generator tries to improve its generative properties until it is able to fool the discriminator.

In GAIL, the generator approximates a policy π_ω with weights ω. Thus, it takes as input a state s and outputs the probability of using any action $a \in A$. The discriminator, on the other hand, is another NN parameterized by θ, $D_\theta : S \times A \to [0,1]$: it takes as input a (s,a) pair and outputs the probability that the input was generated by the expert policy or π_ω. Note that the cost function depends on the discriminator as follows:

$$c_\theta = \log(D_\theta(s,a)). \tag{2.59}$$

By training the GAN, GAIL obtains a saddle point (π_ω, D_θ) of the following expression:

$$\mathbb{E}_{\pi_\omega}[\log(D_\theta(s,a))] + \mathbb{E}_{\pi^*}[\log(1 - D_\theta(s,a))] - \lambda H(\pi_\omega), \tag{2.60}$$

where $\lambda \geq 0$. Note that (2.60) is equivalent to (2.58) with minor changes that allows an efficient implementation based on GANs. First, a step of TRPO is used to minimize (2.60) with respect to ω: this step improves the policy π_ω towards being similar to the expert policy. The motivation behind using TRPO is that is prevents the policy from changing too much between iterations because it restricts the maximum Kullback-Leibler divergence in the policies between iterations. And secondly, GAIL uses an Adam [122] step with respect to θ in order to maximize (2.60) with respect to D, where we remark that the cost function is (2.59). Note that again, a method based on two steps is used, but now we need not solving the RL problem multiple times. Rather, using a GAN architecture allows solving the RL algorithm only once, by alternatively taking TRPO and Adam steps, although from a convergence perspective it is recommendable training the generator several steps before updating the discriminator.

GAIL presents several advantages. First, it uses NNs to approximate both the cost and policy function, which allows these functions to be very expressive. We have already indicated that a problem that arises when using linear approximations is that the true reward function needs not be part of the subspace generated

by the basis functions. GAIL addresses this problem by making use of NNs, which are universal function approximators [105]. This means that GAIL could imitate arbitrarily complex behaviors from the expert. Second, GAIL can be used in large, even continuous, state and action spaces, as TRPO is able to deal with them. Third, it needs not solving an RL problem multiple times in order to perform the optimization. Due to these properties, we use GAIL in our problems when we need an IRL algorithm.

2.3 Partially Observable Markov Decision Processes

The previous discussion has assumed that the agent has access to the state in each iteration. However, in real life problems, it is frequent that the agent only has access to a partial or noisy observation of the state. In order to deal with this situation, the MDP framework needs to be extended to what is known as Partially Observable Markov Decision Process (POMDP), defined as follows [218]:

Definition 2 (Partially Observable Markov Decision Process)**.** *A POMDP is an 8-tuple* $\langle S, A, O, P, R, Z, b^0, \gamma \rangle$ *where:*

- S *is the state set, defined as in the MDP case.*
- A *is the action set, defined as in the MDP case.*
- O *is the observation set, containing all possible observations* $o \in O$ *that the agent observes.*
- P *is the transition probability function defined as in the MDP case.*
- $R : S \times A \to \mathbb{R}$ *is the reward function defined as in the MDP case.*
- $Z : S \times O \times A \to [0,1]$ *is the observation model if* O *is discrete and* $Z : S \times O \times A \to [0,\infty)$ *if* O *is continuous, where* $Z(o^{n+1}|s^n, a^n)$ *is the probability of observing* o^{n+1} *given that the agent is in state* s^n *and plays action* a^n*.*
- b^0 *is the initial belief, an initial distribution over* S *that denotes the belief of the agent concerning the initial state of the dynamical system.*
- γ *is a discount factor as in the MDP case.*

Note that the main difference between MDPs and POMDPs is that in an MDP, the agent observes the state of the system, but in a POMDP, the agent does not observe the state directly, but an observation related to the state. Thus, the agent is uncertain regarding the current state of the system. This causes the POMDP framework to be more complex than the MDP framework, because both the transition and reward function depend on the state, which the agent does not directly observe. Hence, the agent needs to estimate the current state of the system based on its previous observations and actions, and has to deal with the uncertainty derived of not knowing exactly the current state. Note that, in general, this implies that the optimal policy now depends on the whole history of actions and observations: $\pi(a^n|o^n, a^{n-1}, o^{n-1}, ..., a^0, o^0)$, instead of on the state only as in the MDP case. It also means that the complexity of solving POMDPs increases significantly when compared to solving MDPs: for some special cases, it is shown to have exponential complexity [40]. The contrast regarding the MDP case is notorious, as solving an MDP using DP is known to have a polynomial complexity [174].

A popular approach to solve a POMDP relies on the definition of a sufficient statistic known as belief [218]. This statistic is updated in each time step, and collects all the important information needed about the

history. Note that the history, by definition, is a sufficient statistic. We have the nice property that a POMDP is Markovian over the belief space. So, one approach consists in defining the belief, and in each time step, updating the belief and solving the resulting MDP using any of the techniques presented in the previous Section, replacing the state with the belief. However, as shown in [218], this approach can be computationally costly even in simple cases. An alternative representation consists in using Predictive State Representation [142], which tries to predict future possible trajectories of observations and actions.

There have also been many proposed algorithms to solve approximately a POMDP, such as [94], [144], [186], [93] and [192], to mention some. However, a very powerful approach has recently been proposed in [92], known as Deep Recurrent Q-Networks (DRQN). It consists in modifying the DQN algorithm in order to be able to work with the history of observations and actions, by using Recurrent Neural Networks (RNNs), which we proceed to introduce.

Recurrent Neural Networks

FNNs can be modified to include feedback from the past, giving way to RNNs. These are specially designed to process sequential data [81], as they are able to remember past information. The memory is achieved by updating, as new data arrives, an internal state, which is then concatenated to the NN input. Hence, the output now is not only a function of the NN input data, but also a function of the state, which contains information about the past. One of the most popular architectures to implement an RNN is the Long-Short Term Memory (LSTM) architecture [104]. In an LSTM, there is a cell state c^n that is updated using the data input x^n and the output of the LSTM, denoted by y^n. The superscript n is used to denote the time index. An LSTM is formed by four different NNs, with weights w_i and biases b_i, $i \in \{1,2,3,4\}$ (2.27). These four NNs are used to update y^n and c^n as follows:

- First, c^n is updated with the following expression:

$$c^n = Sigm\left(w_1 \cdot [x^n, y^{n-1}] + b_1\right) \cdot c^{n-1} + Sigm\left(w_2 \cdot [x^n, y^{n-1}] + b_2\right) \cdot Tanh\left(w_3 \cdot [x^n, y^{n-1}] + b_3\right), \quad (2.61)$$

  where *Sigm* denotes the sigmoid function, *Tanh* the hyperbolic tangent function and $[a,b]$ the concatenation of the vectors a and b. Note that the cell state c^n is updated using the previous cell state c^{n-1}, the previous LSTM output y^{n-1} and the current input x^n. The first term in (2.61) is called the forget term: the sigmoid function outputs a nonlinear combination of the current input and previous output in the range $(0,1)$. By multiplying this term element-wise with the previous cell state, we are determining which elements from the previous cell state are forgotten.

  The second term in (2.61) intuitively controls what new information we are adding to the cell state. Note that the hyperbolic tangent term could be considered the new information that the LSTM wants to add to the cell state, whereas the sigmoid term controls again how much of that information will be added to the cell state. Thus, the cell state update consists of two main terms: the first controls how much information from the previous state cell is remembered, and the second how much information from the current input and previous output we are adding to the state cell to remember in the next time steps.

- Second, we obtain the output to the LSTM using the following expression:

$$y^n = Sigm\left(w_4 \cdot [x^n, y^{n-1}] + b_4\right) \cdot Tanh\left(c^n\right), \quad (2.62)$$

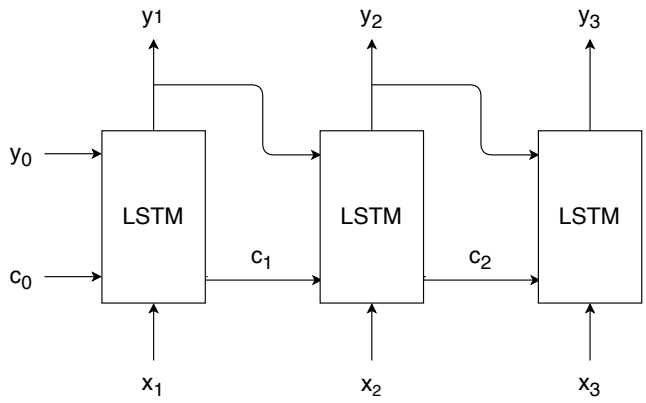


Fig. 2.3 Illustration of the procedure of an LSTM for three time steps. The output y^n is updated in each time step using (2.62) and the cell state c^n is updated using (2.61). The LSTM block is composed of four neural networks, which are the same for all time steps. Note that, in the first time step, it is necessary to provide an initial c^0 and y^0 in order to obtain c^1 and y^1.

where the input depends on the current input and cell state, and the previous output. Note that the cell state is updated using (2.61) prior to obtaining the LSTM output using (2.62). In addition, note that (2.62) shows that the output is a filtered version of the current cell state.

It is important to note that, in each time step, the weights and biases of the four NNs that compose the LSTM architecture are the same. The training of RNNs is different from FNNs, as now the time needs to be taken into account as well, and a modified algorithm known as backpropagation through time is used [235]. This algorithm computes the gradient for the weights and biases, not only taking into account the current time step, but also the previous ones, unrolling the LSTM similarly as shown in Figure 2.3. Thus, sequences of input data are used for training an LSTM.

If FNNs were able to approximate any function, RNNs are able to approximate any Turing machine [205], which makes them suitable for modeling dynamical systems. The main problem with RNNs is that these networks are hard to train, hence, there has been a significant effort addressed to alleviate this problem [214]. Nowadays, the main architecture used to implement an RNN is the LSTM already presented, although other structures, such as Gated Recurrent Unit (GRU), have been proposed recently [42], however, their advantages over LSTM are not clear [85].

Deep Recurrent Q-Networks

After DQN came out, there were several algorithms that proposed changes in the original DQN algorithm in order to adapt it to other environments. A very interesting one is based on Deep Recurrent Q-Networks [92]. It consists on using DQN algorithm with a recurrent NN, namely an LSTM, to approximate the Q function. The main intuition behind this approach is that RNNs allow taking into account the past information, and hence, they take into account the whole history. To put it differently, the RNN internal state is used to store the past information, and hence, it is similar to the belief concept. As shown by [92], DRQN can successfully solve POMDPs with many states and actions, and due to being based on DQN, it works with continuous state spaces and discrete actions. This work then was applied in many Deep RL algorithm using RNNs in order to better address the partial observation that arises in many problems.

However, as we noted before, the main problem with using RNNs is that they can be significantly hard to train. Thus, today there are two main ways to approximate a POMDP by using Deep RL tools, both of which are used in this work and in current literature:

1. Using Recurrent Neural Networks (RNNs), which are able to store past information as we have explained. This solution provides a better approximation at the cost of having a significantly larger computational cost and a more difficult training process.

2. Using a finite vector of past observations as input to an MDP Deep RL method. In other words, we use as state a truncated history vector, where we include past information so that the Deep RL method may take it into account. As [92] shows, this solution is an approximation for solving POMDPs, but it can provide very good results [154].

2.4 Swarms

Until this point, we have assumed that we have a single agent interacting with a dynamical system, which may have either a perfect knowledge of the state or a partial observation. Now, in this section, we turn our attention to the case in which we have more than one agent, but all of them share the same reward function. This is a first step that allows solving the situation in which there are several agents cooperating among them to reach a common goal, while we reserve the more general case in which we have several agents with different interests to the next Section.

Of special interest is the concept of swarm, which is a group of agents which cooperate among them to obtain a common goal in a decentralized fashion. Swarms are inspired by nature, in which small insects are able to cooperate among them to obtain a desired objective that they could not have obtained otherwise. For instance, several ants cooperate to carry a burden that a single ant could not take. Hence, a swarm is a multi agent system, where each agent is simple but by means of cooperation, the system can achieve complex goals. This problem, as we will see, can be modeled under the distributed control perspective: there is a single reward function to be optimized and the action vector has as components the individual actions of several agents.

2.4.1 Dec-POMDP

A first model that allows dealing with many agents and a single reward is the Decentralized Partially Observable Markov Decision Process (Dec-POMDP), a multi-agent model defined as follows [24], [236, Ch. 15], [173]:

Definition 3 (Decentralized POMDP)**.** *A Dec-POMDP is a 9-tuple* $\langle N_p, S, A, O, P, R, Z, b^0, \gamma \rangle$ *where:*

- N_p *is the index set of the agents, where the subscript* $i = 1, 2, ..., N_p$ *indexes each of the* N_p *agents.*
- S *is the state set, defined as in the MDP case.*
- A *is the action set. Each agent* i *has its own action set* A_i *and* $A \triangleq A_1 \times A_2 \times ... \times A_{N_p}$ *is the product of the action spaces of each player. The joint action* $a = (a_1, a_2, ..., a_{N_p}) \in A$ *is a vector formed by the actions of all agents.*
- O *is the observation set. Each agent* i *has its own observation set* O_i *and* $O \triangleq O_1 \times O_2 \times ... \times O_{N_p}$ *is the product of the observation spaces of each player. The joint observation* $o = (o_1, o_2, ..., o_{N_p}) \in O$ *is a vector formed by the observations of all agents.*

- *P is the transition probability function defined as in the MDP case. Note that P depends on a, the joint action vector.*
- *R is the reward function defined as in the MDP case. Note that R depends on the joint action vector a.*
- *Z is the observation model, defined as in the POMDP case. Note that Z uses the joint action and observation vectors a and o respectively.*
- b^0 *is the initial belief as in the POMDP case.*
- γ *is a discount factor as in the MDP case.*

Observe that we use subindexes for agents and superindexes for time. Note that a Dec-POMDP generalizes the POMDP framework to a multi agent setting with a common goal, because there is a single reward function common to all agents. Hence, it is a convenient model for swarms. However, this is a model very hard to solve: in the best case, they are NEXP-hard, while MDPs are P-hard and POMDPs PSPACE-hard [24]. This means that MDPs can be solved in polinomial time, POMDPSs using a Turing machine with polynomial amount of memory and Dec-POMDPs could be solved by a Turing machine in exponential time. However, in the worst case, POMDPs are NEXP-complete, which are the hardest problems in the NEXP class. Hence, having several agents comes at the cost of a significant increase in the problem complexity and tractability. There are several algorithms proposed for solving Dec-POMDPs, some of them based in value functions like [173] and a more recent and efficient algorithm was proposed in [61]. Even though the algorithm in [61] surpasses its predecessors, it works in very limited time horizons. Hence, the main drawback of Dec-POMDPs lies in the complexity associated to solving them. We note that, as Dec-POMDP is a model that generalizes the POMDP model, there exists also a Dec-MDP model [24], which generalizes MDP. Dec-MDP model is similar to Dec-POMDP model except that the agents observe the state directly. As we do not use them in our problems, we do not introduce them.

2.4.2 swarMDP

In order to alleviate the complexity in the Dec-POMDP model, a particularization of this framework for swarms has been proposed in [211]. The model proposed is known as swarMDP:

Definition 4 (swarMDP)**.** *A swarMDP is defined in two steps. First, we define a prototype* $\mathbb{A} = \langle S, A, O, \pi \rangle$*, where* $\mathbb{A}$ *is an instance of each agent of the swarm and where:*

- *S is the set of local states.*
- *A is the set of local actions.*
- *O is the set of local observations.*
- π *is the local policy.*

A swarMDP is a 7-tuple $\langle N_p, \mathbb{A}, P, R, Z, \gamma \rangle$ *where:*

- N_p *is the index set of the agents as in the Dec-POMDP case.*
- $\mathbb{A}$ *is the agent prototype defined before.*

- *P is the transition probability function defined as in the MDP case. Note that P depends on* $a = (a_1, a_2, ..., a_{N_p})$*, the joint action vector, as in the Dec-POMDP case.*
- *R is the reward function defined as in the MDP case. Note that R depends on* $a = (a_1, a_2, ..., a_{N_p})$*, the joint action vector, as in the Dec-POMDP case.*
- *Z is the observation model, defined as in the POMDP case. Note that Z uses* $a = (a_1, a_2, ..., a_{N_p})$*, the joint action vector, and* $o = (o_1, o_2, ..., o_{N_p})$*, the joint observation vector, as in the Dec-POMDP case.*
- γ *is a discount factor as in the MDP case.*

Note that the main difference between the swarMDP and the Dec-POMDP model lies in the fact that the swarMDP explicitly makes all agents equal: they share the same set of local states, actions, observations and policies. Whereas under the Dec-POMDP model each agent could have a different action and/or observation set, under the swarMDP model all agents share the same local states, actions, observations and policy. Due to this characteristic, which is called the homogeneity property, the agents are interchangeable. Also, note that a single agent swarMDP reduces to a POMDP.

2.4.3 Mean embeddings

The homogeneity property simplifies the problem of learning under the swarMDP model, as the order of the agents does not matter to the learning process because they are interchangeable. Note that all agents share the same policy due to the homogeneity property, which means that each agent would act similarly as the others if they observed the same observation o^n. Thus, we can use single agent Deep RL algorithms and a centralized training/decentralized execution method to find a policy that maximizes the cumulative reward [110]. For instance, we could use TRPO and train a single policy π_ω for all agents, which takes as input the local observation of each agent $i \in N_p$, o_i^n and outputs a local action a_i^n for time step n. During training, the local observations of each agent are sent to the central learning algorithm for training. During execution, each agent uses a copy of the learned policy with their own local observation. Note that, as agents do not observe the state because we are in a partial observation environment, we either need to use a recurrent NN for the policy π_ω or include in the observation enough information about the past, as explained in the POMDP Section.

We finally note that the observation vector of agent i may include not only information about agent i, but also information about other agents if the agents are able to communicate. Let us denote by $o_{i,i}^n$ the information available to agent i about itself in time n, and $o_{i,j}^n$ the information available to agent i about agent j, $j \neq i$. A naive way of encoding this information is to build the total observation vector of agent i, o_i^n by simply concatenating $o_{i,i}^n$ and all the vectors $o_{i,j}^n$. However, this concatenation causes a large input space, as well as ignoring the permutation invariance inherent to a homogeneous swarm. A better option consists in using mean embeddings [209], [110]. There are several possible mean embeddings that can be used:

- Neural Networks Mean Embedding (NNME): In this approach, each $o_{i,j}^n$ is used as input to an NN, which outputs $\phi(o_{i,j}^n)$, where ϕ denotes the transformation done by the NN. The total observation vector of agent i, o_i^n, is built by concatenating $o_{i,i}^n$ to the mean of all $\phi(o_{i,j}^n)$. That is, we obtain the features using the NN first, and use the averaged value of such features as input to the policy. A very interesting characteristic of this approach is that it allows training the mean embedding NN together with the policy, and hence, this NN will be trained to extract the information that the policy needs, thus being able to adapt to a concrete problem setup.

- Mean-based Mean Embeddings (MME): Under this approach, we average $o^n_{i,j}$ and concatenate it to $o^n_{i,i}$. This vector is the input to the policy network. Note that this case is a simpler approach, in which we do not perform any feature extraction process and which does not adapt to a concrete setting. Its success will strongly depend on how informative the average values of the observation vectors are for the policy.

Note that mean embeddings are thus insensitive to the number of agents to which each agent i can communicate and is also insensitive to their order. We remark that this tool allows us to solve swarMDP problems by making use of the single agent methods presented in previous Sections.

2.5 Game theory

In this section, we turn our attention to the case in which there are several agents, each of them with possibly different reward functions. That is, each agent may have different interests, and the actions that each of them chooses affect the rewards of the rest of the players. The branch of mathematics that studies this problem is Game Theory (GT). Currently, GT is a mature field with many important works covering different aspects of the theory, such as [76], [19], [146] or [150]. Closely related to this field is the field of Multi-Agent Learning (MAL), which seeks to find optimal policies for agents that are interacting with others. Even though it is possible to pose the MAL problem from the GT perspective [204], we note that MAL can be studied using other tools than game theory ones [213].

In this Section, we start by presenting the simple case in which the agents interact a single time, known as static game. We then move on to study the case in which the agents interact repeatedly over time, introducing three models that take into account the time, with different complexities: repeated games, stochastic games and partially observable stochastic games. And finally, we introduce the concepts of imperfect and incomplete information, which will be of capital interest in the incoming Chapters.

2.5.1 Static games

We start studying the case in which the agents interact a single time. We introduce minor notations changes regarding the previous Sections: an agent may be called also a player, and the reward function may be called also outcome or payoff. Keeping that in mind, a static game is defined as follows:

Definition 5 (Static game)**.** *A static game is a* 3*-tuple* $\langle N_p, A, r\rangle$*, where:*

- N_p *denotes the number of players, where the subscript* $i = 1,2,...,N_p$ *indexes each of the* N_p *players.*
- A *is the action set. Each player* i *has its own action set* A_i *and* $A \triangleq A_1 \times A_2 \times ... \times A_{N_p}$ *is the product of the action spaces of each player. The joint action* $a = (a_1, a_2, ..., a_{N_p}) \in A$ *is a vector formed by the actions of all players.*
- r *is a continuous function that gives the game payoffs as:*

$$r : A \to \mathbb{R}^{N_p} \tag{2.63}$$

where $r_i(a_i, a_{-i})$ *denotes the payoff or reward obtained by player* i *when she plays action* a_i *and the rest of the players play* a_{-i}*, where the index* $-i$ *is a shorthand for all players except* i*.*

In this Section, we consider discrete sets of actions. Hence, the payoff functions will be also discrete. If an action set is discrete, each of its components is denoted as pure action. Mixed actions are distribution

probabilities for each player that map each pure action of the player with the probability that the player plays that action.

Observe that the payoff functions denote the nature of the game. It can be purely competitive: the gains of some players are the losses of the others and thus, $\sum_i r_i = 0$, which is the case known as zero-sum game. It can be purely cooperative if all players share the same payoff function. It can finally range between these two extreme cases: these games are known as general-sum or nonzero sum games.

Nash equilibrium concept

We note here that static games can be seen as a generalization of the optimization field, in which we have several agents trying to maximize their reward. However, now it is possible that, as the interest of agents may be in conflict, they are not able to attain their maximum reward, and a different solution concept is needed for games: the concept of equilibrium. There are several equilibrium notions, and in this work, we introduce two of them: the Nash Equilibrium (NE), and the Correlated Equilibrium (CE).

A Nash equilibrium (NE) [161] of an N_p-player game is an action vector such that no player can gain by deviating unilaterally. Mathematically, for a static game, an action vector a is a Nash ε_i-equilibrium of the game G, where $\varepsilon_i \geq 0, \forall i \in N_p$ if:

$$r_i(a_i, a_{-i}) \geq r_i(a_i', a_{-i}) - \varepsilon_i, \quad \forall i, \forall a_i' \neq a_i, \tag{2.64}$$

where a_i denotes the possibly mixed action of player i and A_{-i} the action of all players but player i. When $\varepsilon_i = 0, \forall i$, we have an NE. A nonzero sum game is guaranteed to have at least one NE in mixed actions [19], which assumes that each player has access to a randomizing device which outputs which action the player should play with a given probability. This probability is the mixed NE. In general, nonzero sum games may become hard to solve and might have more than one NE [19, Ch. 3]. A general review on these games and the different algorithms proposed for finding equilibria can be found in [149].

We now restrict our NE analysis to the two player case, and the literature shows that this is still an area of research. In [226] there are some methods used to compute two player NE, and in [16], the authors present two different methods to derive all the NE in a two player game. From here on, we focus on the concrete case that $N_p = 2$ and both players can choose between two possible actions. Hence, the payoff function for each player might be represented using a 2×2 matrix, and that is the reason these games are sometimes known as bimatrix games. If we denote the payoff matrix for player 1 as R_1 and R_2 for player 2, we have:

$$R_1 = \begin{pmatrix} r_{1,11} & r_{1,12} \\ r_{1,21} & r_{1,22} \end{pmatrix} \quad R_2 = \begin{pmatrix} r_{2,11} & r_{2,12} \\ r_{2,21} & r_{2,22} \end{pmatrix}, \tag{2.65}$$

where $r_{a,bc}$ denotes the reward that player a receives when player 1 plays her action b and player 2 plays her action c. Hence, note that we consider that the row player is the player 1, while the player 2 is the column player. As the payoffs contained in these matrices are rewards, each player tries to maximize her own payoff. As mentioned in (2.64), if we only take into account pure actions, the payoffs $(r_{1,j^*k^*}, r_{2,j^*k^*})$ in pure actions NE must satisfy the following conditions:

$$\begin{matrix} r_{1,j^*k^*} \geq r_{1,jk^*} \\ r_{2,j^*k^*} \geq r_{2,j^*k} \end{matrix}, \tag{2.66}$$

where j and k index the actions of player 1 and 2 respectively. The existence of a pure action NE depends on the reward matrices, and note that such NE needs not exist. However, we know that there exist at least a mixed actions NE. Let us define y as the probability that player 1 chooses her action 1, and $1-y$ the probability that she chooses action 2; and for player 2, we define in an equivalent form z and $1-z$. It is possible to refer to y as the mixed action of player 1, and z as the mixed action of player 2. Using this, the conditions in (2.66) can be expressed as follows:

$$\begin{aligned} (y_v^*)^T R_1 z_v^* &\geq y_v^T R_1 z_v^* \\ (y_v^*)^T R_2 z_v^* &\geq (y_v^*)^T R_2 z_v \end{aligned}, \tag{2.67}$$

where $y_v = (y, 1-y)$, $z_v = (z, 1-z)$ and y_v^T denotes the transposed of vector y_v. In both (2.66) and (2.67) it is assumed that both players are maximizers: otherwise, the inequality should be reversed. The system of inequalities in (2.67) can be solved as shown in [19, pp. 85-87] and, by using the expressions in (2.65), it yields the following values for an NE:

$$\begin{aligned} y^* &= \frac{r_{2,22} - r_{2,21}}{r_{2,11} + r_{2,22} - r_{2,21} - r_{2,12}} \\ z^* &= \frac{r_{1,22} - r_{1,12}}{r_{1,11} + r_{1,22} - r_{1,21} - r_{1,12}} \end{aligned}, \tag{2.68}$$

where these expressions yield a valid solution if $y^* \in [0,1]$ and $z^* \in [0,1]$. These expressions are the same whether the players are minimizing or maximizing. It is interesting noting that each player optimizes only taking into account the other player's payoff matrix.

A final consideration on (2.68) arises when the denominator becomes null, that is, either $r_{2,11} + r_{2,22} - r_{2,21} - r_{2,12} = 0$ or $r_{1,11} + r_{1,22} - r_{1,21} - r_{1,12} = 0$. Substituting these conditions in (2.67) shows that the NE will be in pure actions, as $y, z \in \{0,1\}$, which means that all the probability is assigned to one of the actions that a player can choose.

Correlated equilibrium concept

The CE concept, owed to Aumann [13], generalizes the NE concept. The CE is based on the possibility that players might communicate and coordinate themselves. It assumes that there is a correlating device that produces a signal sent to all players. They use this signal in order to coordinate and obtaining a higher reward than if they did not coordinate. Each signal of the correlating device corresponds to a pure action for each player. The CE is defined so that no player has any advantage if she deviates from the prescription of the correlating device. More formally, a CE for N_p players is defined as a distribution probability $\phi(a)$ over the set of joint pure actions of the players $A = A_1 \times A_2 \times ... \times A_{N_p}$, where $a = (a_1, a_2, ..., a_{N_p})$ is a vector of pure actions such that $a \in A$. The equilibrium condition that must be satisfied for every player $i \in N_p$ is [13] [76]:

$$\sum_{a \in A} \phi(a) r_i(a_i, a_{-i}) \geq \sum_{a \in A} \phi(a) r_i(a_i', a_{-i}) - \varepsilon_i, \quad \forall a_i' \in A_i, \quad a_i \neq a_i', \tag{2.69}$$

where the distribution ϕ is the ε_i-CE of the game. For $\varepsilon_i = 0, \forall i$, we have a CE. The signal that coordinates the players follows the distribution $\phi(a)$ and the equilibrium is reached if no player has an advantage by deviating.

An equivalent way of expressing (2.69) is found in [76] and is the following, for $\varepsilon_i = 0$:

$$\sum_{a_{-i} \in A_{-i}} \phi(a_{-i}|a_i) r_i(a_i, a_{-i}) \geq \sum_{a_{-i} \in A_{-i}} \phi(a_{-i}|a_i) r_i(a'_i, a_{-i}), \quad \forall a'_i \in A_i, \quad a_i \neq a'_i, \tag{2.70}$$

where A_{-i} is the set of joint pure actions of all players except player i. This way of expressing a CE highlights that it is not profitable for player i to deviate, if every other player follows the action prescribed by the correlating device.

CE are a generalization of NE: every NE will have an equivalent CE, but not all CE have an equivalent NE. However, CE are significantly less expensive to compute than NE, as it is shown in [79] and [80]. In general, CE yield a region of valid equilibrium payoffs, as a function of the distribution $\phi(a)$. Hence, the problem of selecting an equilibrium point arises, as in the NE case. This problem is known as bargaining in literature and there are different mechanisms proposed for equilibrium selection, as the Nash bargaining solution [160], the Kalai-Smodorinsky solution [118] or the egalitarian solution [117].

Regret Matching

The Regret Matching (RM) algorithm, proposed by Hart and Mas-Colell [90] [91], is an algorithm which is used to learn the CE of a game by playing it several times. The players adapt their strategy in such a way that guarantees that the joint distribution of play converges to the set of CE of the underlying game if each player plays a regret matching strategy [91].

The rule that governs how a player plays according to an RM strategy is: “switch next period to a different action with a probability that is proportional to the regret for that action, where regret is defined as the increment in the payoff had such a change always been made in the past" [91]. Hence, the main idea is to keep and update a regret measure for each player, which is a vector with as many components as the number of pure actions of the player. The game is played repeatedly over time, and each player updates his regret depending on the outcome they obtain each time they play. This algorithm requires that player i knows the actions of all players, as well as her payoff function, but note that player i needs not knowing the payoff functions of the rest of the players. The regret that each player has after playing is the difference between the payoff they would have obtained if they had played differently and what they actually obtained. In each iteration, the regret $W_i(a'_i)$ is obtained as:

$$W_i(a'_i) = r_i(a'_i, a_{-i}) - r_i(a_i, a_{-i}), \quad \forall a'_i \in A_i, \tag{2.71}$$

where a_i is the pure action played by player i, a'_i is used to denote all pure actions available to player i and A_i is the set of pure actions for player i. Obviously, the regret of playing action a_i is 0. If the regret of an action is positive, RM will assign positive probability to that action to be played, because the player would have gained in the past if she had used this action. On the other hand, if the regret of an action is negative, the player will assign probability 0 to that action, as the player would not have improved her payoff by using that action in the past and hence, she will not use it. At the beginning of the game, the regrets are initialized to 0, and they are updated with each repetition of the static game following:

$$W_i^{n+1}(a'_i) = W_i^n(a'_i) + W_i(a'_i), \quad \forall a'_i \in A_i, \tag{2.72}$$

where $W_i^n(a'_i)$ is the regret at the beginning of the previous iteration n and $W_i(a'_i)$ is obtained using (2.71). At the beginning of each time step n, each player chooses a pure action randomly following a distribution

$p_i(a_j)$, where $p_i(a_j)$ is the probability that player i uses pure action a_j. The probability $p_i(a_j)$ is obtained at the beginning of each time step as follows:

- If all regrets are less or equal than zero, then choose a pure action randomly following the uniform distribution $p_i(a_j)$:

$$p_i(a_j) = \frac{1}{|A_i|} \tag{2.73}$$

  where $|A_i|$ stands for the number of pure actions available to player i.

- If there are regrets strictly higher than zero, then choose a pure action randomly following this distribution $p_i(a_i)$:

$$p_i(a_j) = \begin{cases} \frac{W_i^n(a_j)}{W} & \text{if} \quad W_i^n(a_j) > 0 \\ 0 & \text{if} \quad W_i^n(a_j) \leq 0 \end{cases} \tag{2.74}$$

  where

$$W = \sum_{a_j \in A_i | W_i^n(a_j) > 0} W_i^n(a_j) \tag{2.75}$$

  that is, W is the sum of all positive regrets in time step n. Observe that W is computed in each time step, as the vector W_i^n is updated in each time step. This definition of W guarantees that the distribution in (2.74) adds up to 1 and has nonnegative components, which are the two requisites to have an actual distribution.

2.5.2 Repeated games

The simplest way of taking time into account in game theory consists in playing a static game several times sequentially. This is the simplest dynamic game, known as Repeated Game (RG) [76], [146]. Formally, an RG is built using a static game, which is played repeatedly over N periods. This static game is called stage game. In an RG, the stage game is played on the periods $n \in \{0,1,2,...,N-1\}$. We consider only RGs of infinite horizon, that is, $N = \infty$. We define an RG as:

Definition 6 (Repeated game)**.** *An RG is a* 6*-tuple* $\langle N_p, A, r, \sigma, \mathscr{H}^t, \gamma \rangle$ *where:*

- N_p, A *and* r *are defined as in the static game.*
- $\mathscr{H}^n$ *denotes the set of histories. A history* $h^n \in \mathscr{H}^n$ *is a list of n-action profiles played in periods* $\{0,1,...,n-1\}$.
- *A strategy for player i is a mapping from the set of all possible histories into the set of actions:* $\sigma_i : \mathscr{H} \rightarrow A_i$.
- *The average discounted payoff to player i, is given by:*

$$V_i(\sigma) = (1-\gamma) \sum_{n=0}^{\infty} \gamma^n r_i^n(\sigma_i, \sigma_{-i}), \tag{2.76}$$

  where $\gamma \in (0,1)$ *is the discount factor. In case of having mixed actions, the average discounted payoff is computed taking expectations.*

We focus on RGs of perfect monitoring [146]: the history h^n is known to all players. In case of mixed actions, this means that the output of the randomizing device of each player is also observed by other players

[4]. Note that payoff is normalized with the term $1-\gamma$, which allows comparing payoffs in the RG with the ones in the stage game. Also, observe that static games are concrete cases of RGs, namely, when $N=1$. We consider only games with equal discount factor for all players.

Finally, similarly to previous Sections, it is possible to work with an average payoff instead of a discounted one, although that case is out of the scope of this work. Discounted payoff schemes are generally more realistic than average payoff ones: the discount factor might reflect the balance between present and future rewards and also, might be used as a measure on the uncertainty in the length of the game, whereas average payoff scheme assumes that the game duration is known, which in certain environments might not be a realistic assumption [123]. The difference between both payoff schemes acutes when the discount factor is low, i.e., γ is not close to 1. In Chapter 3, we discuss some differences among these two ways to obtain the total payoff and their impact in our problems.

Subgame perfect equilibrium

An NE is a set of actions such that is the best response to the strategies of other players, as we saw in static games. That concept can be extended to RGs as follows [146]:

Definition 7 (Nash equilibrium)**.** *A strategy profile σ is an NE of an RG if $\forall i$ and $\forall \sigma_i'$, $V_i(\sigma_i,\sigma_{-i}) \geq V_i(\sigma_i',\sigma_{-i})$*
,

where the main difference with the static case is that the NE is now defined in terms of the averaged discounted payoff (2.76). Note that, as the game is repeated, there are many possible strategies, that is, many different action paths that may be taken by the players. Hence, we are now interested in sequences of actions that provide an equilibrium to all players. Regarding previous sections, strategy is the same concept as policy, although strategy is preferred in the GT field.

In RGs, the NE concept is strengthened by imposing additionally the sequential rationality requirement: the behavior followed by the players must be optimal in all circumstances [146]. Hence, the NE concept is refined and gives place to the Subgame Perfect Equilibrium (SPE) concept: a strategy profile σ is an SPE it is an NE for every possible subgame of the RG. The task of checking whether a concrete strategy profile σ is an SPE might become intractable: there are infinity possible deviations to analyze. In order to simplify this task, the one-shot deviation principle is used, where a one-shot deviation for player i from strategy σ_i is a strategy $\hat{\sigma}_i \neq \sigma_i$ such that there is a unique history $\tilde{h}^n \in \mathscr{H}$ such that for all $h^\tau \neq \tilde{h}^n$, $\sigma_i(h^\tau) = \hat{\sigma}_i(h^\tau)$. That is, the strategy $\hat{\sigma}_i$ agrees with the original strategy everywhere except at one point where the deviation occurs. Yet this deviation can have significant impact on the outcome. A one-shoot profitable deviation is defined as:

Definition 8 (One-shot profitable deviation)**.** *A one-shot deviation $\hat{\sigma}_i$ from strategy σ_i is profitable if, at the history h^n for which $\hat{\sigma}_i(h^n) \neq \sigma_i(h^n)$ and for a fixed σ_{-i},*

$$V_i(\hat{\sigma}_{i|h^n},\sigma_{-i|h^n}) > V_i(\sigma_{i|h^n},\sigma_{-i|h^n}), \quad \forall \sigma_{-i},$$

where the main interest of the concept of profitable deviation is the following Lemma, whose proof is in [146]:

Lemma 3 (The one-shot deviation principle)**.** *A strategy profile σ is subgame perfect if and only if there are no profitable one shot deviations.*

In order to simplify even more the analysis of RGs, it can be noted that the histories can be grouped into equivalence classes: each history belonging to a concrete equivalence class induces an identical continuation strategy. This allows describing a strategy using an automaton $(\mathscr{W}, w^0, f, \tau)$, where:

- $\mathscr{W}$ is a set of states, where each state represents an equivalence class.
- $w^0 \in \mathscr{W}$ is the initial state.
- $f : \mathscr{W} \to A$ is an output or decision function that maps states to action profiles. It happens that $f(h^n) = \sigma(h^n)$.
- $\tau : \mathscr{W} \times A \to \mathscr{W}$ is a transition function that identifies the next state of the automaton as a function of the present state and the realized action profile. It happens that $\tau(h^n, a) = h^{n+1}$. A state is accessible from another state if the transition function links both with some action.

Note that this automaton is a Markov Process, as the transition depends only on the current state. The advantage of using an automaton is that often, the set of states $\mathscr{W}$ is finite, whereas the set of histories is not. The set $\mathscr{W}$ is a partition on $\mathscr{H}$, grouping together the histories that lead to identical continuation strategies. Using the automaton definition shown before, we can recursively define the averaged discounted payoff for player i in a game that starts in state w using Bellman's equation as:

$$V_i(w) = (1-\gamma) r_i(a) + \gamma V_i(\tau(w,a)), \tag{2.77}$$

where we assume pure action strategies: in case of using mixed strategies, we would take mathematical expectations in equation (2.77). The Bellman formulation allows solving the RG as if they were static games, as the next Lemma, whose proof is in [146], shows:

Lemma 4. *Suppose that a strategy profile σ is described by an automaton $(\mathscr{W}, w^0, f, \tau)$. The strategy profile σ is a subgame perfect equilibrium if and only if for all $w \in \mathscr{W}$ accessible from w^0, $f(w)$ is a Nash equilibrium of the normal form game described by the payoff functions $g_w : A \to \mathbb{R}^{N_p}$ where*

$$g_{w,i}(a) = (1-\gamma) r_i(a) + \gamma V_i(\tau(w,a)).$$

In other words, we can test a strategy σ by obtaining the equivalent normal form game, i.e., static game, which uses payoffs g_w and checking for existence of NE. Note that we assume that the strategy profile is described by the automaton: hence, we use Markovian strategies over the states w. Unluckily, Lemma 4 only provides us with a way to check whether a strategy is an SPE, but it does not define a way to find such strategies. Note that one possible candidate strategy would be always playing a static NE of the stage game. Lemma 4 shows that the players would obtain their stage Nash payoff, independently of the value of γ. Hence, we have the same payoff that we had in the static case, or in other words, the stage NE is also an SPE in the RG. We note here that, by a slight abuse of notation, we use indistinctly NE and SPE to refer to the SPE in RGs in the rest of the work.

However, this payoff could be improved, as a family of results known as Folk theorems asserts [146, Ch. 3], [76, Ch. 5]. Folk Theorem is the informal name that was given to a set of results which were widely known among game theory researchers prior to their formal publication. Roughly speaking, the Folk theorems state that in an RG, for a γ value sufficiently close to 1, there might be other payoffs than the stage NE that could be equilibria of the game and that could yield a higher payoff to all players. The discount factor gives a measure on how "patient" a player will be, meaning how much weight a player puts on future payoffs when compared to

the actual payoff. Intuitively, the Folk theorems state that a player patient enough will be able to obtain better payoffs. An RG may have infinitely many strategies that are an SPE of the RG, that will yield payoffs equal or better than the stage NE payoff to every player.

There are many well-known strategies that are used to take advantage of the Folk theorems, such as Nash reversion, tit-for-tat, grim trigger or forgiving strategies [146], [103]. Lemma 4 is used to test the conditions under which these strategies are SPE of the game: note that we need to define a strategy first, and then we test which payoffs are provided by that strategy. Many of the proposed RG strategies make use of a strategy that all players should follow and a punishment strategy which arises if any of the players deviates. Hence, the ability to obtain better payoffs by taking into account future play is closely related to being able to detect deviations instantaneously, and hence, are valid only for the case in which all players observe perfectly the actions of the other players, known as perfect monitoring or perfect information. In this work, we use as strategy unforgiving Nash reversion (UNR): both players start playing an agreed strategy (y_o, z_o) that provides them a payoff higher than their stage Nash payoff. If a deviation is observed, all players switch to play strategy (y_n, z_n), their stage NE strategy. This phase lasts forever, that is: if a player deviates, all players switch to play their stage NE strategy indefinitely.

Correlated equilibrium

The set of CE in the static case could be obtained using (2.69) or (2.70). In case of having an RG, we can use the same idea that lies behind Lemma 4 in order to obtain the set of CE, as shown in [158]. We define a static game which is equivalent to the RG using Bellman's equation, as in (2.77). This allows us to solve a static game which takes into account also future play if a certain strategy is followed, and also considers the effects of possible deviations using the one-shot deviation principle. This is justified in [146, p. 31], where the automaton representation and their derived properties are shown to hold in the CE case with minor modifications, the main difference being that now we use the CE condition. Hence, we seek for equilibrium distributions ϕ that satisfy the CE condition (2.70), which for RGs of perfect monitoring becomes:

$$\sum_{a_{-i}\in A_{-i}} \phi(a_{-i}|a_i) V_i(a_i, a_{-i}) \geq \sum_{a_{-i}\in A_{-i}} \phi(a_{-i}|a_i) V_i(a_i', a_{-i}), \quad \forall a_i' \in A_i, \quad a_i \neq a_i', \tag{2.78}$$

where we use Bellman's equation to define the payoff of the players as follows:

$$V_i(a_i, a_{-i}) = (1-\gamma) r_i(a_i, a_{-i}) + \gamma V_i'(a_i, a_{-i}), \tag{2.79}$$

where (a_i, a_{-i}) is the vector containing the actions of the players, $V_i(a_i, a_{-i})$ is the payoff that player i expects to obtain if she plays action a_i and all other players play a_{-i}. Finally, we note that, in case of using CE, the set of equilibria is a convex set and there exist algorithms that can be used to approximate this set, as QPACE [58].

2.5.3 Stochastic Games

The main difference between RGs and the single agent models that we have presented is that RGs have no state, or more concretely, they have a single state. Note that this means that the rewards that each player receives depend only on the action vector, and thus, the same action vector yields always the same rewards. In the MDP case, for instance, the reward depends on the action and the state, and hence, the same action in different states may yields different rewards. Thus, RGs are a useful tool for the case in which the same interaction is repeated over and over, but in many real world situations, the dynamical system has more than a single state.

Before continuing, a further clarification is needed. Note that RGs strategies need not be Markovian, as they depend on the whole previous history; but as we have shown, we can use Markovian strategies over the classes of equivalence w. The class of equivalence is not the same as the state s in the single agent models we have presented. Observe that s is the state of the dynamical system, while w is related to the strategy / policy in an RG. Also, in an RG there are infinitely many possible $\mathscr{W}$ spaces, as they depend on the concrete strategy chosen, while in a dynamical system, S does not depend on the policy.

It is possible to obtain a generalization of RGs which, at the same time, is a generalization of MDPs: the Stochastic Game (SG) [200], [70], defined as:

Definition 9 (Stochastic Game)**.** *An SG is a 6-tuple $\langle N_p, S, A, P, R_i, \gamma \rangle$ where:*

- *N_p is the index set of the agents, where the subscript $i = 1, 2, ..., N_p$ indexes each of the N_p agents, as in the Dec-POMDP case.*
- *S is the state set, defined as in the MDP case.*
- *A is the action set. Each agent i has its own action set A_i and $A \triangleq A_1 \times A_2 \times ... \times A_{N_p}$ is the product of the action spaces of each player. The joint action $a = (a_1, a_2, ..., a_{N_p}) \in A$ is a vector formed by the actions of all agents, as in the Dec-POMDP case.*
- *P is the transition probability function defined as in the MDP case. Note that P depends on a, the joint action vector.*
- *$R_i : S \times A \to \mathbb{R}$ is the reward function for each player $i \in N_p$, where each $r_i(s^n, a^n)$ is defined as in the MDP case. Note that each R_i depends on the joint action vector a.*
- *γ is a discount factor as in the MDP case.*

Note that an RG is an SG with a single state [6], an MDP is an SG with a single agent, and a static game is an SG with a single state and time step. The procedures proposed for solving SGs are similar to the ones in the single agent case, but now the players have to reach an equilibrium: each agent has to solve a control problem coupled with the control problems of the rest of the players. A lot of work is devoted to finding efficient methods to solve SGs [70], and there exist many algorithms to find equilibria for these games, such as Minimax-Q [140], WoLF [33], CE-Q [84], OPVar-Q [6] and Pepper [49]. A problem which arises with stochastic games is the dimensionality: a large number of states significantly hardens the problem of learning. To avoid this, several algorithms based on approximations have been used, such as FAL-SG [65] or AlphaGo Zero [207]. The recent advances in deep learning have also brought several new algorithms to the MAL problem [127]. Unluckily, SGs are in general complex to solve: even for concrete cases, a polynomial bound is not achievable [89].

2.5.4 Partially Observable Stochastic Games

The final model that we introduce generalizes the SG model to the case in which the agents are unable to observe the state. This framework is known as Partially Observable Stochastic Game (POSG) and is defined as follows [88]:

Definition 10 (Partially Observable SG)**.** *A POSG is a 9-tuple $\langle N_p, S, A, O, P, R_i, Z, b^0, \gamma \rangle$ where:*

- *N_p, S, A, O, P, Z, b^0 and γ are defined as in the Dec-POMDP case.*
- *R_i is the reward function for player $i \in N_p$, as in the SG case.*

Thus, a POSG is the generalization of a Dec-POMDP where the agents do not share a common interest, and a generalization of an SG when the agents have partial observability. Again, this model is very hard to solve, and actually, it has been solved only for very concrete cases, such as the Dec-POMDP [88], or using approximations as Libratus does [35].

2.5.5 Imperfect and incomplete information games

In a POSG we have partial observability, that is, the state is not observed directly, but a partial observation of it. In GT, it is also possible that the actions are not observed directly, but only partially. In general, the situation in which the actions and / or the state is not directly observed is denoted as imperfect information or imperfect monitoring [146]. In this work, we make interchangeable use of the terms imperfect information and partial observability, as the latter is more frequent in control theory, whereas the former is more used in GT contexts.

We also note that there are incomplete information games, which are such that one player does not know the payoff function of the rest. Hence, the player does not know the type of game that she is facing: it could range from extreme cooperation to extreme competition. In these games, it is frequent using the concept of Bayesian Equilibrium, which is an extension of the NE concept that assigns a certain probability to each player of belonging to a certain type of player, which is denoted as belief, and these beliefs are updated as the players act. These games are even more complex to solve than POSG, as they require conditioning on the beliefs of the players.

Thus, we note here the rationale of the title of this thesis. All of the problems we study deal with adversarial situations between one or more attackers and a defense mechanism in a WSN. In the best case, all players have perfect and complete information, i.e., they all know the type of the other players and observe their actions and the state if there is any. But in real environments, these assumptions need not hold, so in general, our security problems will be of imperfect and incomplete information, and they will be games because there is a conflict among the attackers and the defense mechanism. The complexity of solving such a game analytically prevents us from exploring that path, rather, we use control tools in order to obtain solutions which are computationally tractable, as Chapters 4-7 show.

2.6 Conclusions

In this Chapter, we have lain the mathematical foundations for this work, by introducing several frameworks that allow dealing with sequential decision-making, under a wide variety of conditions: for a single and multiple agents, with and without observing the state, and with and without conflict among agents. In Figure 2.4, it is possible to observe a schematic relation between the models explained, that summarizes the content of this Chapter. In the incoming Chapters, we apply the models explained in this Chapter to concrete problems in communication networks.

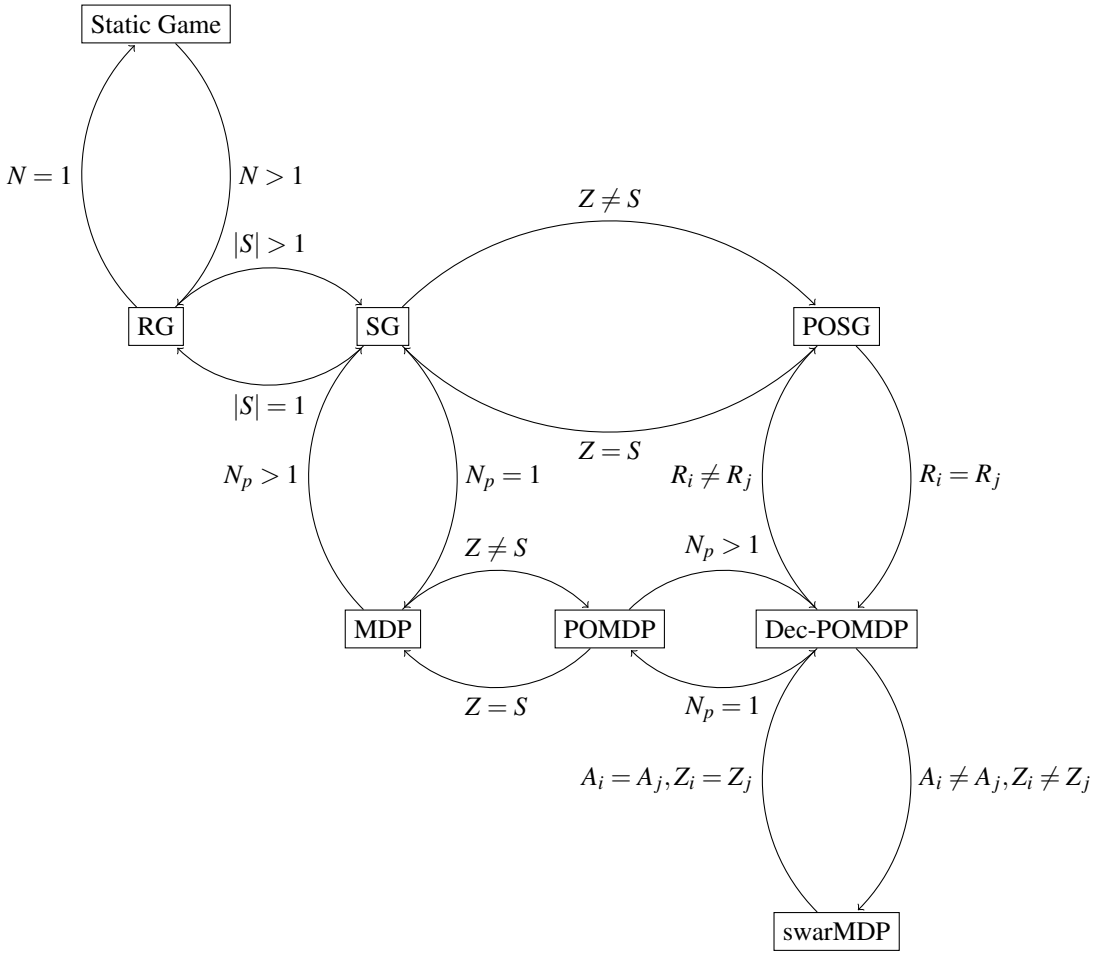


Fig. 2.4 Schematic illustration of the relations between the explained frameworks. Note that there are single agent models (MDP, POMDP) and multi-agent models (Static Game, RG, SG, Dec-POMDP, swarMDP, POSG); there are also models which assume a perfect observation, either of the states (MDP, SG) or the actions (RG), and models which assume partial observability (POMDP, Dec-POMDP, swarMDP, POSG). Also, in the multi-agent models, there are models which assume a common reward function shared by all agents (Dec-POMDP, swarMDP) or different reward functions for each agent (Static Game, RG, SG, POSG). Finally, note that all the models presented are dynamic except for the Static Game model.

Chapter 3

Discounted repeated games algorithms

3.1 Introduction

As we have indicated in Chapter 2, a frequently used mathematical framework to study the MAL problem is GT [204], the branch of mathematics oriented to study the potential conflicts that arise between different players that interact with possibly different objectives. However, as Chapter 2 hinted, this approach does not come free of problems. A first problem with GT is that the classic solution concept, the NE, is computationally hard to obtain [54]. This problem could be alleviated by using other solution concepts, such as the CE [79]. A second problem is that in real environments a player may not know the objectives of the other players, that is, the payoff functions of the other players: this situation is known as incomplete information [76]. Thus, they do not know which kind of players they face and the game can range from extreme competition to extreme cooperation. These two problems motivate trying to approach the problem from the learning perspective: each player would follow a certain procedure in order to maximize its cumulative reward.

There are many algorithms proposed for this problem, and the number is continuously increasing because this is a field of active research, as shown in [97]. In this Chapter we focus only on RGs. As we have explained, the total payoff that players receive can be obtained by averaging the rewards received in each stage, which is known as average payoff, or by using a discount factor to sum them, which is known as discounted payoff and it is the case that we have thoroughly described. In current literature, however, the former is more popular than the latter and hence, there is a significant gap in current MAL algorithms: the discounted payoff case has not been thoroughly addressed yet.

Discounted payoff schemes are generally more realistic than average payoff ones: the discount factor might reflect the balance between present and future rewards and also, might be used as a measure on the uncertainty in the length of the game, as the game may end at any stage with probability $1-\gamma$ [103]. Also, the average payoff scheme assumes that the game duration is known, which in certain environments, such as wireless networks, might not be a realistic assumption [123]. The difference between both payoff schemes acutes when the discount factor is low, i.e., is not close to 1. For instance, in communication networks, in which RGs are widely applied [103], there may appear very low discount factors, as in [128], [246] or [171]. In Section 3.2 we analytically show two important differences that arise when using discounted payoffs with respect to averaged payoff that affect the learning process, namely, the time to achieve a certain payoff and the variance in payoffs, both of which depend on the discount factor γ.

$$\begin{pmatrix} (1,-1) & (-1,1) \\ (-1,1) & (1,-1) \end{pmatrix}$$
(a) Matching pennies (MP).

$$\begin{pmatrix} (2,2) & (-1,3) \\ (3,-1) & (0,0) \end{pmatrix}$$
(b) Prisoner's dilemma (PD).

$$\begin{pmatrix} (1,1) & (0,0) \\ (0,0) & (2,2) \end{pmatrix}$$
(c) Cooperative game (CG).

Fig. 3.1 Payoff matrices for the example games. Player 1 is the row player and player 2 is column player. In each matrix, the payoff entries for each par of actions $a = (a_1, a_2)$ are $(r_1(a), r_2(a))$.

Then, we give two steps towards filling the gap in the algorithms available for discounted payoff. First, in Section 3.3, we introduce an algorithm that we call LEWIS (LEarn WIth Security). It is an algorithm specifically designed for online learning in RGs under a discounted payoff scheme with security, which means that the learning algorithm provides a lower bound on the player payoff. LEWIS takes into account the peculiarities of learning with discounted payoffs presented in Section 3.2 by using a conservative security condition that provides a low bound on the expected payoff. LEWIS is designed to work in incomplete information settings since it only needs to know its own reward. LEWIS may also cooperate if the game allows this, otherwise, it competes; and the trade-off between security and high payoffs can be adjusted.

Then, in Section 3.4, we introduce another algorithm that we call CA (Communicate and Agree), which is designed for negotiating equilibria in RGs of incomplete information. It is a fully distributed algorithm in which each agent finds candidate equilibrium points which are shared with the rest of agents: the final equilibrium is a Pareto-efficient one chosen among all the points which are valid equilibria for all players. It allows working using the NE and CE conditions, as well as using different strategies, even though for simplicity, we work only with UNR strategy. Note that while LEWIS is an online learning algorithm, in which it learns as it plays, CA negotiates an equilibrium prior to starting to play. Hence, both algorithms follow different approaches, and both are shown to provide good empirical results.

3.1.1 Example games

Now, we introduce some RGs that we use in this Chapter to illustrate our discussion. Each game has $N_p = 2$ players and each player has 2 possible actions. The payoff matrices of the chosen games are in Figure 3.1. The first game is matching pennies (MP), a zero-sum game. This means that $r_1 = -r_2$ and hence, the gains of one player are the losses of the other. The second game is the prisoner's dilemma (PD), which is a classic non-zero sum game: choosing the first action provides the highest reward for both players, but if the other plays her second action, the reward received is the worst possible. Finally, the third game is a cooperative game (CG), in which both players have the same reward function. Note that these games are chosen on purpose in order to cover a broad set of situations that may arise in RGs.

3.2 Discounted vs average payoffs

First, we recall the discounted payoff expression for RGs (2.76):

$$V_i(\sigma) = (1-\gamma) \sum_{n=0}^{\infty} \gamma^n r_i^n(\sigma_i, \sigma_{-i}). \tag{3.1}$$

In this section, we briefly introduce the most common payoff metric used in the current learning algorithms, which is the average payoff, and study two effects that have an impact on learning schemes. The average payoff

scheme is defined as follows:

$$U_i(\sigma) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{\infty} r_i^n(\sigma_i, \sigma_{-i}), \tag{3.2}$$

where we use U_i for the average payoff and V_i for the discounted payoff (3.1). In the incoming Sections 3.2.1 and 3.2.2, for the sake of mathematical tractability, we consider that the players use a fixed mixed strategy σ. That is, σ_i is a fixed probability distributions over A_i that does not change with time. Thus, the sequence of actions $a_i^0, a_i^1, ..., a_i^n$ is composed by independent and identically distributed actions for each stage. Note also that the actions are independent among players, for each player samples their action sequence using their own σ_i. And finally, observe that the sequence of rewards $r_i^0, r_i^1, ..., r_i^n$ has also independent and identically distributed components.

3.2.1 Time to achieve a certain payoff

The main difference between the average payoff U_i and the discounted payoff V_i, for the same rewards sequence $r_i^0, r_i^1, r_i^2...$, is that the discounted payoff is a weighted mean, with weights $(1-\gamma), (1-\gamma)\gamma, (1-\gamma)\gamma^2, ...$ Since the weights are decreasing, the first difference between average and discounted payoff is that the discounted payoff scheme emphasizes the first several elements of the payoff sequence. Moreover, this effect depends on γ. Note that since we consider that player i uses a fixed strategy σ_i, then $\mathbb{E}[r_i^t]$, the expected reward for player i, is constant. Let us start by defining the metric n^M:

Definition 11. *We define $n^M, M \in [0, 100]$ as the stage n of the RG in which the M% of the expected discounted payoff of player i has already been assigned in the discounted payoff case if player i uses a fixed strategy σ_i:*

$$n^M = \left\{ \min n \,\middle|\, \mathbb{E}\left[(1-\gamma) \sum_{k=0}^{n} \gamma^k r_i^k \right] \geq \frac{M}{100} \mathbb{E}[V_i] \right\}. \tag{3.3}$$

This definition of n^M allows us to obtain the following result:

Theorem 2. *In a discounted RG with infinite time horizon, with $\gamma \in (0,1)$, if a fixed strategy σ_i is played in all stages, we can obtain n^M as:*

$$n^M = \left\lceil \frac{\log\left(1 - \frac{M}{100}\right)}{\log(\gamma)} - 1 \right\rceil, \tag{3.4}$$

where $\lceil x \rceil$ denotes that x is rounded up to the next integer

Proof. The problem we have to solve, using (3.3) and the definition of V_i (3.1), is to obtain the minimum n^M that satisfies:

$$\mathbb{E}\left[(1-\gamma) \sum_{k=0}^{n^M} \gamma^k r_i^k \right] \geq \frac{M}{100} \mathbb{E}\left[(1-\gamma) \sum_{k=0}^{\infty} \gamma^k r_i^k \right],$$

which becomes:

$$(1-\gamma) \sum_{k=0}^{n^M} \gamma^k \mathbb{E}\left[r_i^k \right] \geq \frac{M}{100} (1-\gamma) \sum_{k=0}^{\infty} \gamma^k \mathbb{E}\left[r_i^k \right],$$

and since σ_i is fixed, $\mathbb{E}\left[r_i^k\right]$ is constant and hence:

$$(1-\gamma) \sum_{k=0}^{n^M} \gamma^k \geq \frac{M}{100} (1-\gamma) \sum_{k=0}^{\infty} \gamma^k. \tag{3.5}$$

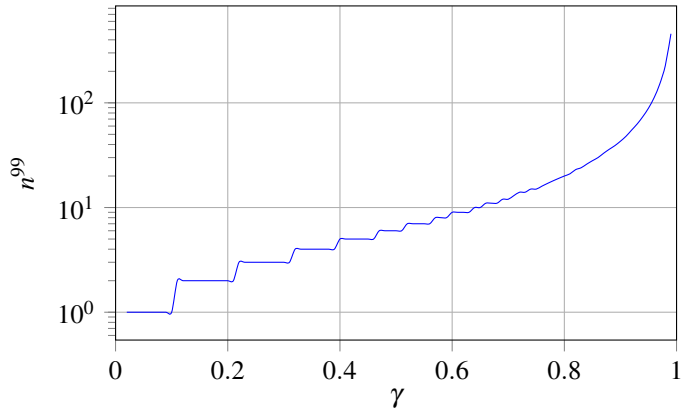


Fig. 3.2 Evolution of n^{99} as a function of γ values using (3.4). The horizontal axis represents the γ values, and in the vertical axis, we plot n^{99}, the number of stages needed to assign the 99% of the discounted payoff. Note that for low values of γ the major part of the total payoff is achieved with a few stages.

Now, we can use the following expression for geometric sums:

$$\sum_{n=n_0}^{n_1} \gamma^n = \frac{\gamma^{n_0} - \gamma^{n_1+1}}{1-\gamma}, \quad \gamma \neq 1, \tag{3.6}$$

to manipulate (3.5) and obtain:

$$1 - \gamma^{n^M+1} \geq \frac{M}{100}, \tag{3.7}$$

and the minimum n^M that solves this expression is (3.4). □

Note that (3.4) can be used to study the part of the payoff that has been assigned on time stage n^M, which depends on the γ value as plot in Figure 3.2 for $M = 99$. Small values of γ mean that the major part of the payoff is assigned in a short number of stages, whereas γ values close to 1 take more time stages to assign the payoff. Note that under average payoff, $n^M = \infty$ in the limit for $M \in (0, 100]$. The impact that this has on a learning schema is that under average payoff, the learning algorithm may converge in a large time stage n and it would not affect significantly the payoff. However, under a discounted payoff, the learning rate is key: a learning algorithm that converges slowly yields poor payoffs.

3.2.2 Variance

A second difference between discounted and average payoffs is the variance of the total payoff when a mixed strategy is used: the variance depends on the discount factor, as the next Theorem shows.

Theorem 3. *In a discounted, RG with infinite time horizon, with $\gamma \in (0,1)$, if a fixed mixed strategy σ is played in all stages, the following expressions hold for the discounted payoff case:*

$$\begin{aligned} \mathbb{E}[V_i] &= (1-\gamma^N)\mathbb{E}[r_i^n] \\ Var[V_i] &= \frac{1-\gamma}{1+\gamma}\left(1-\gamma^{2N}\right) Var[r_i^n] \end{aligned}, \tag{3.8}$$

and for the average payoff case:

$$\begin{aligned} \mathbb{E}[U_i] &= \mathbb{E}[r_i^n] \\ Var[U_i] &= \frac{1}{N} Var[r_i^n] \end{aligned}, \tag{3.9}$$

where $\mathbb{E}[r_i^n]$ is the expected reward of player i and $Var[r_i^n]$ the variance of the reward of player i.

Proof. Since each player follows a mixed fixed strategy, the reward r_i^n of player i follows a distribution probability which depends on the product of probabilities for each player for each action vector a, since each player chooses her mixed action independently of the rest following her strategy. Thus, if we define $X_{n,i} = (1-\gamma)\gamma^n r_i^n$ as the random variable that models the reward that player i receives in stage n, we obtain:

$$\begin{aligned} \mathbb{E}[X_{n,i}] &= \mathbb{E}[(1-\gamma)\gamma^n r_i^n] = (1-\gamma)\gamma^n \mathbb{E}[r_i^n] \\ Var[X_{n,i}] &= Var[(1-\gamma)\gamma^n r_i^n] = (1-\gamma)^2 \gamma^{2n} Var[r_i^n] \end{aligned}. \tag{3.10}$$

The expected discounted reward that a player obtains after N stages is:

$$\mathbb{E}[V_i] = \mathbb{E}\left[\sum_{n=0}^{N-1} X_{n,i}\right] = \sum_{n=0}^{N-1} (1-\gamma)\gamma^n \mathbb{E}[r_i^n], \tag{3.11}$$

and thus, using (3.6), (3.10) and (3.11), and taking into account that the variance of the sum of independent random variables is the sum of the variances and that $\mathbb{E}[r_i^n]$ is constant, we obtain the following result for the discounted payoff (3.1)

$$\begin{aligned} \mathbb{E}_n[V_i] &= \mathbb{E}\left[\sum_{n=0}^{N-1} (1-\gamma)\gamma^n r_i^n\right] = (1-\gamma^N)\mathbb{E}[r_i^n] \\ Var[V_i] &= Var\left[\sum_{n=0}^{N-1} (1-\gamma)\gamma^n r_i^n\right] = \sum_{n=0}^{N-1} Var[(1-\gamma)\gamma^n r_i^n] = \frac{1-\gamma}{1+\gamma}\left(1-\gamma^{2N}\right) Var[r_i^n] \end{aligned}. \tag{3.12}$$

And solving for the average payoff case (3.2), we obtain:

$$\begin{aligned} \mathbb{E}[U_i] &= \mathbb{E}\left[\frac{1}{N}\sum_{n=0}^{N-1} r_i^n\right] = \mathbb{E}[r_i^n] \\ Var[U_i] &= Var\left[\frac{1}{N}\sum_{n=0}^{N-1} r_i^n\right] = \frac{1}{N^2}\sum_{n=0}^{N-1} Var[r_i^n] = \frac{1}{N} Var[r_i^n] \end{aligned}. \tag{3.13}$$

□

Observe how in the average payoff case (3.9), the mean value of the total payoff coincides with the mean value of the reward, and the variance of the total payoff tends to zero with sufficiently large time stages. However, in the discounted payoff case (3.8), note that the mean value and variance of the total payoff depend on the γ value. Also, observe that for sufficiently large values of N, i.e., such that $\gamma^N \to 0$, the mean value of the total payoff coincides with the mean value of the reward, but the variance still depends on the γ value and only tends to 0 if $\gamma \to 1$. In other words, the discount factor does also have an impact on the mean and variance of the total payoff. It is possible to understand this effect intuitively by realizing that the discount factor gives a higher weight to the firsts r_i^n values to obtain V_i: with a discount factor, say, $\gamma = 0.1$, r_i^0 has a weight $(1-\gamma)\gamma^n = 0.9$,

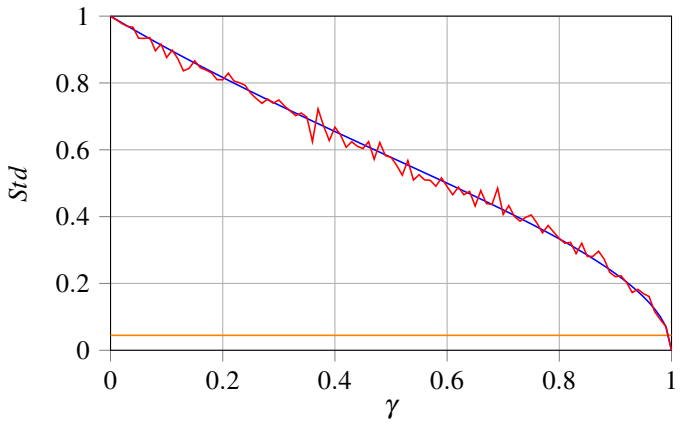


Fig. 3.3 Results of the standard deviation comparison simulation using MP. The horizontal axis represents the γ values, and in the vertical axis, we plot the standard deviation of the payoff. Orange line is for the theoretical average payoff case, using (3.9). Blue line is for the theoretical discounted payoff case, using (3.8). Red lines are the empirical standard deviation obtained under simulation. Note how the standard deviation depends on γ under the discounted payoff case. Also, note that the average payoff case gives in general lower deviations, except when $\gamma \to 1$.

r_i^1 has a weight 0.09 and r_i^2 has a weight 0.009; that is, the first three r_i^n values concentrate the 0.999% of the total V_i. With such a small amount of samples dominating the average payoff, it is easy to understand why low γ values cause large variances in V_i.

We further illustrate this effect using as example the MP game. Recall that in this game there is a single mixed NE in which each player plays each of her actions with probability 0.5. Thus, $\mathbb{E}[r_i^n] = 0$ and $Var[r_i^n] = 1$. We simulated 100 different sequences with length $N = 500$, for 101 equispaced values of $\gamma \in [0,1]$ and obtained the standard deviation for each γ value. Also, we obtained the theoretical standard deviation value using (3.8). For comparison purposes, we added the theoretical standard deviation under the average payoff scheme using (3.9). The results can be observed in Figure 3.3, where we can observe that the standard deviation in the discount payoff scheme depends on the γ value and is larger than the standard deviation under the average payoff scheme except when $\gamma \approx 1$.

3.2.3 The impact of the discount factor

Using discounted payoff introduces several differences with respect to the average payoff case. We have highlighted two of them which have a strong impact on the learning mechanisms. The first one indicates that the $M\%$ total payoff under the discounted schema is determined by the first n^M actions. Note that this means that the first actions are key in a learning procedure that is interested in obtaining good payoffs. If a learning mechanism is very good at finding equilibrium strategies but it takes a long time to find such a strategy, then this algorithm is not adequate for the discounted payoff case: we need algorithms that are really fast in learning. For instance, in [65] the authors develop a fast learning algorithm for average payoff stochastic games.

Secondly, we have shown that using a discounted payoff also impacts the variance of the total payoff obtained. In most cases, this variance will be larger than the one obtained using an average payoff scheme. Thus, a learning algorithm might be fast and however, it may provide a high variance on the total payoff.

Since current algorithms deal with the case of average payoffs, these concerns related to the discounted case have not yet been thoroughly addressed. In the next Sections, we introduce two novel algorithms that are able

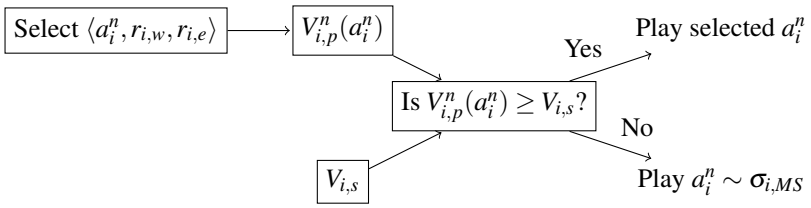


Fig. 3.4 LEWIS block diagram.

to successfully deal with discounted payoff situations, being an important step towards algorithms specifically designed for the discounted payoff case.

3.3 Learning with Security: the LEWIS algorithm

Now, we turn our attention to describe our first algorithm for learning in discounted RGs, which is called LEWIS, as an acronym of LEarning WIth Security. It is an online algorithm that learns by interacting. It has a security strategy, and the algorithm tries to obtain a payoff higher than the payoff induced by the security strategy when possible. Some LEWIS highlights are:

- LEWIS implicitly uses the Folk Theorem, and hence, it can obtain better payoffs than the ones obtained by simply repeating the static game equilibrium.
- LEWIS is designed for secure online learning in discounted setups, that takes into account the effects presented in Section 3.2.
- LEWIS can solve games of incomplete information, as each player needs not knowing the payoff functions of the others. Actually, each player needs only knowing its own reward function and keeping a track of her past actions.

3.3.1 The LEWIS algorithm

The basic idea behind LEWIS is that it will play a certain action a_i^n if and only if (1) the expected payoff by using a_i^n is larger than a security payoff and (2) the worst case payoff of using a_i^n does not fall below a certain threshold. A block diagram of LEWIS can be found in Figure 3.4, which we proceed now to explain.

Action selection block

The first key component of LEWIS is the action selection block, which returns a triple $\langle a_i^n, r_{i,w}, r_{i,e}\rangle$ as a recommended action a_i^n, the worst reward that could be obtained by playing action a_i^n, $r_{i,w}$, and the expected reward of playing action a_i^n, $r_{i,e}$. The action selection block we propose is based on the past rewards received by agent i. For each discrete action $a_i \in A_i$, we keep a measure of the reward that player i has obtained by playing this action in the past, $\hat{r}_i^n(a_i)$. The index n is used because the estimation is updated in each stage n as follows:

$$\hat{r}_i^n(a_i) = \begin{cases} (1-\alpha)\hat{r}_i^{n-1}(a_i) + \alpha r_i^n & \text{if} \quad a_i = a_i^n \\ \hat{r}_i^{n-1}(a_i) & \text{if} \quad a_i \neq a_i^n \end{cases}, \tag{3.14}$$

where $\alpha \in [0,1]$ is a parameter that controls how much weight we give to the current reward. Note that (3.14) obtains an exponential weighted average of the received payoffs, where the exponential decay is controlled by α.

Larger α values provide a faster update, but also larger variance. We follow [212] by initializing the estimation optimistically to facilitate exploration as $\hat{r}_i^{-1}(a_i) = \max_{a_{-i}} r_i(a_i, a_{-i})$, that is, the estimation is initialized to the maximum reward value for each action.

In each stage n, this block recommends using the action which provides a larger reward estimation, that is, $a_i^n = \arg\max_{a_i} \hat{r}_i^{n-1}(a_i)$ and $r_{i,e} = \hat{r}_i^{n-1}(a_i^n)$. The worst case reward is the minimum reward that player i would obtain by playing a_i^n, that is, $r_{i,w} = \min_{a_{-i}} r_i(a_i^n, a_{-i})$. LEWIS may decide to use a_i^n or not, and then a reward r_i^n would be received, which allows updating $\hat{r}_i^n(a_{i,n})$ using (3.14) and the actual action chosen by LEWIS. Hence, note that LEWIS is not only valid for incomplete observation settings, but also for imperfect observation ones, as each player only needs to observe her own actions.

Security condition

A main feature of LEWIS is the security property: LEWIS guarantees a minimum payoff for the player, by comparing with a security payoff. Since each player knows only her own payoff, it is not possible for her to compute the NE of the game and use this as security payoff. We choose to use the minmax strategy (MS): the player finds a strategy such that is maximizes the worst reward that she could obtain. In other words, this strategy considers that the other players are extreme competitors that will try to minimize her payoffs, so she chooses her strategy accordingly:

$$\sigma_{i,MS} = \arg\max_{\sigma_i} \min_{\sigma_{-i}} V_i(\sigma_i, \sigma_{-i}), \tag{3.15}$$

where we define $V_{i,MS}$ as the minmax strategy payoff for player i, that is, the payoff that player i would receive if all players used their minmax strategy. Note that the minmax strategy is the NE in case that the game was zero-sum, which is the worst situation for a player, as this is the maximum competition situation. Note that for two players, two action games, the MS can be obtained using a linear program which depends only on the payoffs of one player, as described in [14, Ch. 20]. Thus, by definition, the minmax payoff maximizes the worst reward that the player could obtain and hence, the MS can be used as a security strategy to guarantee a payoff $V_{i,MS}$ at worst, regardless of what other players do. This choice is used in other algorithms that provide security, such as M-Qubed, which we will introduce shortly.

Note that by following the MS, player i would consider that the game is zero-sum. Consider, for instance, the PD game in Figure 3.1: if one player always uses her minmax strategy, i.e., the second action, the other player would also use her minmax strategy and both players would receive a payoff $V_{MS} = 0$, regardless of γ. However, if γ is sufficiently large, both players may use their first action instead to achieve a larger payoff. The problem is that if a player uses her first action, she risks to having a reward of -1 and hence, having a lower payoff than V_{MS}.

In other words, there needs to be a compromise between the security payoff and the ability to cooperate with other players. We model this compromise by using a parameter $\varepsilon \geq 0$ to define $V_{i,s}$, the security payoff for player i as:

$$V_{i,s} = \mathbb{E}[V_{i,MS} - \varepsilon] = \mathbb{E}\left[(1-\gamma)\sum_{n=0}^{\infty} \gamma^n r_{i,MS}\right] - \varepsilon = \mathbb{E}[r_{i,MS}] - \varepsilon, \tag{3.16}$$

where we substitute $\mathbb{E}[r_{i,MS}]$, the expected minmax reward, in (3.1), and use (3.6).

Note that in (3.16), if $\varepsilon = 0$, player i would have to always use $\sigma_{i,MS}$ in order to guarantee herself a security payoff $V_{i,s} = \mathbb{E}[V_{i,MS}]$. However, if ε is a positive value, player i could be willing to use actions that do not follow $\sigma_{i,MS}$ if the worst case payoff provided by these actions is larger than the security payoff, which now is

Algorithm 8 LEWIS algorithm for player i

Input: γ, r_i, ε

```
Obtain the minmax values: r_{i,MS} and σ_{i,MS}
for n = 0, 1, 2, ... do
    Obtain ⟨a_i^n, r_{i,w}, r_{i,e}⟩
    Obtain V_{i,p}^n(a_i^n) using (3.17)
    if V_{i,p}^n(a_i^n) is secure using Definition 12 then
        Play a_i^n
    else
        Play a_i^n ~ σ_{i,MS}
    Observe the actions and rewards
    Update action selection block using (3.14)
```

smaller than the MS payoff by ε. Note that our use of ε is similar to the concept of ε NE, although they are different because we have no guarantee that LEWIS learns such equilibrium.

In each stage n, LEWIS first obtains a recommended action a_i^n and then it has to decide whether to play this recommended action or not. In order to make that decision, LEWIS obtains the worst predicted payoff for a_i^n, $V_{i,p}^n(a_i^n)$, as follows:

$$\begin{aligned} V_{i,p}^n(a_i^n) =& \mathbb{E}\left[(1-\gamma)\left(\sum_{k=0}^{n-1}\gamma^k r_i^k + \gamma^n r_{i,w} + \sum_{k=n+1}^{\infty}\gamma^k r_{i,MS}\right)\right], \\ =& (1-\gamma)\sum_{k=0}^{n-1}\gamma^k r_i^k + (1-\gamma)\gamma^n r_{i,w} + \gamma^{n+1}\mathbb{E}[r_{i,MS}] \end{aligned} \tag{3.17}$$

where $V_{i,p}^n(a_i^n)$ is the expected payoff that player i would obtain if she plays a_i^n and obtains the worst possible reward for this action, $r_{i,w}$, and in the rest of the game, player i follows its minmax strategy. By using (3.16) and (3.17), we have the following Definition that states whether an action a_i^n is secure or not:

Definition 12. *In a discounted RG with infinite time horizon, with $\gamma \in (0,1)$, and $V_{i,s}$ and $V_{i,p}^n(a_i^n)$ defined as in* (3.16) *and* (3.17) *respectively, an action a_i^n is secure if:*

$$V_{i,p}^n(a_i^n) \geq V_{i,s}. \tag{3.18}$$

This condition simply denotes that the estimated payoff needs to be greater or equal than the security payoff. If that condition is not satisfied, then, the action a_i^n is not considered secure and LEWIS would play an action $a_i^n \sim \sigma_{i,MS}$, that is, an action following its minmax strategy. Note that LEWIS is implicitly built upon the Folk Theorem: there may exist payoff better than the minmax ones for the player, and hence, LEWIS is an implementation that tries to find such payoffs online.

Algorithm overview

An overview of the whole LEWIS procedure is in Algorithm 8. As input, each player i needs only her own payoff function r_i, the discount factor γ and the ε value that will be used to set the security payoff. In each stage n, player i obtains a_i^n and checks whether this action is secure or not using Definition 12. If a_i^n is secure, then the player plays, it, otherwise, she plays the minmax action. After that, the action selection block is updated.

A final word is required regarding convergence. We give no guarantee that LEWIS converges to an equilibrium, and this would be largely dependent on the kind of game and the action blocks used, as we note

that it is possible to include action selection blocks different from the one we have proposed. Note that LEWIS does not learn a strategy σ, but rather, it decides at each stage whether to play a recommended action or the minmax action. This choice is done purely in terms of payoff. Note that, as [159] shows, in RGs there might be many possible strategies that lead to different action sequences that provide similar payoffs. This fact is used by LEWIS, by choosing only actions that provide the player with a certain payoff larger than the security payoff. As we describe in an incoming Section, our simulations show that LEWIS does achieve good payoffs both in self play and against other learning algorithms.

3.3.2 Similar works

There are many algorithms proposed for learning RGs. A recent survey on MAL collects more than twenty algorithms [97], and it is a topic subject to an intense research from different perspectives as recent works show. To mention some, it has been studied from a physics point of view [217], [182], from a social and natural sciences perspective [183] and it has also been studied under a computer science perspective [97]. This amount of research has produced many algorithms to learn the stage game equilibrium, i.e., without using the Folk Theorem, such as Regret Matching [90] [91], ReDVaLeR [18] or AWESOME [48]. There are also algorithms that explicitly use the Folk Theorem without discounting, as [141] or [56]. M-Qubed [50] makes an implicit use of the Folk Theorem by setting a bound on the maximum losses that the players are willing to take.

However, as [97] shows, there are several gaps to be filled in the field of learning RGs. There are algorithms that do not take into account that there are other players learning that will affect their own learning procedure, such as Fictitious Play [34] or JAL [47]. However, the major gap we find in current RG learning algorithms is related to not taking into account the discount factor in the learning process. Even though many learning algorithms based in Q-learning use a discount factor in the Q-function update, they end up using the average reward as total payoff. The main difference between using average and discounted payoff is that under the average paradigm, all rewards equally contribute to the total payoff, i.e., all the rewards have the same weight on the total payoff; whereas under a discounted payoff, the firsts rewards have a larger weight on the total payoff, as shown by Theorem 2. We have argued that using discounted payoffs may be more realistic in some settings. Most of the learning algorithms we know are not designed to deal with discounted payoffs, such as GIGA-WoLF [32], CoLF [55], WPL [3], FAQL [116], LFAQ [31], R-Max# [100], RUQL [2], ReDVaLeR [18], Manipulator algorithm [188], AWESOME [48], M-Qubed [50], RSRS [52], CMLeS [39], MDP-CL [98], DriftER [99] or PI-POMDP [241].

As shown in [50], two important concerns arise when a player is learning an RG and the player can only observe her payoffs and the actions of the other players. The first concern is security, which is included by design in LEWIS. Some algorithms take into account this concern, as GIGA-WoLF [32], ReDVaLeR [18], M-Qubed [50] and RSRS [52]. However, the security in these algorithms is related to the average payoff concept. If we use discounted payoff, the security requirement becomes more exigent, since losing at the first stages of the game might not be compensated afterwards. The second concern is the ability to coordinate and cooperate when the players share common interests. These two concerns are opposite and to the best of our knowledge, only M-Qubed and LEWIS address both of them.

The most similar algorithm to LEWIS is M-Qubed [50]. M-Qubed compares between a Q-learning estimation of the value of taking a concrete action and the minmax strategy, and mixes them depending on how the total accumulated payoff compares to the minmax payoff. This allows obtaining a tradeoff between security and cooperation, as LEWIS does. However, the security condition in M-Qubed is related to the average payoff: a parameter L_{tol} is defined that accounts for the maximum loss in the average payoff tolerated. It is not

straightforward transforming this to a discounted payoff setup, although some manipulation on the maximum losses tolerated could be explored. In contrast, LEWIS security condition, as shown in Definition 12, is defined specifically for discounted payoffs. Also, LEWIS is a much simpler algorithm, with fewer parameters to adjust and it also performs better than M-Qubed in our testbench, as we will shortly present.

LEWIS also presents similarities to other algorithms. For instance, LEWIS may choose among several strategies, as in Manipulator algorithm [188], if there were several strategy blocks. LEWIS is designed for RGs with discounting and incomplete information as [184], however, this work applies only to large discount factor values. An important concern of LEWIS is learning fast, as in [65], which applies to stochastic games and does not study the effects of the discount factor.

Note also that there are important differences to other algorithms as well. We do not need to observe the mixed actions as in ReDVaLeR [18]. We adapt to the learning of the rest of the players, contrary to Fictitious Play [34] or JAL [47]. We do not need a priori knowledge about game attributes, as in RUQL [2]; the only a priori information we use is the reward scheme of the player. Yet the main difference to all existing algorithms, as we have discussed, is the fact that LEWIS is specifically designed to deal with discounted payoff setups.

3.3.3 Empirical results

Now, we validate LEWIS by using several simulations with the following objectives:

- We first want to study the influence of ε in the payoffs.
- Then, we study the behavior of LEWIS in self play, in order to observe whether it achieves cooperation when possible.
- We test the security of LEWIS afterwards by testing it on the worst case situation.
- We finally compare LEWIS to other algorithms for RGs, in order to study its performance against other players.

In all these simulations, we use $N = 500$ stages per game, and average each of them for 100 different game realizations. The maximum γ we use is $\gamma = 0.99$, and by using Theorem 2, we know that $n^{99} = 458$ for $\gamma = 0.99$ and hence, by running each game for 500 stages we make sure that more than 99% of the total payoff is already allocated in all of our simulations. In all these simulations, we set $\alpha = 0.5$ for the LEWIS strategy block. We use as testbench the three games shown in Figure 3.1. These are two players, two action games, for which the MS can be obtained using a linear program, as described in [14, Ch. 20].

Simulation 1: The effect of ε

First, we explore the effect of ε in LEWIS. We simulate in the PD game for $\gamma = \{0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$ and 50 values of $\varepsilon \in [0, 0.5]$. Both players use LEWIS and the results are plot in Figure 3.5, where we observe how larger values of γ and ε lead to larger payoffs. The former is due to the Folk Theorem and the latter is due to the fact that ε controls the tradeoff between security and cooperation: a larger ε facilitates cooperation when possible, at the cost of having a lower security payoff. Note that large γ and ε values allow satisfying the security conditions in Definition 12 easier. In this case, since the game allows cooperation, a large ε is desirable, however, as we will see, a large ε also means a lower payoff in the worst case.

Note that Figure 3.5 does not have a monotonic increase with γ. This is due to $V_{i,p}^n$ not being monotonic with γ, as taking the derivatives with respect to γ in (3.17) shows, depending on the stage n and the values of

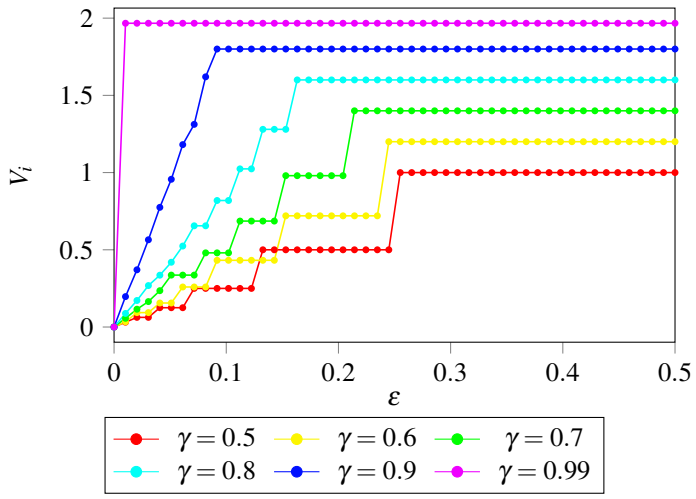


Fig. 3.5 Payoff results as a function of ε and γ in the PD game, using LEWIS. In the horizontal axis we represent ε and in the vertical axis, the payoff of the players V_i. In this case, players learn to cooperate except when $\gamma = 0$ and both receive the same payoffs. Note how larger values of γ and ε lead to larger payoffs.

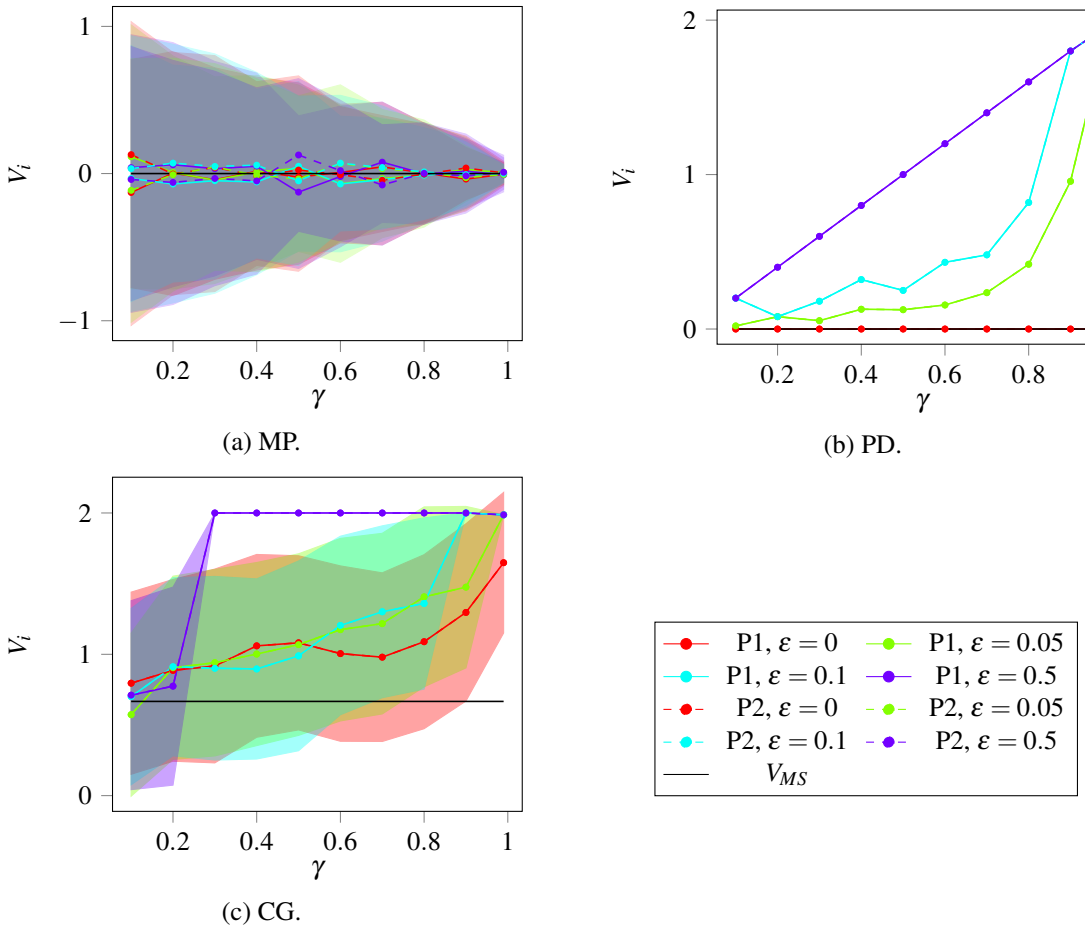
$r_{i,w}$ and $r_{i,MS}$. Thus, it may happen that better payoffs are achieved with lower γ values for the same ε; however note that Figure 3.5 shows that this appears rarely. Thus, in general, larger γ and ε values lead to larger payoffs.

Simulation 2: LEWIS in self play

Now, we turn our attention at how LEWIS performs in self play, that is, when all players are using LEWIS. In this case, we simulate for PD, MP and CG, using $\gamma = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$ and $\varepsilon = \{0, 0.05, 0.1, 0.5\}$. The results can be observed in Figure 3.6, where we plot the payoffs achieved as a function of γ. In MP, which is a zero-sum game, we know that there is no possible gain with respect to V_{MS}, which is the result that LEWIS achieves in mean. Note that there is a variance that decreases with γ, as predicted by Theorem 3. In PD, when $\varepsilon = 0$ the players do not learn to cooperate, as expected, but for positive values of ε, note how the players start cooperating, achieving larger payoffs as γ and ε increase, which is the result expected as we showed in the previous simulation. CG achieves similar results to PD: players learn to cooperate with large values of ε and γ. Thus, in self play, LEWIS is able to cooperate (CG, PD) and compete (MP) successfully.

Simulation 3: LEWIS against a minmax player

Now, we turn our attention at how LEWIS performs in the worst case, that is, against a player that is continuously minmaxing. Again, we simulate for PD, MP and CG, using $\gamma = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$ and $\varepsilon = \{0, 0.05, 0.1, 0.5\}$. The results can be observed in Figure 3.7, where we plot the payoffs achieved as a function of γ. Note how in MP and CG, LEWIS is able to achieve a payoff close to V_{MS} in all cases, and also, since these games have mixed MS strategies, observe how the variance decreases with γ, as predicted by Theorem 3. The PD plot is really interesting, because we can clearly observe how the security property of LEWIS holds: the maximum loss with respect to V_{MS} is clearly bounded by ε. Thus, LEWIS satisfies their design restrictions: it is a secure algorithm, in which it provides a minimum security payoff, and it also allows cooperating when possible; and the tradeoff between both properties is controlled by the ε parameter.



(a) MP.

(b) PD.

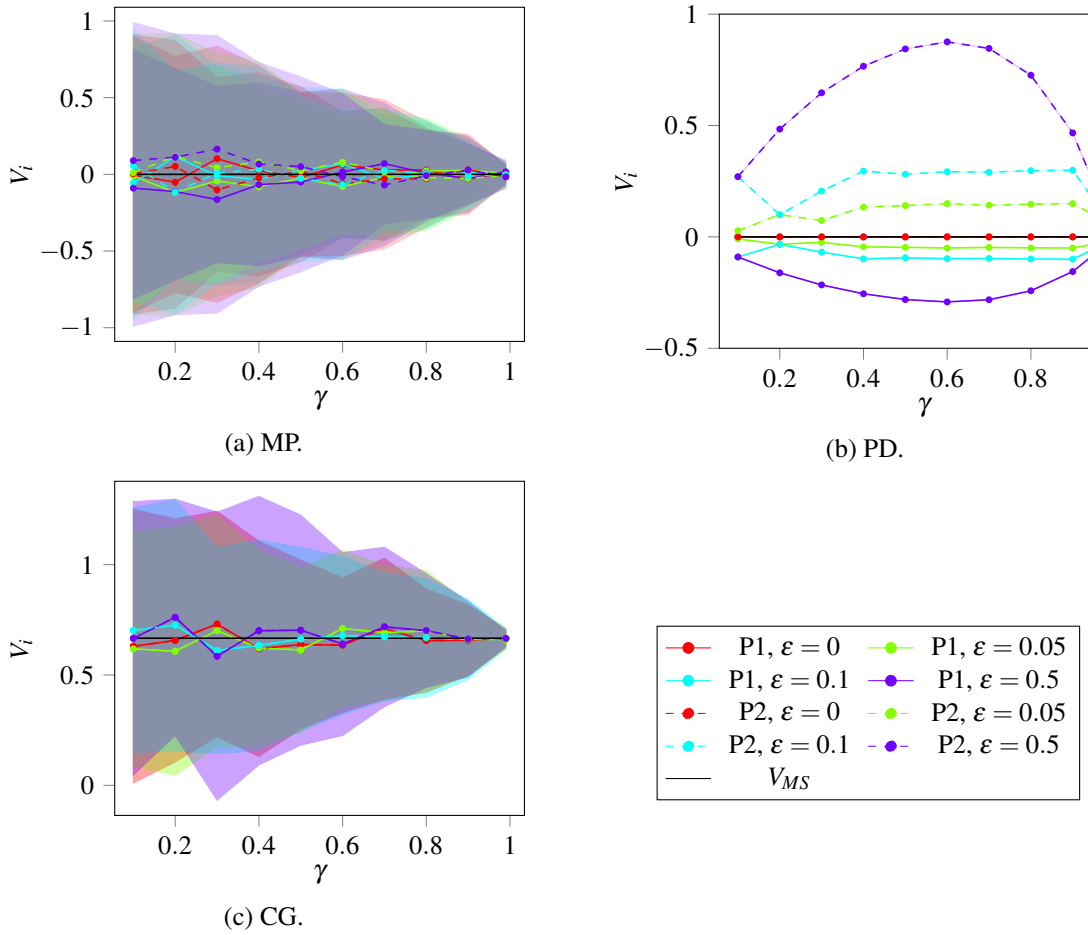
(c) CG.

Fig. 3.6 Results of the simulation of LEWIS in self play, when both player 1 (P1) and player 2 (P2) use LEWIS. The shadowed region is the standard deviation. The horizontal axis is γ and the vertical axis shows the payoff achieved by each player. Note that LEWIS is able to cooperate in PD and CG with a sufficiently large ε and γ values. In MP, cooperation is not possible as this is a zero-sum game and thus $V_i = V_{MS}$. Finally, note how in MS, the variance decreases with γ, as predicted by Theorem 3.

Simulation 4: LEWIS against other agents

Now, we turn our attention at how LEWIS performs against a set of different algorithms proposed for learning RGs. We choose three algorithms that are close to some design features of LEWIS. The first one is CoLF [55], which tries to find Pareto payoffs: these payoffs may be better than stage NE payoffs in repeated cooperative games. The second one is LFAQ [31], which uses insights from evolutionary game theory to learn Pareto strategies. And the third one is M-Qubed [50], which is the most similar to LEWIS in that it tries to find cooperative strategies while also being secure in payoffs, although M-Qubed security bounds are related to the average payoff, as we explain in Section 3.3.2. For each algorithm, the parameters used are the ones proposed in each of their articles.

Again, we simulate using $\gamma = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99\}$ in PD, MP and CG. For LEWIS, we use $\varepsilon = 0.5$. The results can be observed in Figure 3.8, where we plot the payoffs achieved as a function of γ and include LEWIS in self play for comparison. Note how in all cases, LEWIS is secure and is usually the algorithm that gives the best payoff, except in MP. Of special interest is the PD case, in which LEWIS



(a) MP.

(b) PD.

(c) CG.

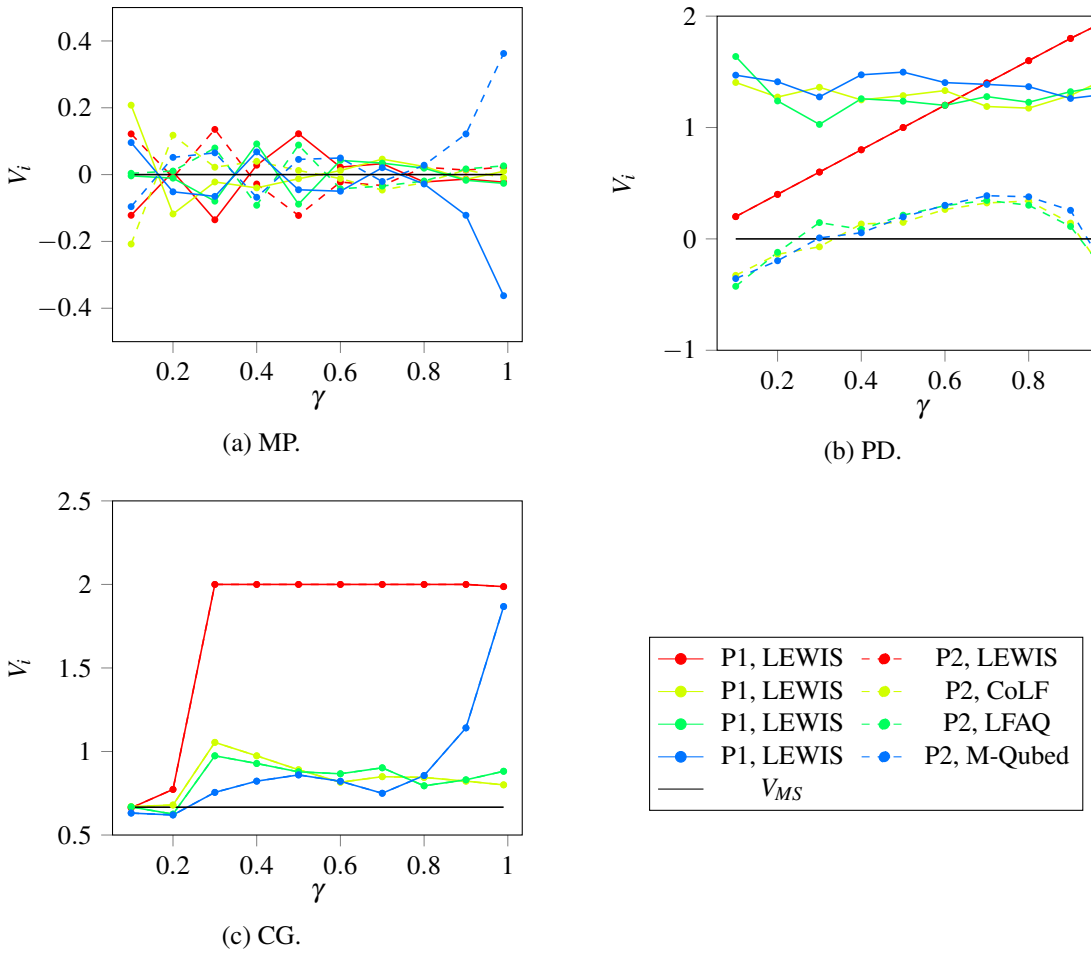
Fig. 3.7 Results of the simulation of LEWIS against a minmax player, where the player 1 (P1) uses LEWIS and the player 2 (P2) follows the MS. The shadowed region is the standard deviation. The horizontal axis is γ and the vertical axis shows the payoff achieved by each player. Note how the security property of LEWIS holds: this can be specially observed in the PD case, when the maximum loss with respect to V_{MS} is clearly bounded by ε.

consistently gives the highest payoff, regardless of which algorithm it is facing. Also, note how in CG, LEWIS in self play is also the algorithm that gives the best results. Thus, the comparison with these algorithms show that LEWIS is a very competitive algorithm.

In conclusion, LEWIS is an online learning algorithm specifically designed to deal with discounted RGs. It is a simple algorithm, but nonetheless able to cooperate when possible and at the same time, maintain a security payoff. It addresses the special characteristics of learning in a discounted game, as we have presented in Section 3.2. And finally, when compared with other learning algorithms, its results are also very competitive.

3.4 Negotiating an equilibrium: Communicate and Agree

The algorithms mentioned so far are based on the idea of online learning: players learn how to play based on the previous behavior and rewards obtained. In this approach, guaranteeing a certain performance bound while learning is important, as we have extensively discussed. A different approach could be based in negotiation:



(a) MP.

(b) PD.

(c) CG.

Fig. 3.8 Results of the simulation of LEWIS against other algorithms, where the player 1 (P1) uses LEWIS and the player 2 (P2) uses different algorithms. The standard deviation is not showed for the sake of clarity. The horizontal axis is γ and the vertical axis shows the payoff achieved by each player. Observe how LEWIS is really competitive, providing the largest payoffs in PD.

prior to play, the players interchange messages in order to negotiate a certain strategy. This approach is used, for instance, in [106].

In this Section, we introduce Communicate & Agree (CA), a novel algorithm that, as LEWIS, aims to fill a gap in current MAL algorithms: CA is a MAL algorithm that computes equilibria of RGs with discounting using the Folk Theorem and negotiation. Some of CA highlights are:

- CA is a fully distributed algorithm, which does not need a central entity to control the negotiation process.
- CA explicitly uses the Folk Theorem, and hence, it can obtain better payoffs than the ones obtained by simply repeating the static game equilibrium.
- When there are several payoffs that could be achieved by making use of Folk Theorem tools, CA selects payoffs that are Pareto efficient.
- CA can be applied to compute different kind of equilibria, such as NE or CE.

- CA can solve games of incomplete information, as each player needs not knowing the payoff functions of the others. Yet we assume that all players can observe the actions of the other players, that is, we assume perfect information [146].

3.4.1 The CA algorithm

CA requires the following inputs for each player:

- The discount factor γ.
- The payoff function for player i, r_i. Each player does not need to know payoff function of the other players, which in turn means that the player does not know what kind of opponents she is facing, i.e., it is a game of incomplete information. CA is able to work in a wide variety of environments: from extreme competition to extreme cooperation games, without needing a priori knowledge of the kind of game. Each player also needs to know *na*, the dimension of the action vectors $a \in A$. This parameter is required to sample the action space A.
- The punishment action for player i, $a_{i,p}$ and its punishment payoff, $r_{i,p}$. This is the payoff that CA will try to improve by using RG tools. We use a stage equilibrium which can be obtained, for instance, using RM algorithm [90]. Other options are possible, such as the minmax strategy (3.15). Note that this requirement appears in other MAL algorithms, such as [48].
- The number of players of the game, N_p.
- N_c, the maximum number of communications allowed. This parameter allows controlling the negotiation time.

For simplicity, we assume that all players use the same equilibrium concept, either NE or CE, and that they all use the UNR strategy. CA is based in negotiation, and hence, it assumes that players communicate among them, as in [106]. We use this communication so that players interchange strategies that they are willing to follow: player i proposes a strategy that leads her to an equilibrium, and if this strategy is also an equilibrium to the other players, then it constitutes an equilibrium of the RG.

CA proceeds in two steps. The first step is called action space sampling. All players sample the set of actions A and interchange messages in order to obtain A_s, the set of sampled actions which are valid equilibria for all players. In order to do this, each player samples actions profiles and tests them to check which yield an equilibrium for her, i.e., using Lemma 4 with the NE condition (2.64) or (2.79) in the CE case. When a player finds an equilibrium strategy a, it communicates it to other players, and they test whether a is also an equilibrium for them or not. If it is an equilibrium for all players, all players do $A_s = A_s \cup a$. This is repeated a certain number of times, controlled by N_c.

The second step is called Pareto pruning: the players must choose one strategy from A_s. In order to do so, they coordinate to randomly choose a strategy $a_c \in A_s$. Then, all players discard all strategies $a \in A_s$ so that $V_i(a) < V_i(a_c)$, i.e., they discard all strategies that are Pareto dominated by strategy a_c. Hence, A_s is pruned by eliminating all Pareto-dominated strategies. This procedure is repeated until A_s is a single strategy. By construction, this strategy is guaranteed to be Pareto-efficient, because if it were Pareto-dominated, it would have been pruned.

A schematic description of CA can be found in Algorithm 9, where $|A_s|$ denotes the number of elements in the set A_s. Each of the two main steps is detailed in the following sections, as well as in Algorithms 10 and 11.

Algorithm 9 CA algorithm for each player i

Input: γ, r_i, $a_{i,p}$, $r_{i,p}$, N_p, N_c, na

$A_s \leftarrow sample-actions(\gamma, r_i, a_{i,p}, r_{i,p}, N_p, N_c, na)$
if $A_s = \emptyset$ **then**
  $A_s = a_{i,p}$
else
  while $|A_s| > 1$ **do**
    $A_s \leftarrow pareto-prune(A_s, r_i)$

Output: A_s

Action space sampling

Action space sampling is the first component of CA algorithm. Roughly speaking, we try to obtain the subset of actions that leads all players to an equilibrium. Note that other approaches try to find the achievable set of payoffs, as [58]. More formally, in case of NE:

$$A_s : \Big\{ a \in A, r_i(a) \geq (1-\gamma) \max_{a_i'} r_i(a_i', a_{-i}) + \gamma r_{i,p}, a_i' \neq a_{i,o} \Big\}, \tag{3.19}$$

which are the conditions from Lemma 4 with the NE condition (2.64) particularized to UNR strategy. Using (2.79) we reach the condition for CE; in this case, the sampling applies to ϕ distributions instead of actions a:

$$A_s : \Big\{ \phi \in \Phi, \text{so that} \quad (2.79) \quad \text{is satisfied} \Big\}. \tag{3.20}$$

In general, A_s is not easy to obtain. We propose using an independent sampling scheme to approximate A_s: each player uses a possibly different sampling method to obtain $\tilde{A}_i$, a sampled version of A, where the subscript i emphasizes that each player might obtain a different $\tilde{A}_i$ set. Then, player i checks which action profiles $a \in \tilde{A}_i$ are valid equilibria for her. Then, player i shares her equilibrium points with the rest of the players. If this point is a valid equilibrium for all players, then they all add a to A_s, that is, $A_s = A_s \cup a$. Note that for NE, we sample actions a, and for CE, ϕ distributions.

Observe that each player can use a different sampling scheme. We propose three different sampling methods. The first is equispaced sampling on the action space A or the ϕ space. The second is random sampling: each player randomly obtains a sampled space $\tilde{A}_i$ following a certain distribution. Those two methods are brute-force ones. The third method we propose is using an intelligent sampling method, based on optimization. Each player samples A trying to maximize her payoff. We define the following reward function for each player i:

$$f_i(a) = \lambda z_i(a) + (1-\lambda) z_{-i}(a), \tag{3.21}$$

where $z_i(a)$ is a function that measures how good action a is to player i, $z_{-i}(a)$ does the same for the rest of the players and $\lambda \in [0,1]$ is a parameter that allows modeling how much $f_i(a)$ takes into account player i reward and the rest of the players. We use:

$$z_i(a) = \begin{cases} ||r_i(a) - r_{i,p}|| & \text{if} \quad r_i(a) \geq r_{i,p} \text{ and} \\ & a \text{ is an equilibrium} \\ -||r_i(a) - r_{i,p}|| & \text{if} \quad \text{otherwise} \end{cases}, \tag{3.22}$$

$$z_{-i}(a) = \sum_{j \neq i} z_j(a)$$

where $||x||$ is the Euclidean norm of vector x. Our definition of $z_i(a)$ is positive only if the payoff that action a provides to player i is higher than the punishment payoff: the highest this payoff is, the highest $z_i(a)$ will be. Also observe that $z_{-i}(a)$ provides an average on the payoff gain of the rest of the players; other metrics, such as the minimum payoff gain, could be used as well. Observe that in (3.22), each player computes $z_i(a)$ and then shares it to the rest of the players. If this is not desired or possible, we set $\lambda = 1$, and thus, $f_i(a) = z_i(a)$: each player does not take into account the rest of the players.

The intelligent sampling proposed is based on each player maximizing (3.21). Intuitively, we sample A so that we maximize (3.21): this sampling is intelligent because it finds actions a or ϕ distributions with high probabilities to be equilibria to all players. As optimization algorithm, we will use Simultaneous Optimistic Optimization (SOO) [157]. SOO is a non-convex optimization algorithm that allows maximizing a deterministic function when the function is smooth around one of its global maxima, using a limited number of evaluations.

SOO is a very adequate algorithm for this sampling method. First, because our objective function (3.21) is deterministic and possibly unknown: each player only knows her own payoff, and hence, the term $z_i(a)$; but she does not know the payoff of the other players. This means that, unless $\lambda = 1$, each player does not know a term of the objective function (3.21). Second, because it allows finding an approximation to a maximum with a finite number of evaluations: this means that SOO will try to find a maximizer as good as possible with a fixed number of samples. And third, because SOO does not require the objective function (3.21) to be convex, but only to be locally smooth around a local maximum, which (3.21) and (3.22) satisfy.

We limit the maximum number of communications that each player can initiate to N_c, i.e., each player can ask up to N_c times whether an action is a valid equilibrium or not to other players. This assumes that the cost of evaluating whether a point is an equilibrium or not is negligible when compared to the cost of communicating. If that were not the case, we can limit the maximum number of points sampled. We set this limit in order to control the execution time.

Observe that our sampling methods also allow to exploit the heterogeneity of the players: some players might have a higher computational capacity than the rest. The computationally more powerful players might sample using more complicated schemes than the other players, which benefits them and also may benefit the other players. Also, observe that the aim of the players is to distributedly obtain A_s. They are only allowed to ask other players whether an action vector is a valid equilibrium for them. In this way, each player needs not knowing what is the payoff function of other players, and hence, CA works in incomplete information games.

Finally, it might happen that no equilibrium point is found, i.e., $A_s = \emptyset$. This might occur for two reasons: either the sampling is not fine enough or there is no possibility to obtain a better payoff than the punishment payoff. The former case could be solved by performing a denser sampling, which increases the computational cost. The latter is the case in which the static equilibrium used as punishment cannot be improved, such as in zero-sum games or in cases where γ values are not high enough to satisfy the Folk Theorem. When $A_s = \emptyset$, each player makes $A = a_{i,p}$. Hence, CA guarantees to return a strategy that provides all players with a payoff equal or higher to the punishment payoff.

The action space sampling procedure is summarized in Algorithm 10. Note that players may simultaneously question others about a point a and at the same time, be asked about other point. Due to this, we have put the tasks of asking, answering and updating A_s as separate threads.

Pareto pruning

The second component of CA algorithm is Pareto pruning. This mechanism selects one of the valid equilibria found in the action space sampling stage. Hence, the problem is distributedly choosing a strategy from A_s for

Algorithm 10 Action space sampling for player i

Input: γ, r_i, $a_{i,p}$, $r_{i,p}$, N_p, N_c, na

{Questioning thread}
Initialize $A_s = \emptyset$
for N_c iterations **do**
  $a = obtain - sample(na)$ {Use the sampling scheme desired}
  if a is an equilibrium ((3.19) or (3.20)) and $r_i(a) \geq r_{i,p}$ **then**
    Ask other players if a is a valid equilibrium
    if All player answer 'YES' **then**
      $A_s = A_s \cup a$
      Tell all players that a is a valid equilibrium
{Answering thread}
for all a that player is asked about **do**
  if a is an equilibrium ((3.19) or (3.20)) and $r_i(a) \geq r_{i,p}$ **then**
    Answer 'YES'
  else
    Answer 'NO'
{Updating thread}
for all a that other players tell as valid equilibria **do**
  $A_s = A_s \cup a$

Output: A_s

all players. CA algorithm assumes that all players seek to optimize their own payoff functions selfishly. Hence, choosing an equilibrium $a \in A_s$ is not straightforward: each player may prefer different equilibria and no player should dominate others when choosing.

We solve this by using a jointly controlled lottery [15]. A jointly controlled lottery is a procedure that allows obtaining a random outcome distributedly. In [146], the following join controlled lottery is proposed for two players: each player simultaneously chooses a random number that follows a uniform distribution in $[0,1]$, that is, $w_i \sim U[0,1]$. Then, we obtain w as:

$$w = \begin{cases} w_1 + w_2 & if \quad w_1 + w_2 < 1 \\ w_1 + w_2 - 1 & if \quad w_1 + w_2 \geq 1 \end{cases}, \tag{3.23}$$

and w will follow a uniform distribution in $[0,1]$. If one player chooses w_i deterministically, w would still follow a uniform random distribution if the other takes w_i randomly. Thus, player i ensures a random w by simply using a random w_i. Now, let us assume that A_s has a certain indexing equal for all players. If we have L actions in A_s, where each action has an index $l \in \{1,2,...,L\}$, players select the action with index l:

$$l = \lfloor 1 + (L-1)w \rfloor, \tag{3.24}$$

where $\lfloor x \rfloor$ denotes the integer part of x. The action index l follows a uniform distribution in the interval $[1,L]$ and hence, a random action a_l will be selected. It is important to remark that a_l has been randomly selected among all valid actions in A_s to avoid a player dominating this choice. This procedure can be extended to more than two players.

After choosing a_l, each player prunes the actions $a \in A_s$ such that $r_i(a) < r_i(a_l)$. That is, each player eliminates the actions that are Pareto-dominated by a_l. When a player erases an action a, it communicates to other players, so that all players can update A_s by erasing a as well. After each pruning procedure, it may

Algorithm 11 Pareto pruning

Input: A_s, r_i
while $|A_s| > 1$ **do**
  Run jointly-controlled lottery (e.g., use (3.23))
  Obtain random action $a_l \in A_s$ (e.g., use (3.24))
  for all $a \in A_s, \quad a \neq a_c$ **do**
    if $r_i(a) < r_i(a_l)$ **then**
      $A_s = A_s \setminus a$
      Inform other players that a is not a valid equilibrium
  {Listening thread}
  for all Actions a that other players communicate as non valid equilibrium **do**
    $A_s = A_s \setminus a$
Output: A_s

happen that $|A_s| = 1$ (i.e., $A_s = \{a_l\}$), which means that a_l is an action that Pareto-dominates the rest of actions $a \in A_s$ and hence, it is Pareto-efficient and returned as the UNR strategy. Otherwise, $|A_s| > 1$ means that there is another action that Pareto-dominates a_l. In that case, the process starts again: a new jointly controlled lottery is performed, a new action a_l is chosen and the set A_s is pruned again, until $|A_s| = 1$.

A description of the Pareto-pruning procedure is in Algorithm 11. Note that in each pruning, players may simultaneously inform and be informed: due to this, we have separated the questioning and the listening tasks as separate threads. Also, all players must wait each other to have fully pruned A_s before checking whether $|A_s| > 1$ and prune again or not.

Observe that CA scalability depends on two aspects: the ability of each player to sample the actions space and check if a point is a valid equilibrium (computational cost, which increases with the number of players and actions) and also, on the efficiency of the communications among players, which depend on the network topology and protocols used. If we assume that the former cost is negligible compared to the communications cost, we can model the scalability of CA by observing that it is an example of the atomic-commitment problem [237]. The atomic-commitment problem appears in a distributed system, in which different subsystems have to apply an operation if and only if all subsystems apply it successfully; otherwise, the operation is reversed. In our case, the operation is checking whether a joint-action vector a is an equilibrium for player i and adding it to A_s if and only if a is an equilibrium for all players. In [237], it is shown that in absence of communication failures, there is an efficient, polynomial time algorithm that minimizes this cost. Thus, the total cost of CA depends on N_c for the action space sampling and it is polynomial for the Pareto pruning, assuming that the computational cost is negligible when compared to the communication cost among players.

3.4.2 Error bounds in CA algorithm

Now, we proceed to study the error that CA algorithm induces. As we show, this error comes from the sampling procedure, and it affects only the NE case. We provide a theoretical bound for that error, as well as particularized expressions for the two players, two actions case.

General theoretical bounds

In this Section, we study the error introduced by CA algorithm. We start with the NE error, whose equilibrium condition is (3.19). Observe that the error comes from the sampling method: if we were able to sample all points in A, that is, $A = \tilde{A}$, there would be no error. Let us assume that each player samples with a method

that guarantees that the maximal distance between two actions for player i in $\tilde{A}_i$ is Δa_i. Thus, for two sampled actions $\tilde{a}$, $\tilde{a}' \in \tilde{A}_i$, with $\tilde{a}_k = \tilde{a}'_k$ if and only if $k \neq i$, i.e., the two actions vectors differ only in the action of player i:

$$\max ||\tilde{a_1} - \tilde{a_2}|| \leq \Delta a_i. \tag{3.25}$$

Now, observe (3.19). CA algorithm checks this equilibrium condition after sampling, which means that the equilibrium condition that each player computes, for two actions $\tilde{a}$, $\tilde{a}' \in \tilde{A}_i$, with $\tilde{a}_k = \tilde{a}'_k$ if and only if $k \neq i$ is:

$$(1-\gamma) r_i(\tilde{a}') + \gamma r_i(a_p) - r_i(\tilde{a}) \leq 0, \quad \forall \tilde{a}'. \tag{3.26}$$

We can simplify by observing that the equilibrium condition for player i assumes that the actions of the other players are fixed, hence, it is a condition that only affects the actions of player i, a_i:

$$(1-\gamma) r_i(\tilde{a}'_i) + \gamma r_i(a_{p,i}) - r_i(\tilde{a}_i) \leq 0, \forall \tilde{a}'_i. \tag{3.27}$$

However, due to sampling, there might be an action a_i, which has not been sampled, such that $r_i(a_i) > \max_{\tilde{a}'_i} r_i(\tilde{a}'_i)$. We define $\Delta r_i = r_i(a) - r_i(\tilde{a}')$, as the difference in payoffs. This would mean that the equilibrium that CA algorithm computes would not be anymore an NE, but a Nash ε_i-equilibrium, when $\Delta r_i > 0$. Hence, (3.27) would become:

$$(1-\gamma) \left[r_i(\tilde{a}'_i) + \Delta r_i \right] + \gamma r_i(a_{p,i}) - r_i(\tilde{a}_i) \leq \varepsilon_i, \forall \tilde{a}'_i. \tag{3.28}$$

Using (3.27) and (3.28), we obtain the following bound for ε_i:

$$\varepsilon_i = (1-\gamma) \Delta r_i. \tag{3.29}$$

In order to bound Δr_i, we will assume that r_i functions are Lipschitz-continuous in the action set A, that is:

$$||r_i(a_1) - r_i(a_2)|| \leq C_i ||a_1 - a_2||, \forall a_1, a_2 \in A, \tag{3.30}$$

where C_i is the Lipschitz constant for the function r_i. The lowest C_i that satisfies (3.30) is called the best Lipschitz constant, and will be denoted by C_i^*.

Let us assume that in (3.30), $a_1 = \tilde{a}_i$ is a sampled action and $a_2 = a_i$ is an action which was not sampled. In the worst case, according to (3.25), $||\tilde{a}_i - a_i|| = \Delta a_i$ and hence, using (3.30), $||r_i(\tilde{a}_i) - r_i(a_i)|| \leq C_i^* \Delta a_i$. Since the function r_i is a function returning real numbers, $||r_i(\tilde{a}_i) - r_i(a_i)|| = |r_i(\tilde{a}_i) - r_i(a_i)| = \Delta r_i$ if $r_i(a) > r_i(\tilde{a}')$. Hence, $\Delta r_i \leq C_i^* \Delta a_i$. Thus, we can bound the error as:

$$\varepsilon_i = (1-\gamma) C_i^* \Delta a_i. \tag{3.31}$$

Observe that Δr_i measures the error induced by the possible actions a_i which are not sampled and which cause that the sampled action $\tilde{a}_i$ is not an equilibrium, but an ε_i-equilibrium. Hence, the case when $r_i(a) < r_i(\tilde{a}')$ is not of interest to us, because the action not sampled returns a lower payoff than the sampled and hence, (3.27) holds for $\varepsilon_i = 0$.

The result in (3.31) means that there are three factors that contribute to the error in the equilibrium obtained. The first one is the discount factor: as γ tends to 1, the error decreases. It also depends on C_i, which is an upper bound on the variation of the function as can be observed in (3.30). Since C_i is an upper bound, the tightest ε_i will be achieved with the lowest C_i possible, which is C_i^*. Finally, the last component is the maximal distance

between a sampled and a not sampled action. As the number of actions sampled tends to infinity, this term will tend to 0 and hence, $\varepsilon_i \to 0$, which means that CA algorithm finds an NE asymptotically.

In the case of CE, CA does not introduce any error. Note that for any value of $\tilde{\phi}$, if (3.20) holds, it will be a CE, with no error. In CE, we sample distributions $\tilde{\phi}$ instead of actions, thus, when a sampled $\tilde{\phi}$ distribution satisfies the equilibrium condition (3.20), it will be an exact CE.

Finally, we note that we do not provide any guarantees on whether CA will be able to find an RG equilibrium. This depends on the discount factor, the kind of game, i.e., zero-sum or general sum, and the sampling method used. But we do assure that when CA finds a NE, it will be an ε_i-equilibrium for each player bounded by (3.31) and when CA finds a CE, it will contain no error.

Theoretical bounds on a 2 player, 2 action game, equispaced sampling

In this section, we particularize (3.31) for the case in which there are $N_p = 2$ players and each player has 2 pure actions. We denote the pure actions payoffs that player i receives as $r_{i,jk}$, where i denotes the player, j denotes the pure action of player 1 and k denotes the pure action of player 2. Note that $i, j, k = \{1, 2\}$. We denote the mixed action of player 1 as $(y, 1-y)$, where y is the probability that player 1 assigns to her pure action 1 and $1-y$ the probability assigned to her pure action 2. The mixed action of player 2 is defined equivalently by $(z, 1-z)$. Thus, the mixed action space of the game A is the unit square $A = A_1 \times A_2 = A = [0,1] \times [0,1]$, where one axis are the y values and the other, the z values.

We sample A using equispaced sampling, with K_i samples in each dimension. Hence, for each player, the sampled mixed actions are $\{0, \frac{1}{K_i-1}, \frac{2}{K_i-1}, ..., 1\}$. The maximum distance between a sampled action and a not sampled one will take place when the not sampled action lies in the middle of two sampled actions, hence, $\Delta a_i = \frac{1}{2(K_i-1)}$. The payoff function r_i has the following form:

$$\begin{aligned} r_i(y,z) =& yzr_{i,11} + y(1-z)r_{i,12} + (1-y)zr_{i,21} + (1-y)(1-z)r_{i,22} \\ =& A_i yz + B_i y + C_i z + D_i \end{aligned}, \tag{3.32}$$

where

$$\begin{aligned} A_i =& r_{i,11} - r_{i,12} - r_{i,21} + r_{i,22} \\ B_i =& r_{i,12} - r_{i,22} \\ C_i =& r_{i,21} - r_{i,22} \\ D_i =& r_{i,22} \end{aligned}. \tag{3.33}$$

Since the payoff functions $r_i(y,z)$ are polynomials, they are continuous, derivable and with bounded derivatives in A, which is a convex subset of $\mathbb{R}^2$. A continuous function $f(x)$ with bounded derivatives is Lipschitz-continuous with $C^* = \sup_x ||\nabla f(x)||$ (see [109, Lemma 2.18]). Now, there are two possible approaches. Remark that player i is interested in computing ε_i, a bound on the error she is committing when she evaluates an NE. Since she knows the actions of other players, player i can fix the actions of other players in r_i, so that r_i becomes a one variable function that only depends on a_i. In that case:

$$C_i^* = \max_{a_i} \left\{ \left. \frac{dr_i(a_i, a_{-i})}{da_i} \right|_{a_{-i}} \right\}$$

$$\begin{pmatrix} (1,-1) & (-1,1) \\ (-1,1) & (1,-1) \end{pmatrix}$$
(a) Matching pennies (MP).

$$\begin{pmatrix} (2,2) & (-1,3) \\ (3,-1) & (0,0) \end{pmatrix}$$
(b) Prisoner's dilemma (PD).

$$\begin{pmatrix} (2,1) & (0,0) \\ (0,0) & (1,2) \end{pmatrix}$$
(c) Battle of sexes (BS).

$$\begin{pmatrix} (-10,-10) & (1,-1) \\ (-1,1) & (0,0) \end{pmatrix}$$
(a) Chicken game (CG).

Fig. 3.9 Payoff matrices for the four games proposed. Player 1 is row player, and player 2 is column player, hence, the first row stands for pure action 1 of player 1, and row 2 for her pure action 2. The first column contains the pure action 1 of player 2, and the second column, her pure action 2. In each matrix, the payoff entries for each pair of pure actions are (r_1, r_2).

Yet this implies that player i bound, ε_i, depends on a_{-i}, the actions of the other players. A second option is the worst case one by using $C_i^* = \sup_{(a_i, a_{-i})} ||\nabla r_i(a_i, a_{-i})||$. This option yields a higher Lipschitz constant and hence, a less tight ε_i value, but it provides an upper bound for ε_i independent of the actions of other players. Using (3.32), we obtain $\nabla r_i = (A_i z + B_i, A_i y + C_i)$ and $||\nabla r_i|| = \sqrt{(A_i z + B_i)^2 + (A_i y + C_i)^2}$. Hence, we obtain the following ε_i bound using (3.31):

$$\varepsilon_i = \frac{1-\gamma}{2(K_i - 1)} \left\{ \max_{0 \le y \le 1, 0 \le z \le 1} \sqrt{(A_i z + B_i)^2 + (A_i y + C_i)^2} \right\} \tag{3.34}$$

3.4.3 Empirical results

Now, we proceed to test CA on a set of four RGs, two of which are the same as in the LEWIS case, and the other two are chosen because they help to illustrate how CA distributedly tries to improve a given equilibrium. In all games, there are $N_p = 2$ players with 2 pure actions each. We choose games with very different characteristics, whose payoff matrices are in Figure 3.9. Remark that a static NE for these games will have the form $a_1 = (y, 1-y)$, $a_2 = (z, 1-z)$ and each static NE is also a CE of the form $\phi = (yz, y(1-z), (1-y)z, (1-y)(1-z))$.

The first game is matching pennies (MP), a zero-sum game. This means that $r_1 = -r_2$ and hence, the gains of one player are the losses of the other. This game has only one static NE: $a_1 = a_2 = (1/2, 1/2)$ (the equivalent CE is $\phi = (1/4, 1/4, 1/4, 1/4)$), which yields each player a payoff of $V_1 = V_2 = 0$. No gain in payoffs can be achieved by repeating the game with respect to the static equilibrium.

The second game is the prisoner's dilemma (PD), which is a non-zero sum game, used frequently to illustrate the Folk Theorem [146]. There is only one static NE, which is $a_1 = a_2 = (0, 1)$, which provides each player with a payoff of $V_1 = V_2 = 0$ (the equivalent CE is $\phi = (0, 0, 0, 1)$). However, when the game is repeated for a sufficiently high value of γ, new equilibria arise, and in this case, it is possible to achieve a payoff as high as $V_1 = V_2 = 2$ using UNR strategy: for a theoretical analysis, see [146, Ch. 2 - 3].

The third game is the Battle of sexes (BS), which is a non-zero sum game with three different static NE, namely, $a_1 = (2/3, 1/3)$, $a_2 = (1/3, 2/3)$, which yields the players a payoff of $V_1 = V_2 = 2/3$, $a_1 = (1, 0)$, $a_2 = (1, 0)$, which yields payoffs $(V_1, V_2) = (2, 1)$, and $a_1 = (0, 1)$, $a_2 = (0, 1)$, which yields them payoffs $(V_1, V_2) = (1, 2)$. Note that the two pure action equilibria are Pareto-efficient, but the mixed equilibrium is not.

The fourth game is the chicken game (CG), which is a non-zero sum game with three different static NE, namely, $a_1 = a_2 = (1/10, 9/10)$, which yields the players a payoff of $V_1 = V_2 = -1/10$, $a_1 = (1, 0)$, $a_2 = (0, 1)$, which yields payoffs $(V_1, V_2) = (1, -1)$, and $a_1 = (0, 1)$, $a_2 = (1, 0)$, which yields them payoffs $(V_1, V_2) = (-1, 1)$.

Simulation 5: The influence of the sampling method

First, we study the differences among the sampling methods proposed. We use the PD game with $\gamma = 0.9$. The analytical solutions to PD game can be found in [146]. For the γ value we are using, the Pareto-efficient region is:

$$V_p = \begin{cases} V_2 = \frac{-V_1+8}{3} & \text{if} \quad V_1 \in [0,2] \\ V_2 = -3V_1 + 8 & \text{if} \quad V_1 \in [2,8/3] \end{cases} . \tag{3.35}$$

We define ξ as the distance between a payoff and the Pareto frontier:

$$\xi = \min ||V(a) - V_p||, \tag{3.36}$$

where $V(a)$ is the RG payoff vector to all players by playing action vector a and V_p is the Pareto region (3.35). We use ξ to compare the performance of the three sampling methods we proposed: equispaced, random and SOO. In order to show empirically the advantages of using SOO, we will limit in equispaced and random sampling the number of communications to N_c, and in SOO, we only sample N_c times (see section 3.4.1). We test for $N_c \in [5, 200]$.

First, we obtain a stage equilibrium using RM algorithm, which provides a static equilibrium for all players. We use 10^3 iterations for RM. The equilibrium that RM returns is used as punishment strategy for CA algorithm. Then, we run CA using all the sampling schemes proposed. In case of NE, the mixed action space is a square $A = [0,1] \times [0,1]$. For equispaced sampling, we used $K_i = 50$ samples in each dimension. In the case of CE, we equispacedly sample a simplex of dimension 3: note that $\sum_{k=1}^{4} \phi_k = 1$. We use approximately the same number of points that for the NE case, 2500, i.e., we test approximately 2500 ϕ distributions. Then, we also test using random sampling, following a uniform distribution. In the case of NE, each player randomly generates a pair of actions following a uniform distribution between 0 and 1, that is, $(y,z) \sim (U[0,1], U[0,1])$. We limit the maximum number of actions tested to 10^4 for each player: if no equilibrium is found, the sampling procedure is exited. For the CE concept, each player samples uniformly in a simplex and again, we limit to 10^4 the maximum number of samples if no equilibrium is found. Finally, we test SOO sampling method. We use (3.21) and (3.22), with $N_c = 10$ and $\lambda = \{0.5, 1\}$, for both NE and CE. Finally, in order to test in NE whether there are profitable deviations or not (see Definition 8), we sample $a_i' \in [0,1]$ equispacedly using 50 samples. This is used to check the NE deviation condition every time that is needed.

For each value of N_c or N_s in SOO case, we run 100 times CA, and the resulting payoff errors are computed using (3.36) and (3.35). The results can be observed in Figure 3.10, where we observe that CA approaches the Pareto frontier as the number of communications increases. We also observe that SOO sampling method outperforms the others, even though it has a stricter limitation: number of samples instead of number of communications. Thus, SOO intelligent sampling presents a clear advantage over the other methods proposed.

Simulation 6: CA Performance

We also simulated the performance of CA in the four games described above. We use three possible values for the discount factor: $\gamma = \{0.1, 0.5, 0.9\}$. For each γ value and each game, we run 100 different repetitions: in each repetition we use the same 9 algorithms that we used in the previous simulation: regret matching and 8 different instances of CA, 4 for NE and other 4 for CE. We use the same parameters as in the previous simulation, except that we fix $N_c = 100$ for equispaced and random sampling, and $N_s = N_c$ for SOO sampling.

We run the simulations and obtain the region of payoffs for each algorithm and the payoff that, in average, is obtained in each setup. The results can be observed in Figure 3.11, where we plot the payoff increase in

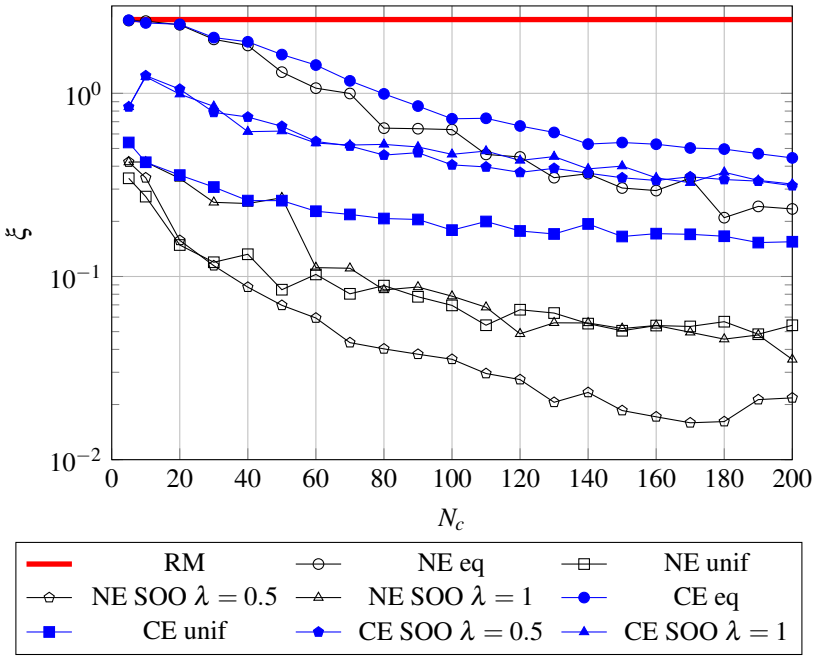
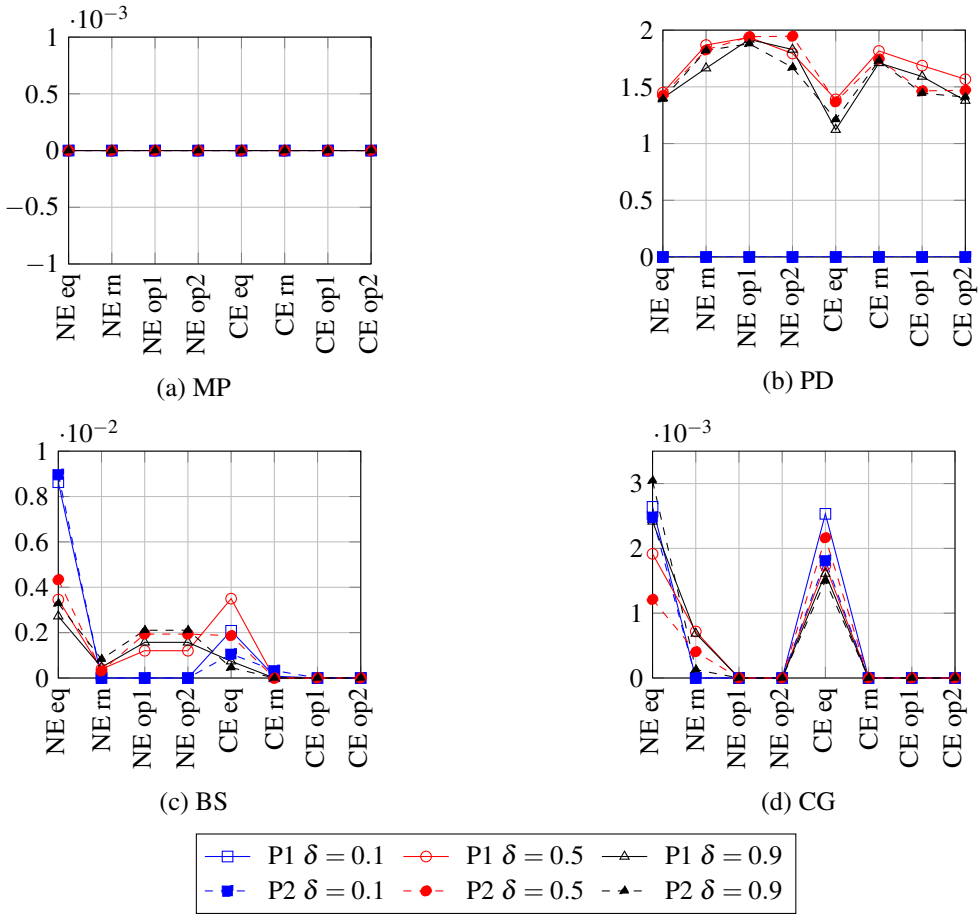


Fig. 3.10 Average values of ξ for each sampling method. Equispaced sampling is 'eq', random uniform sampling is 'unif' and 'RM' stands for regret-matching results. We observe that, as we increase the number of communications allowed N_c, the error ξ decreases. Recall that ξ measures how far the CA results are from the theoretical Pareto frontier (see (3.36)). Thus, lower is better, as it implies that the players achieve a payoff closer to the Pareto frontier. Note that a greater N_c allows getting closer to the Pareto frontier. The sampling methods order, from the worst to the best performance, are equispaced, random uniform and SOO. Even though in SOO we use a stricter limitation, as we limit in samples instead of communications, it outperforms the other sampling methods.

the four games described that CA yields for the static equilibrium that RM provides. For MP, BS and CG, it is possible to observe that there is no significant increment in payoffs between CA and RM, as we expected. MP is a zero-sum game and hence, the Folk Theorem does not apply. In BS and CG, RM returns a static, Pareto-efficient payoff: since the payoff is already efficient, CA does not find a better one. The case of PD is the most important and interesting one, because it is a game in which both players can benefit of repeating the game. It is possible to observe that CA does not improve the theoretical payoff for $\gamma = 0.1$, because with that discount factor, the only equilibrium of the RG is the static one. Yet as γ increases, new equilibria arise and the gains of using CA appear, as the Folk Theorem conditions are satisfied.

As an example, in Figure 3.12 we include some of the payoff regions returned by CA algorithm. We observe that in case of PD game, as γ increases, a whole region of new payoffs appears: these payoffs can be achieved by repeating the game following the UNR strategy proposed. Although this needs not be the case in all games: in the case of BS game, the static equilibrium used as punishment is already Pareto-efficient and hence, CA cannot improve it. We also show how SOO provides similar results in terms of Pareto-efficient payoffs, with significantly fewer samples and equilibria evaluations.

In Table 3.1, it is possible to observe the different ε_i values obtained, using (3.31), for equispaced sampling and NE. The ε_i values are low, except in the game of the chicken, due to the higher values of the derivatives in this payoff matrix.



(a) MP (b) PD

(c) BS (d) CG

Fig. 3.11 Payoff results: for each game, we represent the average payoff increment ΔV_i between CA and RM for different values of γ. Thus, higher is better, as it means that CA provides better payoffs than RM. We use four sampling methods for CA: equispaced (eq), random uniform (rn), SOO with $\lambda = 0.5$ (op1) and SOO with $\lambda = 1$ (op2), for NE and CE. When CA takes advantage of the Folk Theorem, it outperforms RM, as happens in PD. And when using the Folk Theorem provides no advantage in payoffs, as in MP, BS and CG, CA is not worse than RM, as expected.

Recall that in all simulations, we used as CA input the RM equilibrium, in order to be used as punishment equilibrium and explore the possibility of obtaining a better payoff in the RG by using UNR strategy. In our game testbench, that was the case only of PD, and in that case, CA algorithm outperforms clearly RM, because it exploits the new equilibria that appear when repeating the game, as the Folk Theorem states. However, this increment does not come at no cost. Observe that for games where the best possible equilibrium is the static one, CA algorithm does not improve RM. Hence, CA does not make sense if we know for sure that we can obtain no gain by repeating the game. Yet if we know or hope that new equilibria may arise by repeating the game, CA may find equilibria which yield better payoff for all players, such as in PD. Recall that CA needs no a priori information on the kind of game being played and hence, it works on incomplete information games.

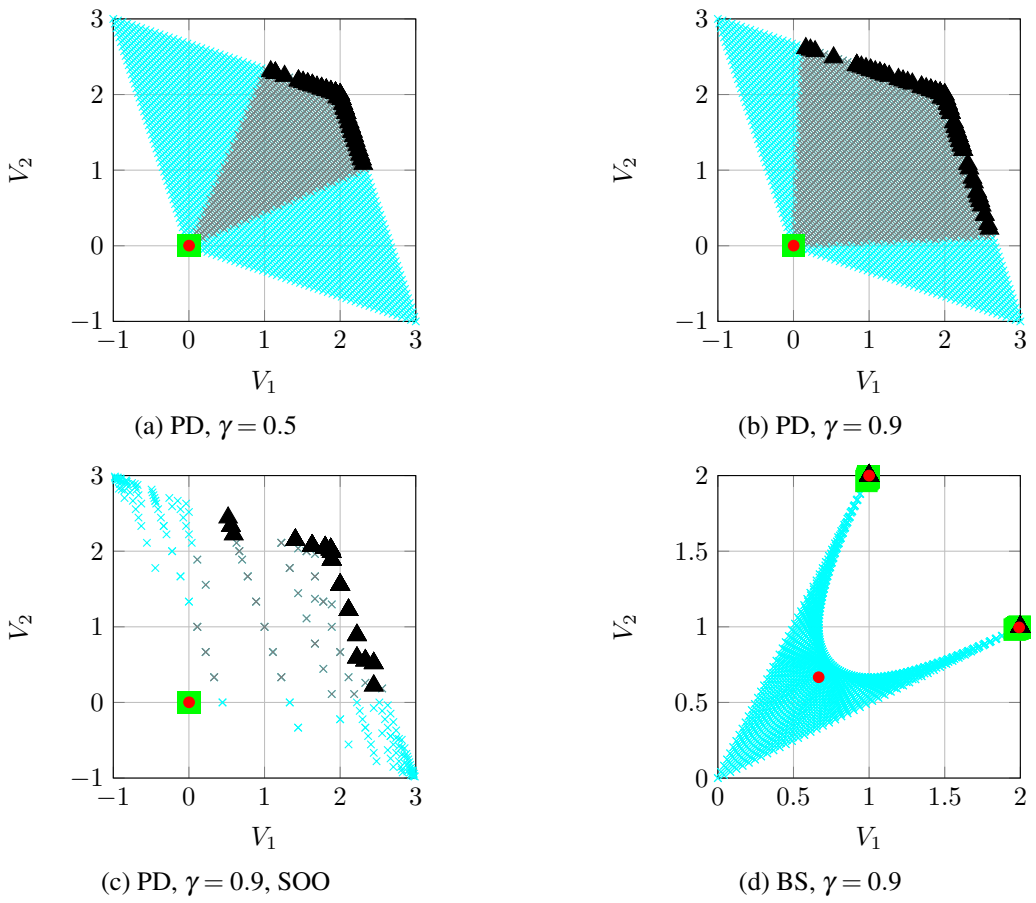
Also, in the case of PD, observe that there are differences between different implementations of CA. In general, NE yields higher payoff than CE, because the region of NE, a square in the plane, is smaller than the CE region, a simplex with 3 dimensions. Some aspects that may help in practice are:

Game	$\gamma = 0.1$	$\gamma = 0.5$	$\gamma = 0.9$
MP	0.0509	0.0283	0.0057
PD	0.0569	0.0316	0.0063
BS	0.0509	0.0283	0.0057
CG	0.2558	0.1421	0.0284

Table 3.1 Comparison of theoretical ε_i values for the Nash equilibrium concept, when using equispaced sampling, according to (3.31), where $K_i = 50$. In all cases, $\varepsilon_1 = \varepsilon_2$, that is, both players had the same bound.

- First, think whether CE makes sense in our game setup. This means that either we have access to a correlating device or to a jointly-controlled lottery [15]. CE has two advantages over NE: first, the region of CE always contains NE, so there might be games in which there are CE that yield a better payoff than NE. And secondly, CA algorithm guarantees that any CE found will be exact, whereas NE have a bounded error, according to (3.31), see Table 3.1. However, since the CE region has a higher dimensionality than the NE region, the sampling schemes may perform poorly.
- The computational capacity for sampling purposes. Using SOO and limiting the number of samples allows performing fewer sampling operations, see Figure 3.12. However, it implies implementing SOO algorithm [157].
- A final aspect is related to the ability of detecting a deviation. UNR strategy needs to detect deviations immediately, i.e., it needs perfect information. In the case of NE, this means having access to the mixed actions of each player. In the case of CE, since a deviation is not following the recommendation ϕ, if we have an entity that sends the ϕ recommendation to each player, this device can detect whether a player deviates or not instantaneously, i.e., it player i does not play the pure action recommended. Thus, CE eases detecting a deviation.

Finally, we note that CA could be modified to use other strategies or even equilibrium conditions, by changing the equilibrium conditions (3.19) or (3.20).



(a) PD, $\gamma = 0.5$

(b) PD, $\gamma = 0.9$

(c) PD, $\gamma = 0.9$, SOO

(d) BS, $\gamma = 0.9$

Fig. 3.12 Comparison of NE payoff regions in PD and BS games. In light blue, we observe the possible payoff region, the gray darker region is the set of payoff equilibria in the RG. The red circles are the theoretical static payoff equilibria, the green squares are the payoff equilibria returned by RM and the black triangles are the payoff equilibria returned by CA. Note that RM always provides a static equilibrium payoff. Sampling in regions (a), (b) and (d) is equispaced with 2500 samples, whereas region (c) was sampled using SOO with $\lambda = 1$. We note that (1) increasing γ might provide a larger payoff equilibria region, as the Folk Theorem says: compare (a) and (b); (2) if a static equilibrium is already Pareto-efficient, CA cannot improve it, as shown in (d); (3) SOO provides similar equilibria to equispaced sampling taking much fewer samples: compare (b) and (c). Thus, CA with SOO sampling produces the best results both in terms of payoffs and samples taken.

3.5 Conclusions

In this Chapter, we first have shown theoretically that using discounted payoff instead of average payoff has an impact both in the speed of learning and in the variance of the total payoff achieved. The first is due to the fact that most of the payoff is assigned in the first stages of the game (Theorem 2), and this means that an algorithm for RGs needs to be fast, for the first game stages are crucial in terms of payoffs. The second means that the variance in payoffs does not only depend on the stages of the game, but also on the discount factor value (Theorem 3). And nowadays, most learning schemes for RGs are designed for average payoff schemes, rather than discounted ones. As we use in our problems the discounted case, we have introduced two algorithms that are specifically designed to deal with the discounted payoff case.

Our first algorithm is LEWIS, which is derived for discounted RGs with incomplete information. LEWIS allows a fast and secure learning in such games, and our simulations show that it is able to cooperate in self play, provide a minimum payoff in the worst case, improves the security payoff when possible and is also competitive when facing other algorithms designed for RGs. As we showed, there are applications in which low discount factor arise, and hence, LEWIS would be a good algorithm for these problems.

Our second algorithm is CA, which is a negotiation-based algorithm that allows computing equilibria of RGs of perfect information using the averaged discounted payoff criterion. CA is based on the idea that players can communicate each other the strategies they are willing to use. This allows, if possible, to reach an RG equilibrium. Our results show that CA is a powerful and flexible algorithm, with plenty of positive features: it is completely distributed, it is valid for N_p players, it is valid for incomplete information games, it improves when possible an input static equilibrium, it chooses Pareto-efficient payoffs, it works both using NE or CE and it may be adapted to different strategies by modifying the equilibrium conditions (3.19) or (3.20), it can use intelligent sampling methods and finally, CA takes advantage of heterogeneous computational capacity of each player in the sampling stage. In case of NE, it returns an ε_i-equilibrium for all players, where ε_i is bounded. In case of CE, it returns an exact equilibrium. And finally, we remark that CA takes advantage of the new equilibria that may arise by repeating the game, according to the Folk Theorem, in order to improve the payoff that all players can obtain in an RG.

These two algorithms are used in Chapter 4 in order to address the first security problem that we study: a backoff attack, that affects the CSMA/CA multiple access mechanism when some sensors use a different backoff rule to transmit. We show that this situation can be modeled using an RG, and we obtain solutions to it both in the static case and the repeated one, for NE and CE, using the algorithms presented in this Chapter.

Chapter 4

Backoff attack under a Repeated Game approach

4.1 Introduction

In this Chapter, we present the first attack to WSNs that we study in depth using tools from the Chapters 2 and 3: the backoff attack. We note that the remarkable advances and proliferation in wireless networks in the last years have brought a significant interest in the security and threats to WSNs: they can be the target of many attacks due to the limited capabilities of the sensors, as some recent surveys show [72], [257]. One of these attacks is the backoff attack, which affects to the Medium Access Control (MAC) layer when a CSMA/CA (carrier-sense medium access with collision avoidance) scheme is used to regulate the access to the medium. The backoff mechanism minimizes the risk of collision, i.e., that two or more sensors transmit simultaneously, by deferring transmissions during a certain random time period: the backoff window. In a backoff attack, a sensor uses a lower backoff window than the rest of the sensors, thus obtaining a higher throughput at expense of the other sensors [21].

Backoff attacks are a real threat to WSNs. Firstly, because network adapters are highly programmable [37], thus allowing sensors to modify their backoff parameters. And secondly, because many MAC layer protocols proposed for WSNs make use of CSMA as medium access mechanism, for instance, SMAC [255], WiseMAC [67], TMAC [224] and DSMAC [137]. Actually, two surveys on MAC layer protocols, [57] and [249], show that CSMA is the most common access mechanism in contention based MAC protocols.

There are works on backoff attacks, such as [229] and [219], which focus only on the defense mechanism. However, any attack is a conflict between the attackers and the defense mechanism. In order to better model this conflict, we make use of GT tools in this Chapter. This approach is already popular: [7] is a survey on GT approaches to multiple access situations in wireless networks and [78] is another survey focused on CSMA methods.

Two important works which study backoff attacks in wireless networks are [123] and [37]. Our approach in this Chapter differs from these works in the following points:

- We assume that the defense mechanism lies in a central server, to which the sensors communicate; thus, we assume a star network topology, in which a central server receives the packets of the rest of the network. We model the conflict individually between each sensor, which can follow the backoff procedure or attack, and the server. We consider that there are sensors that always follow the backoff

Chapter	CSMA/CA	CSS	Player	Information	Observation (A/S)	Behavior
4	Yes	No	Attack Defense	Complete Complete	Mixed / - Mixed / -	Static Static
5	Yes	Yes	Attack Defense	Complete Incomplete	- / State Realization / -	Dynamic Static
6	Yes	Yes	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / -	Dynamic Static
7	Yes	No	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / Observation	Dynamic Dynamic

Table 4.1 Table comparing the different setups used in Chapters 4-7. CSMA/CA. i.e., the backoff attack, and CSS, i.e., the SSDF attack, denote whether each of these setups is used in the Chapter. Information denotes whether each player knows the target of the other player (Complete) or not (Incomplete). Observation refers to what each agent observes with respect to the actions / states of the other players: regarding actions, they observe the mixed actions or the actions realizations, and regarding states, they observe the state or an observation of the rest of players: this is related to having perfect or imperfect information. Behavior refers to whether the player adapts its behavior with time or not.

procedure, which are Good Sensors (GSs), and other sensors that may attack, the Attacking Sensors (ASs). Note that the star topology appears, for instance, in hierarchical routing protocols [83]: in these protocols, the sensors are clustered in order to be energy efficient [252] and each cluster follows a star topology. Even though our approach could be adapted to other network topologies, we focus on the star topology in this Chapter for simplicity. By differentiating between ASs and the server, we use a heterogeneous network model: the ASs are greedy and want the maximum individual throughput they can obtain, whereas the server tries to divide fairly the total throughput among the sensors that communicate with it. This makes our model different from [123] and [37]: each sensor may have different interests, which is a more complex and realistic situation.

- We use Bianchi's model to estimate the total network throughput and use this metric as game payoff: we try to enforce a fair use of the network total throughput. By modeling the total throughput, we contribute to provide a deeper insight on how different parameters influence the fairness of the network. Namely, we show that fairness is related to the backoff parameters and the number of ASs.

- We solve our game both analytically and empirically. On one side, we provide a theoretical framework to the backoff attack problem based on GT and solve it analytically. On the other side, we use LEWIS and CA, the two algorithms designed to learn the solution of a repeated game, as well as RM, to find the solution to the game when there are more than two players. These algorithms are simple to implement, even in sensors with low computational capabilities. This makes our model both well theoretically founded and also, practical to implement in real life situations.

We note that we refer to the sensors as stations if we study the network from the MAC protocol perspective or as players or agents if we are studying the network from the GT perspective. In this Chapter, we assume (1) that the attackers and the defense mechanism know the target of the others, and hence, the defense mechanism knows which sensors may attack, i.e., complete information; (2) that all agents are able to observe the mixed actions of the others, which means that they can detect instantaneously any deviation from a negotiated strategy and the defense mechanism detects instantaneously deviations from the backoff procedure, i.e., perfect information; and (3) that all agents respect a predefined behavior rule and hence, can be considered static, as shown in Table

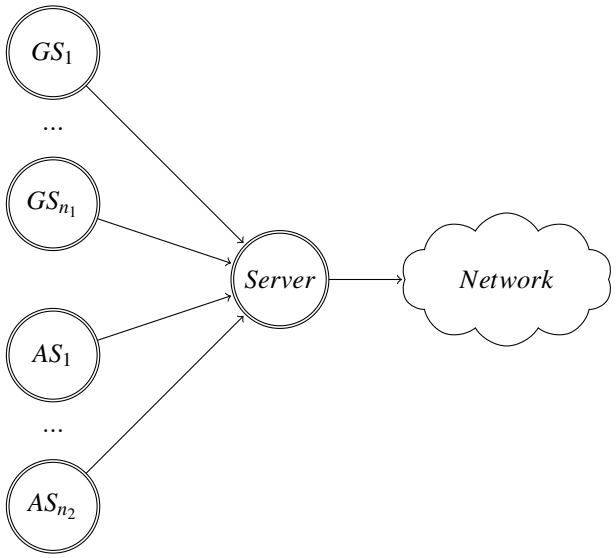


Fig. 4.1 Network scheme for the case that there are n_1 GSs and n_2 ASs. GSs respect 802.11 binary exponential backoff, whereas ASs can choose to use it or to use a uniform backoff.

4.1. Hence, we assume that we are in a perfect and complete information game. Note, however, that both CA and LEWIS are able to deal with incomplete information situations as well, although in this Chapter we introduce the simplest version of the backoff attack, leaving more complex situations for Chapters 5-7.

We start by describing the CSMA/CA mechanism as it is implemented by the IEEE 802.11 standard in Section 4.2. Then, Section 4.3 obtains the total network throughput under CSMA/CA mechanism when some sensors in the network do not follow the backoff procedure: as we will see, they have a deep impact with respect to the fair distribution of the network resources. In order to alleviate this situation, we model this as a game and solve it using both Static Game tools in Section 4.4 and Repeated Game tools in Section 4.5, where we deeply analyze the two player case and also use algorithms from Chapters 2 and 3 to generalize to the N_p players case. Then, in Section 4.6 we discuss two possible ways in which we can detect instantaneously deviations from a prescribed mixed action without having access to the randomizing device of each player. Finally, we draw some conclusions in Section 4.7 and point towards some aspects of interest of this Chapter that are treated in the next ones.

4.2 CSMA/CA in IEEE 802.11

We start by describing the CSMA/CA mechanism in the 802.11 standard. Note that we base on this implementation because it is a well-known one. The IEEE 802.11 standard [111] defines the MAC and physical (PHY) layer specifications for a wireless local area network (WLAN). Each device connected using this standard is known as station. The access to the shared medium can be regulated using the Distributed Coordination Function (DCF), which uses CSMA/CA to access the medium. In this Chapter, we use the network topology shown in Figure 4.1, where there are n_1 GSs and n_2 ASs connected to a server using a star topology.

The basic mechanism used by the DCF in IEEE 802.11 standard is CSMA/CA to control the medium access and a positive acknowledgment frame (ACK): if no ACK is received, there is a retransmission. CSMA/CA operates using two procedures: a carrier sense (CS) which determines whether the channel is busy, i.e., other

station is transmitting, or idle, i.e., no other station is transmitting; and a backoff procedure which determines when a station should start transmitting.

A station willing to transmit invokes the CS mechanism to determine whether the channel is idle or not. If it is busy, the station defers the transmission until the channel is idle without interruption for a fixed period of time. After, the station starts a counter, called backoff, for an additional deferral time before transmitting: the station transmits when its backoff counter reaches 0. This procedure minimizes collisions among multiple stations that have been deferring to the same event. The backoff follows a uniform random variable in the interval $[0, CW-1]$, where CW stands for contention window. If a collision is detected while a station transmits, its CW is duplicated, which is known as binary exponential backoff, and the backoff procedure starts over. When the station has transmitted the packet, it waits for an ACK; if none is received in a certain time, the station starts the transmission procedure again. This mechanism is known as Basic Access (BA), and is based on a two-way handshaking.

The standard also defines an alternative procedure, based on a four-way handshaking, called request-to-send/clear-to-send (RTS/CTS). In this case, the transmitter station sends an RTS frame to the receiver, using the BA mechanism described above. The RTS frame is used to reserve the medium: when the receiver station receives an RTS, proceeds to reserve the channel for some time, sending a CTS frame to indicate that the channel reservation was successful. When the transmitter receives the CTS frame, starts transmitting its packet; when it finishes, if the transmission was successful, the receiving station sends a positive ACK. While the channel is reserved, the rest of stations remain silent. The RTS/CTS procedure helps to ease the hidden node problem [36], [190] and provides a higher throughput than the BA mechanism when the MAC payload is large [28].

4.3 Network Throughput under Backoff Modification

The previous Section described the backoff mechanism that all sensors are supposed to follow. However, ASs may use a different backoff rule in order to obtain a benefit, namely, that they transmit more often. In this Section, we proceed to thoroughly study this case for the concrete case that the ASs follow a uniform backoff, instead of the binary exponential mechanism just described. As we will see, the impact of the ASs misbehavior on the network distribution of the resources can be huge.

4.3.1 Theoretical Network Throughput

The 802.11 standard does not provide a way to estimate the network throughput. The best-known model to estimate the throughput in a network is Bianchi's model [28], which provides expressions both for BA and RTS/CTS mechanisms. The main advantage of this model is that it provides analytical expressions to determine the network throughput. It assumes saturation of the network, that is, that each sensor always has a packet to transmit. This assumption could be relaxed using more complex models [147].

The CSMA/CA mechanism described in Section 4.2 assumes that all sensors respect the backoff procedure. However, the stations can modify their backoff in such a way that they can obtain a higher throughput, at expense of other stations [37], [123]. In order to analyze these effects, we use Bianchi's model [28] to estimate the total network throughput. The results are used in posterior Sections to study how to enforce network throughput fairness using GT tools. This model relies on the computation of the following system, where we

assume that our WSN has I sensors willing to transmit:

$$\begin{cases} \tau_i = \frac{2}{1+W_i+p_iW_i\sum_{j=0}^{m_i-1}(2p_i)^j} \\ p_i = 1-\prod_{j\neq i}(1-\tau_j) \end{cases}, \tag{4.1}$$

where $i \in \{1,2,...,I\}$ indexes the sensor and p_i is the collision probability for sensor i: the probability that sensor i observes a collision while transmitting a packet, which Bianchi's model assumes to be constant. Also, τ_i is the probability that sensor i transmits a packet. The system (4.1) assumes a binary exponential backoff, where the contention window CW lies in the interval $[W, CW_{max}]$, where $CW_{max} = 2^m W$, where m is the maximum backoff stage and W is the minimum size of the contention window.

Let us assume that we have a network with I sensors, split into two different classes. There are n_1 GSs characterized by using a binary exponential backoff as described by IEEE 802.11 standard, and thus, following (4.1). Also, there are $n_2 = I - n_1$ ASs using a uniformly distributed backoff in the range $[0, W_2 - 1]$, whose expression [28] is:

$$\tau_i = \frac{2}{1+W_i}. \tag{4.2}$$

The probabilities τ_i and p_i are the same for all the members of each class. Hence, (4.1) becomes:

$$\begin{cases} \tau_1 = \frac{2}{1+W_1+p_1W_1\sum_{j=0}^{m_1-1}(2p_1)^j} \\ \tau_2 = \frac{2}{1+W_2} \\ p_1 = 1-(1-\tau_1)^{n_1-1}(1-\tau_2)^{n_2} \\ p_2 = 1-(1-\tau_1)^{n_1}(1-\tau_2)^{n_2-1} \end{cases}, \tag{4.3}$$

where the subscript $i \in \{1,2\}$ denotes the class of a sensor: the index 1 refers to GSs, and the index 2 refers to ASs. Now, we obtain the total throughput of the network [28] [147]. The probability that there is at least one sensor transmitting is denoted as P_{tr}:

$$P_{tr} = 1-\prod_{i=1}^{I}(1-\tau_i) = 1-(1-\tau_1)^{n_1}(1-\tau_2)^{n_2}, \tag{4.4}$$

and hence, $1-P_{tr}$ is the probability that no sensor is transmitting. The probability that there is exactly one sensor of class i transmitting, $P_{s,i}$, is:

$$\begin{cases} P_{s,1} = \tau_1(1-\tau_1)^{n_1-1}(1-\tau_2)^{n_2} \\ P_{s,2} = \tau_2(1-\tau_1)^{n_1}(1-\tau_2)^{n_2-1} \end{cases}, \tag{4.5}$$

and the probability that there are two or more sensors transmitting simultaneously, i.e., the collision probability, denoted by P_c, is obtained as the total probability minus the probabilities of having exactly none or one sensor transmitting:

$$P_c = 1-\sum_i P_s - (1-P_{tr}) = P_{tr} - n_1P_{s,1} - n_2P_{s,2}. \tag{4.6}$$

Now, we obtain the expected duration of a slot time, T_{slot}. We define T_s as the time to count down a backoff unit, i.e., the time that lies between two consecutive calls to the CS method when the channel was sensed idle. T_t is the time duration of a successful transmission and T_c is the time duration of a collision. We assume that the sensors of both classes share the same duration of a successful transmission and the same duration of a

collision. Thus, T_{slot} is:

$$T_{slot} = (1-P_{tr})T_s + (n_1 P_{s,1} + n_2 P_{s,2})T_t + P_c T_c. \tag{4.7}$$

We consider T_p the payload information time duration in a successful transmission, and we assume that all sensors share the same T_p. We define S_i, the throughput ratio for sensor i, as the fraction of time used by sensor i to successfully transmit payload bits. S_i is obtained as:

$$S_i = \frac{P_{s,i}T_p}{T_{slot}} = \frac{P_{s,i}T_p}{(1-P_{tr})T_s + (n_1 P_{s,1} + n_2 P_{s,2})T_t + P_c T_c}. \tag{4.8}$$

In (4.8), we could use units of time for the magnitudes T_p, T_s, T_t and T_c, or measure its length in bits, as long as the units are the same for the four parameters. Finally, the total network throughput, defined as the fraction of the time spent by all the sensors transmitting successfully payload bits, is:

$$S = \sum_{i=1}^{I} S_i = n_1 S_1 + n_2 S_2. \tag{4.9}$$

The parameters T_s, T_t and T_c are obtained from the 802.11 standard. T_s is the empty slot time. In case of using BA mechanism, we have [28]:

$$\begin{cases} T_c^{ba} = H + T_p + DIFS + \delta \\ T_t^{ba} = H + T_p + SIFS + \delta + ACK + DIFS + \delta \end{cases}, \tag{4.10}$$

where H is the total header transmission time, obtained by adding PHY and MAC layers headers; $DIFS$ and $SIFS$ are interframe spacing defined in the standard, ACK is the transmission time of an ACK and δ is the propagation delay. We also consider that all payloads have the same size, whose transmission time is T_p. In case of using RTS/CTS mechanism, we have [28]:

$$\begin{cases} T_c^{rts} = RTS + DIFS + \delta \\ T_t^{rts} = RTS + SIFS + \delta + CTS + SIFS + \delta + T_t^{ba} \end{cases}. \tag{4.11}$$

Comparing (4.10) and (4.11), we see that BA mechanism uses less time for a successful transmission, whereas the time spent in a collision depends on the payload size. Intuitively, in case of large payloads and a high collision probability, RTS/CTS could achieve a higher throughput, since less time is spent on retransmissions and that might compensate the longer time spent on transmitting. Indeed, this result is observed in [28].

4.3.2 Simulation 1: Network Throughput and Fairness

Now, we will make use of the expressions derived in the previous Section to analyze the impact of having n_2 ASs that follow a uniform backoff, and hence, do not respect the binary backoff procedure. The values used for each time duration are the same as in [28], extracted from 802.11 standard, and can be seen in Table 4.2. Observe that we consider two different payload lengths, a short one, $T_{p,s}$, and a long one, $T_{p,l}$. We consider that GSs follow the IEEE 802.11 standard binary backoff mechanism with $W_1 = CW_{min,1} = 32$, $CW_{max,1} = 1024$ and hence, $m_1 = 5$. ASs will follow a uniformly distributed backoff in the interval $[0, W_2 - 1]$.

With these values, we obtain the throughput for each sensor using (4.8) and (4.9) for these cases:

- Using the large payload: $T_p = T_{p,l}$. We test four cases: first, we consider that $n_2 = 0$, that is, all stations follow the binary exponential backoff, and we vary the number of stations for $I \in [1, 20]$. Then, we fix the

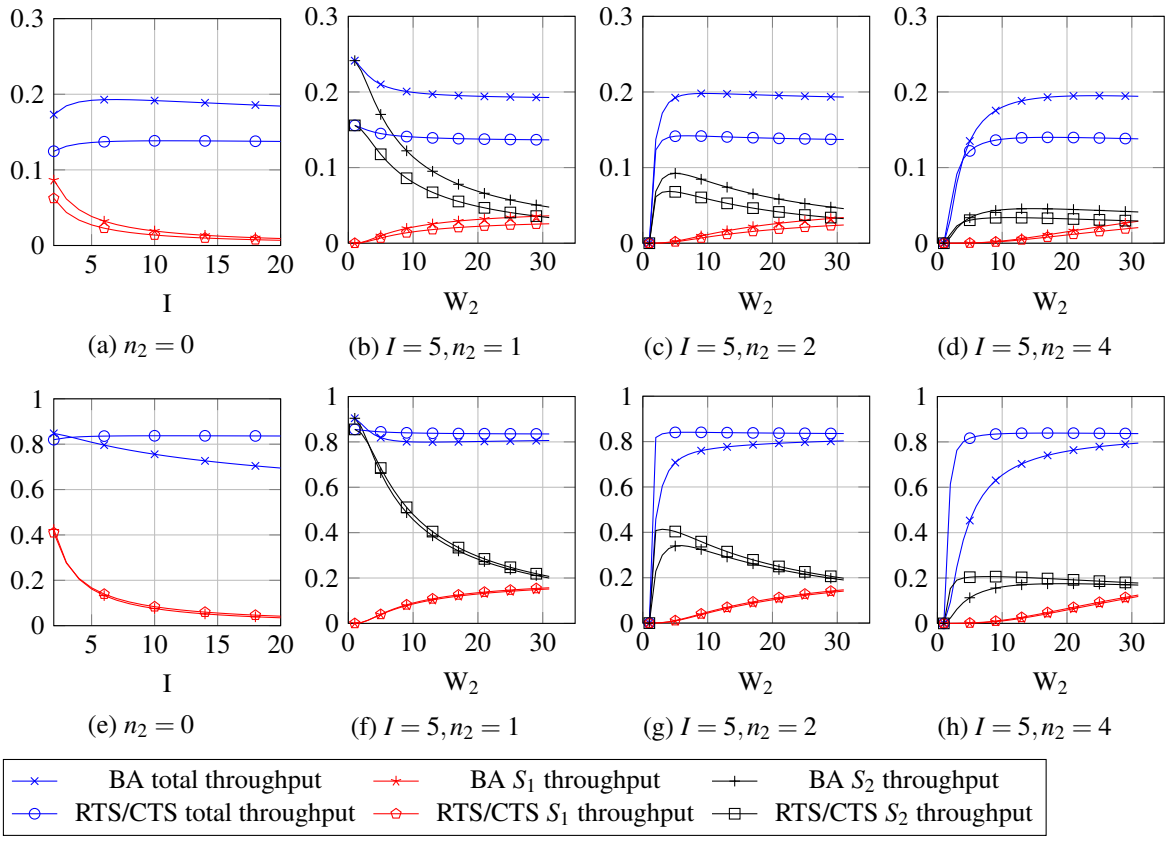


Fig. 4.2 Throughput S results for the simulation, using Bianchi's model with short payload, $T_{p,l}$, (a-d), and long payload, $T_{p,l}$, (e-h). In cases (a) and (e), there are no ASs; in cases (b-d) and (f-h) there are ASs. S_1 is the throughput of normal stations, S_2 the throughput of malicious stations. Note how having ASs significantly decreases the throughput of GSs.

number of stations to $I = 5$ and simulate for $n_2 \in \{1,2,4\}$, that is, for respectively 1, 2 and 4 malicious stations. We show the results for different values of W_2, namely, for $W_2 \in [1, W_1]$. The obtained results are in Figure 4.2.

- Using the short payload, $T_p = T_{p,s}$. We test the same four cases than we did for the large payload case. The obtained results are in Figure 4.2.

4.3.3 Discussion

The results presented in Figure 4.2 show that:

- The throughput of GSs decreases significantly for low values of W_2. This is independent of the number of ASs, the mechanism used, i.e., BA or RTS/CTS, and the payload size. This happens because ASs use lower backoffs and hence, they have higher chances to win the contention procedure against GSs. This causes that the throughput is not fairly distributed among sensors. As W_2 increases, the ASs behave more similarly to the GSs and the difference in throughput becomes smaller.
- If there is only one AS, this sensor consumes the major part of the network throughput for low W_2, because it usually wins the contention procedures. This is independent of the mechanism used, i.e., BA

Parameter	Value	Parameter	Value
$T_{p,s}$	256 bits	$T_{p,l}$	8184 bits
MAC header	272 bits	PHY header	128 bits
ACK	112 bits + PHY header	RTS	160 bits + PHY header
CTS	272 bits + PHY header	Bit rate	1 Mbps
δ	1 μs	T_s	50 μs
SIFS	28 μs	DIFS	128 μs

Table 4.2 Values used for simulation 1.

or RTS/CTS, and the payload size. However, when there are more than one AS, the total throughput becomes 0 for $W_2 = 1$, because there are several stations trying to access the network that will always collide. As the W_2 value increases, we observe that the throughput for the ASs also increases, presenting a maximum value which depends on the total number of sensors in the network and the W_2 parameter. Also, as n_2 increases, the throughput that an AS obtains decreases: it is better for an AS to be the only AS in the network.

- RTS/CTS mechanism provides higher throughput when using larger payloads: in Figure 4.2 (e-h), RTS/CTS curves are always above BA curves. The opposite happens when using short payloads.

Hence, if in a WSN using CSMA/CA there is one or more sensors which can modify the binary exponential backoff procedure used by 802.11, the throughput that each sensor gets can be seriously affected. This happens using both BA or RTS/CTS mechanisms. The results obtained in this Section show that network fairness is seriously affected by a backoff attack; the next Sections propose a solution to this situation using GT tools.

4.4 Solving the backoff attack using Static Games

We use the network scheme in Figure 4.1 to model the CSMA/CA problem that arises when some sensors modify their backoff procedure. There are n_1 GSs, which always follow the binary exponential backoff, and n_2 ASs, which can choose between using the binary exponential backoff or the uniform backoff. We denote by I the number of stations, with $I = n_1 + n_2$. All I sensors are connected to a server which forwards their packets to a network. The sensors communicate with the server: we only consider the uplink in the problem. Observe that this problem arises in a situation in which a star topology is used.

The players of the game are the server on one side, and the ASs on the other. Thus, there are $N_p = n_2 + 1$ players in the game. Each AS tries to maximize the throughput available to it, whereas the server tries to enforce that all stations in the network obtain a fair throughput. By fair, we mean that no sensor is getting a higher throughput at expense of others. Under the saturation condition imposed before, this means that all ASs receive the same proportion of the total throughput.

The ASs have two different actions: either they behave selfishly (s) by using the uniform backoff, or they do not behave selfishly (ns) by using the binary exponential backoff. The server will also have two actions: it can use a test to detect if the network throughput is being fairly distributed (d) or not to test the network (nd). If the server detects and catches an AS behaving selfishly, it will drop its packet, as punishment. This means that the AS has to send again the packet, and the throughput advantage it had obtained by modifying its backoff vanishes. We must also take into account that this detection procedure cannot be free of charge for the server: there must be a cost associated to the detection procedure in terms of computational resources.

Two of the possible schemes that could be used to detect this selfish behavior are [219], which is based in Kolmogorov-Smirnov (K-S) statistics, and [229], which is based on a modified Cramer-von Mises (C-M) test [10]. To simplify the modeling, we assume that the server is able to perfectly detect when a sensor behaves selfishly: in Chapters 5 and 6 we will show the deep implications that this assumption has.

4.4.1 Obtaining the payoff functions

In order to model the backoff attack using GT tools, we first need to obtain the payoff functions for each player. We obtain them first for the two player case, and then generalize our results for an arbitrary number of players.

Two Player Case

Now, we center in the case when $n_2 = 1$, that is, there are only two players in the game: the server and one AS. We proceed to describe the payoffs for each player. We denote by S_1^{ns} the throughput that the AS can obtain by playing *ns*. In that case, the n_1 normal stations will obtain each a throughput $S_{n_1}^{ns} = S_1^{ns} = S^{ns}$, that is, all stations obtain the same amount of throughput, i.e., cases (a) and (e) in Figure 4.2. If the AS plays *s*, it obtains a throughput S_c^1 if the server plays *nd*. This causes the normal stations to have a lower throughput, $S_{n_1}^s < S_1^s$, as observed in Figure 4.2.

We define $-k_d$ (with $k_d > 0$) as the cost of detecting malicious behavior for the server. We model the cost function for AS and server as a linear function of the throughput, with k_s and k_j as a constant for the server and the AS respectively. The payoff functions that the agents try to maximize are defined as follows:

- If they play (nd, s), where the first action corresponds to the server, the second to the AS, the AS is modifying its backoff and hence, the throughput in the network. The server does not detect this modification, and hence, does not punish the AS. Thus, the AS obtains a throughput increment, which provides it a gain of $k_1(S_1^s - S^{ns})$. The server has a cost proportional the throughput loss that the normal stations suffer: $k_s n_1(S_{n_1}^s - S^{ns})$.
- If they play (d, s), the AS modifies its backoff, but it is caught by the server because it tests the network and hence, the server drops the packet of the AS. This causes the AS a loss of $k_1(0 - S^{ns}) = -k_1 S^{ns}$. The server has a gain proportional to the throughput that the normal stations would have lost minus the cost of the detection: $k_s n_1(S^{ns} - S_{n_1}^s) - k_d$.
- If they play (d, ns), the AS does not modify its backoff and hence, does not affect the throughput. Hence, it has no gain nor lose. But the server tests the network, and hence, it incurs in the cost of detection, expressed as $-k_d$.
- If they play (nd, ns), the AS again has no gain nor lose. The server does not test, and hence, it incurs in no cost since the AS is behaving properly: it also has no gain nor loss.

All of these payoffs do not vary along the game, provided that there is no modification of the game conditions, e.g., the number of players. Since $N_p = 2$, we can pose the game as a bimatrix, non-zero sum, static game, whose game payoffs as a function of the player actions are in Table 4.3.

In order to simplify, we will replace the payoff values in Table 4.3 for the following constants, where R_1 is the payoff matrix for the server and R_2 for the AS:

$$R_1 = \begin{pmatrix} -\alpha_m & 0 \\ \alpha_c & -\alpha_f \end{pmatrix} \quad R_2 = \begin{pmatrix} \beta_s & 0 \\ -\beta_c & 0 \end{pmatrix}. \tag{4.12}$$

	s	ns
nd	$\Big(k_s n_1(S^s_{n_1} - S^{ns}), k_1(S^s_1 - S^{ns})\Big)$	$\Big(0,0\Big)$
d	$\Big(k_s n_1(S^{ns} - S^s_{n_1}) - k_d, -k_1 S^{ns}\Big)$	$\Big(-k_d, 0\Big)$

Table 4.3 Payoffs values for the game posed, when $n_2 = 1$. The payoff vectors are of the form $r = (r_1, r_2)$, where r_1 is the payoff of the server and r_2 is the payoff of the AS.

Observe that all parameters in (4.12) are strictly positive, that is, $\alpha_c, \alpha_m, \alpha_f, \beta_s, \beta_c \in (0, +\infty)$. This arises because:

- k_1, k_c, k_d, n_1 and all the throughput values $S^s_{n_1}$, S^s_1 and S^{ns} are strictly positive parameters.
- The throughput of the AS must be higher if it behaves selfishly than if it does not. If that were not the case, this would mean that the AS achieves higher throughput by following the exponential binary backoff, and from Figure 4.2, we see that this is not the case if there is only one AS ($n_2 = 1$). This means that $S^s_1 > S^{ns}$.
- The throughput of the GSs must decrease when the AS behaves selfishly with respect to their throughput when the AS follows the binary exponential backoff. As we observe in Figure 4.2, that is indeed the case if there are malicious stations, i.e., $n_2 \geq 1$. This means that $S^{ns} > S^s_{n_1}$.
- It must happen that $k_s n_1(S^{ns} - S^s_{n_1}) > k_d$: observe that the previous point showed that the left-hand side is positive. This simply means that the cost of detecting is lower than the gain of detecting a deviation from the AS. If that did not happen, it would be counter intuitive: the server incurs in a loss when it successfully detects a deviation from the AS.

Observe that our model includes the case in which there is no selfishness in the AS as a particular case. If the server knows that the AS will always play *ns*, i.e., like a GS, then the server will always play *nd* and hence, both players receive a payoff of 0.

Extension to more than two players

The payoff functions derived in the previous Section for the case that there are only two players can be extended to the general case when there are more than two players. In this case, again, there is one server which can choose between two pure actions (*d*, *nd*) and there are $n_2 > 1$ ASs, each AS being able to choose between two pure actions (*s*, *ns*). In the general case, the payoff function of each player is a multidimensional array of dimensions $na_1 \times na_2 \times ... \times na_{N_p}$, where na_i denotes the number of pure actions available to player $i \in \{1, N_p\}$. Observe that when $N_p = 2$, the payoff function of each player is a matrix.

We define a vector of pure actions as $a_p = (a_{p,1}, a_{p,2}, ..., a_{p,N_p})$. Observe that the payoff multidimensional array contains a payoff value for each possible vector a_p. In order to obtain the payoff function of each player, for each a_p, we define n^s_2 as the number of ASs that play pure action *s* and $n^{ns}_2 = n_2 - n^s_2$ as the number of ASs that play pure action *ns*. The payoff each player receives is coupled with the actions of the rest of the players: in general, it is a function $f_i(a_p)$, where *i* denotes a concrete player. There is a payoff function for each a_p.

The payoff function for the server depends on a_p as follows. If the server plays *d*, the payoff function of the server is $k_s n_1(S^{ns} - S^s_{n_1}) - k_d$. Remark that S^{ns} is obtained considering that there are $n = n_1 + n_2$ stations. Also, there are different possible values of n^s_2, thus $S^s_{n_1}$ depends on the number of ASs playing *s*. Finally, observe that

if all ASs played *ns*, $n_2^s = 0$ and hence, $S^{ns} = S^s_{n_1}$; thus, the payoff of the server in this case is $-k_d$. If the server plays *nd*, the payoff value for each a_p is $k_s n_1 (S^s_{n_1} - S^{ns})$. It is the same as when the server played d, but now there is no cost k_d and the sign is reversed.

The payoff for AS $j, j \in \{1,...,n_2\}$ if it plays *ns* is 0. If AS j plays s and the server plays d, the payoff for AS j is $-k_j S^{ns}$. If AS j plays s and the server plays *nd*, the payoff for AS j is $k_j(S^s_j - S^{ns})$. Observe that S^s_j depends on n_2^s.

We follow this procedure for each a_p value in order to obtain the payoff values. Observe that if $n_2 = 1$, all the expressions in this Section reduce to the ones given in the previous Section.

4.4.2 Analysis for the two players case

In this Section, we solve analytically the backoff attack or CSMA/CA game that we have just described for the two player case using the Static Game concepts that we advanced in Chapter 2.5.1.

Nash Equilibrium Solution

The CSMA/CA game can be solved using the mixed NE concept. Using (2.68), the game presents the following NE, where the payoff matrices used are (4.12):

$$y^* = \frac{\beta_c}{\beta_c + \beta_s} \qquad z^* = \frac{\alpha_f}{\alpha_f + \alpha_m + \alpha_c}. \tag{4.13}$$

This means that the server plays d with probability $1 - y^*$ and *nd* with probability y^*. The AS plays s with probability z^* and *ns* with probability $1 - z^*$. We define the expected payoff that each player obtains if they play mixed strategies with probability $(y, 1-y)$ for the server and $(z, 1-z)$ for the AS as:

$$\begin{aligned} r_1(y,z) =& (y,1-y)R_1(z,1-z)^T = -zy(\alpha_m + \alpha_c + \alpha_f) + z(\alpha_c + \alpha_f) + \alpha_f(y-1) \\ r_2(y,z) =& (y,1-y)R_2(z,1-z)^T = zy(\beta_s + \beta_c) - z\beta_c \end{aligned}, \tag{4.14}$$

which means that the payoff that each player receives by playing their mixed NE strategy, using (4.13) and (4.14), is:

$$r_1 = -\frac{\alpha_f \alpha_m}{\alpha_m + \alpha_c + \alpha_f} \qquad r_2 = 0. \tag{4.15}$$

The values in (4.15) show that the equilibrium payoff for the AS is 0, and the equilibrium payoff for the server depends on the values that the α parameters take. This means that the AS can guarantee itself a throughput as good as if it behaved normally. The server has always a loss, derived from the cost of detecting, which is k_d, collected by the parameter α_f.

Correlated Equilibrium Solution

The CSMA/CA game can be solved using the CE concept, where we note that there are several possible ways in which a correlator might be implemented in a WSN, that we present later in this Chapter. The equilibrium

condition for the CE case, (2.70), becomes in our setting:

$$\begin{array}{c}\sum\limits_{a_2=\{s,ns\}}\phi(a_2|d)r_1(d,a_2)\geq\sum\limits_{a_2=\{s,ns\}}\phi(a_2|d)r_1(nd,a_2)\\ \sum\limits_{a_2=\{s,ns\}}\phi(a_2|nd)r_1(nd,a_2)\geq\sum\limits_{a_2=\{s,ns\}}\phi(a_2|nd)r_1(d,a_2)\\ \sum\limits_{a_1=\{d,nd\}}\phi(a_1|s)r_2(s,a_1)\geq\sum\limits_{a_1=\{d,nd\}}\phi(a_1|s)r_2(ns,a_1)\\ \sum\limits_{a_1=\{d,nd\}}\phi(a_1|ns)r_2(ns,a_1)\geq\sum\limits_{a_1=\{d,nd\}}\phi(a_1|ns)r_2(s,a_1)\end{array}. \tag{4.16}$$

Replacing the payoffs from (4.12), (4.16) becomes:

$$\begin{array}{c}\alpha_c\phi(s|d)-\alpha_f\phi(ns|d)\geq-\alpha_m\phi(s|d)+0\phi(ns|d)\\ -\alpha_m\phi(s|nd)+0\phi(ns|nd)\geq\alpha_c\phi(s|nd)-\alpha_f\phi(ns|nd)\\ -\beta_c\phi(d|s)+\beta_s\phi(nd|s)\geq0\phi(d|s)+0\phi(nd|s)\\ 0\phi(d|ns)+0\phi(nd|ns)\geq-\beta_c\phi(d|ns)+\beta_s\phi(nd|ns)\end{array}. \tag{4.17}$$

We know that the following is satisfied:

$$\phi(a|b)=\frac{\phi(a\cap b)}{\phi(b)}\qquad\phi(a\cap b)=\phi(b\cap a), \tag{4.18}$$

and hence, we can simplify (4.17) using (4.18). We use the following shorthand notation: $\phi_{11}=\phi(nd\cap s)$, $\phi_{12}=\phi(nd\cap ns)$, $\phi_{21}=\phi(d\cap s)$ and $\phi_{22}=\phi(d\cap ns)$. Observe that this is the joint distribution probability, considering that the first subscript refers to the pure action of the server, and the second, to the pure action of the AS. We also consider that pure action 1 for the server is *nd*, and pure action 2, *d*; for the AS, *s* is its pure action 1 and *ns* its pure action 2. Using all these ideas, (4.17) becomes:

$$\begin{array}{c}-\alpha_m\phi_{11}+0\phi_{12}\geq\alpha_c\phi_{11}-\alpha_f\phi_{12}\\ \alpha_c\phi_{21}-\alpha_f\phi_{22}\geq-\alpha_m\phi_{21}+0\phi_{22}\\ \beta_s\phi_{11}-\beta_c\phi_{21}\geq0\phi_{11}+0\phi_{21}\\ 0\phi_{12}+0\phi_{22}\geq\beta_s\phi_{12}-\beta_c\phi_{22}\end{array}, \tag{4.19}$$

where we assumed that $\phi(nd)>0$, $\phi(d)>0$, $\phi(s)>0$ and $\phi(ns)>0$. By taking into account that all α and β parameters are greater than 0, that is, $\alpha,\beta\in(0,+\infty)$; and also constraining ϕ to be a valid distribution, we

obtain the following simplified CE conditions from (4.19):

$$\begin{aligned} \phi_{11}\phi_{22} &= \phi_{12}\phi_{21} \\ \frac{\beta_s}{\beta_c} &= \frac{\phi_{22}}{\phi_{12}} \\ \frac{\alpha_c+\alpha_m}{\alpha_f} &= \frac{\phi_{22}}{\phi_{21}} \\ \phi_{11}+\phi_{12}+\phi_{21}+\phi_{22} &= 1 \\ \phi_{ij} &\geq 0, \quad i=\{1,2\}, \quad j=\{1,2\} \\ \alpha,\beta &\in (0,+\infty) \end{aligned} \quad . \tag{4.20}$$

The system in (4.20) has only one solution:

$$\begin{aligned} \phi_{11} &= \frac{\alpha_f}{\alpha_c+\alpha_m+\alpha_c}\frac{\beta_c}{\beta_c+\beta_s} \qquad \phi_{12} = \frac{\alpha_c+\alpha_m}{\alpha_c+\alpha_m+\alpha_c}\frac{\beta_c}{\beta_c+\beta_s} \\ \phi_{21} &= \frac{\alpha_f}{\alpha_c+\alpha_m+\alpha_c}\frac{\beta_s}{\beta_c+\beta_s} \qquad \phi_{22} = \frac{\alpha_c+\alpha_m}{\alpha_c+\alpha_m+\alpha_c}\frac{\beta_s}{\beta_c+\beta_s} \end{aligned} . \tag{4.21}$$

Thus, there is only one CE which corresponds to the mixed NE we have already found in (4.13): observe that $\phi_{11} = y^*z^*$, $\phi_{12} = y^*(1-z^*)$, $\phi_{21} = (1-y^*)z^*$ and $\phi_{22} = (1-y^*)(1-z^*)$. This happens with all games following the payoff matrices from (4.12). The payoff for each player if they follow the CE is:

$$r_1 = -\alpha_m\phi_{11} + \alpha_c\phi_{21} - \alpha_f\phi_{22} \qquad r_2 = \beta_s\phi_{11} - \beta_c\phi_{21}, \tag{4.22}$$

where the payoffs obtained using CE by replacing (4.21) in (4.22) are the same that were obtained using mixed NE for our setting, in (4.15). This is obvious, as both are the same equilibrium.

4.4.3 Solving for more than two players

We propose using RM (see Chapter 2.5.1) to solve the CSMA/CA game for more than two players, as the theoretical analysis would be involved. We know that RM converges to the set of correlated equilibria [90], which in our game, for the two player case, is only one point (4.21). This CE is also the only NE of the game (4.13). Hence, in the CSMA/CA game with two players, RM converges to the NE, but in the general case, it converges to a CE of the game.

4.4.4 Simulation 2: The static CSMA/CA game

Now, we simulate the static CSMA/CA game in order to observe and compare the theoretical developments derived in previous sections. We define a network using the model in Figure 4.1, where we set the number of stations to $I = 5$, we use BA mechanism and $T_{p,l}$ in order to estimate the network throughput using Bianchi's model. The parameters of GSs, denoted by subscript 1 will be $W_1 = 32$, $CW_{max,1} = 1024$, and hence, $m_1 = 5$, as in the IEEE 802.11 standard. The ASs, denoted with subscript 2, uses the uniform random mechanism modification described in Section 4.3, with a window length $W_2 = 8$. The rest of IEEE 802.11 parameters are in Table 4.2, taken from [28]. We solve equations (4.3) to (4.10), and obtain the throughput values for different number of ASs: $n_2 \in \{1,2,3,4\}$.

	s	ns
nd	$\big(-0.3668, 0.3608)\big)$	$\big(0,0\big)$
d	$\big(0.2668, -0.1617\big)$	$\big(-0.1, 0\big)$

Table 4.4 Payoffs values for the game when $n_1 = 4$ and $n_2 = 1$. The first entry of the payoff vector is the server payoff, the second is the AS payoff.

n_2	Server r	ASs r
1	-0.0493	-0.0015
2	-0.0504	$(-0.0011, -0.0012)$
3	-0.0502	$(-0.0011, -0.0011, -0.0013)$
4	-0.0499	$(-0.0008, -0.0008, -0.0004, -0.0003)$

Table 4.5 Empirical payoffs obtained using RM for each value of n_2. Observe that payoffs do not significantly vary as the number of players increase. This is consistent with Figure 4.4: the game tends to the two player situation, even if there are more players.

We also need to define the parameters that are used to model the payoff functions. We use $k_s = k_c = 1$, $k_d = 0.1$. The payoff functions are obtained using Table 4.3 for the case of two players and the procedure described in Section 4.4.1 for the case $N_p > 2$. For two players, $S^{ns} = 0.1617$, $S^s_n = 0.0700$, $S^s_c = 0.5225$, and the payoff matrix obtained is in Table 4.4.

We can use (4.13) and Table 4.4 to obtain the theoretical solutions for the two player game. The mixed equilibrium actions are $y^* = 0.3095$ and $z^* = 0.1364$, which yield a payoff of -0.05 for the server and 0 for the AS. Recall that the CE, obtained using (4.21), yields the same equilibrium.

Then, we simulate using RM algorithm for $n_2 \in \{1,2,3,4\}$. We set the number of iterations $T = 2000$, and run the learning process 50 times. The empirical payoffs obtained are in Table 4.5, and in Figure 4.3, the histogram of the mixed actions obtained is represented for all the n_2 cases tested. We can compare to the theoretical results expected in the two player case, by computing the difference between the actions and payoff obtained using RM and the theoretical values using (4.13) and (4.15). The mean difference in mixed actions is -0.0224 ± 0.0183 (mean $\pm$ standard deviation) for the server and 0.0056 ± 0.0087 for the AS. The mean difference in payoffs is also small: 0.0007 ± 0.0024 for the server and -0.0015 ± 0.0013 for the AS. Thus, RM provides a very good approximation to the analytical game values.

It is of special interest noting that, for $n_2 \geq 2$, each of the ASs distribution presents two peaks, clearer as n_2 grows; one of them is nearly 0. We observe that in each game realization all ASs but one tend to behave as normal stations, i.e., they tend to play *ns*, as can be observed also in Figure 4.4 for $n_2 = 4$: AS 1 plays a mixed action around $z = 0.5$ and the rest of ASs tend to play $z = 0$, that is, they tend to always play *ns*. This means that the game tends to the two player case, even if there are more than two players. This might be due to having payoffs such that they do not encourage having more than one player behaving selfishly at once. As we saw in Figure 4.2, as the number of ASs increased, the advantages of playing *s* for the ASs decreased: the difference between the normal behavior throughput and the throughput obtained when using a different backoff diminished. Since the payoff of the ASs is proportional to this difference, it is not enough gain for them to play *ns*: the loss when they play *s* and the server plays *d* does not compensate the gains when they play *s* and the server plays *nd*; hence, it is better for them playing *ns*.

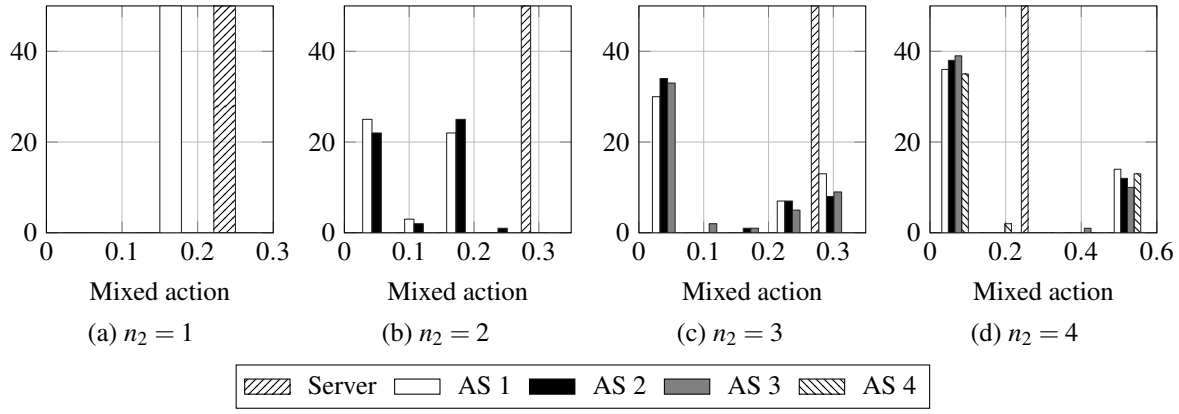


Fig. 4.3 Histogram of actions obtained using RM algorithm, for $I = 5$ sensors and variable number of ASs. Each histogram is computed using 5 bins. Observe that the action of the server does not vary significantly, whereas the actions of the ASs do. Also, observe how as n_2 increases, the ASs histogram presents two peaks: the biggest close to 0 and a smaller peak at another mixed action value. This hints that the game tends to the two player case when there are many ASs: all but one AS tend to behave as GSs.

4.5 Solving the backoff attack using Repeated Games

In the previous Section, we have posed and solved the CSMA/CA game using Static Game tools. We note that this game is a nonzero sum game, and hence, it may benefit from the Folk Theorem: by repeating the game, players might have better payoffs. Note that repeating the game is natural in the WSN situation described, as each sensor does not transmit only once. Hence, we now study the CSMA/CA game using the tools presented in Chapter 2.5.2

4.5.1 Analysis for the two-player case

In this Section, we solve analytically the CSMA/CA game treating it as an RG, for the two player case.

SPE solution to the CSMA/CA game

We solve the CSMA/CA game using the ideas from Section 2.5.2. We start demonstrating the validity of the UNR strategy with the server, using Proposition 4 and the expected payoff values from (4.14). UNR strategy is an SPE for the server if:

$$(1-\gamma)r_1(y_o,z_o)+\gamma V_1(y_o,z_o) \geq (1-\gamma)r_{1,max}(y,z_o)+\gamma V_{1,n}, \tag{4.23}$$

where $r_{1,max}(y,z_o)$ is the maximum payoff that the server can obtain from a unilateral deviation, $V_1(y_o,z_o)$ is the payoff that the server expects to obtain by playing y_o when the AS plays z_o and $V_{1,n}$ is the payoff that the server expects to obtain if it deviates, which is the stage NE payoff. Observe that $V_1(y_o,z_o)$ is the payoff if both players follow the UNR strategy without deviation, that is, $V_1(y_o,z_o) = r_1(y_o,z_o)$. Hence, (4.23) becomes:

$$r_1(y_o,z_o) \geq (1-\gamma)r_{1,max}(y,z_o)+\gamma V_{1,n}, \tag{4.24}$$

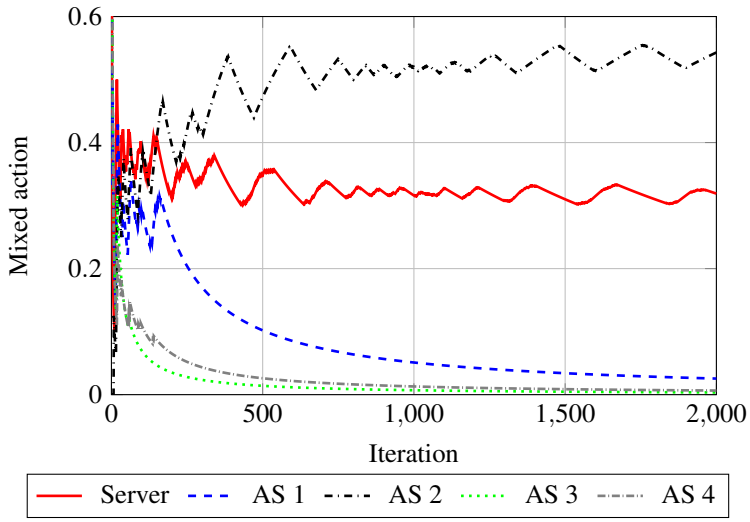


Fig. 4.4 Example the evolution of the mixed action for each player, using RM algorithm. In each simulation, all ASs tend to play *ns*, except for one. This one randomly arises at each simulation using RM algorithm. This means that the game tends to the two player situation.

which means that the discount factor must satisfy:

$$\gamma \geq \frac{r_{1,max}(y,z_o) - r_1(y_o,z_o)}{r_{1,max}(y,z_o) - V_{1,n}}, \quad r_{1,max}(y,z_o) > V_{1,n}. \tag{4.25}$$

Now, we turn to the AS. We know that the stage NE payoff for the AS is $V_{2,n} = 0$. Hence, UNR strategy is an SPE for the AS if:

$$r_2(y_o,z_o) \geq (1-\gamma) r_{2,max}(y_o,z), \tag{4.26}$$

which means that the discount factor must satisfy:

$$\gamma \geq 1 - \frac{r_2(y_o,z_o)}{r_{2,max}(y_o,z)}, \quad r_{2,max}(y_o,z) > 0. \tag{4.27}$$

Hence, from (4.25) and (4.27), UNR strategy is an SPE strategy for the CSMA/CA game if the following set of conditions are satisfied:

$$\begin{aligned} &\gamma \geq \max\left(\frac{r_{1,max}(y,z_o) - r_1(y_o,z_o)}{r_{1,max}(y,z_o) - V_{1,n}}, 1 - \frac{r_2(y_o,z_o)}{r_{2,max}(y_o,z)} \right). \\ &\gamma \in (0,1), \quad r_{1,max}(y,z_o) > V_{1,n}, \quad r_{2,max}(y_o,z) > 0 \end{aligned} \tag{4.28}$$

Observe that if players followed UNR without deviating, their payoff would be $(V_1(y_o,z_o), V_2(y_o,z_o)) = (r_1(y_o,z_o), r_2(y_o,z_o))$. Both players must choose the strategy values (y_o,z_o) so that the conditions from (4.28) are satisfied. There might happen that $(y_o,z_o) = (y_n,z_n)$, i.e., no UNR strategy gives higher payoff than stage NE, or that there are one or more valid $(y_o,z_o) \neq (y_n,z_n)$. In other words, this problem might have multiple solutions.

We interpret $r_{1,max}(y,z_o)$, the maximum payoff for the server if it deviates (equivalently, $r_{2,max}(y_o,z)$ for the AS) as the expected payoff of deviating by using the mixed action y in case of the server (and z in case of the AS). After we have fixed $r_1(y_o,z_o)$ and $r_2(y_o,z_o)$, we compute y_o and z_o using (4.14), and then, we use (4.14)

again in order to obtain $r_{1,max}(y,z_o)$ and $r_{2,max}(y_o,z)$ as the solutions to:

$$\begin{aligned} r_{1,max}(y,z_o) &= \max_y r_1(y,z) \quad s.t. \quad z = z_o \\ r_{2,max}(y_o,z) &= \max_z r_2(y,z) \quad s.t. \quad y = y_o \end{aligned}, \tag{4.29}$$

whose solution, using (4.14), is:

$$\begin{aligned} r_{1,max} &= \begin{cases} z_o(\alpha_f+\alpha_c)-\alpha_f & \text{if} \quad z_o > z_n \\ -z_o\alpha_m & \text{if} \quad z_o < z_n \end{cases} \\ r_{2,max} &= \begin{cases} y_o(\beta_s+\beta_c)-\beta_c & \text{if} \quad y_o > y_n \\ 0 & \text{if} \quad y_o < y_n \end{cases} \end{aligned}. \tag{4.30}$$

Correlated equilibrium solution to the CSMA/CA game

We compute the CE of the CSMA/CA game, using (2.78) and (2.79). We consider UNR strategy: both players commit to use a strategy that yields a payoff $V_o = (V_{1,o}, V_{2,o})$, and if one of the players deviates, the other switches to its stage NE strategy, which yields a payoff $V_n = (V_{1,n}, V_{2,n})$. The CE condition, thus, using (2.78) becomes:

$$\begin{aligned} \sum_{a_2=\{s,ns\}} \phi(a_2|d)V_1(d,a_2) &\geq \sum_{a_2=\{s,ns\}} \phi(a_2|d)V_1(nd,a_2) \\ \sum_{a_2=\{s,ns\}} \phi(a_2|nd)V_1(nd,a_2) &\geq \sum_{a_2=\{s,ns\}} \phi(a_2|nd)V_1(d,a_2) \\ \sum_{a_1=\{d,nd\}} \phi(a_1|s)V_2(s,a_1) &\geq \sum_{a_1=\{d,nd\}} \phi(a_1|s)V_2(ns,a_1) \\ \sum_{a_1=\{d,nd\}} \phi(a_1|ns)V_2(ns,a_1) &\geq \sum_{a_1=\{d,nd\}} \phi(a_1|ns)V_2(s,a_1) \end{aligned}. \tag{4.31}$$

Using (4.12) and (2.79), and considering that $V_i' = V_{i,o}$ if there is no deviation and $V_i' = V_{i,n}$ if there is a deviation, the expressions in (4.31) become:

$$\begin{aligned} ((1-\gamma)\alpha_c+\gamma V_{1,o})\phi(s|d)+(-(1-\gamma)\alpha_f+\gamma V_{1,o})\phi(ns|d) &\geq \\ (-(1-\gamma)\alpha_m+\gamma V_{1,n})\phi(s|d)+(0+\gamma V_{1,n})\phi(ns|d) & \\ (-(1-\gamma)\alpha_m+\gamma V_{1,o})\phi(s|nd)+(0+\gamma V_{1,o})\phi(ns|nd) &\geq \\ ((1-\gamma)\alpha_c+\gamma V_{1,n})\phi(s|nd)+(-(1-\gamma)\alpha_f+\gamma V_{1,n})\phi(ns|nd) & \\ (-(1-\gamma)\beta_c+\gamma V_{2,o})\phi(d|s)+((1-\gamma)\beta_s+\gamma V_{2,o})\phi(nd|s) &\geq \\ (0+\gamma V_{2,n})\phi(d|s)+(0+\gamma V_{2,n})\phi(nd|s) & \\ (0+\gamma V_{2,o})\phi(d|ns)+(0+\gamma V_{2,o})\phi(nd|ns) &\geq \\ (-(1-\gamma)\beta_c+\gamma V_{2,n})\phi(d|ns)+((1-\gamma)\beta_s+\gamma V_{2,n})\phi(nd|ns) & \end{aligned}. \tag{4.32}$$

And as in the Static Game case, we use (4.18) to simplify (4.32), with the same notation, that we repeat for convenience: $\phi_{11} = \phi(nd \cap s)$, $\phi_{12} = \phi(nd \cap ns)$, $\phi_{21} = \phi(d \cap s)$ and $\phi_{22} = \phi(d \cap ns)$. This is the joint distribution probability, considering that the first subscript refers to the pure action of the server, and the second, to the pure action of the AS. We consider that pure action 1 for the server is *nd*, and pure action 2, *d*; for the

AS, s is pure action 1 and ns pure action 2. Using all these concepts, (4.32) becomes:

$$\begin{array}{r}(1-\gamma)\big\{(\alpha_c+\alpha_m)\phi_{11}-\alpha_f\phi_{12}\big\}+\gamma(V_{1,n}-V_{1,o})(\phi_{11}+\phi_{12})\leq 0\\ (1-\gamma)\big\{(-\alpha_c-\alpha_m)\phi_{21}+\alpha_f\phi_{22}\big\}+\gamma(V_{1,n}-V_{1,o})(\phi_{21}+\phi_{22})\leq 0\\ (1-\gamma)\big\{-\beta_s\phi_{11}+\beta_c\phi_{21}\big\}+\gamma(V_{2,n}-V_{2,o})(\phi_{11}+\phi_{21})\leq 0\\ (1-\gamma)\big\{\beta_s\phi_{12}-\beta_c\phi_{22}\big\}+\gamma(V_{2,n}-V_{2,o})(\phi_{12}+\phi_{22})\leq 0\end{array}, \tag{4.33}$$

where we again assumed that $\phi(nd)>0$, $\phi(d)>0$, $\phi(s)>0$ and $\phi(ns)>0$. The constraints on the joint probability distribution ϕ, i.e., that all its components are non negative and add up to 1; and the payoff that each player would obtain by following UNR strategy, obtained doing the mathematical expectation on ϕ of the payoffs in (4.12), are:

$$\begin{array}{l}\phi_{11}+\phi_{12}+\phi_{21}+\phi_{22}=1\\ 0\leq\phi_{ij}\leq 1,\quad i\in\{1,2\},\quad j\in\{1,2\}\\ V_{1,o}=-\alpha_m\phi_{11}+\alpha_c\phi_{21}-\alpha_f\phi_{22}\\ V_{2,o}=\beta_s\phi_{11}-\beta_c\phi_{21}\end{array}. \tag{4.34}$$

The expressions in (4.33) and (4.34) define the region of CE and the payoffs that players would obtain in the RG case.

4.5.2 Solving for more than two players

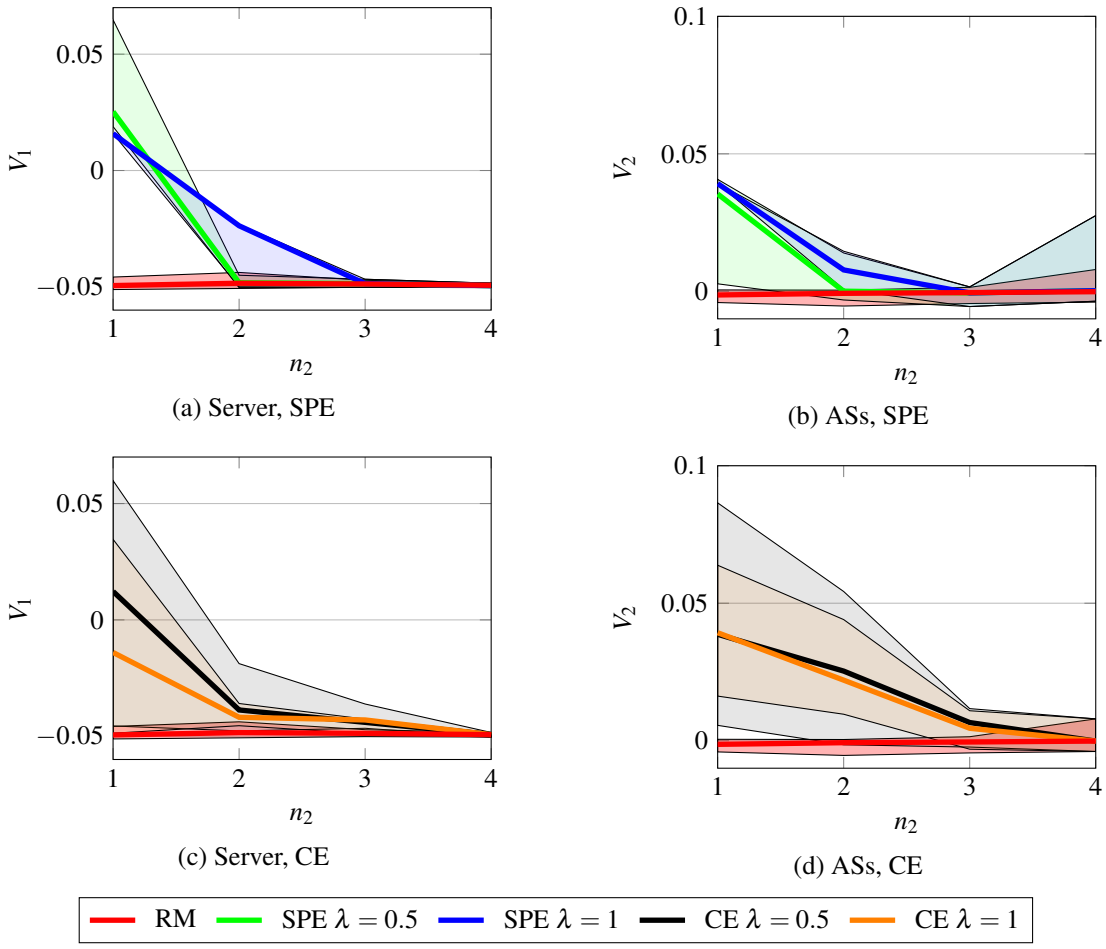
The analytical derivations from the previous Sections may become intractable when the game is composed by many players. For these cases, we propose using the two algorithms developed in Chapter 3, CA and LEWIS, to learn solutions of the CSMA/CA game. Note that the philosophies of these two algorithms are very different, as CA is based on negotiating an equilibrium, while LEWIS relies on online learning.

A very important consideration is that RM algorithm does not learn an RG equilibrium using the tools provided by the Folk theorems. RM can be used for learning equilibria in RGs, since static NE and CE are equilibria of the RGs. But stage equilibria payoffs need not be the best payoffs that players might achieve: the main reason to use the Folk theorems tools is that they allow providing all players with a payoff strictly higher than the ones they obtain by following a static strategy. Hence, note that CA does take into account the Folk theorem, and LEWIS tries also to improve a security payoff, whereas RM does not make use of the Folk Theorem and hence, it only obtains stage game equilibria.

4.5.3 Simulation 3: The repeated CSMA/CA game

In order to compare the stage and the RG payoffs of the CSMA/CA game, we run a set of simulations comparing the solutions that RM algorithm provides with the solutions given by CA (Section 3.4.1) and LEWIS (Section 3.3.1). We fix the discount factor value to $\gamma=0.99$ for all cases.

For CA, we allow $N_c=100$ communications per player. As sampling procedure, we use SOO optimization approach, as we know that it provides the best results. As SOO samples in a hypercube, this is appropriate for the SPE case: we have two actions per player, hence, the mixed actions vector for N_p players lies in the hypercube of dimension N_p, whose components lie in the range $[0,1]$, i.e., the mixed actions vector a is so that $a\in[0,1]^{N_p}$. However, the CE solution is a distribution ϕ that has, in our case, 2^{N_p} components. It must satisfy



(a) Server, SPE

(b) ASs, SPE

(c) Server, CE

(d) ASs, CE

Fig. 4.5 Payoff V obtained for the server and ASs, using CA. The error bars show the maximum and minimum values achieved. For ASs, we plot the mean values, computed among the n_2 ASs in the setup. We can observe that CA never performs worse than RM, and when there is a low number of ASs it provides a significant payoff gain to both server and ASs.

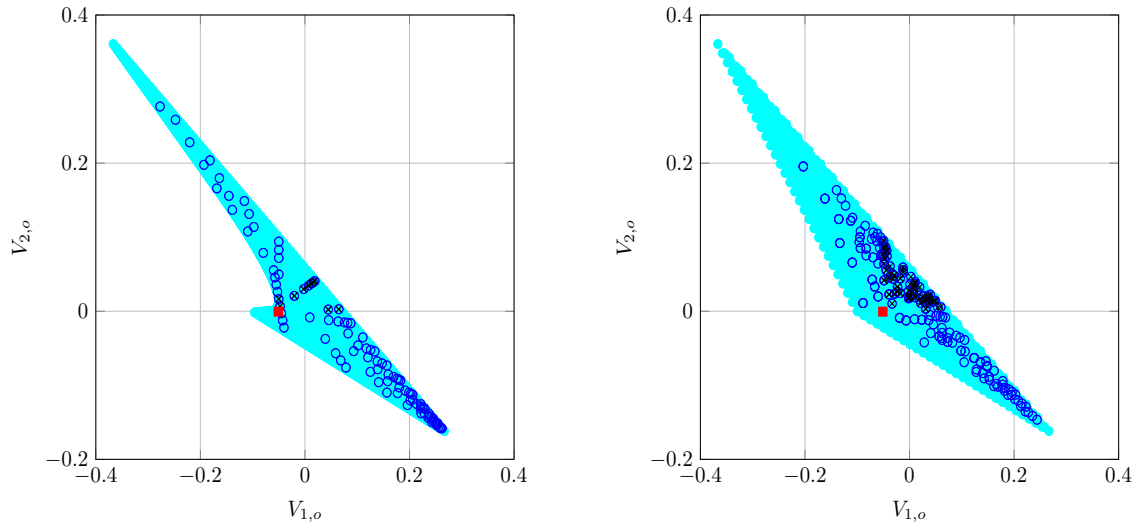
that $\phi_k \geq 0$ and $\sum_k \phi_k = 1$, and hence, it is a simplex, not a hypercube. This means that, as N_p grows, if we sample a hypercube, we lose many of points because they do not belong to the valid region of the distribution ϕ. In order to solve this problem, we use a mapping from a hypercube to the simplex region containing ϕ. For a vector x that belongs to the hypercube of dimension $N_p - 1$, we compute $s = \sum_k x_k$ and $m = \max(x_k)$ and obtain the point x' as follows:

$$x' = x\frac{m}{s}, \tag{4.35}$$

where x' satisfies that $x'_k \geq 0$ for its $N_p - 1$ components, and $\sum_k x'_k \leq 1$. Hence, we can define a candidate equilibrium distribution ϕ_c as:

$$\phi_c = \left(x'_1, x'_2, ..., x'_{N_p-1}, 1 - \sum_k x'_k\right), \tag{4.36}$$

where we recall that x' was obtained from the hypercube of dimension $N_p - 1$. By doing this we ensure that ϕ_c satisfies the conditions to be a valid distribution.



(a) SPE (b) CE

Fig. 4.6 Payoff region when $n_2 = 1$, using SPE and CE. The light region are all possible payoffs, the red square is the static NE that RM provides, the blue circles are the points that CA samples and the circles with a black cross are those that are valid equilibria for the RG, i.e., there is a greater payoff for both players than their stage NE payoff. Observe that the SPE region is contained in the CE region.

Sampling using SOO has a $\lambda \in [0,1]$ parameter, which models how much a player takes into account how good an equilibrium point is for the other players. We simulate using $\lambda = 1$, i.e., the player ignores the rest of the players, and $\lambda = 0.5$, i.e., the player takes into account the information of all players. Also, for the SPE, we must define a grid of actions to test for deviations; in our case, we provide a uniformly distributed grid in the range $[0,1]$ with 30 samples.

We test CA for both CE and SPE concepts, using $\lambda = \{0.5, 1\}$ and for $n = 5$ stations in the network. We consider that $n_2 = \{1,2,3,4\}$. For each of these cases, we first obtain a static equilibrium using RM algorithm with $T = 2000$ iterations, and the results of RM are given as input to CA algorithm. After CA algorithm has been run, we obtain a possibly higher payoff. We repeat 50 times the whole procedure for each n_2 value, and the results are in Figure 4.5. Observe that (1) as expected by design, CA never provides a lower payoff than RM, (2) the payoff increases are bigger and with higher variability when n_2 is lower, that is, when there are fewer ASs, (3) CE and SPE provide similar results, with an advantage for CE in the case of the ASs and (4) the payoff gains are smaller for the ASs than for the server.

We also include a representation of the payoff regions in Figure 4.6 for both SPE and CE, for the case in which $n_2 = 1$, using the expressions derived in Section 4.5.1. Observe that the region of valid payoffs (i.e., those which yield a greater payoff than the static NE) is not too large. This explains why, in Figure 4.5, the increments in payoffs that CA returned were small: they cannot be too large due to the characteristics of the payoff region.

We also simulate the CSMA/CA game when LEWIS is used. We test for $\varepsilon = \{0, 0.1\}$, where we remind that ε is the parameter that controls the risk in LEWIS, and simulate 50 different runs of LEWIS, with $N = 500$ time steps each, in the same setup used in CA. Each player uses LEWIS, and the results can be seen in Figure 4.7, and comparing to the static results from Table 4.5 and the CA results in Figure 4.5, we can observe that the ASs do not improve their payoff when compared to the static case. The server does present a payoff increment,

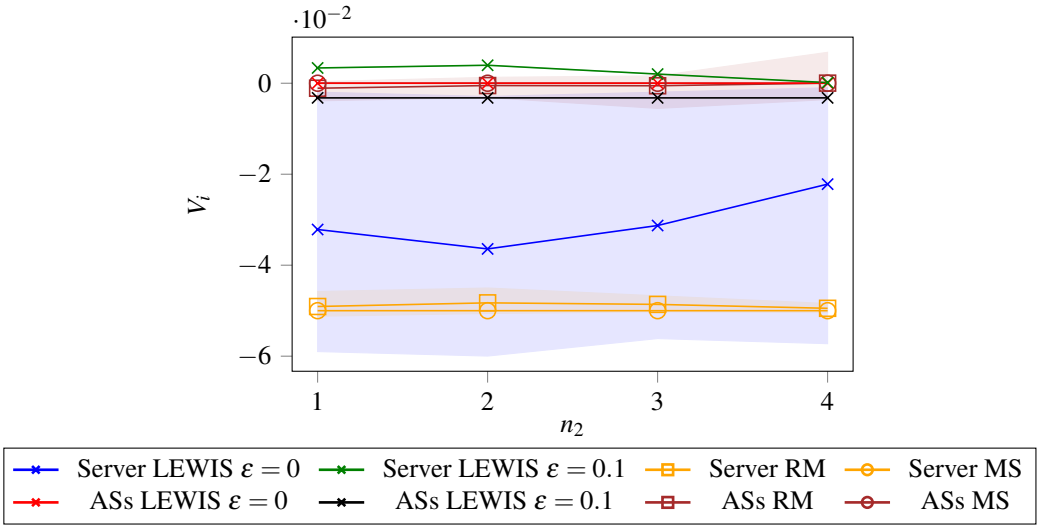


Fig. 4.7 Payoff V obtained for the server and the ASs, using LEWIS for $\varepsilon = \{0, 0.1\}$, compared to the security payoff and the RM payoff. The shadow regions represent the maximum and minimum values obtained: note that in some cases LEWIS acts deterministically. The security condition of LEWIS is satisfied in all cases: note that this condition depends on the minmax strategy payoff (MS) and the ε value. In case of the ASs, the security payoff, the RM payoff and the LEWIS payoff when $\varepsilon = 0$ are nearly the same: note that the ASs have some loss when $\varepsilon = 0.1$, although the security condition holds, as the loss is lower than ε. In case of the the Server, the security payoff and the RM payoff are very close again, but the server is able to improve its payoff by using LEWIS for all ε values tested.

although it is not as large as using CA. Hence, CA negotiating approach is able to outperform LEWIS online learning procedure. Note that this is to be expected, as CA has an initial stage of negotiation in which each agent tries to cooperate in order to improve its payoff. However, LEWIS tries to learn with security, and the prime interest of LEWIS is to balance not risking too much in terms of payoff with trying to improve its payoff when this improvement does not jeopardize its payoff. Hence, the conservative behavior of LEWIS is to be expected, which means that it provides worse payoffs when compared to CA. Also, we note that the resources required by CA are larger, as it requires first negotiating, whereas LEWIS is a fully online learning algorithm.

Discussion

The results of the previous simulations have an impact on practical implementations of the defense mechanism proposed. The first question is whether to implement a static or RG solution. We have shown, in Figure 4.5 and 4.7, that the repeated solution might provide higher payoffs to all players. This increment, as shown in Figure 4.6, could be very significant depending on the payoff region of the game. But this payoff gain comes at the cost of more computational resources in case of CA due to the communication phase and the sampling procedure. Note that LEWIS has a similar complexity to RM, as both are online learning algorithms. We also must take into account that CA requires a stage NE as input, so it can be thought of as an additional cost after having a stage NE. LEWIS needs as input a security payoff, which in our case was the minmax payoff, which can be efficiently computed using a linear program. In short, there is a trade-off between computational time and payoff gain. If we are more interested in having a low computational load, as may be the case in a sensor network with low computational resources or strict constraints in battery life, then RM or LEWIS are more sensible options.

If we decide to use a repeated solution based in CA algorithm, then two more questions arise. The first is related to the concrete parameters of the algorithm to use: λ, N_c and the sampling procedure. These parameters have an effect on the equilibrium that CA returns as shown in Chapter 3.4.3; and hence, we have to find a set of parameters that performs adequately in our concrete setup, as a function of the computational resources, the network topology and the payoff gain desired.

When using CA, we observe that CE is preferable to SPE for different reasons. First, Figure 4.5 shows that CE performs similarly in terms of payoff gain. Second, Figure 4.6 shows that SPE region is contained into the CE region, so any NE will have a corresponding CE, but the reverse needs not be true. Third, as we noted before, CE is more efficient to compute. However, CE is based on a correlating device, which obtains realizations of the equilibrium distribution ϕ and sends the action to play to each player. For instance, in the context of IEEE 802.11, this task could be performed by the HCF (Hybrid Coordination Function), a centralized network coordinator whose task in this case would be obtaining realizations of the distribution ϕ and sending each player its corresponding action. This means that the CE solution reminds of a centralized scheduler such that no sensor gains by deviating from its recommendations. This scheduler can be implemented distributedly as well, as the next Section shows.

Finally, we have derived equilibrium conditions for CA which are valid only in a perfect information environment. This means that players are able to detect deviations instantaneously. In the case of CE, this is straightforward: the correlating device, at each stage, sends each player the pure action that she should play: if any player deviates, the correlating device would know at the end of that stage. The case of SPE is much harder: players play mixed strategies y_o and z_o, which mean that the other players can detect a deviation instantaneously if they have access to the correlating device of the rest of the players. This might not be practical in terms of implementation, and it is another reason to see CE as superior to SPE in practical terms. However, as the next Section shows, there are several ways in which we could detect instantaneously a deviation from a mixed action without having access to the randomizing device of the players.

4.6 Detecting deviations with unobservable mixed actions

It is important to note that the RG implementation provided in this Chapter is, in principle, unable to deal with imperfect information situations in which the mixed actions are unobservable, that is, the other agents only observe the action realizations, which is frequent in real life situations. The equilibrium conditions depend on the ability to detect deviations instantaneously: even though this is not a problem in case of CE, it is a problem in case of NE if we do not have access to the randomizing device of the players. Now, we present two possible solutions to the problem of RGs with unobservable mixed actions. The first one consists in using deterministic sequences. The second one uses a pseudo-random number generator in order to create a distributed correlator that allows detecting any instantaneous deviation. Even though there are other possible solutions to this problem, as shown in Sections 3.7 and 3.8 in [146], we present these two because of their simplicity.

4.6.1 Deterministic sequences

A possible solution consists is using deterministic sequences of actions that behave as the mixed actions would. This solution was proposed in [75]. The idea behind this is that all players know what the other player is going to do in each time step, and hence, detecting a deviation is straightforward. One option is simply to generate a sufficiently large vector of actions, that is shared to all players. This allows all players knowing in advance which is going to be their payoffs. This solution can require a very large amount of memory to store the actions

vector, thus, a different approach would use periodical sequences in order to require less memory. For instance, in [75], the authors derive an algorithm that obtains a periodical vector of actions which allows achieving rational mixed actions.

Thus, this solution requires that the players obtain a deterministic sequence of actions that allows them to detect any deviation instantaneously. This is a very simple and straightforward solution, with two main drawbacks. The first one consists in deciding who is going to obtain the sequences and how: all players must agree on this procedure. The second problem is that this solution requires sharing and storing in memory a potentially very large action vector, since a sequence can be arbitrarily long, even if the sequence is periodical because the periods can be very long. Thus, this simple solution turns out to require a potentially large amount of memory and communications capacity.

4.6.2 PRNG based correlator

Another solution consists in using a correlator device. One approach could be using a public randomization device, or equivalently, that all players allowed access to their randomization devices in order that all players could observe a deviation. This was suggested in [74]. This solution, however, might be hard to implement, requiring either access to the randomizing device of each player or creating a trusted, centralized device that generates the mixed action for each player.

We overcome these problems by using a Pseudo-Random Number Generator (PRNG). A PRNG is an algorithm that allows obtaining a sequence of numbers that has similar properties to a sequence of random numbers. But the sequence can be completely determined by an initial value, called seed. If the seed is known, the whole sequence can be obtained. We focus on Linear Congruential Generators (LCG), due to their simplicity. An LCG generates the sequence of numbers x^n using the following recursive formula [108]:

$$x^{n+1} = (bx^n + c) \mod m, \tag{4.37}$$

where b, c and m are the parameters of the LCG, and $\mod$ denotes the module operation. The value x^0 is the seed, which can take any value. If we want $x \in [0,1)$, we must divide each value x^n by m: by doing so, the values x/m approximately follow a uniform distribution in the interval $[0,1)$. As it is shown in [108], the sequence generated using (4.37) is periodical, with at most period m; and the sequence has period m for all seed values if and only if:

1. m and c are relatively prime.
2. $b-1$ is divisible by all prime factors of m.
3. $b-1$ is divisible by 4 if m is divisible for 4.

The expression in (4.37) is easy and fast to compute, and this is one of the main advantages of LCG, together with their low memory consumption. However, LCG also presents some known problems [66], [130], that make them not suitable for applications where high randomness quality is required. However, that is not our case, so an LCG will suffice for our purposes.

We make use of the Inverse Transform method, which allows transforming a uniform random variable $U \sim Unif[0,1]$ in a random variable following another distribution. For instance, we can obtain a Bernoulli random variable X of parameter θ from U as follows: $x = 0$ if $U \leq 1-\theta$ and $x = 1$ if $U > 1-\theta$. This procedure can be generalized to multinomial random variables. Observe that the mixed actions of a game are the outcomes

of a binomial distribution when a player has two actions, and a multinomial in case that there she has three or more actions.

In terms of practical implementation, we assume that all players know the equilibrium mixed action a, i.e., the mixed actions of all players, and they also know the LCG parameters to use, i.e., m, b, c and x^0. The first thing each player does is using the Inverse Transform method to obtain the actions that each player should play depending on the value of the variable U. Then, each time that an action is played, each player generates a pseudorandom number x/m using the LCG generator. Observe that this number is the same for all players and also, observe that this number follows a uniform distribution. By using the Inverse Transform method, each player knows which action she has to play and also, by observing the past action realizations of the rest of the players, each player can detect any deviation instantaneously.

This procedure is easy to implement, computationally fast and does not require a high amount of memory. Also, note that it could be easily extended for Correlated equilibria, in which the actions follow a multinomial distribution: thus, it allows implementing a distributed correlating device. Note that all players need to use the same LCG parameters, which could be fixed beforehand. However, an important drawback comes from finite precision effects: if two players operate using different precisions, eventually, they will obtain different pseudorandom numbers x/m and this may lead to error. This problem could be alleviated by using, for instance, a periodical reset to the seed or sharing the x value periodically.

4.7 Conclusions

In this Chapter, we study a CSMA/CA based WSN under a backoff attack: some sensors deviate from the defined contention mechanism and this causes the network throughput not to be fairly distributed. This impact is studied using Bianchi's model and posed as a game of perfect and complete information. We first solve this game using static solution concepts, and then we use RG tools in order to take into account the fact that there is more than one transmission in the network. We first provide an analytical solution to the RG in the two player case, using both CE and SPE equilibrium concepts, and then we also use several algorithms that can be used to learn strategies in the case in which there are two or more players. By using simulations, we are able to check that using RG tools allows the players to have better payoffs, as the Folk Theorem advances.

Our approach shows that there is a trade-off between modeling complexity and computational complexity. By making use of payoff matrices, we alleviate this trade-off: the game theoretic solutions we provide are agnostic regarding where these payoffs come from. That is, we could use Bianchi's model as we do to relate rewards with the throughput, or we could relate rewards to other network parameters, as delay or any other measure of the quality of service, and yet our game model method would be valid: we should only replace the payoff matrix and solve the game, with these new matrices. Hence, we believe that we introduce a framework simple enough to accommodate different situations, but also complex enough as to model the conflict and the actions of the different stations involved by using game theory tools.

Our approach in this Chapter has relied on a perfect and complete information game, although we have also discussed two possible ways to extend our results to an imperfect information setting in which players do not observe the mixed actions, but the actions realizations instead. The framework we introduce in this Chapter can be further deepened in different ways. We could increase the complexity of the defense mechanism by introducing states: it may make use of different states in order to enhance its detection capabilities, for instance, keeping a track of the sensors which have not respected the contention procedure. This would mean introducing states in our dynamical system, and hence, making use of the MDP tools explained in Chapter 2. We do this in Chapters 5-7.

Also, note that we have assumed that the game was of complete information: the defense mechanism knew which agents were ASs. However, this needs not be the case in a security problem in a real environment: it is normal that the defense mechanism does not know which sensors are ASs and which are GSs. Hence, we would have an incomplete information situation, in which the defense mechanism tries to detect which sensors are ASs, and we deal with this setup in Chapters 5-7. Moreover, note that the defense mechanism needs not be able to instantaneously detect the deviations from the backoff procedure: this would be an imperfect information problem, and the consequences of this partial observability are explored in Chapters 6 and 7.

Hence, in this Chapter we study a simplified backoff attack under the assumptions of a perfect and complete information game. In Chapter 5, we assume that the defense mechanism has imperfect and incomplete information, while the attacker still has complete and perfect information. As we will see, this has a strong impact on the success possibilities of the attacker.

Chapter 5

Intelligent attacks against known defense mechanisms

5.1 Introduction

In Chapter 4, we considered complete information games in which each player had knowledge about how the actions of the rest of the players affected her. However, in real life environments this assumption needs not hold, as either the attacker or the defense mechanism may have more information than the other part, and hence, it may happen that one player has complete information and the other does not. In this Chapter we turn to asymmetric situations, as we now assume that the attacker has more capabilities than the defense mechanism, and hence, this will allow it to exploit the defense mechanism. Specifically, we assume that (1) the attacker has complete information of the defense mechanism, while the defense mechanism has incomplete information of the attacker, (2) the attacker has perfect information of the defense mechanism, while the defense mechanism has imperfect information of the attacker, and (3) the defense mechanism follows static procedures, i.e., that do not change with time, while the attacker may change and adapt with time. More formally, we use the MDP framework to model the defense mechanism and the attacker uses Control tools from Chapter 2 that allows it to exploit the defense mechanism.

Note that assumption (1) is not unrealistic, as an attacker may know the defense mechanism used in a certain setup. Assumption (3) is also realistic, as most defense mechanisms follow fixed rules that are not updated with time. However, assumption (2) may be unrealistic in some situations, as the attacker may have only a partial observation of the defense mechanism. In the concrete problems of this Chapter, assumption (2) holds; but we note here that Chapter 6 includes the case in which the attacker has incomplete and imperfect information about the defense mechanism, and hence, assumptions (1) and (2) do not hold. In other words, in this Chapter the attacker knows the defense mechanism, is able to observe its state and is dynamic in behavior, while the defense mechanism does not know whether a sensor is an AS or a GS, observes the action realizations of the sensor and is static in behavior, as shown in Table 5.1.

In order to address the situation in which the defense mechanism has an imperfect and incomplete information from the attacker, we focus on defense mechanisms based on Hypothesis Tests (HTs) in this Chapter. The defense mechanism has access to a stream of data following a certain distribution, and it must decide whether the behavior exhibited by the data stream corresponds to an expected behavior or to a behavior under attack. This is a problem that has been widely studied in the signal theory field [121]. One of many possible

Chapter	CSMA/CA	CSS	Player	Information	Observation (A/S)	Behavior
4	Yes	No	Attack Defense	Complete Complete	Mixed / - Mixed / -	Static Static
5	Yes	Yes	Attack Defense	Complete Incomplete	- / State Realization / -	Dynamic Static
6	Yes	Yes	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / -	Dynamic Static
7	Yes	No	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / Observation	Dynamic Dynamic

Table 5.1 Table comparing the different setups used in Chapters 4-7. CSMA/CA. i.e., the backoff attack, and CSS, i.e., the SSDF attack, denote whether each of these setups is used in the Chapter. Information denotes whether each player knows the target of the other player (Complete) or not (Incomplete). Observation refers to what each agent observes with respect to the actions / states of the other players: regarding actions, they observe the mixed actions or the actions realizations, and regarding states, they observe the state or an observation of the rest of players: this is related to having perfect or imperfect information. Behavior refers to whether the player adapts its behavior with time or not.

taxonomies to classify HTs is based on the sample size: whether the needed number of samples to make a decision is fixed in advance or not. The former case is the most usual: the well-known Neyman-Pearson HT belongs to this kind [163], [121], among many others, such as Rao or Wald tests [44]. The latter case is known as sequential hypothesis test, and traces back to the work of Wald on the Sequential Probability Ratio Test (SPRT) [227], [228]. In an SPRT, a sample from the signal of interest is collected at each time step n and used to update a statistic. The updated statistic may be used either to make a decision if there is enough information or to collect another sample. Note that SPRT is very attractive in many scenarios, as in WSN [41], [201], in which the ability to make a decision requiring fewer communications among sensors means a lower battery and bandwidth consumption. SPRT also allows making a decision as soon as possible and adapts easily to working with online data.

Thus, it is not a surprise that many WSN mechanisms make use of SPRT when an HT is needed. It has been used for Cooperative Spectrum Sensing (CSS), in which several sensors send to a central entity their local spectrum sensing report and SPRT mechanisms are used to implement the information fusion [201], [41], [238]. In [102], [53] and [223], SPRT is used for detecting sensors that have been compromised and replicated. In [131] and [77], SPRT is used to detect a selective forwarding attack, in which a compromised sensor drops packets. SPRT can also be used for Distributed Denial of Service (DDoS) attack detection [62] and spam detection [27]. Thus, SPRT finds many applications currently in WSN, specially when trying to detect a malicious behavior, and this justifies choosing it for our problem. We introduce SPRT in Section 5.2.

However, the formulation of SPRT introduced in Section 5.2 raises an important challenge, as it assumes that the underlying distribution of the signal under test does not change with time. Since SPRT is widely used to detect malicious behavior, this means that the malicious behavior is assumed to be static. As we will see, this is a dangerous assumption that can be used to exploit such systems, as Section 5.3 shows, where we derive an optimal attack against SPRT. To address this attack, we develop a novel defense mechanism in Section 5.4. A second challenge that arises is that SPRT does not make use of prior information about the sensors that could be available. In order to address this problem, we make use of a Bayesian framework in Section 5.5 that allows us deriving a very efficient sequential test which permits including prior information into the test, which however, is also shown to be vulnerable to the attack strategy derived before.

Later, Section 5.6 presents several simulations in which the ideas presented in the previous Sections about attack and defense mechanisms are applied to the backoff attack presented in Chapter 4, and also to a novel attack that we introduce in Section 5.2 against a CSS WSN. The use of the CSS problem allows us showing how the research from this work may generalize to several problems in WSN, as it does not only apply to a single problem. Finally, the conclusions of this Chapter, presented in Section 5.7, indicate that asymmetric attack situations as the ones described in this Chapter may suppose an important challenge to current defense mechanisms.

5.2 Sequential tests

In this Section, we describe an SPRT based detection mechanism which does not only apply to our problem, but it is also very similar to the ones present in current literature, such as [27], [102], [41], [223], [131], [62] or [53]. Even though concrete details differ between these works, the main lines of the detection mechanisms are similar to the mechanism that we introduce in this Section. Also, note that though we only provide results for this model, we strongly believe that they could be extended to different signal characterizations, such as the one in [201].

We focus on Bernoulli distributions, which have indeed many applications in the signal processing field. For instance, it appears in radar applications [203], pattern identification [69] and fusion in sensor networks [169], [170], [45], [189]. For the concrete case of Bernoulli distributions, the Neyman Pearson test [121] reduces to the Counting Rule for equal confidences. Other tests which are also used are the Rao and Wald tests [44]. Note that all these works use fixed sample tests, while we are focusing on sequential tests.

We start by introducing our Bernoulli HT, and after presenting the common counting rule, we explain the SPRT mechanism. Then, we introduce the SSDF attack, which is our second main study case in this work. Note that we introduce the SSDF attack here because several defense mechanisms proposed against SSDF are based on modified versions of SPRT. We finally point out a potential vulnerability of SPRT, which as we show in incoming Sections, can be used to exploit SPRT based tests.

5.2.1 The detection problem

We assume a discrete time signal x^n, where $n = 0,1,2,...$ is again the time index. The detection problem consists in collecting enough information in order to decide between two hypotheses H_0 and H_1:

$$\begin{cases} H_0 : x^n \sim Q_0, & n = 0,1,2,... \\ H_1 : x^n \sim Q_1, & n = 0,1,2,... \end{cases}, \tag{5.1}$$

where Q_0 and Q_1 are statistical distributions that characterize the behavior of the signal x^n under normal and malicious behavior respectively. In our case, we assume that each x^n follows a Bernoulli distribution of parameters θ_0 under H_0 and θ_1 under H_1, and hence:

$$\begin{cases} H_0 : x^n \sim B(\theta_0), & n = 0,1,2,... \\ H_1 : x^n \sim B(\theta_1), & n = 0,1,2,... \end{cases}. \tag{5.2}$$

The concrete malicious behavior varies depending on the setting, for instance:

- In the CSMA/CA problem we presented in Chapter 4, note that a modification of the frequency with which the AS does not follow the backoff procedure provides a better payoff to it, at the expense of the

network fair distribution of resources, as seen in Section 4.5.1. Note that $\theta_0 = z_o$, while θ_1 represents a deviation from the prescribed mixed action.

- In [53], the authors describe an attack in which sensors could have been compromised. A compromised sensor would send a piece of information x which can be accurate ($x = 0$) or inaccurate ($x = 1$). They try to detect as soon as possible inaccurate sensors, and note that ASs try to provide as much inaccurate information to the network as possible.
- In [223], the authors also try to make a difference between legitimate and illegitimate sensors in a network. In order to do so, they define a binary variable x that combines information of distance and signal power. Under this setting, an attacker tries not to be discovered while trying to compromise other sensors, which means that they will often cause $x = 1$.
- In [131], the authors try to detect a selective forwarding attack, in which $x = 0$ denotes a successful forwarding and $x = 1$ a packet drop. Note that an attacker tries to drop as many packets as possible.
- In [27], the authors propose a spammer detection algorithm, in which x denotes whether the emails sent by a user are spam ($x = 1$) or not ($x = 0$). A user which surpasses a certain threshold is blocked, hence, a spammer will try to send as much spam as possible while also trying not to be detected.

In all these cases, the condition $\theta_0 < \theta_1$ is satisfied, which means that the malicious agent tries to use $x = 1$ as often as possible. Thus, $x^n \in \{0,1\}$, and $P(x^n = 1) = \theta$ and $P(x^n = 0) = 1 - \theta$, and:

$$\begin{cases} H_0 : \theta = \theta_0 \\ H_1 : \theta = \theta_1 \end{cases}. \tag{5.3}$$

In an HT, there are two different errors: the type I error or false alarm probability is the probability that H_0 is rejected, provided that H_0 is actually true. The type II error is the probability of accepting H_0, provided that H_1 is actually true. We denote by α the type I error probability and β the type II error probability. The values of α and β determine the stopping rule that is used in the test. There is always a tradeoff between having a low false alarm probability and a high power test, which is defined as $1 - \beta$ and is the probability of correctly rejecting the null hypothesis.

5.2.2 The Counting Rule

The counting rule is a decision rule used in many WSN works due to being simple but nonetheless able to outperform more complex mechanisms [45], because it is the universally most powerful test [44] for the fixed length hypothesis test (5.3). It uses $s^n = \sum_{i=0}^{n} x^i$ as test statistic for a predetermined value of n, and decides H_1 if $s^n \geq \delta$, where $P(s^n \geq \delta | H_0) \leq \alpha$ allows fixing the decision threshold δ as a function of the significance level or type I error probability, α, fixed a priori. The power of the test depends on α and n: a larger n brings a higher power to the test, at the cost of a longer delay to the decision and a larger number of communications required. Hence, note that there is a tradeoff between precision and resources consumption.

5.2.3 Sequential Probability Ratio Test

The SPRT for our signal model presents the following test statistic for the sample n:

$$LR^n = \frac{\theta_1^{s^n}(1-\theta_1)^{n+1-s^n}}{\theta_0^{s^n}(1-\theta_0)^{n+1-s^n}}, \tag{5.4}$$

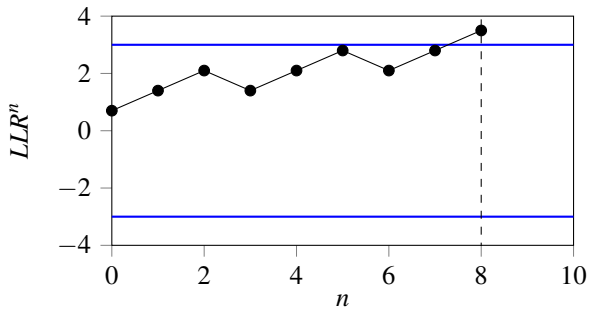


Fig. 5.1 Illustration of an SPRT. The upper blue line is h, the lower blue line is l. The black continuous line is the LLR^n, the test statistic of the SPRT. The dashed line indicates $N-1$, the time in which a decision is made by the SPRT. In this case, since $LLR^n \geq h$, H_0 the test decision is to reject H_0. Note that in samples $n \leq 7$, SPRT does not have enough information to make a decision and hence, another sample is collected.

where $s_n = \sum_{i=0}^{n} x^i$, $s^n \in [0, n+1]$ and $n \in \{0, 1, ...\}$. It is usual working with the log-likelihood ratio: the SPRT from (5.4) becomes:

$$LLR^n = s^n \log\left(\frac{\theta_1}{\theta_0}\right) + (n+1-s^n)\log\left(\frac{1-\theta_1}{1-\theta_0}\right), \tag{5.5}$$

and the decision rules of test (5.5) can be approximated [228] as:

$$\begin{cases} \text{Reject } H_0 & \text{if} \quad LLR^n \geq h \\ \text{Accept } H_0 & \text{if} \quad LLR^n \leq l \\ \text{Take another sample} & \text{if} \quad \text{otherwise} \end{cases}, \tag{5.6}$$

where h and l are defined as:

$$h = \log\left(\frac{1-\beta}{\alpha}\right), \quad l = \log\left(\frac{\beta}{1-\alpha}\right). \tag{5.7}$$

Note that (5.6) means that the SPRT procedure gathers new samples until a certain threshold in the statistic LLR^n is surpassed. An illustration is found in Figure 5.1: note that LLR^n produces a random walk and it is finished when it surpasses a certain threshold. Also, note that we can rewrite (5.5) as:

$$LLR^n = s^n \log\left(\frac{\theta_1(1-\theta_0))}{\theta_0(1-\theta_1)}\right) + (n+1)\log\left(\frac{1-\theta_1}{1-\theta_0}\right), \tag{5.8}$$

and then define:

$$A = \log\left(\frac{\theta_1(1-\theta_0))}{\theta_0(1-\theta_1)}\right), \quad B = \log\left(\frac{1-\theta_1}{1-\theta_0}\right), \tag{5.9}$$

and these ideas allow us to rewrite (5.8) as:

$$LLR^n = As^n + B(n+1). \tag{5.10}$$

Also, a sequential formulation for (5.5) can be obtained by noting that:

$$LLR^n = \begin{cases} LLR^{n-1} + A + B & \text{if} \quad x^n = 1 \\ LLR^{n-1} + B & \text{if} \quad x^n = 0 \end{cases}, \tag{5.11}$$

or even more compactly as:

$$LLR^n = LLR^{n-1} + B + Ax^n, \quad LLR^0 = B. \tag{5.12}$$

Note that (5.12) facilitates an easy and sequential implementation of the SPRT test that we have defined, which updates the LLR^n statistic using only simple operations. In case that $x^n = 1$ is received, the LLR^n adds up $A+B$, and in case that $x^n = 0$, only B is added.

Finally, even though SPRT was originally developed to test simple hypotheses, several approaches have been proposed to deal with composite hypothesis, as shown in [126]. Also, a unified framework for treating composite hypotheses is found in [125], where the author also proposes a nearly-optimal Bayes sequential test for the case of one-sided composite hypothesis. However, for simplicity, in this Chapter we limit to the case of simple hypothesis.

5.2.4 Fusion rules with reputations

As we mentioned in the introduction of this Chapter, SPRT and its derivates are widely used in defense mechanisms for WSN. We now introduce our second main security problem in WSN: the Spectrum Sensing Data Falsification (SSDF) attack. This attack takes place in CSS situations: there is a WSN in which each sensor measures the spectrum and sends its information to a central Fusion Center (FC), which makes a decision on the spectrum state. This problem is of special interest in Cognitive Radio (CR) setups, in which each sensor would be a secondary node that would cooperate with other secondary nodes to transmit when there is no primary transmitting. It might happen that one or more sensors in the CSS WSN are ASs, as each sensor senses the spectrum locally and sends this data report to a centralized FC which uses a certain fusion rule to make a decision on whether the communications channel is busy or idle. These schemes are vulnerable to SSDF attacks, in which false reports are given by attacking sensors. A lot of effort has been addressed to design defense mechanisms against such attacks, as [41], [260], [151], [167], [256], [172], [229], [250], [23], [238] or [239].

Let us assume a WSN with I sensors: n_1 is the number of GSs and n_2 the number of ASs, where $I = n_1 + n_2$. This WSN wants to estimate the channel state in the times $k = 0, 1, 2, ..., K$. The actual state of the channel can be $u^k = 1$ if the primary is transmitting or $u^k = 0$ if the primary is idle. At each k, the FC asks several sensors m for a report u_m^k, $m \in 1, 2, ..., I$, which may be $u_m^k = 0$ if sensor m senses the channel idle or $u_m^k = 1$ if sensor m detects a primary transmitting. When there are enough reports to make a decision, the FC uses a predefined fusion rule in order to obtain the channel state estimation u_d^k. This case is known as hard fusion: the information that the FC receives from the sensors are binary reports. The FC makes an error if $u^k \neq u_d^k$. Note that we use n for the stages of a sequential test, and k to index each time that a sequential test procedure is invoked.

There are several fusion rules that can be used by the FC to make a decision. A majority rule can be used, in which a maximum sample size N_{mr} is fixed: the FC collects N_{mr} reports and makes its decision by majority [41]. Note that the FC needs not having N_{mr} reports in order to make a decision: whenever it has $(N_{mr}+1)/2$ equal reports, the decision could be made. Another popular fusion rule is based on SPRT in order to have a defense mechanism against SSDF attacks, where examples of these algorithms are SPRT and WSPRT [41], EWSPRT [260], RWSPRT [238] or S0/1 [239]. In this Chapter, we will use the majority rule for its simplicity and EWSZOT (Enhanced Weighted Sequential Zero/One Test) because it has a higher performance than other schemes based on sequential tests [260], it is also simple to implement, it is fast in deciding [260], it is energy efficient [43] and it is mathematically tractable, which will be important in Chapter 6. In short, EWSZOT is an advanced centralized data fusion scheme against SSDF attacks, see [72], [257], [132] or [143]. We do not use RWSPRT and S1/0 because they require additional information from the transmissions to make a decision.

EWSZOT description

EWSZOT data fusion scheme is a hard fusion scheme based on reputations. At each stage k of EWSZOT, an HT is run: each stage k consists in the FC asking reports to certain sensors and taking a decision based on these reports, where these reports are indexed by n. At each stage k, the reputation of the sensor m, z_m^k, is updated based on whether the report of sensor m was consistent or not with the decision taken by the FC, u_d^k. Mathematically:

$$z_m^k = \begin{cases} z_m^{k-1} + 1 & \text{if} \quad u_m^k = u_d^k \\ z_m^{k-1} - 1 & \text{if} \quad u_m^k \neq u_d^k \end{cases} . \tag{5.13}$$

In the initial stage, all reputations are initialized to 0. The decision rule used by EWSZOT HT is:

$$\begin{cases} u_d^k = 1 & \text{if} \quad W^k \geq q \\ u_d^k = 0 & \text{if} \quad W^k \leq -q \\ u_d^k = 1 & \text{if} \quad -q < W^k < q \text{ and } n = N \\ \text{Ask for another report} & \text{if} \quad -q < W^k < q \text{ and } n < N \end{cases} , \tag{5.14}$$

where q and N are predefined thresholds. Observe that the first three conditions from (5.14) are the final conditions of the test: they finish the HT and lead to a decision. Also, W^k is the HT statistic, following:

$$W^k = \sum_{m=1} (-1)^{u_m^k + 1} w_m^k, \tag{5.15}$$

where w_m^k are weights related to the reputation of each sensor m that will be defined later. Note that EWSZOT HT is similar to a sequential test. The decision rule at stage k consists in asking sensor m to give a report: if its report is $u_m^k = 0$, then W_k is decreased w_m^k units, and if its report is $u_m^k = 1$, then W_k is increased w_m^k units. Note that the aggregated stream of data from the m sensors is the equivalent stream of data to x^n in SPRT. This process is repeated until:

1. W^k surpasses a threshold q. In this case, the decision is immediately taken using the first two lines of (5.14).
2. N sensors have been called and W^k has not surpassed the threshold q. This means that the test result is uncertain and we follow a conservative decision rule: to decide that the channel is occupied, which benefits Primary Users (PUs) in a CR environment, which is the original environment in which EWSZOT was proposed. This test truncation is added in [260] in order to avoid lockouts, and is frequent in sequential tests implementations.

Finally, the reputation of each sensor has an impact on the HT through the weights w_m^k, defined as:

$$w_m^k = \begin{cases} 0 & \text{if} \quad z_m^k < -g \\ \frac{z_m^k + g}{avg(z_m^k) + g} & \text{if} \quad z_m^k \geq -g \end{cases} , \tag{5.16}$$

where $avg(z_m^k)$ is the average reputation of all sensors and g is a small positive value. The purpose of the weights scheme (5.16) is that sensors with better reputation have a higher influence on the HT. The use of g allows GSs to have a slightly negative reputation, caused by their sensing error. Note that reputations also determine the order in which sensors are asked to give their reports. EWSZOT calls up to N sensors in descending order of reputations. Thus, we ensure using the sensors with best reputations to take the decision. The whole procedure is summarized in Algorithm 12, where an implementation of EWSZOT is presented.

Algorithm 12 EWSZOT algorithm implementation.

Input: N, q, g
Initialize $r_m^{-1} = 0, \forall m$
for Stages $k = 0, 1, 2, ...$ **do**
  Obtain weights using (5.16)
  Select the N sensors with highest reputations
  Set $W^k = 0$
  for Sensors selected **do**
    Ask report from sensor
    Update W^k using (5.15)
    if $W^k \geq q$ or $W^k \leq -q$ or N sensors have been called **then**
      Take decision u_d^k using (5.14)
      Exit loop
  Update reputations using (5.13)

5.2.5 Overview of attacks against SPRT

Although SPRT is widely used in defense mechanisms, it has a very dangerous underlying assumption because it considers that the statistical behavior of the signal x^n does not change with n. If the attacker is able to change x^n dynamically, it would be possible to exploit an SPRT based defense mechanism. However, in many works this limitation of SPRT is not taken into account. For instance, in [102], [41], [223], [131] or [62], SPRT is used in environments in which the attackers may use a dynamic attack strategy and hence, compromise the defense mechanism. To the best of our knowledge, the only study on such dynamic attacks is a work of ours [176] which we present in Chapter 6, where we exploit a CSS WSN which makes use of EWSZOT as defense mechanism.

In current literature, there are several tools designed for dealing with changes in the statistical behavior of a signal, such as quickest detection tools [187] or, for discrete time signals, repeated hypothesis tests [20]. In this Chapter, we follow an innovative framework where we make use of a well-known tool in the field of novelty detection as the One Class Supporting Vector Machine (OCSVM) [247]. This method allows detecting signals whose features differ from the ones with which the OCSVM was trained [195]. In a misbehavior setting as ours, standard SVM needs to have access both to examples of normal and malicious behavior, as in [107], and hence, it becomes specialized in detecting a single attack type. However, the main advantage of OCSVM is that they need only have access to normal behavior examples in order to be trained, which in our security setting means that they could potentially detect any type of misbehavior.

5.3 Optimal attacks against SPRT

In this Section, we pose and solve a control problem in order to obtain the optimal policy that an attacker should follow to exploit the Bernoulli SPRT already presented. We have chosen the Bernoulli distribution because it appears in the CSMA/CA problem described in Chapter 4 and also in many WSN defense mechanisms, such as [27], [102], [41], [223], [131] or [62], as mentioned before.

5.3.1 Attacker model

In this Section, for simplicity, we assume that there is a single AS in our WSN that can modify the signal x^n, either directly or indirectly. Note that, in our CSMA/CA setup, the signal x^n corresponds to the AS actions,

hence, we treat x as the stream of actions of the AS. We have assumed that a malicious behavior means that the AS uses $x^n = 1$ as often as possible. Thus, we assume that the agent receives an instantaneous reward of $+1$ each time that $x^n = 1$, where we use a simple reward of $+1$, but our results hold for any positive reward. Note that in the CSMA/CA problem, the AS receives a positive reward when it ignores the backoff procedures, and hence, it tries to use $x^n = 1$ as often as possible without being detected. Note that this reward scheme causes that the agent tries to increase the mean value of the signal x and thus, $\theta_1 > \theta_0$, as in the tests already presented.

The agent tries to maximize its total discounted reward R, defined as:

$$R(x^n) = \sum_{n=0}^{\infty} \gamma^n x^n, \tag{5.17}$$

where $\gamma \in (0,1)$ is again the discount factor that gives more weight to the rewards obtained in closer time steps than in the future. As we discussed in Chapter 2, the use of γ fits volatile environments such as WSN, as an AS does not know how long it will be able to attack and thus, it cannot be infinitely patient. Also, γ allows that the total reward remains finite, as the minimum value for R using (5.17) is $R = 0$ for $x^n = 0, \forall n$, and the maximum value is $R = (1-\gamma)^{-1}$, for $x^n = 1, \forall n$, where we used (3.6).

5.3.2 Optimal camouflage algorithms as a control problem

Now, we proceed to show the optimal control that the AS should use in order to maximize its reward when facing an SPRT detection mechanism. We denote by N the number of timesteps required by the SPRT to make a decision, and as mentioned before, in practical implementations, N is usually bounded to avoid system lockouts, although this truncation is suboptimal [225]. That is, in a truncated SPRT, there is a maximum number of samples that the test will gather before making a decision. A predefined decision is fixed beforehand in case that sample $N-1$ is reached without having made a decision and $l < LLR^{N-1} < h$. We consider that if the truncated SPRT test reaches sample $N-1$ without making a decision, H_0 is rejected. The problem that the agent must solve is the following one, where we use (3.6) and consider $N-1$ as the time in which the SPRT makes a decision:

$$\begin{aligned} \max_{x^n} \quad & \sum_{n=0}^{\infty} \gamma^n x^n = \sum_{n=0}^{N-1} \gamma^n x^n + \frac{\gamma^N}{1-\gamma} \\ \text{s.t.} \quad & x^n \in \{0,1\}, \quad s^n = \sum_{i=0}^{n} x^i \\ & LLR^n < h, \quad \forall\, n \leq N-1 \\ & LLR^{N-1} \leq l < h \end{aligned} \tag{5.18}$$

Note that in (5.18):

- The function that the agent needs to maximize is split in two terms. The first term refers to the timesteps in which the SPRT detection mechanism is active, in which the agent needs to find an optimal control law for x^n such that it is not discovered. The second term includes the timesteps after a decision is made: at these timesteps the SPRT mechanism is not active, and hence, the agent can always use $x^n = 1$.
- The constraint $LLR^n < h$ allows the agent not to be discovered by the SPRT mechanism. It forces SPRT to never reject H_0.

- The constraint LLR^{N-1} simply indicates that, at timestep $N-1$, in which a decision is made, H_0 is accepted, which implies that the AS has not been discovered yet.
- An assumption that we make is that $l < LLR^0 < h$, that is, that the *LLR* initial value does not allow SPRT to make a decision. This condition is usually satisfied by normal SPRT parameters and in our case, using (5.9) and (5.10) turns out to be:

$$\frac{\beta}{1-\alpha} < \frac{1-\theta_1}{1-\theta_0} < \frac{1-\beta}{\alpha}. \tag{5.19}$$

  Note that this means also that $l < h$ in the last constraint.

It is important noting that the problem formulated in (5.18) can be modeled using the MDP framework shown in Chapter 2, where we would have that the states are $s^n = LLR^n$, the actions would be $a^n = x^n$, the reward function is $r(s^n, a^n) = a^n$ and the transition function is deterministic and can be obtained using (5.11). Thus, we could also apply Dynamic Programming methods to obtain the optimal policy. In other words, our situation can be modeled using MDP tools: the target of the agent is finding a policy such that it can attack and camouflage while the SPRT mechanism is running. When the SPRT mechanism is not running, i.e., $n \geq N$, the agent can attack without needing to camouflage: under the MDP notation, when the final state LLR^{N-1} is reached, the agent always uses the attack action. Note that in truncated SPRT, N is fixed, while in SPRT, N is not fixed and hence, the actions of the agent will determine the final time N, which means that N is another value to optimize. Finally, we observe that the asymmetry in this problem comes from the fact that the agent can change its behavior dynamically, as a function of the state, as it is solving a control problem; whereas the defense mechanism is fixed and follows a static behavior. An important result is that SPRT can be successfully exploited by a dynamic attacker, as the next Theorem shows:

Theorem 4. *Consider the discrete time control problem described in* (5.18)*, in which the controller chooses x^n and may choose $N-1$. In this problem, the optimal control for $n \in [0, N-1]$ depends on whether $N-1$ is fixed or not:*

- *If $N-1$ is fixed:*

$$\begin{cases} x^n = 1 & \text{if} \quad LLR^{n-1} + A + B < h \text{ and } LLR^{n-1} + A + B(N-n) \leq l \\ x^n = 0 & \text{if} \quad \text{otherwise} \end{cases}.$$

- *If $N-1$ is not fixed:*

$$\begin{cases} x^n = 1 & \text{if} \quad LLR^{n-1} + A + B < h \\ x^n = 0 & \text{if} \quad \text{otherwise} \end{cases}.$$

Let us visualize the result of the control law obtained in Theorem 4. Note that the basic idea is that the agent is able to get close to the SPRT threshold without surpassing it. We provide an illustration in Figure 5.2, in which we can observe how the control law proposed is able to effectively attack the SPRT mechanism while not being detected. The next two Sections provide the justification to the control law in Theorem 4.

5.3.3 Optimal control to attack a truncated SPRT

Let us start by assuming that there is a truncated SPRT, which means that $N-1$ is fixed, as this is the most common option in practice. A first option to derive the optimal control would be using DP tools, namely, Lemma 1. However, it is possible to obtain a simple formulation for the optimal policy by following the next

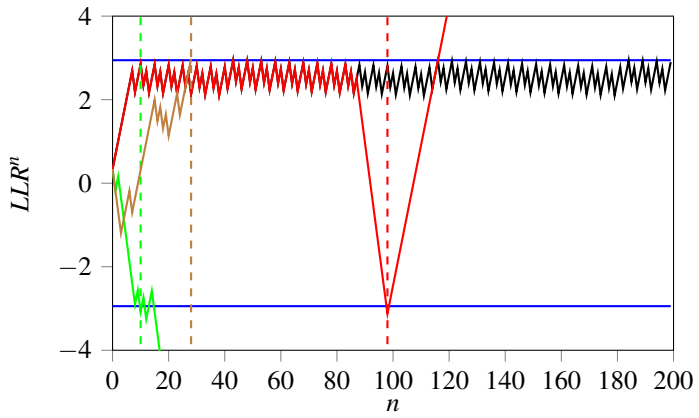


Fig. 5.2 Example of control under several situations. For all cases, $\theta_0 = 0.5$ and $\theta_1 = 0.7$. The blue lines are the LLR^n thresholds from (5.6), for $\alpha = \beta = 0.05$. Green line is the case in which there is no attack, i.e., $x^n \sim Bernoulli(\theta_0)$. Brown line is the case in which there is a naive attack, i.e., $x^n \sim Bernoulli(\theta_1)$. Red line is the case in which the attacker follows the control law from Theorem 4 when the SPRT test finishes after 100 samples. Black line is the case in which the attacker follows the control law from Theorem 4 when the SPRT does not have a predefined finishing time. The dashed vertical lines indicate when each test ends. While SPRT is able to detect the naive attack, is unable to detect the control law we describe in Theorem 4, independently on whether the SPRT test is truncated or not.

reasoning. First, note that the agent prefers using $x^n = 1$ as often as possible because that way, its reward is maximized. Also, note that since rewards are discounted, if the agent has to choose between using $x = 1$ at timestep n or at timestep $m > n$, the agent always prefer the former because it provides a larger reward due to the discount factor: $\gamma^n \cdot 1 > \gamma^m \cdot 1$ for $\gamma \in (0,1)$. Intuitively, this means that the agent will try to use $x = 1$ whenever possible.

However, the agent cannot always use $x^n = 1$ for all n. Using (5.12), the agent can predict its LLR^n value depending on its action. Since we considered that $\theta_1 > \theta_0$, then we obtain from (5.9) that $B < 0$ and $A + B \geq 0$. This means that:

- If the agent uses $x^n = 0$, $LLR^n = LLR^{n-1} + B < LLR^{n-1}$. In other words, the LLR^n value decreases by using $x^n = 0$.
- If the agent uses $x^n = 1$, $LLR^n = LLR^{n-1} + B + A \geq LLR^{n-1}$. In other words, the LLR^n value is non-decreasing by using $x^n = 1$.

There are two constrains in problem (5.18) that may prevent the agent from using $x^n = 1$. The first is that $LLR^n < h$. As we just noted, by using $x^n = 1$ the agent may increase its LLR^n and hence, it may, eventually, violate that constraint. In order to avoid that, the agent can play $x^n = 1$ if $LLR^{n-1} + A + B < h$. This is illustrated in Figure 5.3.

Another constraint in problem (5.18) that may prevent the agent from using $x^n = 1$ is that $LLR^{N-1} \leq l$. We assume that $l < h$, which is satisfied if $\alpha + \beta < 1$, which is our case. Note that the agent can only decrease LLR^n by using $x^n = 0$, thus, in order to satisfy $LLR^{N-1} \leq l$, the agent will have to play $x = 0$ sometimes. As we noted before, the agent prefers using $x = 1$ as often as possible, and hence, it will delay using $x = 0$ to satisfy $LLR^{N-1} < l$ as many timesteps as possible. Namely, if the agent is at timestep n, it could play $x^n = 1$ and then, use $x = 0$ at time steps $[n+1, N-1]$ and satisfy $LLR^{N-1} \leq l$ if $LLR^{n-1} + A + B(N-n) \leq l$. This is illustrated

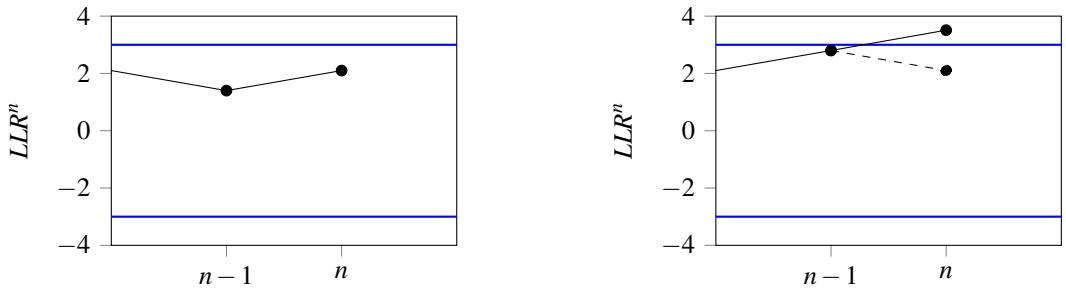


Fig. 5.3 Illustration of the constraint that $LLR^n < h$ in problem (5.18), where h is the upper blue line and the black lines represent LLR^n. In both plots, we show what would happen if the agent used $x^n = 1$. In the left plot, $LLR^n = LLR^{n-1} + A + B < h$ and hence, the agent could play $x^n = 1$. However, in the right plot, $LLR^n = LLR^{n-1} + A + B > h$ (solid line) and if the agent played $x^n = 1$, H_0 would be rejected and the agent would be discovered. Instead, the agent should use $x^n = 0$, which would decrease the LLR^n value (dashed line).

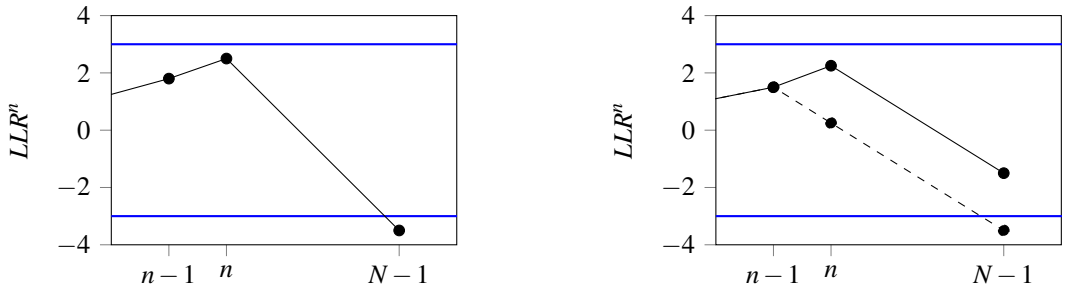


Fig. 5.4 Illustration of the constraint that $LLR^{N-1} \leq l$ in problem (5.18), where l is the lower blue line and the black lines represent LLR^n. In both plots, the solid black lines indicate the evolution of the LLR^n if the agent used $x^n = 1$ and then $x = 0$ for $n \in [n+1, N-1]$. In the left plot case, the agent satisfies that $LLR^{N-1} \leq l$, thus, it can use $x^n = 1$. However, in the right plot, the agent does not satisfy $LLR^{N-1} \leq l$ if $x^n = 1$, and hence, the agent would have to use $x^n = 0$ to satisfy the constraint (dashed line).

in Figure 5.4, and intuitively means that the attacker tries to delay using $x = 0$ as many timesteps as possible in order to satisfy the constraint on l.

Thus, the optimal control for the agent is:

$$\begin{cases} x^n = 1 & \text{if} \quad LLR^{n-1} + A + B < h \text{ and } LLR^{n-1} + A + B(N-n) \leq l \\ x^n = 0 & \text{if} \quad \text{otherwise} \end{cases} . \tag{5.20}$$

5.3.4 Optimal control to attack a non-truncated SPRT

In case that the SPRT is not truncated, note that the actions of the agent determine $N-1$, the time in which the SPRT makes a decision. In the previous Section, we showed that for fixed $N-1$, the agent would need to use $x = 0$ at several timesteps. As we noted, the agent would rather use $x = 1$, and this would imply that $N-1 \to \infty$. In other words, the agent would cause the SPRT to never make a decision, and its optimal control would be, see (5.20):

$$\begin{cases} x^n = 1 & \text{if} \quad LLR^{n-1} + A + B < h \\ x^n = 0 & \text{if} \quad \text{otherwise} \end{cases} . \tag{5.21}$$

5.4 Improved SPRT defense mechanism against intelligent attacks

Now, we present OCSVM-SPRT: a modification on the SPRT mechanism that makes use of an OCSVM that is able to deal with the novel optimal attacker we propose in the previous Section. Since an OCSVM is used, OCSVM-SPRT may potentially deal with any attacker whose spectral features do not match the ones of GSs.

Parallel SPRT

As we have already observed, SPRT is not a good option when the attacker may change its behavior, as it is vulnerable to attacks. A solution proposed to face this problem is given in [20], where several simultaneous SPRT tests are run in parallel, in order to detect changes in the signal x. This would imply that, for each new sample x^n that arrives to the defense mechanism, a new SPRT test is initiated, and up to n SPRT tests are updated. Note that this is a computationally demanding mechanism, since many SPRT tests must be run in parallel.

Also, note that an approach like this would not detect an AS following the control law from Theorem 4. As we showed in Figure 5.2, the attacker is as close as possible to the detection threshold without surpassing it. A second SPRT test would simply be a downshift in the LLR^n curve, pushing it down and hence, making impossible that the second SPRT test detects the attacker. This reasoning extends to subsequent SPRT tests, which are unable to detect the attacker. Note, however, that if the agent is unaware that there are several SPRT tests running, it could be detected after the last SPRT sample. But that can be easily overcome: an AS could exploit such mechanism simply initiating a control law for each sample n and choosing the most restrictive one. Hence, a parallel SPRT is not only computationally expensive, but also is unable to detect adequately the attacker we have developed. Note that we assume that the AS knows the defense mechanism parameters: we delay the case in which this does not hold to Chapter 6.

One-class SVM

We propose using a well-known tool in the field of sequence classification in order to modify the SPRT defense mechanism: a One-Class Supporting Vector Machine (OCSVM) [247]. As described in [195], a OCSVM is an algorithm that maps an input vector z according to whether z belongs to a set Z or not as follows:

$$f(z) = \begin{cases} +1 & \text{if} \quad z \in Z \\ -1 & \text{if} \quad z \notin Z \end{cases} . \tag{5.22}$$

The algorithm takes a set of $z \in Z$ points, and then obtains f by solving the following optimization problem:

$$\begin{aligned} \min_{w,\xi,\rho} \quad & \frac{1}{2}\|w\|^2 + \frac{1}{\nu l}\sum_i \xi_i - \rho \\ s.t. & (w \cdot \Phi(z_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0 \end{aligned} , \tag{5.23}$$

where $\Phi(z)$ is the feature map obtained by using a certain kernel, i indexes the training inputs z_i, l is the total number of training inputs, ξ are the slack variables and $\nu \in (0,1)$ is a parameter that corresponds to the fraction of outliers in the input data set Z [231]. Thus, note that an OCSVM only needs a training data set of valid data, in order to provide a decision function that later can be used for anomaly detection.

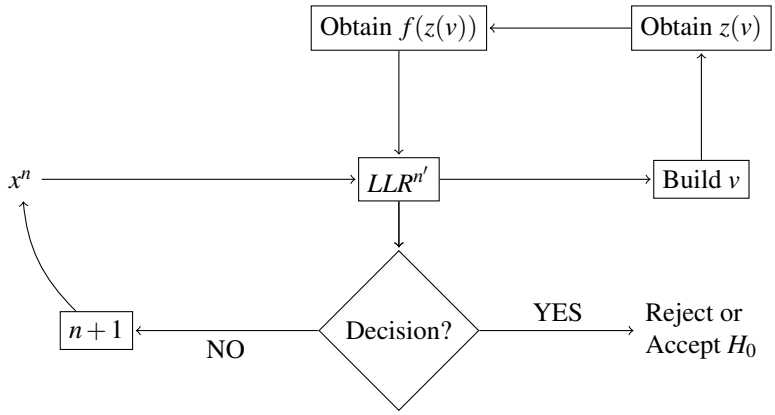


Fig. 5.5 Flow diagram for the proposed OCSVM-SPRT defense mechanism, where the $LLR^{n'}$ block implements (5.25).

SPRT - OCSVM defense mechanism

We make use of the capabilities of the OCSVM to propose a modified SPRT defense mechanism. First, we need to define which are the features z that we want to use. Note that these features need to characterize a statistical signal, and a possible way of characterizing such signals is by using the power spectrum [63]. We use as feature the power spectrum of the LLR^n signal, which is a random walk. We denote by v a subsequence of LLR^n, and in order to avoid errors caused by the mean value of the signal we subtract the mean of v: note that the LLR^n signal does not have a constant mean. Mathematically, we propose using the following feature vector z:

$$z(v) = \left| FT\left(AC\left(v - \frac{\sum_{m=1}^{M} v_m}{M} \right) \right) \right|, \tag{5.24}$$

where FT denotes the Fourier Transform, AC is the estimator of the aucotorrelation function, and v is a vector formed by the M most recent values of LLR^n. We use the full autocorrelation, hence, z has a length $2M-1$. Note that z is the estimated Power Spectrum of v with the mean subtracted. As we said, this choice of z is sensible given the fact that LLR^n is a random signal, and as we show in Section 5.6.3, the results provided by this characterization are quite good against the attack described in the previous Section. We note that the OCSVM training can be done offline by generating sequences of $x^n \sim Bernoulli(\theta_0)$, then obtaining the LLR^n vector v by using (5.5) and then obtaining z by using (5.24). That way, we train the OCSVM to detect any sequence not generated by following a Bernoulli of parameter θ_0. This means that the OCSVM will be able to detect, not only the attack we propose, but potentially any attack with a different spectral characterization from the signal that the OCSVM has been trained with.

Now, we need to decide how to include the additional information that the OCSVM provides. We propose using a modified SPRT with the following statistic $LLR^{n'}$:

$$LLR^{n'} = LLR^{n-1'} + B + Ax^n + \rho(A+B)\left|\min\left(f\left(z\left(v\right)\right),0\right)\right|, \quad LLR^{0'} = B \tag{5.25}$$

where $LLR^{n-1'}$ is the previous value of the test statistic, $B+Ax^n$ is the standard LLR^n update for the SPRT as shown in (5.12), ρ is a small positive parameter that control how much we make use of the information given by the OCSVM, and f is the OCSVM as defined in (5.22) using (5.24). Note that we include an additional term which depends on $A+B$, the increase on the LLR^n value when $x^n = 1$ (5.12). We do so because the value

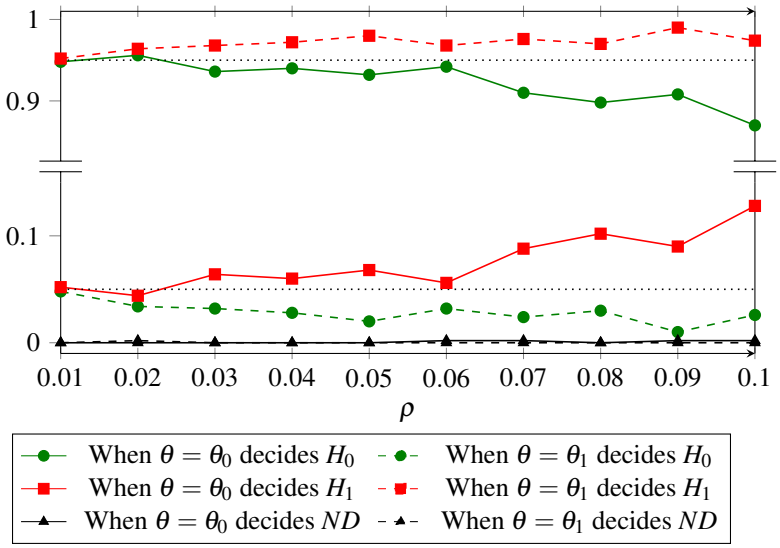


Fig. 5.6 Example on the influence of ρ on the modified SPRT-OCSVM scheme proposed, for $\theta_0 = 0.5$ and $\theta_1 = 0.7$. The details on the OCSVM are in Section 3.3.3. We consider $\alpha = \beta = 0.05$, without truncation and finishing the test after 200 iterations. The dotted black lines represent the type I and II error of the SPRT without modification: note that our modified test performs gives an increasing performance under H_1 as ρ grows, while its performance under H_0 decreases as ρ grows. This is to be expected: the OCSVM modification helps to detect a deviation from H_0, however, under H_0 the OCSVM modification introduces an additional error since it increases the LLR^n value.

of $A+B$ depends on θ_0 and θ_1 (5.9), and thus, we enforce that the last term in (5.25) depends on the concrete test parameters by being relative to the increase when $x^n = 1$. A flowchart illustrating the whole process can be seen in Figure 5.5.

Intuitively, the modified SPRT test in (5.25) works as follows. We first obtain the standard LLR^n value, and then, we use a OCSVM to obtain a second opinion on whether the LLR^n subsequence v has been generated by a GS or an AS. If the OCSVM detects a sequence that differs from these it has been trained to detect, it returns $f = -1$. In that case, the LLR^n is increased by a $\rho(A+B)$ value, which is proportional to the increase in the LLR^n signal when $x^n = 1$. If the OCSVM detects a normal sequence, then no modification is done. Note that the effect of the OCSVM is cumulative, and hence, our modified SPRT test detects faster an attacker if the OCSVM continuously confirms the detection. On the other hand, note that if the OCSVM states erroneously that a sequence comes from an attacker, the probability of type I error is increased compared to the standard SPRT test. This effect can be observed in Figure 5.6: what we have done is obtaining a more precise test under H_1, which however, gives a higher error under H_0.

5.5 Improving defense mechanisms using prior information

A challenge which arises with the SPRT approach already described is that the detection mechanism cannot incorporate prior information about H_0 and H_1 to the HT, which may be useful given the fact that the defense mechanism often has information about the past behavior of each sensor. Thus, in this Section, we develop a different sequential test that is able to incorporate prior information based on the Bayes Factor [119], which traces back to the work of Jeffreys [113], [114]. This approach involves the use of prior probability distribution

functions which must be integrated and thus, may cause this approach to be computationally very expensive if there are no closed form expressions of these integrals, which unfortunately happens often.

As we have noted, many WSN works today still use fixed length tests, such as [45], [189] or [155]. However, a variable length test may bring some advantages over a fixed length one in issues of capital concern for wireless sensor network such as battery consumption or bandwidth use. In [46], a framework for change detection in sensor networks which uses a non-parametric model is proposed, however, they do not make use of any prior information that may be available. And in [196] there is a study on Bayes Factor sequential probability ratio test, which however is computationally costly and hence, is not adequate for WSN.

The major problem with a Bayesian approach in WSNs is the computational cost. In order to avoid this problem, we use Beta prior distributions, which allow us not only to obtain closed form expressions of the probability distributions involved, but also, allows us to develop an updating rule very efficient both in terms of time and computational resources. Thus, our algorithm can be used as an alternative to the simple Counting Rule and SPRT, which (1) may make use of prior information because it is a Bayesian approach, (2) is implemented sequentially, hence, offering all the advantages of sequential tests for WSNs, and (3) has a very high computational efficiency.

5.5.1 Bayes factor using beta priors

Under a Bayesian scheme, we assume to know a prior probability distribution for each hypothesis, $p(H_0)$ and $p(H_1) = 1 - p(H_0)$. Using Bayes theorem, it is possible to obtain $p(H_k|x^n)$ as:

$$p(H_k|x^n) = \frac{p(x^n|H_k)p(H_k)}{\sum_k p(x^n|H_k)p(H_k)}, \quad k = \{0,1\}, \tag{5.26}$$

and this expression can be manipulated to obtain:

$$\frac{p(H_1|x^n)}{p(H_0|x^n)} = \frac{p(x^n|H_1)}{p(x^n|H_0)}\frac{p(H_1)}{p(H_0)} = B_{10}\frac{p(H_1)}{p(H_0)}. \tag{5.27}$$

In (5.27), we observe that the posterior odds are the prior odds times a B_{10} term , which is the Bayes factor (BF). Intuitively, the BF carries information about how likely is the data x^n to have been generated under models H_0 or H_1. When the BF surpasses a certain threshold, a decision is made, similarly to the SPRT model already presented, where there are several thresholds proposed for the value of the BF in order to take a decision [119].

The densities $p(x^n|H_k)$ need to be computed in order to obtain the BF. By assuming that each hypothesis is modeled using a distribution $p(\theta|H_k)$ with an unknown parameter θ, the densities $p(x^n|H_k)$ can be obtained by integration as:

$$p(x^n|H_k) = \int p(x^n|\theta, H_k)p(\theta|H_k)d\theta, \quad k = \{0,1\}. \tag{5.28}$$

By taking into account that in our problem x^n follows a Bernoulli distribution, (5.28) becomes:

$$p(x^n|H_k) = \int_0^1 \theta^{s^n}(1-\theta)^{n-s^n}p(\theta|H_k)d\theta, \quad k = \{0,1\}. \tag{5.29}$$

A major problem to use the BF is that (5.29) can be hard to obtain, see for instance [119], where several numerical methods are reviewed. In the best case, (5.29) can be analytically evaluated and hence, there is no need of numerical methods. This also brings a significant improvement both in computational efficiency and precision. In order to choose a prior $p(\theta|H_k)$ analytically evaluable, the family of conjugate distributions is

of special interest because the conjugacy property holds: the posterior distribution is in the same family of distributions as the prior [60]. These reasons motivate us to choose Beta distributions as priors, which belong to the exponential family. As we will see, this choice will allow us to design a simple and efficient sequential update algorithm to obtain (5.29). The beta distribution has two parameters $\lambda_1 > 0$ and $\lambda_2 > 0$, which we fix a priori. The pdf of the Beta distribution is:

$$Beta(\theta|\lambda_1,\lambda_2) = \frac{\theta^{\lambda_1-1}(1-\theta)^{\lambda_2-1}}{B(\lambda_1,\lambda_2)}, \tag{5.30}$$

where the normalization factor $B(\lambda_1,\lambda_2)$ is the Beta function, also known as Euler integral of first kind:

$$B(\lambda_1,\lambda_2) = \int_0^1 \theta^{\lambda_1-1}(1-\theta)^{\lambda_2-1}d\theta = \frac{\Gamma(\lambda_1)\Gamma(\lambda_2)}{\Gamma(\lambda_1+\lambda_2)}, \tag{5.31}$$

where $\Gamma(a)$ is the gamma function of a.

We use as prior a weighted sum of L Beta distributions, because they allow us to model complicated priors, such as multimodal distributions. The prior parameters are chosen to adapt to the prior information. We define λ^k as the matrix of parameters of the prior distribution of θ under hypothesis H_k, with L rows containing the beta distribution parameters $(\lambda_1^{l,k},\lambda_2^{l,k})$. Using (5.30), our prior becomes:

$$p(\theta|\lambda) = \sum_{l=1}^{L} w_l Beta(\theta|\lambda_1^{l,k},\lambda_2^{l,k}) = \sum_{l=1}^{L} w_l \frac{\theta^{\lambda_1^{l,k}-1}(1-\theta)^{\lambda_2^{l,k}-1}}{B(\lambda_1^{l,k},\lambda_2^{l,k})}, \tag{5.32}$$

where $\sum_{l=1}^{L} w_l = 1$ and each $w_l \geq 0$, so that (5.32) defines a distribution. We now can compute the posterior probabilities $p(x^n|H_k)$ using (5.29) and (5.32) as follows:

$$\begin{aligned} p(x^n|H_k) &= \int_0^1 \theta^{s^n}(1-\theta)^{n-s^n} \sum_{l=1}^{L} w_l \frac{\theta^{\lambda_1^{l,k}-1}(1-\theta)^{\lambda_2^{l,k}-1}}{B(\lambda_1^{l,k},\lambda_2^{l,k})} d\theta \\ &= \sum_{l=1}^{L} w_l \frac{\int_0^1 \theta^{s^n+\lambda_1^{l,k}-1}(1-\theta)^{n-s^n+\lambda_2^{l,k}-1}d\theta}{B(\lambda_1^{l,k},\lambda_2^{l,k})} = \sum_{l=1}^{L} w_l \frac{B(s^n+\lambda_1^{l,k},n-s^n+\lambda_2^{l,k})}{B(\lambda_1^{l,k},\lambda_2^{l,k})} \end{aligned}. \tag{5.33}$$

5.5.2 Bayes factor update algorithm

The expression obtained for $p(x^n|H_k)$ in (5.33) allows obtaining an efficient sequential algorithm to update the prior in a sequential test, as new samples x^n arrive. First, we express (5.33) in terms of the Gamma function using (5.31) as follows:

$$p(x^n|H_k) = \sum_{l=1}^{L} w_l S_{l,k}^n, \tag{5.34}$$

where

$$S_{l,k}^n = \frac{\Gamma(s^n+\lambda_1^{l,k})}{\Gamma(\lambda_1^{l,k})} \frac{\Gamma(n-s^n+\lambda_2^{l,k})}{\Gamma(\lambda_2^{l,k})} \frac{\Gamma(\lambda_1^{l,k}+\lambda_2^{l,k})}{\Gamma(n+\lambda_1^{l,k}+\lambda_2^{l,k})}. \tag{5.35}$$

The values for $S_{l,k}^n$ in (5.35) can be obtained recursively with the help of Lemma 5.

Lemma 5. *The following identity holds for $a \in \{0,1,2,3...\}$ and $k > 0$:*

$$\frac{\Gamma(k+a)}{\Gamma(k)} = \begin{cases} \prod_{i=k}^{k+a-1} i & if \quad a \geq 1 \\ 1 & if \quad a = 0 \end{cases}$$

Proof. For $a = 0$, the proof is straightforward:

$$\left.\frac{\Gamma(k+a)}{\Gamma(k)}\right|_{a=0} = \frac{\Gamma(k)}{\Gamma(k)} = 1.$$

For $a \geq 1$, we will use the following property of the gamma function which holds for any real number $z > 0$:

$$\Gamma(z+1) = z\Gamma(z) \tag{5.36}$$

Proceeding by induction, for $a = 1$ and $k > 0$:

$$\frac{\Gamma(k+1)}{\Gamma(k)} = \frac{k\Gamma(k)}{\Gamma(k)} = k,$$

where we used (5.36). For $a = 2$, we have that:

$$\frac{\Gamma(k+2)}{\Gamma(k)} = \frac{(k+1)\Gamma(k+1)}{\Gamma(k)} = (k+1)k,$$

where again we used (5.36). Now, we assume that for $a > 1$, the following holds:

$$\frac{\Gamma(k+a)}{\Gamma(k)} = \prod_{i=k}^{k+a-1} i,$$

and proceed to obtain the value for $a+1$:

$$\frac{\Gamma(k+a+1)}{\Gamma(k)} = \frac{(k+a)\Gamma(k+a)}{\Gamma(k)} = (k+a)\prod_{i=k}^{k+a-1} i = \prod_{i=k}^{k+a} i,$$

which finishes the proof. □

Lemma 5 allows obtaining the values for $S_{l,k}^n$ in (5.35) sequentially, as new data x^n arrives. Observe that (5.35) can be expressed as:

$$S_{l,k}^n = S_{1,l,k}^n S_{2,l,k}^n \left(S_{3,l,k}^n\right)^{-1}, \tag{5.37}$$

where:

$$\begin{cases} S_{1,l,k}^n = \dfrac{\Gamma(\lambda_1^{l,k} + s^n)}{\Gamma(\lambda_1^{l,k})} \\ S_{2,l,k}^n = \dfrac{\Gamma(\lambda_2^{l,k} + n - s^n)}{\Gamma(\lambda_2^{l,k})} \\ S_{3,l,k}^n = \dfrac{\Gamma(\lambda_1^{l,k} + \lambda_2^{l,k} + n)}{\Gamma(\lambda_1^{l,k} + \lambda_2^{l,k})} \end{cases}. \tag{5.38}$$

In these expressions, s^n, $n - s^n$ and n are natural numbers and hence, we can apply Lemma 5 to recursively update $S_{l,k}^n$ as new data x^n arrives. Observe that all λ parameters are greater than zero, because the Beta distribution parameters must be positive. Thus, all conditions from Lemma 5 are satisfied.

The updating procedure depends on each new x^n. If $x^n = 1$, then s^n and n increase one unit with respect to their previous values, whereas $n - s^n$ remains the same. Hence, we must update $S_{1,l,k}^n$ and $S_{3,l,k}^n$ only. If $x^n = 0$, then $n - s^n$ and n increase one unit with respect to their previous values, whereas s^n remains the same. Hence, we must update $S_{2,l,k}^n$ and $S_{3,l,k}^n$ only. Thus, when a sample x^n arrives, we always update $S_{3,l,k}^n$ and depending

Algorithm 13 Sequential Bayes test

Input: $\lambda^k, w_l, B_{t,0}, B_{t,1}$
Initialize $stop = False$, $n = 0$, $s^{-1} = 0$
Initialize $S^{-1}_{1,l,k} = S^{-1}_{2,l,k} = S^{-1}_{3,l,k} = 1$
while $stop$ is $False$ **do**
  Obtain a new sample x^n
  Update $s^n = \sum_{i=1}^{n} x_i = s^{n-1} + x^n$
  for $k = 0, 1$ **do**
    for $l = 1, 2, ..., L$ **do**
      $S^n_{3,l,k} = (\lambda_1^{l,k} + \lambda_2^{l,k} + n - 1) S^{n-1}_{3,l,k}$
      if $x^n = 1$ **then**
        $S^n_{1,l,k} = (\lambda_1^{l,k} + s^n - 1) S^{n-1}_{1,l,k}$
        $S^n_{2,l,k} = S^{n-1}_{2,l,k}$
      if $x^n = 0$ **then**
        $S^n_{2,l,k} = (\lambda_2^{l,k} + n - s^n - 1) S^{n-1}_{2,l,k}$
        $S^n_{1,l,k} = S^{n-1}_{1,l,k}$
  Obtain $p(x^n|H_k)$ using (5.34) for $k = \{0, 1\}$
  Obtain $B^n_{10} = \frac{p(x^n|H_1)}{p(x^n|H_0)}$
  if $B^n_{10} > B_{1,t}$ **then**
    Decide H_1 and set $stop = True$
  else if $B^n_{10} < B_{0,t}$ **then**
    Decide H_0 and set $stop = True$
  else
    Set $n = n + 1$
Output: Decision taken, n

on whether $x^n = 1$ or $x^n = 0$ we update $S^n_{1,l,k}$ or $S^n_{2,l,k}$. Note that these updatings are straightforward according to Lemma 5. If $s^n = 0$, then $S^n_{1,l,k} = 1$. And if $s^n \geq 1$, then $S^n_{1,l,k} = (\lambda_1^{l,k} + s^n - 1) S^{n-1}_{1,l,k}$. A similar reasoning applies to $S^n_{2,l,k}$ and $S^n_{3,l,k}$.

With all this, we propose an Algorithm that makes use of the procedure described above to sequentially update the marginal distributions $p(x^n|H_k)$ in order to obtain a sequential test, based on the Bayes factor B_{10}. We provide as inputs to the algorithm the λ^k prior values and the w_l weights for each l value, as well as the threshold values $B_{t,0}$ and $B_{t,1}$ that we wish to establish as stopping rules. When a new sample x^n arrives, the algorithm updates the two marginal distributions $p(x^n|H_0)$ and $p(x^n|H_1)$, obtains the Bayes factor at sample n B^n_{10} and compares it to the two thresholds. If $B_{10} > B_{1,t}$, the test stops and H_1 is accepted. If $B_{10} < B_{0,t}$, the test stops and H_0 is accepted. Otherwise, a new sample is obtained. The whole procedure is summarized in Algorithm 13. Observe that this algorithm is very efficient, because:

- We do not need to evaluate any gamma function.
- Only sums, products and divisions are involved in each algorithm iteration.
- The updating is based in a constant number of operations as each new sample x^n arrives, thus, our algorithm has a linear number of operations with L and constant with n.

All these reasons make Algorithm 13 very suitable for an environment with limited resources, as a WSN.

5.5.3 Bayes Factor vulnerability to intelligent attacks

However, the BF test we just presented can also be vulnerable to the intelligent attack from Theorem 4. From (5.27), we have that:

$$\log(B_{10}) = \log\left(\frac{p(x^n|H_1)}{p(x^n|H_0)}\right) = \log\left(p(x^n|H_1)\right) - \log\left(p(x^n|H_0)\right), \tag{5.39}$$

and, from (5.34) and (5.37), we know that

$$p(x^n|H_k) = \sum_{l=1}^{L} w_l S_{l,k}^n = \sum_{l=1}^{L} w_l \frac{S_{1,l,k}^n S_{2,l,k}^n}{S_{3,l,k}^n}. \tag{5.40}$$

If $x^n = 0$, using Algorithm 13, we can formulate (5.40) as:

$$p(x^n|H_k) = \sum_{l=1}^{L} w_l \frac{\lambda_2^{l,k} + n - s^n - 1}{\lambda_1^{l,k} + \lambda_2^{l,k} + n - 1} \frac{S_{1,l,k}^{n-1} S_{2,l,k}^{n-1}}{S_{3,l,k}^{n-1}}, \tag{5.41}$$

and equivalently for $x^n = 1$:

$$p(x^n|H_k) = \sum_{l=1}^{L} w_l \frac{\lambda_1^{l,k} + s^n - 1}{\lambda_1^{l,k} + \lambda_2^{l,k} + n - 1} \frac{S_{1,l,k}^{n-1} S_{2,l,k}^{n-1}}{S_{3,l,k}^{n-1}}. \tag{5.42}$$

If we particularize for $L = 1$, we obtain that:

$$p(x^n|H_k) = \begin{cases} \dfrac{\lambda_2^{1,k} + n - s^n - 1}{\lambda_1^{1,k} + \lambda_2^{1,k} + n - 1} p(x^{n-1}|H_k) & \text{if} \quad x^n = 0 \\ \dfrac{\lambda_1^{1,k} + s^n - 1}{\lambda_1^{1,k} + \lambda_2^{1,k} + n - 1} p(x^{n-1}|H_k) & \text{if} \quad x^n = 1 \end{cases}, \tag{5.43}$$

which, by taking logarithms, becomes:

$$\log\left(p(x^n|H_k)\right) = \begin{cases} \log\left(\lambda_2^{1,k} + n - s^n - 1\right) - \log\left(\lambda_1^{1,k} + \lambda_2^{1,k} + n - 1\right) + \log\left(p(x^{n-1}|H_k)\right) & \text{if} \quad x^n = 0 \\ \log\left(\lambda_1^{1,k} + s^n - 1\right) - \log\left(\lambda_1^{1,k} + \lambda_2^{1,k} + n - 1\right) + \log\left(p(x^{n-1}|H_k)\right) & \text{if} \quad x^n = 1 \end{cases}, \tag{5.44}$$

and hence, (5.39) can be rearranged using (5.44) in a similar way to (5.11). Thus, it would be possible to adapt Theorem 4 also to exploit our BF sequential test. Note that in case that $L > 1$, the sums in (5.41) and (5.42) make that the results of Theorem 4 do not apply directly, as the presence of more complex prior functions significantly complicates finding the optimal control law. In this case, a possible approach would be using the DP approach shown in Lemma 1 in order to obtain the optimal attack control policy for the AS.

5.6 Empirical results

In this Section, we provide empirical results on the theoretical developments of this Chapter:

- First, we focus on the CSMA/CA problem presented in Chapter 4 and evaluate the impact that an intelligent attacker following Theorem 4 may have on the defense mechanism.

θ_1	Attack	H_0	H_1	ND	Lenght	$R, \gamma = 0.5$	$R, \gamma = 0.7$	$R, \gamma = 0.9$
0.25	No attack	0.256	0.01	0.734	185.42	0.39	0.66	1.98
	Naive	0.016	0.286	0.698	182.01	0.48	0.82	2.48
	Intelligent	0	0	1	200.0	2.00	3.31	7.94
0.3	No attack	0.854	0.042	0.104	99.02	0.36	0.61	1.95
	Naive	0.034	0.854	0.112	93.83	0.62	1.02	3.00
	Intelligent	0	0	1	200.0	1.99	3.10	6.23
0.4	No attack	0.964	0.036	0.0	30.33	0.44	0.70	2.00
	Naive	0.044	0.956	0.0	26.88	0.81	1.32	3.44
	Intelligent	0	0	1	200.0	1.89	2.69	5.15

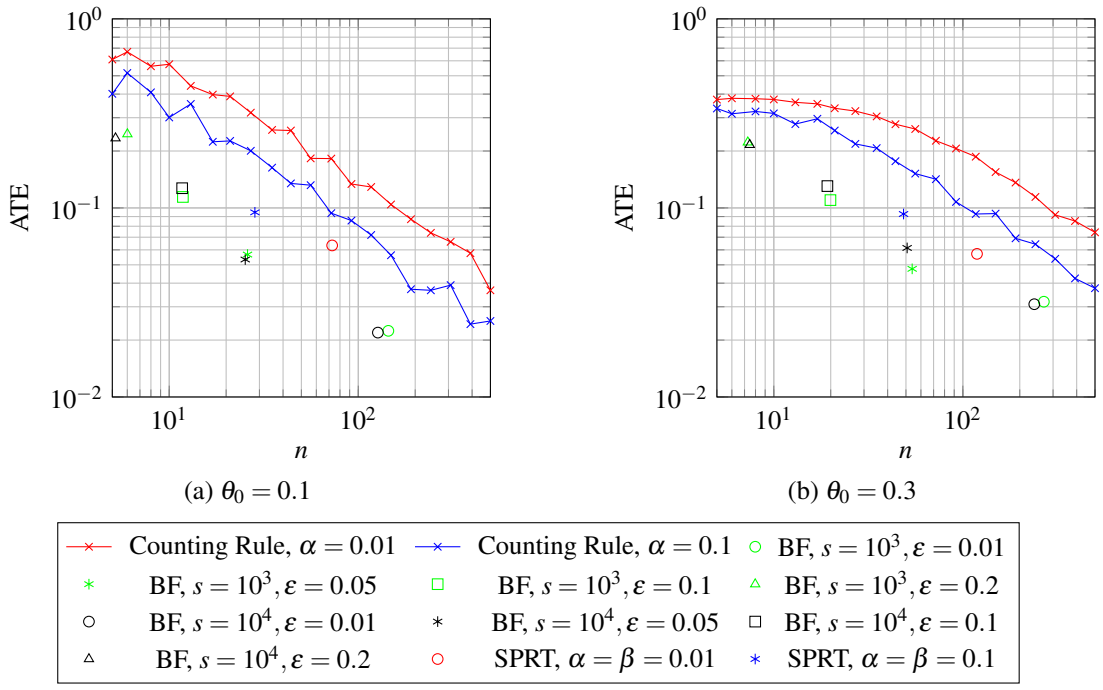
Table 5.2 Results for the CSMA/CA detection problem using SPRT without truncation, when $\theta_0 = 0.2$ and $\theta_1 = \theta_0 + \{0.05, 0.1, 0.2\}$. H_0, H_1 and ND are the probabilities that the SPRT decides H_0, rejects H_0 and does not reach a decision respectively. Length is the average samples needed to make a decision. R is the total reward, computed using (5.17), for different values of γ. Note that the intelligent attack described in the previous Section is able to successfully overcome an SPRT based defense mechanism.

- Then, using the same CSMA/CA problem background, we study the impact that having prior information may have on the speed and error of the sequential test by comparing the counting rule, SPRT and our BF test.
- We then study the effects of using our OCSVM-SPRT defense mechanism in the CSMA/CA problem. We discuss both the influence of the OCSVM-SPRT parameters on the total error and its performance when compared to SPRT.
- Finally, we introduce the SSDF attack and show how our OCSVM-SPRT procedure can be used together with other defense mechanisms in order to improve the resistance against attacks.

We remark that the defense mechanism has imperfect and incomplete information about the sensors: it has incomplete information because it does not know which are GSs and which are ASs, and it has imperfect information because it does not observe the mixed actions of a sensor, but its actions realizations.

5.6.1 Simulation 1: Intelligent attacks against SPRT in the backoff attack

First, we evaluate the impact that an intelligent attacker as the one already described would have on the CSMA/CA problem of Chapter 4. Let us assume that we have a single AS, and that the Server only has access to the actions realizations of the sensors in the WSN. As in Chapter 4, we assume that the defense mechanism is able to detect instantaneously when an agent deviates from the binary exponential backoff, hence, the actions of the agent are a binary variable, whose values correspond to the case in which the agent follows the binary exponential backoff or not. We assume that the repeated game strategy is UNR, and the Server wants to know whether each sensor is following $\theta_0 = z_o$ by using an SPRT without truncation. In this case, for simplicity, we assume that $z_o = \theta_0 = 0.2$ and that $\theta_1 = \theta_0 + \{0.05, 0.1, 0.2\}$ in order to test several cases. We test for the case in which there is no attack, i.e., the behavior that a GS would follow in which $z_o = \theta_0$, the case in which there is a naive attack, i.e., the sensor uses $z_o = \theta_1$, and the intelligent attacker already described, where we consider that SPRT is not truncated, but for practical reasons, we interrupt the simulation after $N = 200$ SPRT stages: we note that the test might not have decided by that time. For each case, we average 500 runs of each test and the results are in Table 5.2, where we observe that:



(a) $\theta_0 = 0.1$ (b) $\theta_0 = 0.3$

Fig. 5.7 Simulation result curves. Note that our proposed BF approach obtains a lower averaged total error using fewer samples n than the counting rule and SPRT, for all the s and ε values tested. In the BF approach, the tested values of ε have a greater impact than the values of s on the test ATE. This is to be expected, since ε controls the sensitivity of the test. For all the values tested, our BF approach significantly outperforms the counting rule and SPRT.

- In case that there is no attack, the result of the SPRT correctly detects that $x^n \sim H_0$. However, note that as $\theta_1 - \theta_0$ decreases, the number of samples required to make a decision increases and hence, the number of tests which have not made a decision after 200 samples increases. This is an expected result: when both θ values are close, the decision is harder to make as the distributions under both hypotheses overlap significantly.

- The naive attack is easily detected in most cases, especially as $\theta_1 - \theta_0$ increases. Note that, again, as $\theta_1 - \theta_0$ increases, the number of samples required to make a decision decreases. Observe that in all cases, the reward obtained by the naive attack is larger than the reward obtained if there is no attack: hence, even a naive attack is an option for an attacker against an SPRT.

- The intelligent attack presented in the previous Section is able to successfully overcome the defense mechanism: it is never detected and its reward is significantly higher than in the other two cases, for all γ and θ_1 values.

Hence, if in the CSMA/CA problem, the deviations were detected using an SPRT mechanism, it would be vulnerable, not only to our intelligent attack, but also to a naive attack: note that there is an increase in the reward under both cases.

5.6.2 Simulation 2: Bayes Factor test performance in the backoff attack

Now, let us assume that the server is testing the stream of actions of a GS, that is, we consider that now there is no attack, in order to compare the BF test we propose with the Counting Rule, which, as we indicated, is commonly used and hard to beat due to being the universally most powerful test [44]; and also to the SPRT procedure already described. Our main objective is to compare the performance of these three tests when there is no attack. Since the counting rule and SPRT performances are not directly comparable to BF, we use an Averaged Total Error (ATE) metric to compare them:

$$ATE = \frac{\sum Dec_e}{\sum Dec}, \tag{5.45}$$

where Dec_e are the erroneous decisions and Dec the decisions made by an HT. For this simulation, we employ a set of 21 Bernoulli parameters θ_{test} linearly spaced in the interval $[0, 0.5]$. Then, we fix θ_0 and we perform 100 hypothesis tests for each θ_{test}, using both the counting rule and our BF method. After each of these hypothesis tests, we obtain the ATE by adding the number of erroneous decisions taken and dividing by the 2100 simulations performed. Note that this means that ATE (5.45) gives us an averaged measure of the decision error over the θ_{test} values. We simulate for $z_o = \theta_0 \in \{0.1, 0.3\}$. For the counting rule, we use $\alpha \in \{0.01, 0.1\}$ and employ 20 logarithmically spaced values for n in the interval $[5, 500]$. For SPRT, we use $\alpha = \beta \in \{0.01, 0.1\}$ and define $\theta_1 = \theta_0 + 0.1$, which as seen in Table 5.2, provides a good balance between samples taken and test accuracy. Also, for SPRT, we truncate the test after 500 samples, and returns H_1 if no decision has been achieved at the end of the test.

For the BF approach, we use as decision thresholds $B_{0,t} = 3^{-1}$ and $B_{1,t} = 3$, following [119]. We also need to choose the prior parameters. For the simulations, we use $L = 1$ Beta prior distribution for simplicity, and we define the Beta parameters as a function of two values: the strength of the prior, defined as $s = \lambda_1^k + \lambda_2^k$, which denote the confidence we have in the prior, and $\varepsilon = \theta_1 - \theta_0$, which controls the sensitivity of the BF test. Since the mean of the beta distribution is:

$$\mu = \frac{\lambda_1}{\lambda_1 + \lambda_2},$$

we set $\mu = \theta_0$ and obtain the Beta distribution parameters for the set $\langle s, \theta_0, \varepsilon \rangle$ as:

$$\begin{cases} \lambda_1^{1,0} = s \cdot \theta_0 \\ \lambda_2^{1,0} = s \cdot (1 - \theta_0) \\ \lambda_1^{1,1} = s \cdot (\theta_0 + \varepsilon) \\ \lambda_2^{1,1} = s \cdot (1 - \theta_0 - \varepsilon) \end{cases}. \tag{5.46}$$

In our simulations, we use (5.46) to define the prior distributions with $\varepsilon = \{0.01, 0.05, 0.1, 0.2\}$ and $s = \{10^3, 10^4\}$, that is, we test for two different confidence values in the prior and for different sensitivities for the BF test. The results are in Figure 5.7, where we can observe that our proposed BF approach performs significantly better than the counting rule in both ATE and the number of samples required to take a decision, and is also better than SPRT. This means that using a sequential BF hypothesis test provides a lower average error of decision taking a smaller number of samples, and both are crucial in a WSN.

We also observe that the total error using BF and SPRT is distributed around $\theta_{test} = \theta_0$, while the counting rule strongly concentrates its error on $\theta_{test} > \theta_0$ in order to satisfy the restriction $P(H_1|H_0) \leq \alpha$. That is, the counting rule provides a bound in the type I error, i.e., α, but the type II error depends on n and increases as n decreases: lower sample sizes yield a higher type II error. Finally, we note that the BF test performance could

	SPRT	SPRT-OCSVM
NA	97/3/0	93,2/6.8/0
SA	3.6/96.4/0	2/98/0
OWT	100/0/0	0/100/0
ONT	0/0/100	0/100/0

Table 5.3 Test results for $\theta_0 = 0.5$ and $\rho = 0.05$, for all the tests simulated. Each table entry is the percentage of times that H_0 was decided / H_0 was rejected / no decision was taken. Observe how when facing the control law from Theorem 4, SPRT is totally unable to detect the AS. However, the exact opposite happens with our proposed SPRT-OCSVM mechanism: it always detects such an AS.

be improved by having a more detailed knowledge about the prior distributions. Observe that we used a simple prior for these simulations, but in real life environments, in which a certain knowledge of the prior may be present [168] [138], our proposed BF algorithm may perform even better. However, as we have noted, SPRT and BF test are vulnerable to intelligent attacks, and hence, their performance might be seriously compromised.

5.6.3 Simulation 3: Testing the performance of OCSVM-SPRT in the backoff attack

Now, we go back to the case in which there might be ASs, and study the influence of the parameters of OCSVM-SPRT on the Server detection capabilities. For this simulation, we fix $\alpha = \beta = 0.05$ for the SPRT; note that these values satisfy the condition (5.19). We then test for 10 values of $z_o = \theta_0$ equispaced in the range $\theta_0 \in [0.1, 0.7]$, and for each θ_0 value, we define $\theta_1 = \theta_0 + 0.2$, that is, we use as θ_1 another 10 equispaced values in the range $\theta_1 \in [0.3, 0.9]$. For each θ_0 value, we train an OCVSVM using 500 different z vectors generated from a Bernoulli distribution with $\theta = \theta_0$, using $\nu = 0.1$ and a Gaussian Kernel; each z vector has a length $M = 5$ and hence each z has a length of 9 samples. We then obtain the validation error using another 500 z vectors. We train 5 different OCSVM for each test and θ_0 value, using the OCSVM that provides the lower value for the sum of the training and validation error.

For each pair of θ_0 and θ_1 value, we average the results for 500 runs of each test, where all the tests are finished for practical reasons after 200 stages, where we note that the test might not have decided by that time. We test for four different situations on the CSMA/CA problem, all of them considering that there is a single sensor under test: (1) a situation of No Attack (NA), in which the sensor is a GS and hence, we have that $x^n \sim Bernoulli(\theta_0)$; (2) a naive, Simple Attack (SA) situation, in which there is a naive AS wuch that $x^n \sim Bernoulli(\theta_1)$; (3) an intelligent attack situation in which the AS uses the control law from Theorem 4 with $N = 100$, i.e., the test is truncated, which we denote as Optimal With Truncation (OWT); (4) an attack situation in which the AS uses the same control law, without truncation, which we denote as Optimal No Truncation (ONT). Each of these different situations are faced to an SPRT defense mechanism that uses (5.12) and also, to our modified SPRT-OCSVM scheme, using $\rho = 0.05$, because, as Figure 5.6 shows, it provides a good tradeoff in the error under both H_0 and H_1.

The results can be observed in Table 5.3 and Figures 5.8, 5.9 and 5.10. First, in Table 5.3 we show the test results for $\theta_0 = 0.5$, and we note how the control law proposed in Theorem 4 allows that the AS is never detected under an SPRT defense mechanism. Note that the AS is able to either make that the SPRT test never makes a decision if no truncation is done, or is able to be detected always as a GS if the SPRT is truncated. However, our proposed modification, SPRT-OCSVM, allows detecting such an AS with high accuracy. As we advanced, SPRT-OCSVM is able to perform better under attack by means of decreasing the test performance

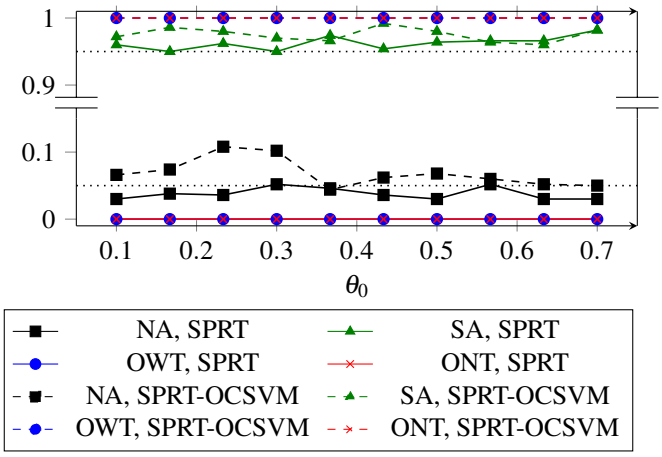


Fig. 5.8 Proportion of H_0 rejections for the different schemes proposed as a function of θ_0. The dotted lines correspond to the α and $1-\beta$ values of the tests. Note that under H_0, i.e., NA, our proposed SPRT-OCSVM performs worse than SPRT, rejecting H_0 more often; and under H_1, SPRT-OCSVM works better than SPRT, as we advanced in Figure 5.6. However, note that the improvement in detecting an AS following the control law from Theorem 4 is dramatic: while SPRT is never able to detect it, SPRT-OCSVM always detects the AS.

under H_0, i.e., when there is no attack. These results apply to all the tested values of θ_0, as can be observed in Figure 5.8, where the proportion of times that each test rejects H_0 is represented as a function of θ_0.

In terms of the total cumulative reward in (5.17), we can observe in Figure 5.9 that an AS using the control law from Theorem 4 is able to obtain a better reward by following that control law if the defense mechanism is an SPRT. However, that control law is not successful against our modified SRPT-OCSVM: note that as $\gamma \to 1$ the AS receives no benefit in attacking, and it would receive a higher reward by behaving as a GS. Note that this dependency on the value of γ comes from the fact that lower values of γ cause that the total reward strongly depends on the rewards at the first time steps. Since any detection method takes some time to make a decision, this problem cannot be easily solved. However, as γ approaches 1, the AS puts a larger emphasis on future rewards and in this case, the ability to camouflage becomes crucial if the AS wants to obtain a large reward.

Finally, in Figure 5.10 we can observe an example of the difference that our proposed approach has when compared to the standard SPRT. Note that if the OCSVM detects a signal that does not follow the expected spectral pattern, the modified $LLR^{n'}$ from (5.25) starts growing with respect to the SPRT LLR^n. Eventually, this means that the AS is detected. Also, observe that the OCSVM brings a very small increase, which is controlled by the ρ parameter. As we noted, a larger ρ brings a higher detection under attack, because it increases the $LLR^{n'}$ faster, but that also means that the error increases under H_0.

5.6.4 Simulation 4: Using OCSVM-SPRT to enhance the defense in an SSDF attack

Now, we turn our attention from the CSMA/CA problem to the SSDF attack presented in the introduction of this Chapter. We consider that we have a CSS WSN with I sensors, which send binary reports u_m^k to the FC in order to make a decision about the channel state u_d^k. Each report from the sensors to the FC is denoted by the binary variable u_m, where $u=1$ means that the channel is busy, $u=0$ means that the channel is idle and $m \in \{1,2,...,I\}$ indexes the sensors. We do not consider the details of the sensing mechanism used by the sensors: see [257] for some possible schemes. Note that each u_m^k may differ from the actual u^k due to errors in the sensing method or due to the presence of ASs. We model the first case by assuming that each sensor

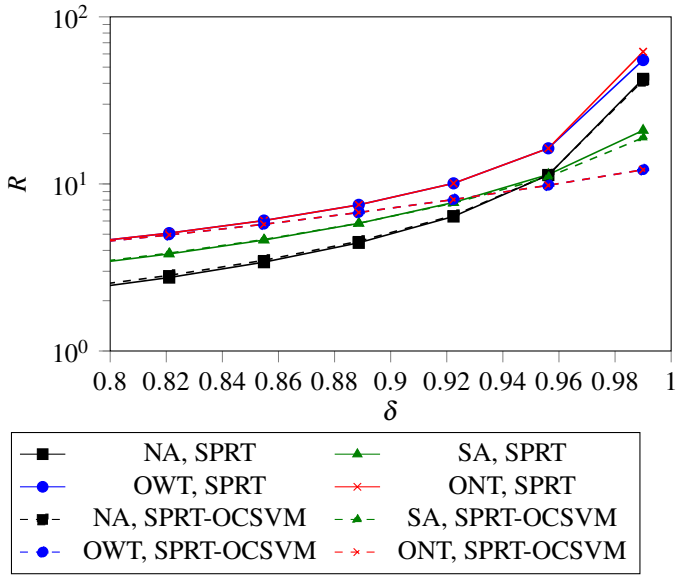


Fig. 5.9 Detail on the total cumulative reward R from (5.9) obtained for an AS under the different schemes proposed. For low values of γ, the use of SPRT or SPRT-OCSVM does not bring significant differences. However, as $\gamma \to 1$, note how SPRT-OCSVM causes the AS to obtain a lower reward than if he did not attack. While the AS obtains an advantage in terms of R against SPRT by using Theorem 4, this advantage vanishes when facing our proposed SPRT-OCSVM mechanism.

has a probability of obtaining a wrong sensing result P_c, which is independent and equal for all sensors. We consider P_c to be constant and independent among the sensors. Thus, in this case, each u_m follows a Bernoulli distribution of parameter P_c if $u = 0$ and $1 - P_c$ if $u = 1$. We base our model in P_c because it simplifies our analysis significantly and it is flexible enough to take into account different phenomena: related to the channel as the shadowing and fading in the sensors or related to the sensing procedure chosen.

Some possible naive SSDF attacks are always yes (AY), i.e., the AS always reports $u_m^k = 1$; always no (AN), i.e., the AS always reports $u_m^k = 0$ or always false (AF), i.e., the AS always reports the opposite of what it has sensed. Even though these are naive attacks, they often appear [41], [260], [238]. To these attack strategies, we add the control law from Theorem 4, which we name Intelligent Attack (IA).

We use SPRT and our OCSVM-SPRT algorithm to enhance the defense mechanism. For each sensor m, we run an instance of SPRT or OCSVM-SPRT in order to detect sensors that are deviating from the expected behavior of a GS, where each x^n is composed by the u_m^k values for sensor m. Note that in this case, if the transmission probability of the primary is P_{tr}, the probability of receiving $u_m^k = 1$, which is used to define the null hypothesis, is $\theta_0 = (1 - P_c) \cdot P_{tr} + P_c \cdot (1 - P_{tr})$. If sensor m is detected as an AS, the sensor is banned from the WSN and is not called again for reports.

The whole FC procedure can be seen in Figure 5.11. At each time step k, the FC asks for a report to a sensor m. This sensor m is randomly selected among the sensors not banned if the majority rule is used, and under EWSZOT sensors are sorted by their reputations, following Algorithm 12. Then, a defense mechanism is used to update the list of banned sensors, and if sensor m has not been banned, its report is used to update the fusion rule. If more information is needed to make a decision, then the process starts over; otherwise, the decision u_d^k is returned.

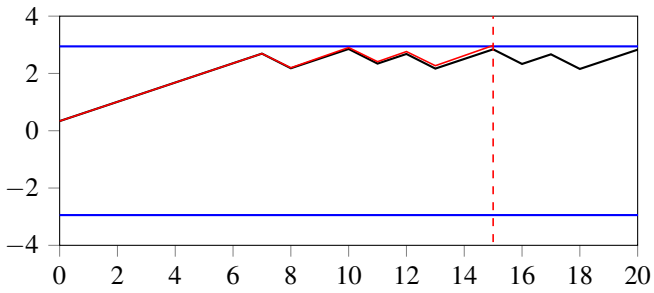


Fig. 5.10 Example of detection for $\theta_0 = 0.5$ and $\theta_1 = 0.7$. The blue lines are the LLR^n thresholds from (5.6). In both cases, we compare a realization of the control law from Theorem 4 without truncation, using SPRT (black) and SPRT-OCSVM (red). The dashed vertical line indicate when each test ends. Observe that, as in Figure 5.2, the SPRT is unable to detect the attack. However, SPRT-OCSVM is able to do so: when it believes that there is an AS, it starts increasing slowly the LLR_n value using (5.25). Note that this means that eventually, the AS is detected.

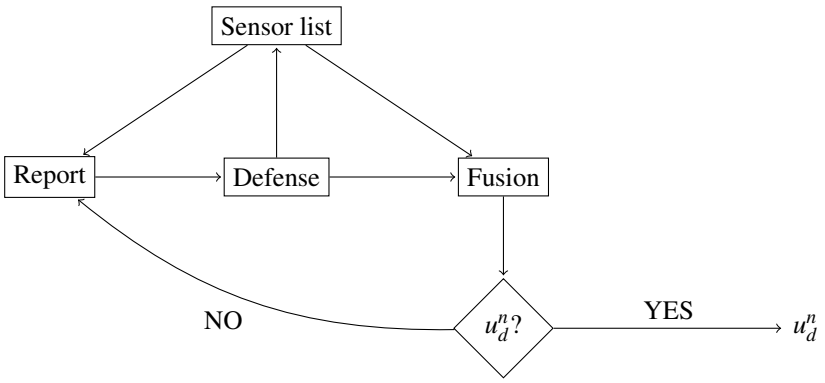
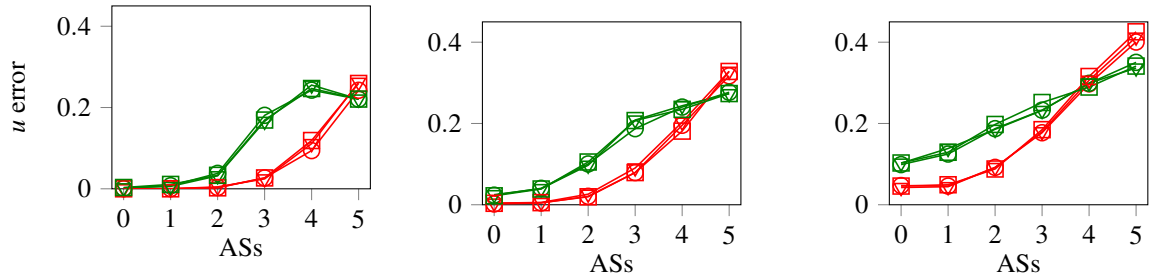


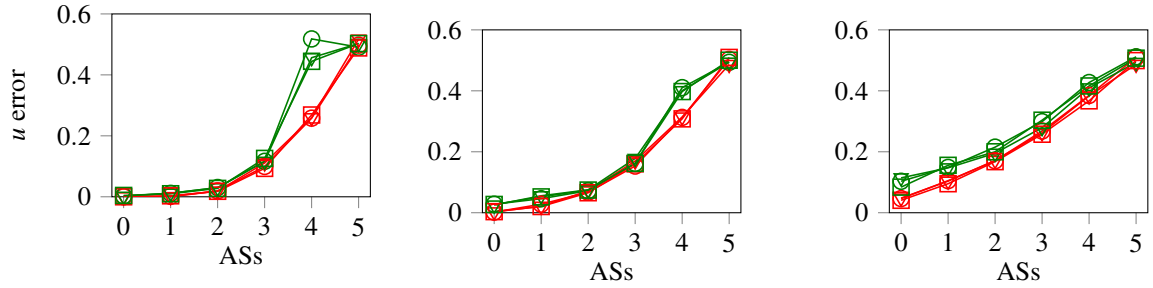
Fig. 5.11 Flow diagram for each time step k of the SSDF problem. The sensor list contains the list of sensors banned and not banned, and hence, it is used to determine to which sensors the FC asks for a report and takes into account in the fusion procedure.

We run this attack in a WSN with $I = 10$ sensors with $P_{tr} = 0.5$. We test for three possible defense mechanisms: no defense mechanism, an SPRT and an OCSVM-SPRT mechanism. The two latter cases use $\theta_0 = (1-P_c)\cdot P_{tr} + P_c \cdot (1-P_{tr})$, $\theta_1 = \theta_0 + 0.1$ and $\alpha = \beta = 0.01$. We use the OCSVM defined in Section 3.3.3 with $\rho = 0.05$. Regarding the fusion rule parameters, we set $N_{mr} = 20$ for the majority rule, and for EWSZOT, we use $g = 5.51$ as in [260] and $q = 2$. We truncate all tests after $N = 20$ reports, as this is the normal case in practical implementations to avoid lockouts.

We define a grid on the number of attackers and the sensing error combining $\{0,1,2,3,4,5\}$ ASs and $P_c = \{0.1, 0.2, 0.3\}$. As attack strategies, we use AY, AN, AF and IA, and as defense mechanism, we use no defense mechanism, SPRT and OCSV-SPRT. For each combination of these parameters, the results are averaged over 50 realizations of each test, with $K = 50$. The results can be observed in Figures 5.12 and 5.13. All attacks are successful and increase their harm with the number of ASs: this is specially remarkable for the IA case, since the control law that the ASs follow was not derived against the fusion rules used, i.e., the fusion rule used by EWSZOT (5.15) differs from the SPRT mechanism (5.4). Note that using our proposed OCSVM-SPRT defense mechanism does not cause any negative impact in the decision error, and it does help against the IA as it provides the best results and helps to decrease the error for all the fusion rules tested. Hence, OCSVM-SPRT



(a) FC average error for the Always No (AN) attack. From left to right: $P_c = \{0.1, 0.2, 0.3\}$. Lower is better.



(b) FC average error for the Always False (AF) attack. From left to right: $P_c = \{0.1, 0.2, 0.3\}$. Lower is better.

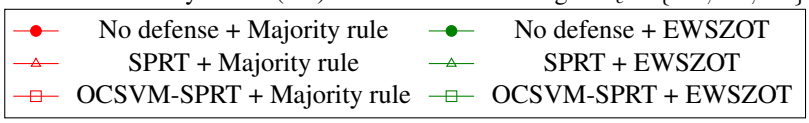


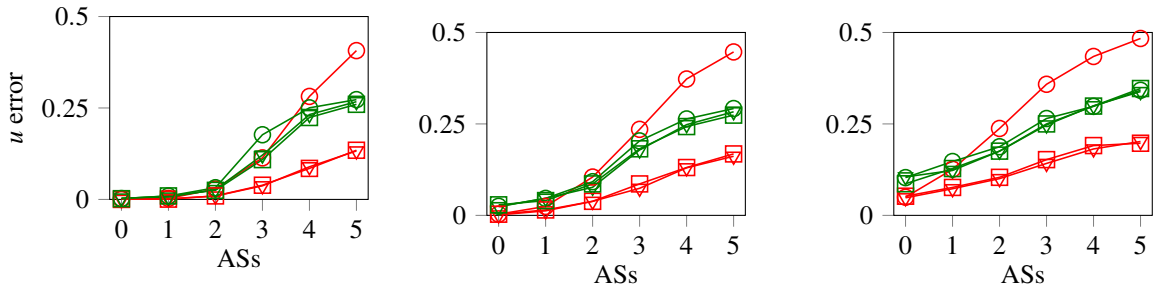
Fig. 5.12 Results for AN and AF attacks. Note that all attack strategies are successful, since the error increases with the number of ASs. In these two attacks, the choice of the defense mechanism does not make a significant difference, as happens in the other two attacks, see Figure 5.13.

can be used as an additional security layer against SSDF attacks, which keeps track of the behavior of each sensor, and bans those who deviate from the expected behavior. Note that OCSVM-SPRT needs an increased computational power in order to obtain this additional security.

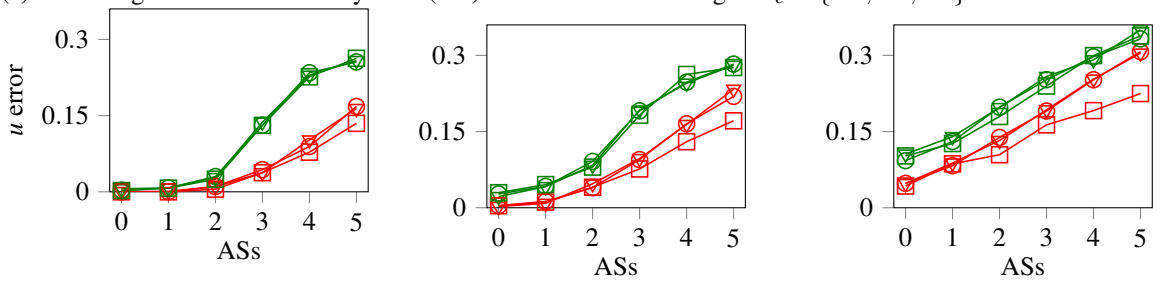
5.7 Conclusions

In this Chapter, we have dealt with asymmetry in attacks, by assuming that the defense mechanism presents a fixed behavior, while the attacker optimizes dynamically its behavior in order to exploit the defense mechanism, and also that while the attacker had perfect and complete information, the defense mechanism had imperfect and incomplete information. The key idea that we have studied are sequential tests, in which a decision is made on whether a data stream follows a certain distribution or not. One popular sequential test is SPRT, which is the base for many current WSN defense mechanisms.

As we have noted, SPRT has a dangerous underlying assumption, which is that the distribution of the data stream does not change with time. We have shown that a dynamic attacker as the one we propose in this Chapter is able to successfully exploit an SPRT procedure without being detected. As SPRT can be studied as an MDP, we can attack it using DP tools; however, in our case we could derive the control law by reasoning on the control problem. Hence, the equilibrium conditions seen in Chapter 4 are not valid anymore, as now the players need to take into account that a deviation in time step n will not be observed at that time step, but possibly many time steps afterwards. This means that the equilibrium conditions become harder to check, as



(a) FC average error for the Always Yes (AY) attack. From left to right: $P_c = \{0.1, 0.2, 0.3\}$. Lower is better.



(b) FC average error for the Intelligent Attack (IA). From left to right: $P_c = \{0.1, 0.2, 0.3\}$. Lower is better.

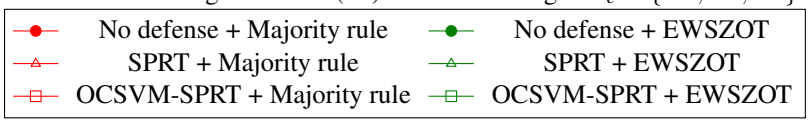


Fig. 5.13 Results for IA and AY attacks. Note that, again, all attack strategies are successful, since the error increases with the number of ASs. In the AY attack, note that not having a defense mechanism and using the Majority rule significantly increases the error. In case of IA, note that as P_c increases, OCSVM yields a lower error.

now there is significantly more room for one-shot profitable deviations. Thus, in many cases the players cannot do better than using their NE, which guarantees them a minimum payoff, and this is due to not having perfect information anymore. A consequence is that the framework of RGs may not be the best for the case in which we cannot detect a deviation instantaneously, which happens often in realistic situations. But also note that, since the defense mechanism does not have complete information, the NE may not be computed, and a different concept, Bayesian Equilibrium, should be used instead.

We also note that the actions for the ASs in the CSMA/CA game from Chapter 4 were following the backoff procedure or not. However, in real environments, the defense mechanism does not know this and needs to perform an HT to detect deviations, as we explain in Chapter 6. Due to all these reasons, it is better to use a Partially Observable framework to model the backoff attack, in order to take into account the imperfect information. In Chapters 6 and 7, we adapt the backoff attack behind the CSMA/CA game to a partially observable framework in order to being able to find strategies for the case in which the players cannot detect deviations instantaneously and without error, that is, an imperfect information setup.

In order to alleviate the effects of the attack, we have used an OCSVM in order to detect deviations from the expected probability distribution of the data stream. Our simulations show that it achieves good results against our proposed attack, significantly increasing the number of detections. Moreover, we have described a CSS WSN, in which the defense mechanisms used were based on SPRT, and there we have seen how our proposed OCSVM-SPRT can be used to enhance the performance of the defense mechanism against several attack types.

However, we may think that changes in the attack strategy may allow an intelligent attacker to use a control law different from the one in Theorem 4 in order to overcome as well the OCSVM-SPRT test. In Chapter 6, we use RL tools that allow an intelligent agent to learn to attack a possibly unknown defense mechanism, i.e., we assume that the attacker now has incomplete information.

Also, we have noted that SPRT is unable to incorporate prior information. In order to address this, we have developed a sequential test based on Bayesian tools which allows incorporating prior information to enhance the test. We get closed form expressions of the probability distributions, which allows us to obtain a sequential test implementation of special interest in WSN. Our algorithm is very efficient, and in terms of performance, it provides a lower average error and requires fewer samples to decide than the counting rule and SPRT. Thus, it is specially suitable for WSN and applications where having a low number of samples is key, and also where there is prior information available for the test: it could be applied, for instance, to sensor fusion problems [178]. However, this test, as we show, is also vulnerable to attackers which can adapt their behavior dynamically.

Hence, in this Chapter we show that an attacker that is able to dynamically adapt its behavior can significantly harm a static and known defense mechanism using the MDP tools introduced in Chapter 2. There are two important questions that arise: the first is what happens if the attackers do not have an explicit or detailed knowledge of the defense mechanism, i.e., they have incomplete information. As we show in Chapter 6, in this case the attackers can use RL tools successfully, provided that they can interact with the system. Hence, Chapters 5 and 6 show that static defense mechanisms, which do not adapt to attackers can be very vulnerable to intelligent attackers. As we note in Chapter 6, this situation is dramatic because many current defense mechanisms are ad hoc designed against specific attacks, but a slight change in the attack may successfully exploit the defense mechanism. Hence, the second important question that arises is what kind of defense mechanisms can be used against such intelligent attackers. In Chapter 7, we propose one intelligent defense mechanism which is based on IRL tools which is able to deal with such intelligent attackers.

Chapter 6

Intelligent attacks against unknown defense mechanisms

6.1 Introduction

In Chapter 5, we have seen how a dynamic attacker is able to successfully exploit a defense mechanism which is static if the attacker has perfect and complete information. As shown, this has a strong impact regarding WSN security in both the CSS and CSMA/CA problems that we are addressing in this work. However, these two assumptions can be challenged in real life environments, as an attacker may only have a partial observation of the defense mechanism state and not know exactly what the defense mechanism is doing. Hence, in this Chapter we move to situations in which the attacker has imperfect and incomplete observation of the defense mechanism, but we still assume that the defense mechanism is static and the attacker dynamic: this is a logical continuation to Chapter 5 as Table 6.1 shows.

As we have already indicated, a lot of effort is devoted nowadays to WSN research [251], [191], [162], where security is one of the key challenges addressed. On one side, the communication protocols and standards used in WSN include security solutions, but most of them are still at a proof-of-concept level according to [220]. On the other side, the existing defense mechanisms are addressed to concrete attacks in concrete setups, such as CSS [72], the 802.15.4 MAC protocol [210] or mechanisms combining different layers [229], to cite some of them. Two related issues that arise with current defense mechanisms are the problem of ad hoc defense and the problem of optimality:

- As it was advanced in Chapter 5, the problem of ad hoc defense arises because defense mechanisms are designed against concrete attacks [72], [210], [229], and hence, changes in the attack may severely affect the performance of the defense mechanism. Indeed, this is the usual procedure followed in many works: a defense mechanism is shown to be vulnerable to a concrete type of attack and an improved defense mechanism is proposed, as in [260], [151], [167], [256], [172], [229], [250] and [23]. This means that a possibly minor attack variation may severely affect the performance of a certain defense mechanism.

- The problem of optimality arises because attack and defense mechanisms are usually complex to model analytically. This means that the efficiency of most defense mechanisms is only evaluated empirically and hence, we do not know if a concrete defense (or attack) mechanism is optimal against a concrete attack (or defense) mechanism, nor we know how far from optimal performance is that mechanism. Note

Chapter	CSMA/CA	CSS	Player	Information	Observation (A/S)	Behavior
4	Yes	No	Attack Defense	Complete Complete	Mixed / - Mixed / -	Static Static
5	Yes	Yes	Attack Defense	Complete Incomplete	- / State Realization / -	Dynamic Static
6	Yes	Yes	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / -	Dynamic Static
7	Yes	No	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / Observation	Dynamic Dynamic

Table 6.1 Table comparing the different setups used in Chapters 4-7. CSMA/CA. i.e., the backoff attack, and CSS, i.e., the SSDF attack, denote whether each of these setups is used in the Chapter. Information denotes whether each player knows the target of the other player (Complete) or not (Incomplete). Observation refers to what each agent observes with respect to the actions / states of the other players: regarding actions, they observe the mixed actions or the actions realizations, and regarding states, they observe the state or an observation of the rest of players: this is related to having perfect or imperfect information. Behavior refers to whether the player adapts its behavior with time or not.

that this problem is closely related to the ad hoc defense problem: since we do not know which one is the optimal attack against a certain defense mechanism, i.e., the attack that harms the most such defense mechanism, we do not know how the defense mechanism performs against a variation of the attack. While in Chapter 5 we have been able to derive the optimal attack against an SPRT defense mechanism, it is not frequent having theoretical results in defense mechanism works.

In this Chapter, we show that these two problems can be used by intelligent and dynamic attackers to exploit unknown defense mechanisms. Along this Chapter, we again use the MDP framework to model several attack mechanisms in WSN. An MDP can be learned by an attacker using RL tools, and hence, they may take advantage of the ad hoc defense problem to exploit a possibly unknown defense mechanism simply by interacting with it. Currently, RL tools are used in WSN several problems, such as routing, data latency, path determination, duty cycle management, QoS provisioning or resource management [9]. The problem of WSN security also takes advantage of the recent advances in Deep Learning [202], [244]. The idea of applying RL to cyber security is not new [38], [86]; to mention some examples, Deep RL tools are used in WSN security to detect spoofing attacks [243], for mobile offloading [245], [242], to avoid jamming [12], [87] and to model Denial-of-Service attacks [134]. As noted, the advances in Deep RL potentially open the door to designing an attacker which learns to exploit a possibly unknown defense mechanism. Thus, the question is, in a WSN in which there are several GSs and one or more ASs, how do current defense mechanisms perform against RL based attackers? Can we learn an attacker using RL tools that exploits unknown defense mechanisms simply by interacting with them? As we show in this Chapter, the answer to the latter is affirmative.

In this Chapter, we make use of three environments. The first one is a hard fusion SSDF attack against a CSS WSN, in which we use as defense mechanism EWSZOT, which was introduced in Chapter 5. The second one is a soft fusion SSDF attack against a CSS WSN: in this case, we assume that the report of each sensor is the energy level thy sense, and hence, the reports are not a binary variable as in the hard fusion case, but a continuous value. Note that this problem presents a larger dimensionality from the point of view of the attacker, as it does not have to choose between two values, but among many possible energy values. And finally, we also include a partial observation backoff environment: we now assume that the defense mechanism only observes the backoff periods of each sensors, and has to make a decision on whether a concrete sensor is following the

binary exponential backoff procedure or it is deviating. We use two statistical tests as defense mechanisms against the soft fusion SSDF attack and the partial observation backoff attack, which are described in Section 6.2.

Then, Section 6.3 is devoted to thoroughly modeling the hard fusion SSDF attack using MDP tools. Note that an MDP model of the backoff attack is presented in [28], hence, we focus on the SSDF attack case, which is not solved yet. Also, note that we choose to model the hard fusion case because the reports, which are related to the actions of the agents, are discrete, which fits the MDP framework presented in Chapter 2. The MDP model obtained can be used to evaluate the theoretical performance of a given attack policy or to obtain the optimal attack strategy. However, the main problem that this approach faces is that the probability transition function may not be possible to obtain analytically if the problem has a large dimensionality, even if we have complete information.

In order to address this situation, in Section 6.4 we focus on the soft fusion SSDF attack and the partially observable backoff attacks, which present a higher dimensionality than the hard fusion SSDF attack solved in Section 6.3. Section 6.4 expands the previous one in several points. First, we do not assume that the state is observable for the ASs: they will only have partial information of the defense mechanism, which is a more realistic setting. Second, we now study two different defense mechanisms: the soft fusion SSDF attack, in which the action space is continuous, and the partially observable backoff attack, in which the action space is discrete, and we propose a Deep RL Attacker (DLA) which is able to work with both types of action spaces, thus, giving it more flexibility. Third, we focus on the case in which there are more than one AS, enabling the ASs to communicate their observations to other ASs in order to better exploit the defense mechanism. Since all the ASs have a common goal, which is to exploit the defense mechanism of the WSN, the ASs are a swarm and hence, we use the swarMDP model [211] presented in Chapter 2.

Then, Section 6.5 presents several simulations which show the potential of the work of this Chapter. First, we validate our MDP model of the hard fusion SSDF attack and obtain optimal strategies. Note that we are able to obtain analytical results with respect to the performance of the attack in this environment because it is a low dimensional problem, that is tractable using DP tools: it is frequent that research on WSN defense mechanism only provides an empirical validation of the approach. One exception is [250], where the effects of the covert adaptive data injection attack is evaluated on a distributed consensus-based CSS network. The efficiency and efficacy of their proposed defense mechanism is assessed both analytically and using simulations. However, we focus on a centralized data fusion scheme. Another exception is found in [115], where the performance of a centralized CSS scheme under an SSDF attack is evaluated. However, they do not make use of a CSS defense mechanism, which significantly simplifies their analysis: our approach allows taking into account defense mechanisms. We also use RL tools to learn the attack strategies and show that they provide quasi-optimal results in the hard fusion problem. And finally, we also test our DLA architecture on the two high-dimensional problems, the soft fusion SSDF attack and the partially observable backoff attack.

Hence, with respect to Chapter 5, we keep on studying intelligent attacks under a control perspective, but in this Chapter we drop the assumption that the attacker knows the defense mechanism parameters: rather, it learns by interacting with it. We also drop the assumption that the attacker observes the state, and now it has access to a partial observation: we move from an attacker with perfect and complete information to another with imperfect and incomplete information. This Chapter shows that the attack strategy that we propose, based on RL tools, is a significant threat against current WSN defense mechanisms.

6.2 Defense mechanisms

In this Chapter, we again continue with the two main attack environments of this work: the SSDF attack in a CSS WSN, and the backof attack when using CSMA/CA in the MAC layer of a WSN. As we already mentioned, we start working with EWSZOT, which has been described in Chapter 5. We obtain a theoretical model for EWSZOT, which will allow us to show (1) that such models are hard to obtain and (2) that RL tools can exploit an unknown defense mechanism. Then, we move on to develop an advanced Deep RL based attacker, which we test on a soft fusion SSDF attack and a backoff attack. We introduce important novelties in these attacks:

- Regarding the SSDF attack, the first part of this Chapter focuses on hard fusion using EWSZOT, as in Chapter 5. However, for the second part of the Chapter, we move on to a soft fusion problem, because (1) we will already have proposed an optimal EWSZOT attack, thus EWSZOT could be considered solved, and (2) the soft fusion problem has a significantly larger dimensionality, as the reports now are the energy measured by the sensors, which is a continuous variable, instead of a binary decision on the channel state as in the hard fusion case.
- Regarding the backoff attack, we already noted in Chapter 5 that in real life environments a defense mechanism does not know instantaneously whether a sensor respects the binary backoff procedure or not. Rather, the defense mechanism can apply a statistical test to the observed backoff periods. In this Chapter, we move to this situation, which is more realistic as the defense mechanism only observes the backoff periods of each sensor and then tries to decide whether each sensor respect the binary exponential backoff or not. Note, also, that we do not assume that the ASs follow a uniform backoff, as they learn an optimal attack policy against the defense mechanism.

6.2.1 Soft fusion SSDF Attack

As we have already explained, the SSDF attack is addressed against a CSS WSN in which several sensors send a report on the channel state to a central FC. Several defense mechanisms have been proposed to deal with SSDF attacks, depending on whether the sensor sends only their decision to the FC, known as hard fusion, or the sensors also include additional information about the certainty of their decisions, known as soft fusion. Some defense mechanisms for the hard fusion case are Weighted Sequential Probability Ratio Test (WSPRT) and EWSZOT [260] and for the soft fusion case, a statistical test based on the energy level distribution [229] and Enhanced WSPRT (EWSPRT) [260]. Many challenges remain open in this field, as [257] shows.

Let us now focus on the soft fusion case, and let us consider that the report sent by the sensor m to the FC is the energy level E_m that it senses. As shown by [221], if the channel is idle, i.e., there is only noise in the channel, E_m follows a chi-square distribution, whereas if a signal is present, E_m follows a non-central chi-square distribution. Thus, it is possible to use an HT to make a decision, where H_0 means that the channel is idle, and H_1 means that the channel is busy:

$$E_m \sim \begin{cases} \chi^2_{2k} & \text{if} \quad H_0 \\ \chi^2_{2k}(2SNR_m) & \text{if} \quad H_1 \end{cases}, \tag{6.1}$$

where k is the time-bandwidth product and SNR_m is the signal-to-noise ratio in sensor m. An illustration of these probability distribution functions can be found in Figure 6.1.

In [229], a soft fusion SSDF attack is proposed, which consists in reporting honestly E_m if $E_m > \xi$ and $E_m + \Delta$ if $E_m \leq \xi$, where Δ and ξ are attack parameters: Δ is the bias in the energy level introduced by the AS

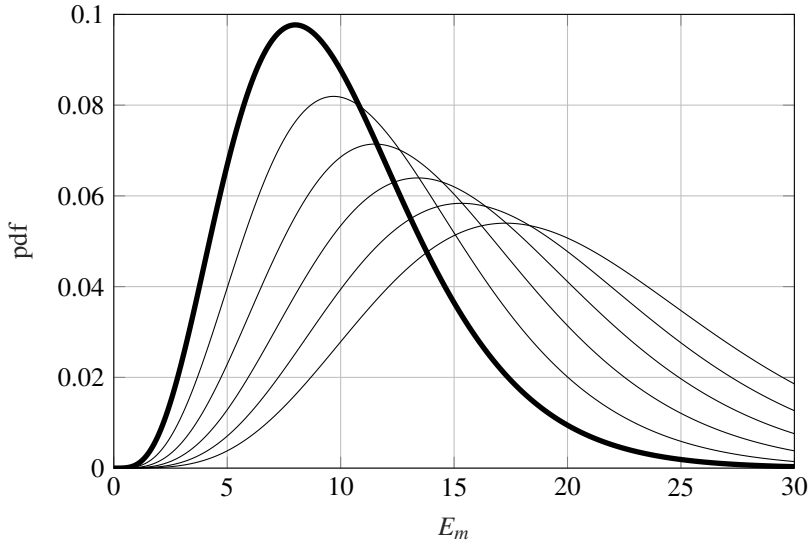


Fig. 6.1 Illustration of the probability distribution function (pdfs) of the chi-squared distributions from (6.1). The thick pdf corresponds to the H_0 case: the chi-squared χ^2_{2k} distribution; and the thinner pdfs correspond to the H_1 case: the non-central chi-squared $\chi^2_{2k}(2SNR_m)$ distributions for SNR values $\{2,4,6,8,10\}$, from left to right in the plot. For all curves, $k = 5$ is the time-bandwidth parameter. Observe that, as the SNR increases, the pdf curves are more separated for H_0 and H_1.

and ξ is the attack threshold. In other words, the AS reports that the channel is busy when it is actually idle when a certain threshold in the energy level is satisfied. We set Δ by using the means of the distributions (6.1). The mean values μ of the distributions under H_0 and H_1 are:

$$\mu_0 = 2k, \qquad \mu_1 = 2k + 2SNR_m. \tag{6.2}$$

We set $\Delta = \mu_1 - \mu_0 = 2SNR_m$, resulting in a defense mechanism that is tuned to detect a bias that tries to simulate the E_m values when the channel is busy. Graphically, according to Figure 6.1, this Δ value translates the H_0 pdf to the right Δ units. Depending on the value of ξ, E_m values that are actually produced under H_1 are also translated and some E_m measurements produced under H_0 are not. Note that if the probability distribution functions under H_0 and H_1 are close and overlap significantly, i.e., SNR_m is low for sensor m, attacking will be unnecessary in many cases.

We denote by G_0 the situation in which an AS does not attack, and G_1 when it attacks. Under attack, the pdf from (6.1) can be written as:

$$E_m \sim \begin{cases} \chi^2_{2k} & \text{if} \quad G_0, H_0 \\ \chi^2_{2k}(2SNR_m) & \text{if} \quad G_0, H_1 \\ \chi^2_{2k} + \Delta & \text{if} \quad G_1, H_0 \\ \chi^2_{2k}(2SNR_m) & \text{if} \quad G_1, H_1 \end{cases}, \tag{6.3}$$

where we approximate the situation G_1, H_0 by a translation of the chi-squared pdf when G_0, H_0. The accuracy of the approximation depends on the threshold ξ value. Also, observe that (6.3) assumes that under H_1 hypothesis there is no attack: again, this assumption is an approximation that depends on ξ.

Against this attack it is possible to use the defense mechanism proposed in [229], which is based on two Neyman-Pearson tests. The first test decides whether sensor m senses a busy channel using reports from other sensors:

$$\prod_{i=1,i\neq m}^{M} \frac{P(E_i = e_i|H_1)}{P(E_i = e_i|H_0)} \underset{H_0}{\overset{H_1}{\gtrless}} \eta \rightarrow \prod_{i=1,i\neq m}^{M} \frac{P(E_i = e_i|\chi^2_{2k}(2SNR_m))}{P(E_i = e_i|\chi^2_{2k})} \underset{H_0}{\overset{H_1}{\gtrless}} \eta, \tag{6.4}$$

where H_1 and H_0 are the energies from (6.1) and η is the threshold of test 1.

The second test is used to individually detect which sensors are providing false reports, i.e., the sensors that are ASs. This test is only used for sensor m if H_0 was the result of test 1, i.e., only noise detected, using the expressions from (6.3) as:

$$\frac{P(E_m = e_m|G_1,H_0)}{P(E_m = e_m|G_0,H_0)} \underset{G_0}{\overset{G_1}{\gtrless}} \zeta \rightarrow \frac{P(E_m = e_m - \Delta|\chi^2_{2k})}{P(E_m = e_m|\chi^2_{2k})} \underset{G_0}{\overset{G_1}{\gtrless}} \zeta, \tag{6.5}$$

where ζ is the threshold of test 2.

Test 2 allows detecting whether a sensor m is attacking or not. The defense mechanism keeps a reputation scheme, in which there are two vectors of size m: r and s. The m-entry of each vector keeps track of how many times sensor m has attacked (s) and how many times sensor m has not attacked (r). The reputation of each sensor t_{PHY} is computed as:

$$t_{PHY} = \frac{r+1}{r+s+2}, \tag{6.6}$$

where, if t_{PHY} falls below a certain threshold λ_{PHY}, the sensor is considered to be an AS and it is banned from the network.

6.2.2 Partially observable backoff attack

We have already introduced the backoff attack in Chapter 4: it is an attack that affects to the MAC layer of any protocol that uses CSMA/CA mechanism, such as IEEE 802.11, [111] and most WSN proposed MAC protocols [57], [249]. If the attack is successful, the ASs reach a higher share of the network throughput by not respecting the backoff procedure.

We now focus on the realistic situation in which the defense mechanism has only access to the past backoff periods used by each sensor: thus, note that we are in a partial observation setup. In order to detect whether a sensor is following the binary exponential backoff procedure or not, we use a modified Cramer-von Mises statistical test [10] as in [229]. This test is fast to compute and allows deciding whether a stream of data is adjusting to a certain distribution, using the cumulative distribution function (CDF). We denote by x_m the observed backoff time, which is the estimated backoff window size that sensor m has used and is observed by the FC: as shown in [229], it can be estimated using the values in (4.10), where the value of each parameter is in Table 4.2.

In order to model the real distribution of the window backoff we follow the binary exponential procedure which was already desribed in Chapter 4 following [111]. If we consider that p_c is the collision probability, n_c is the number of collisions and $U[a,b]$ is the random integer uniform distribution between a and b; the distribution of the window backoff if there is no attack under following the binary exponential backoff procedure, $f_0(x_m)$, is

$$f_0(x_m) = \begin{cases} \sum_{j=0}^{n_c} U[0,2^{5+j}] & \text{w.p.} \quad p_c^{n_c}(1-p_c) \quad \text{if} \quad n_c \leq 5 \\ \sum_{j=0}^{5} U[0,2^{5+j}] + \sum_{j=6}^{n_c} U[0,2^{10}] & \text{w.p.} \quad p_c^{n_c}(1-p_c) \quad \text{if} \quad n_c > 5 \end{cases}, \tag{6.7}$$

where w.p. stands for with probability. The collision probability p_c can be estimated by the FC by counting the number of successful transmissions and the number of collisions, and computing the proportion of collisions. Using (6.7) and the estimated p_c, we can obtain $F_0(x_m)$, the cumulative distribution of $f_0(x_m)$.

As in [229], we need K observations $x_1, x_2, ..., x_K$ from sensor m, which are used to obtain F_1, the empirical CDF of the window size from sensor m. The test also requires L samples $y_1, y_2, ..., y_L$ generated from the real distribution when there is no attack, $f_0(x_m)$. The test statistic θ is obtained as

$$\begin{aligned}\theta_1 &= \sum_{j=1}^{K} \text{sgn}(F_0(x_m) - F_1(x_m))[F_0(x_m) - F_1(x_m)]^2, \\ \theta_2 &= \sum_{j=1}^{L} \text{sgn}(F_0(y_m) - F_1(y_m))[F_0(y_m) - F_1(y_m)]^2, \\ \theta &= \frac{KL}{(K+L)^2}(\theta_1 + \theta_2),\end{aligned} \tag{6.8}$$

where $\text{sgn}(x)$ is the sign function. The value of θ gives a measurement of differences between the two CDFs. The magnitude of θ depends on the difference between the cumulative distributions, i.e., if F_0 and F_1 differ significantly, $|\theta|$ will be large. Also, observe that the sign information is crucial to determine whether F_1 is above or below F_0, a positive θ indicates that F_0 is mostly above F_1, which means that the backoff window values for sensor m are larger than expected, and hence, sensor i is not doing a backoff attack. The opposite happens when θ sign is negative, it indicates that F_0 is mostly under F_1, which means that the observed values of backoff window for the sensor m are smaller than expected and hence, a backoff attack is being detected. Hence, the reputation in the MAC layer of sensor m is determined as follows

$$t_{MAC} = e^{-D^2}, \quad D = \min\{\theta, 0\}. \tag{6.9}$$

In (6.9), t_{MAC} will be 1, i.e., sensor m is completely trusted, when θ is positive, which is the case in which there is no attack. As θ becomes negative, t_{MAC} decreases, indicating that sensor m is less trusted because it might be performing a backoff attack. If t_{MAC} falls below a certain threshold λ_{MAC}, the sensor m is considered to be an AS and it is banned from the network.

6.3 A low dimensional problem: Hard fusion CSS

In this Section, we proceed to illustrate the advantages of the MDP framework. For this reason, we thoroughly study an SSDF attack in a hard fusion CSS WSN. We consider again that we have a WSN with I sensors: n_1 is the number of GSs and n_2 the number of ASs, where $I = n_1 + n_2$. The WSN, again, uses EWSZOT as defense mechanism. EWSZOT is a hard fusion mechanism, thus each report from the sensors to the FC is denoted by the binary variable u_m, where $u = 1$ means that the channel is busy, $u = 0$ means that the channel is idle and $m \in \{1, 2, ..., I\}$ indexes the sensors. Again, each sensor can make a wrong sensing decision with probability P_c, where we consider P_c to be constant and independent among the sensors. Thus, each u_m follows a Bernoulli distribution of parameter P_c if $u = 0$ and $1 - P_c$ if $u = 1$. The optimality criterion we use is the total probability of error $p_{e,t}$ in the FC, that is, the probability that the FC decides that the channel is busy when it is idle and vice-versa. Thus, the ASs will try to maximize $p_{e,t}$. Since EWSZOT is a defense scheme based on reputations, the ASs need to attack and also keep a sufficiently high reputation. In other words, maximizing $p_{e,t}$ will require the ASs to camouflage.

In order to facilitate the mathematical tractability of this Section, we assume that the attacker has perfect information, i.e., that it can perfectly observe the state of the defense mechanism. Also, note that the modeling of the defense mechanism is done assuming complete information, and the reason of doing this is to show that RL tools, which are designed for an incomplete information setting, are able to obtain very good attack policies, quasi-optimal compared to the case of having complete information. Thus, this Section presents an attacker able to deal with situations of incomplete information.

Attacks against EWSZOT

We introduce the three attack strategies that we explore:

- Naive strategies. These strategies consists in always doing a predefined action. Even though they are basic attacks, they are widely used. In [260], it is shown that EWSZOT is successful against the three naive attacks introduced in Chapter 5: report the channel always busy, always idle or always give a false report. To the best of our knowledge, these are the current attack strategies against EWSZOT. Note that the problem of optimality arises here: no study about the optimality of these attacks is found in [260].
- Optimal strategies. These strategies are the result of modeling the defense mechanism and optimizing. In our framework, it means that the strategy is obtained by optimizing an MDP. They are theoretically optimal, but also may be complex to obtain. Note that these strategies may exploit the problem of ad hoc defense: as we will see, minor changes in the attack may lead to a dramatic performance drop in EWSZOT defense mechanism.
- RL strategies. These strategies are obtained using RL tools, and present an interesting trade-off: they are not as complex to obtain as optimal strategies and provide quasi-optimal results. An significant point of this work is that we compare these strategies to the theoretically optimal ones.

We further illustrate the problem of ad hoc defense by noting that EWSZOT does not take into account the possibility of an attack addressed to the communication mechanism between the sensors and the FC. We define two different attack situations: a standard SSDF attack (SA), consisting in sending false reports to the FC. Note that SA is the always false attack proposed in [260], which will serve as baseline to compare our results with.

The second attack situation we consider is a novel, Combined Attack (CoA), consisting on a SSDF attack and also, a jamming attack addressed to the communications link. When the FC asks for reports to the sensors, these reports may not arrive on time or arrive corrupted, due to channel problems, such as shadowing or fading. But the ASs could also jam the communication link to cause the same effect on the reports sent by GSs. A survey on different jamming techniques can be found in [156]. For instance, if a narrowband communication scheme is used, an AS can transmit noise in order to cause a high interference that makes the communication between the sensor and the FC impossible. Or if a CSMA/CA access scheme is used, an AS can jam the communication between a sensor and the FC by simply sending bursts of noise in the backoff periods [194]. In order to keep our scheme as simple and broad as possible, we consider that the group of ASs can jam up to M_j GSs. When the FC does not obtain a report from sensor m, considers that $u_m = 1$, to be consistent with the test truncation used in EWSZOT HT. Observe that in our current framework, we consider for simplicity that all corrupted reports come from jamming, but jamming causes actually only part of them.

6.3.1 Modeling EWSZOT using an MDP

We now describe in detail how the EWSZOT mechanism already presented can be described using the MDP framework presented in Chapter 2. We note that, in order to preserve the MDP notation introduced in Chapter 2, we introduce a change in the notation of EWSZOT with respect to Chapter 5: now, n indexes the EWSZOT stages, whereas in Chapter 5, we used k for that purpose. Thus, note that the truncation value EWSZOT is now M, while N is reserved for the maximum number of times that EWSZOT is called. Note that this change in notation is caused because, while in Chapter 5 we focused on attacking a single sequential test procedure, in this Chapter we focus on attacking the whole EWSZOT procedure.

States definition

We use as state the tuple formed by the reputation vectors of good and attacking sensors, z_g and z_a respectively; and the number of sensors already jammed in case of CoA, m_j. Thus, in stage n, the state is the tuple $s^n = \langle z_g^n, z_a^n, m_j^n \rangle$. The initial state s^0 will be always a vector in which all reputations are set to 0 because that is the initialization value of EWSZOT and $m_j^0 = 0$ because there is no sensor jammed yet.

Actions definition

The definition of the action space A is subtle. The possible actions of the ASs in case of CoA depend on the state due to the jamming. If we are in state s^n that the ASs have already jammed m_j^n sensors, where $m_j^n \leq M_j$, and thus they can only jam up to $M_j - m_j^n$ sensors more. Thus, observe that the action set depends on the state. Note also that if several sensors have the same reputation, the ASs do not know in advance which sensor will be called by the FC. In order to overcome these problems, we define two vector of actions, a_g and a_a. The first is a vector of length M which contains the actions for the case that GSs are called. Each entry of this vector can have two values: 1 if there is jamming and 0 if there is no jamming. Observe that $\sum a_g \leq M_j - m_j^n$, that is, there is a limit on the maximum number of GSs that could be jammed. We limit the length of this vector to M since this is the maximum number of sensors called by the FC: in the extreme case that all sensors called are good, only actions from this vector are used.

The second vector, a_a, is a vector of length $\min\{M, n_2\}$, which contains the actions for the case that ASs are called. Each entry of this vector can have two values: 1 if the sensor gives a false report, i.e., attack, or 0 if the sensor gives a true report. Note that the limit in length responds to the case in which $n_2 < M$, that is, there are less ASs than sensors that can be called in each stage. The action space A is formed by all possible combination of action vectors $a = \langle a_g a_a \rangle \in A$. The dimensionality of A is upper bounded by $2^{2M} = 4^M$, which is the maximum number of actions available to the ASs.

In case of SA, there is no possibility of jamming. Thus $a = a_a$, and hence, the dimensionality of A is upper bounded by 2^M. Note that in SA, the set of actions does not depend on the state.

Transition probabilities definition

Given a state s and an action a, now we turn to obtain $P(s^{n+1}|s^n = s, a^n = a)$, that is, the probability of transitioning from state s^n to state s^{n+1} due to action a^n. This requires to model the HT from the EWSZOT mechanism, which we illustrate in Figure 6.2. We model the HT in stage n using a tree. Each node of the tree represents the possible sequence *seq* of reports that the FC receives from the sensors. Hence, each parent node will have four children nodes, because it can receive as report $u_m = 0$ from a GS (0_g) or from an AS (0_a), $u_m = 1$ from a GS (1_g) or from an AS (1_a). The maximum length of the sequence *seq*, and hence, the maximum

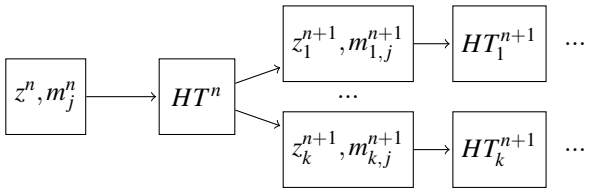


Fig. 6.2 EWSZOT algorithm modeling illustration. Each HT receives as input a reputation vector and a number of jammed sensors and produces a certain number k of updated reputation vectors and jammed sensors. These vectors are used as inputs to new tests in next stages. Each HT has as many k outputs as leaves. Each HT procedure is found using Algorithm 15.

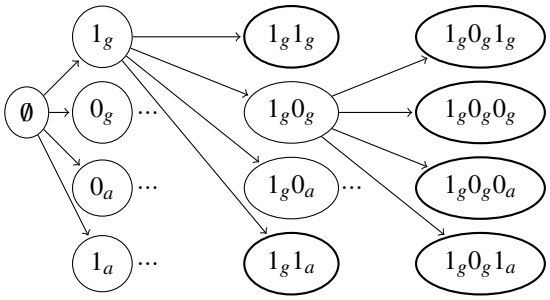


Fig. 6.3 Illustration of EWSZOT HT tree. Each node contains the sequence of reports. For simplicity, we plot part of the tree when $M = 3$. Leaves are the thicker nodes. Observe that the leaves may happen when any of the final conditions from (5.14) is satisfied.

depth of the tree, will be M. Also, observe that a sequence *seq* that satisfies any of the final conditions from (5.14) becomes a leaf: it will have no children nodes. An illustration is shown in Figure 6.3.

Each node of the tree will store the following data: the sequence of reports received *seq*, the probability of the sequence p_s, the updated number of sensors jammed m_j^{n+1}, where $m_j^{n+1} \leq M_j$, and the W^n value. We require as inputs z_g^n, the reputations of the GSs; z_a^n, the reputations of the ASs; the number of sensors already jammed m_j^n; $p_{1,g}$ ($p_{1,a}$), the probability that a GS (AS) reports $u_m = 1$ (observe that these values depend on P_c), the maximum number of sensors that can be jammed M_j; and the parameters g, M and q. Note that the first three parameters are the state: $s^n = \langle z_g^n, z_a^n, m_j^n \rangle$.

We then obtain all the nodes and the data in each node as follows. First, we order the sensors by reputations using z_a^n and z_g^n and obtain $p_g(z)$: the proportion of GSs among all sensors with reputation value z. Then, we proceed to build the tree. We initialize the root node with $seq = \emptyset$, $p_s = 1$, $m_j^{n+1} = m_j^n$ and $W^n = 0$. Then, we call sensors in descending order of reputations z. We build four children nodes for each parent, each children with a sequence which is the concatenation of the sequence of the parent node and each of the reports that can be obtained, i.e., 1_g, 1_a, 0_g or 0_a.

The updating procedure of p_s is detailed in Algorithm 14. We first need to obtain *na* and *ng*, the number of good and attacking sensors already called. This is done by simply looking at the *seq* vector, which stores the sequence of reports. Then, if the report we have obtained comes from a GS, we update p_s using $p_{1,g}$ and $p_g(z)$, and then check the action for the $ng+1$ GS: if it indicates jamming, we change the report value to $rep = 1_g$ and update the number of sensors jammed, m_j^{n+1}. We first update p_s in order to obtain the total probability of the sequence *seq*; if there is jamming, the report is changed to 1 for the update of the HT statistic W^n. If the report came from an AS, and the action for the $na+1$ attacking sensor indicates to attack, we update p_s using $p_{1,m}$ and $p_g(z)$. If the report came from an AS but the $na+1$ action from a_a indicates not to attack, then

Algorithm 14 p_s updating procedure

Input: seq, $a = \langle a_g, a_a \rangle$, m_j^{n+1}, $report$, W^n, $p_{1,m}$, $p_{1,g}$
Obtain na, number of ASs already called, from seq
Obtain ng, number of GSs already called, from seq
if $report$ comes from a GS **then**
  if $report = 1_g$ **then**
    Update $p_s = p_s p_{1,g} p_g(z)$
  else if $report = 0_g$ **then**
    Update $p_s = p_s(1 - p_{1,g}) p_g(z)$
  if The action indicates jamming: $a_g(n_g) = 1$ **then**
    Set $report = 1_g$
    Update $m_j^{n+1} = m_j^{n+1} + 1$
else if $report$ comes from an AS **then**
  if The action indicates attack: $a_a(n_a) = 1$ **then**
    if $report = 1_a$ **then**
      Update $p_s = p_s p_{1,m}(1 - p_g(z))$
    else if $report = 0_a$ **then**
      Update $p_s = p_s(1 - p_{1,m})(1 - p_g(z))$
  else if The action indicates no attack: $a_a(n_a) = 0$ **then**
    if $report = 1_a$ **then**
      Update $p_s = p_s p_{1,g}(1 - p_g(z))$
    else if $report = 0_a$ **then**
      Update $p_s = p_s(1 - p_{1,g})(1 - p_g(z))$
Update $seq = \{seq, report\}$
Update $W^n = W^n + (-1)^{report+1} w_i^n$ (equation (5.15))
Output: p_s, m_j^{n+1}, seq, W^n

we update this sensor using the error probabilities of a GS ($p_{1,g}$ instead of $p_{1,a}$). Finally, we update the HT statistic W^n using (5.15). After a report from sensor i has been received and p_s and W^n have been updated using Algorithm 14, we check whether the node satisfies any of the final conditions from (5.14). If it does, then the node becomes a leaf, otherwise, we set this node as father and repeat the procedure.

Finally, we obtain the information from the leaves. First, we obtain the probability that the test ends with result u_d^n simply by adding the p_s of the leaves that satisfy each of the decision conditions from (5.14). This leads us to obtain $p_{t,1}$, which is the probability that the test ends because $W^n \geq q$; $p_{t,0}$, which is the probability that the test ends because $W^n \leq -q$; $p_{t,nd}$, which is the probability that the test ends because $m = M$; and m_j^{n+1}, the updated number of sensors jammed. Second, we update the reputations using (5.13). Observe that there will be as many updated reputation vectors as leaves. Also observe that the probability of each of these reputations vectors is precisely p_s of the leaf. The whole procedure is summarized in Algorithm 15. Note that Algorithm 15 models each iteration of the outer for loop from Algorithm 12.

Each parent node can have up to four children nodes, one per each possible report that can be received. Some nodes may have no children, i.e., the leaves, or less than four, e.g., all the ASs have already been called and hence $p_g(r) = 1$. In the worst case computationally, which is that all parents have four children, there will be up to 4^M possible s^{n+1} states in case of SA. In case of CoA, we must take into account that there are $M_j + 1$ possible values of m_j^{n+1}, and hence, in the worst case, there will be up to $4^M(M_j + 1)$ possible s^{n+1} states. We denote this value by k: $k \leq 4^M(M_j + 1)$ for CoA and $k \leq 4^M$ for SA.

Algorithm 15 EWSZOT HT modelling in stage n for the MDP.

Input: $s^n = \langle z_g^n, z_a^n, m_j^n \rangle$, $a = \langle a_g, a_a \rangle$, $p_{1,g}$, $p_{1,a}$, M, M_j, q, g
1: Obtain weights w using (5.16)
2: Select the M sensors with highest reputations
3: Initialize tree root: $p_s = 1$, $W^n = 0$, $m_j^{n+1} = m_j^n$, $seq = \emptyset$
4: **for** Each node which is not a leaf **do**
5: **for** Each four possible reports rep: 0_g, 1_g, 0_a, 1_a **do**
6: Create children node
7: Obtain $p_g(z)$ for the reputation of the current sensor
8: Update seq, p_s, m_j^{n+1} and W^n using Algorithm 14
9: **if** $W^n \geq q$ or $W^n \leq -q$ or length of seq is M **then**
10: Make decision u_d^n using (5.14)
11: Make this node a leaf
12: Initialize $p_{t,0} = p_{t,1} = p_{t,nd} = 0$
13: **for** Each leaf **do**
14: Update $p_{t,0}$, $p_{t,1}$ or $p_{t,nd}$
15: Update z_g^{n+1} and z_a^{n+1} using (5.13)
Output: $s^{n+1} = \langle z_g^{n+1}, z_a^{n+1}, m_j^{n+1} \rangle$, $p_{t,0}$, $p_{t,1}$, $p_{t,nd}$, p_s

Using Algorithm 15 to model the HT allows obtaining the exact values of $P(s^{n+1}|s^n, a^n)$: for each possible combination of action a^n and state s^n, we can obtain the probability p_s of transitioning to state s^{n+1}. Note that the Markovian property is satisfied: each HT test output depends only on its input.

Reward definition

We can also obtain the expected reward $r(s^n, a^n)$ at the same time as we obtain $P(s^{n+1}|s^n, a^n)$ using the output of Algorithm 15 as:

$$r(s^n, a^n) = \begin{cases} \frac{1}{N} \sum_{s^{n+1} \in S^{n+1}} p_s (p_{t,1} + p_{t,nd}) & \text{if} \quad u^n = 0 \\ \frac{1}{N} \sum_{s^{n+1} \in S^{n+1}} p_s p_{t,0} & \text{if} \quad u^n = 1 \end{cases}, \tag{6.10}$$

where the reward is the expected error probability conditioned to being in state s^n and taking action a^n. For our problem, we consider that the total reward is:

$$r = p_{e,t} = \sum_{n \in N} r(s^n, a^n). \tag{6.11}$$

Note that EWSZOT is a truncated test as Algorithm 12 shows: in real life, sequential HTs are truncated to avoid lockouts. This means that our problem is of finite horizon, i.e., $N < \infty$, and hence, in (6.11), we consider that $\gamma = 1$. As our problem is of finite horizon, there are two important consequences. The first one is that the optimal policy will be non-stationary, as mentioned in Chapter 2. This means that the optimal policy in state s^n may differ from the optimal policy in the same state in a different n. However, the states in our setup are related to the reputations and in our simulations, we observed that states appearing more than once in the same simulation was rare. Thus, we could make use of stationary policies. This is related to the second consequence: RL is used to learn stationary policies, thus, we can use RL algorithms without further modifications. Also, as we already mentioned in Chapter 2, many problems are approximated using infinite horizon even though they are of finite horizon.

Finally, it is important noting that we can obtain optimal attack strategies against EWSZOT using the DP algorithm from Lemma 1, in order to obtain the maximum error probability that the ASs can achieve and the

optimal policy that they must follow in order to achieve that optimal attack. Hence, modeling the problem as an MDP brings the significant advantage of being able to obtain optimal attacks.

6.3.2 EWSZOT model complexity

Obtaining an MDP model for our defense mechanism allows an attacker to derive an optimal attack by using tools presented in Chapter 2. However, it is possible that such optimal strategies are computationally expensive due to the model complexity, hence, we now proceed to evaluate the complexity of the MDP model already proposed. Let us assume that we want to evaluate a policy, that is, obtaining $V_\pi(s^0)$. For each HT, there are $k \leq 4^M(M_j+1)$ possible transitions to other states. An HT is performed for each state s^n as long as $n \in [0, N-1]$, and we always start from the same s^0. Thus, as we can observe in Figure 6.2, to evaluate a policy we must obtain a tree containing all possible states that can be transitioned from s^0 by following policy π. The depth of this tree is $N+1$ and the maximum number of states that the tree can have is bounded, in case of CoA, by:

$$\sum_{n=0}^{N} k^n = \sum_{n=0}^{N} \left(4^M(M_j+1)\right)^n. \tag{6.12}$$

And in case of SA, the upper bound is:

$$\sum_{n=0}^{N} k^n = \sum_{n=0}^{N} 4^{nM}. \tag{6.13}$$

Thus, the state space, though finite, can be very large. It is possible to speed up the computation by:

- After HT in stage n, we drop out all states s^{n+1} such that $p_s = 0$. That is, we delete all states with zero probability.
- After HT in stage n, we do state aggregation: we merge together all states s^{n+1} that are the same and add up their probabilities p_s.
- After each stage n, we truncate the set of states s^{n+1} by preserving only the T states s^{n+1} which have the highest probability p_s. This truncation can reduce significantly the computational cost by fixing a maximum number of states in each stage, but introduces an error in the results. Larger values of T yield a lower error and a higher computational cost.

The procedure to evaluate a concrete policy, with these three improvements, is summed up in Algorithm 16. Observe that we need to know the actual u in order to obtain the decision error values. The total decision error $p_{e,t}$ is $p_{e,t} = p_{t,1} + p_{t,nd}$ when $u = 0$ and $p_{e,t} = p_{t,0}$ when $u = 1$. Also, observe that Algorithm 16 models theoretically the EWSZOT algorithm described by Algorithm 12. And finally, observe that this algorithm allows obtaining the performance of EWSZOT when there is no attack by simply setting $M_j = 0$ and always using actions without attack as policy.

To obtain the optimal policy, we need to take into account that the number of actions available in each state is bounded by 4^M in case of CoA. This means that we would have, for each state s^n, 4^M possible actions that would cause to transition to a maximum of $k \leq 4^M(M_j+1)$ possible s^{n+1} states. In this case, the dimensionality of the state-action space is bounded, for CoA, by:

$$\sum_{n=0}^{N} \left(4^M k\right)^n = \sum_{n=0}^{N} 2^{4nM}(M_j+1)^n. \tag{6.14}$$

Algorithm 16 EWSZOT algorithm modelling

Input: $M, M_j, q, g, I, n_1, n_2, P_c, u, N, T, \pi$

Initialize $z_g^0 = 1$ and $z_a^0 = 1$ for all sensors and $m_j^0 = 0$
Initialize $s^0 = \langle r_g^0, r_a^0, m_j^0 \rangle$
Initialize $p_{t,0} = p_{t,1} = p_{t,nd} = 0$
Obtain $p_{1,g}$ and $p_{1,a}$ using P_c
for Iterations $n = 0 : N-1$ **do**
    Initialize $S^{n+1} = \emptyset$
    for Each tuple s^n **do**
        Simulate HT: use Algorithm 15) to update the set of S^{n+1} when policy π is used.
        Erase all s^{n+1} such that $p_s = 0$
        Merge all equal s^{n+1} (state aggregation)
        if $Dim(S^{n+1}) > T$ **then**
            Keep only the T tuples $s^{n+1} \in S^{n+1}$ with highest p_s
            Normalize the probabilities p_s of the tuples $s^{n+1} \in S^{n+1}$
    Update $p_{t,0}$, $p_{t,1}$, $p_{t,nd}$

Output: $p_{t,0}$, $p_{t,1}$, $p_{t,nd}$

And for SA, where the number of actions is bounded by 2^{M_m}, the state-action space is bounded by:

$$\sum_{n=0}^{N} \left(2^M k\right)^n = \sum_{n=0}^{N} \left(2^M 4^M\right)^n = \sum_{n=0}^{N} 2^{3nM}. \tag{6.15}$$

Observe that the dimensionality is a major problem when it comes to evaluating or searching for an optimal policy: as we noted in Chapter 2, this is known as the curse of dimensionality. That will be a major limitation regarding computations.

6.4 Two high dimensional problems: Soft fusion CSS and partial observation backoff attack

In the previous Section, we described how an intelligent attacker may model a low dimensionality problem using the MDP framework. As our simulations will show, this allows the agent obtaining optimal attack strategies against the defense mechanisms. But also, our simulations will show that RL procedures are successful against EWSZOT defense mechanism when the agents do not have complete information. However, our approach in the previous Section was limited to 1 AS, with perfect observability of the state, i.e., perfect information, and a reward function hat required knowing the actual channel state. Now, we drop all of these assumptions: we work using very simple rewards that do not require a very specific knowledge, we use partial observations, i.e., imperfect information, and more than 1 AS that can even communicate among them. We make use of RL procedures, as these are able to attack a defense mechanism when the attacker has incomplete information. We also change our problem setting: we leave the hard fusion EWSZOT environment and move to the soft fusion SSDF attack described in Section 6.2.1 because, after modeling EWSZOT using an MDP, the EWSZOT defense mechanism is solved from the point of view of the attacker. We also use the partial observation backoff situation described in Section 6.2.2. Note that both are problems with a higher dimensionality than the EWSZOT setup, and hence, will show better the capabilities of the attacker proposed in this Section. Note that in these cases, we can no longer use the MDP framework as with EWSZOT, since now we have partial observations. Instead,

we make use of the POMDP framework and the swarMDP model introduced in Chapter 2: the former allows dealing with partial observations and the latter with having several ASs with a common objective, which is to successfully attack a defense mechanism. We also deal with continuous and discrete actions: thus, in this Section, we use TRPO as Deep RL algorithm. Hence, note that we introduce in this Section an attacker able to deal with imperfect and incomplete information settings.

An important difference with the previous Section is that now the problem is not solved analytically. Note that modeling the EWSZOT mechanism required a significant work in order to obtain the probability transition functions, it yielded a problem whose dimensionality grew fast and it also required complete information of the defense mechanism. Hence, that means that now we do not have optimal strategies to compare with. However, the DLA architecture we propose is able to exploit both defense mechanisms presented in Section 6.2, and potentially, any other defense mechanism that could be described using the POMDP / swarMDP frameworks, as our simulations will show.

6.4.1 Deep Reinforcement Learning Attacker architecture

We start our discussion of the DLA architecture by relating the attack description from Section 6.2 with the swarm model presented in Chapter 2. Let us assume that we have a swarm of n_2 ASs trying to attack a WSN with n_1 GSs, hence, there are again $I = n_1 + n_2$ sensors. At each time step n, the ith AS has an observation o_i^n and it uses $\pi_\theta(o_i^n)$ to select its action a_i^n. After action is executed, the ith AS receives the common reward r^n and the next observation, o_i^{n+1}. Under this scheme, we have for the SSDF attack:

- A continuous set of actions in the range $[0,1]$. The action indicates the normalized energy that the sensor reports to the FC. Note that the action is continuous: this both means that the problem dimensionality is significantly larger than in the hard fusion case, and also, that we can no longer use DQN/DRQN: rather, we use TRPO, as it is able to deal with continuous actions.

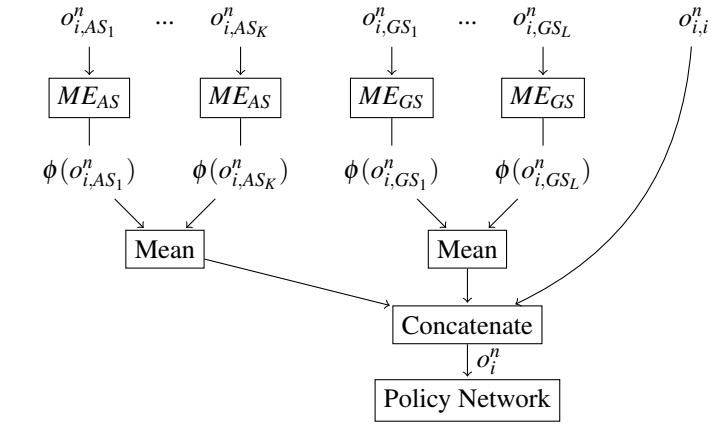
- The reward to each AS is $+1$ if the FC decides that there is a primary transmitting, whereas the reward to each AS is 0 if the FC decides that there is no primary transmitting. We use a maximum number of time steps for each episode and if all ASs are discovered we terminate the episode. The DLA must therefore learn to maximize the number of time steps without being discovered in order to maximize its reward.

- In order to build the observations vector, each agent stores its last 5 actions and a flag indicating whether the agent has been discovered by the defense mechanism or not. We assume that agent i can also access the observations of other sensors $j \neq i$, i.e., ASs can communicate their observations. We choose 5 actions as a tradeoff between keeping the observation dimensionality not excessively large and having enough information to exploit the system, as our simulations will show.

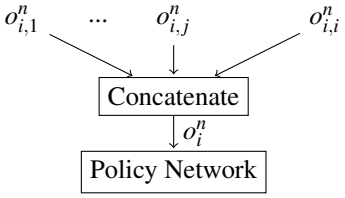
And for the backoff attack, we have:

- Two discrete actions which indicate whether the sensor starts transmitting or not in the current time step.

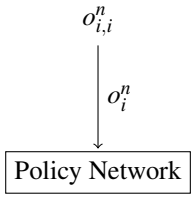
- The reward is -1 in case that a GS starts transmitting, and 0 otherwise. Our choice of the rewards is different as the attacks have different targets: in case of the SSDF attack, we want to detect the primary as often as possible. However, in the backoff attack, we want that GSs transmit as little as possible. We set a fixed simulation time, which is completed regardless of whether the AS are discovered. Thus, the DLA needs to learn not to be discovered and, at the same time, preventing that the GSs transmit.



(a) Mean embedding



(b) Communication without mean embedding



(c) No communication

Fig. 6.4 Sketch of the different DLA architectures. The difference in the architectures lies in how the observation o^n_i is obtained. In (a) and (b), there is communication among the swarm agents and hence, each agent i has access to the local observations of the rest of the agents. (a) shows the architecture when a Mean Embedding is used: note that we use separate Mean Embeddings for ASs and GSs, we assume that there are $K+1$ ASs (agent i is also an AS), L GSs, and o^n_i is the concatenation of the mean values of the Mean Embeddings and the local information of the agent i. (b) shows the architecture when there is communication but we do not use any Mean Embedding: in this case, o^n_i is the concatenation of the observations. (c) shows the no-communication case in which only the local observation is available.

- In order to build the observations vector, we use the time difference between the current time step and the last K transmissions. This difference is normalized by the maximum number of time steps. Also, we add a flag indicating whether the agent has been discovered by the defense mechanism or not.

For both attacks, since we use a model-free Deep RL approach, we do not require a model of the transition probabilities which, as we have seen in the previous Section, is also an important advantage over methods which require explicit models of these probabilities, allowing to significantly ease the required computational load. Also, the states in both attacks are the reputations t_{PHY} for the SSDF attack and t_{MAC} for the backoff attack, which the ASs do not know. Additionally, we want to study whether communicating the observations can be exploited by the DLA agent to learn better attack mechanisms. Therefore, we also compare to the non-communication case.

The set of observations, actions and rewards of each agent in each time step is used to update the common policy π_θ using TRPO. We use as policy an FNN, which takes as input the observation vector o^n_i and outputs a discrete action. The two first layers of the network have 256 neurons and use rectified linear activations. The output of the network are the actions.

We also use mean embeddings in order to combine the observations from the ASs in case that there is communication in a meaningful way, so that this combination is invariant to the number and order of the

agents. Also, note that there are two different kind of sensors: ASs and GSs. In order to combine the sensor observations meaningfully, we concatenate one mean embedding for the observations of the ASs and another with the observations of the GSs: this makes sense as we can assume that ASs know the type of the rest of sensors. Hence, the architectures that we use can be observed in Figure 6.4.

6.5 Empirical results

In this Section, we provide empirical results on the theoretical developments of this Chapter:

- First, we focus on the EWSZOT hard fusion problem and validate our MDP model empirically: this simulation assumes perfect and complete information.
- Then, using our MDP model, we use RL tools in order to obtain attack strategies against EWSZOT and compare them with the optimal ones. As we show, RL tools provide quasi-optimal results. In this simulation, RL agents assume perfect information of the state, but incomplete information of the defense mechanism.
- We then study the soft fusion case, in which our proposed DLA is able to learn in an incomplete and imperfect information setting.
- Finally, we test the attack policy learned by our DLA in the partial observation backoff attack and comment the results on this setup, which is also an incomplete and imperfect information setting.

6.5.1 Simulation 1: Using the MDP model to evaluate attacks against EWSZOT

The MDP model described in Section 6.3 can be used to evaluate the performance of a concrete policy, hence, it allows evaluating the performance of a certain attack. We evaluate the naive strategies using the procedure described in Algorithm 16 to obtain their theoretical performance. Recall that naive strategies consist in always using a predefined action, which we use as baseline. In our problem, we propose the two following naive strategies:

- Always false attack (AFA): it is an SA in which the ASs always give false reports to the FC. It is a current attack against EWSZOT as shown in [260], together with always reporting the channel busy or idle. Note that these two attacks can also be assessed using our approach.
- Jam and false attack (JFA): it is a Combined Attack (CoA) that consists in attacking as long as it is possible: the ASs always give false reports to the FC and jam the communications for the first M_j GSs called by the FC. We remark that this is a novel attack, which exploits the problem of ad hoc defense.

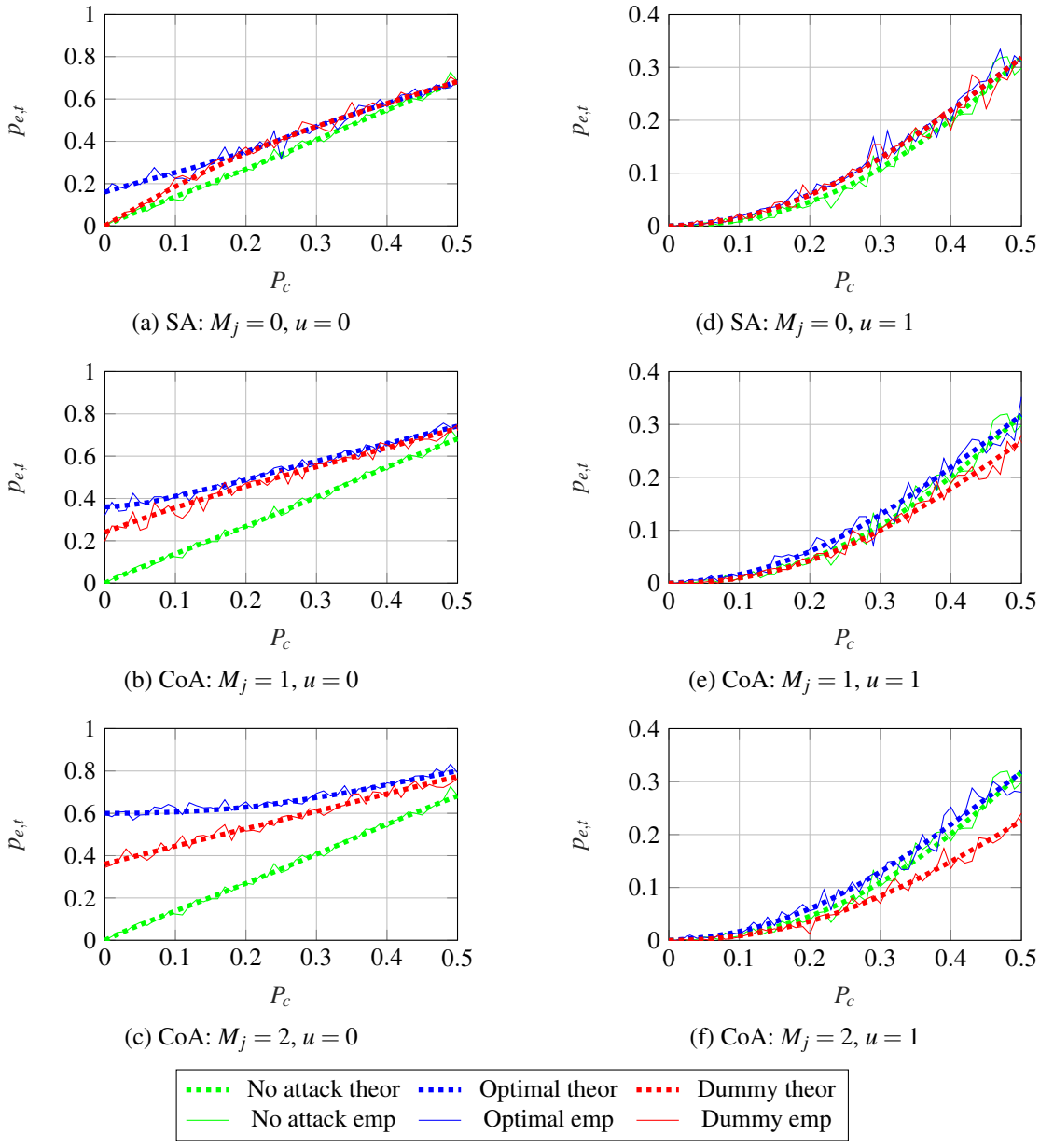
We can also take a step further and obtain the optimal solutions using the MDP model obtained. We use the DP algorithm from Lemma 1 to obtain the maximum error probability that the ASs can achieve and the optimal policy that they must follow in order to achieve that optimal attack. We initialize $V^N(s^N) = 0$ for all states s^N. By doing that, $V^0(s^0) = p_{e,t}$, that is, the value function in the initial state s^0 is the expected total error probability. The main drawback of DP algorithm for our problem lies in the dimensionality of it, as shown in Section 6.3.2. We use small values in our simulations in order to alleviate the computational cost.

Hence, we obtain the theoretical error curves under AFA and JFA using Algorithm 16 with a truncation value $T = 10^3$. We use these values as baseline to compare with. Then, we obtain the optimal attack strategies

using the DP algorithm from Lemma 1. In order to alleviate the curse of dimensionality, we use $M = 4$, $q = 2$ and $g = 5.51$ for the EWSZOT HT. We consider a CSS network with $I = 10$ sensors, of which $n_2 = 1$ and $n_1 = 9$. We test for $M_j = \{0,1,2\}$, that is, for SA and CoA, using 51 equispaced values of P_c in the range $P_c \in [0,0.5]$ and considering $N = 5$ stages, both when $u = 0$ and $u = 1$. Finally, we test and average 100 empirical implementations of the naive and optimal strategies in order to validate our approach, using Algorithm 12 to implement EWSZOT defense mechanism.

The results can be observed in Figure 6.5. Note that optimal strategies always yield the highest error and also, observe that our model predicts correctly the empirical values. When $u = 0$, EWSZOT defense system is severely degraded by the optimal strategy, specially when there is jamming. We also can check that AFA is close to be optimal, specially for $P_c \geq 0.2$, but when jamming is available, JFA harm is surpassed by the optimal strategies. Note that the highest differences in the attacks happen when there is a low P_c: in systems with a low probability of spectrum sensing error, the optimal attacks can notably degrade the system performance. When $u = 1$, we observe that there is a small difference between strategies: the optimal yields the highest error, but the difference is small. When jamming is available, JFA actually decreases the error achieved by EWSZOT, because of the conservative decision when a sensor is jammed: a corrupted report is considered to indicate that the channel is busy. Thus, JFA seriously affects the CSS procedure when the channel is idle. In order to overcome the damage done by JFA, the FC could implement a jamming countermeasure which will depend on the jamming technique used: several methods are proposed in [156]. We remark the influence of the problem of ad hoc defense: by performing a minor change in the attacker capabilities, namely, by having jamming capabilities, even a naive attack can significantly degrade the defense mechanism performance.

Finally, the ASs can significantly harm the system when $u = 0$. The attack efficiency is significantly enhanced by having the possibility of jamming GSs: the effects can be dramatic, as in case (c) in Figure 6.5. We can observe in this case both the problem of ad hoc defense and the problem of optimality: since AFA is not the optimal attack against EWSZOT, an intelligent choice of when to give false reports and when to give true ones provides an attack that degrades the performance of EWSZOT. And by adding jamming capabilities, the attackers are able to achieve an even higher degradation on the defense mechanism. However, the main drawback of the optimal strategies is that they are very costly to compute as the CSS network grows. Observe that we have limited to a case with a low dimensionality for our examples in order to avoid this problem.



(a) SA: $M_j = 0$, $u = 0$

(b) CoA: $M_j = 1$, $u = 0$

(c) CoA: $M_j = 2$, $u = 0$

(d) SA: $M_j = 0$, $u = 1$

(e) CoA: $M_j = 1$, $u = 1$

(f) CoA: $M_j = 2$, $u = 1$

Fig. 6.5 Performance of optimal and naive attack strategies against EWSZOT in terms of $p_{e,t}$ as a function of P_c.Observe that optimal strategies always yield the highest errors, as expected. Note that naive strategies are usually not optimal, specially for low P_c values.

6.5.2 Simulation 2: RL tools to obtain attacks against EWSZOT

Now, we turn to explore the possibilities of learning attack strategies using our MDP model of the EWSZOT hard fusion problem. We make use of RL tools already presented in Chapter 2, where we remind that RL tools need not knowing the transition probability function, and hence, they may provide a significant computational gain, as $P(s^{n+1}|s^n,a^n)$ needs not be obtained. We remark that this means that RL tools may be able to attack any defense mechanism that could be posed as an MDP problem.

Since our action space is discrete, we make use of Q-learning, DQN and DRQN, in the case in which there is a single AS. To use all these algorithms, we need to simulate the environment with which our agent will interact: we do so by using Algorithm 12. Recall that we define the state as the tuple $s^n = \langle z_g^n, z_a^n, m_j^n \rangle$ and the action as the tuple $a = \langle a_g, a_a \rangle$. The environment returns the next state s^{n+1} and the immediate reward obtained r, which is the total error probability p_e: it will be either $p_e = 1/N$ if EWSZOT decided wrongly or $p_e = 0$ if there was no error deciding.

For Q-learning, we follow Algorithm 4, but we use variables α and ε: we initialize each of these values to $\alpha^0 = 0.5$ and $\varepsilon^0 = 1$ respectively and then, after each iteration of the algorithm, we update then by using a decay factor $\alpha_d = \varepsilon_d = 0.9997$, i.e., $\alpha^{k+1} = \alpha^k \cdot \alpha_d$ and $\varepsilon^{k+1} = \varepsilon^k \cdot \varepsilon_d$. This decay is used to met the convergence conditions and to balance the exploration-exploitation trade-off: the agent starts with a high ε and thus, it explores often. As the agent interacts with the environment, the agent explores less and exploits more: eventually, we want to end with an ε value close to 0. This causes that the $Q_\pi(s^n,a)$ values are close to the real ones when using a greedy policy, i.e., ε-greedy policy when $\varepsilon = 0$. We repeat the learning procedure for $n_{epochs} = 2 \cdot 10^4$, and then, we approximate the optimal policy π^* as:

$$\pi(s^n)^* \approx \hat{\pi}(s^n)^* = \arg\max_a Q_\pi(s^n,a) \tag{6.16}$$

For DQN (Algorithm 5), we use a three layers FNN whose structure can be seen in Figure 6.6. Each of the three layers is fully connected to its neighbors. The first layer has as input size the state size, 24 neurons and uses rectifier linear units for activation. The second layer has also 24 neurons and also uses rectifier linear units for activation. The final layer has 24 neurons, an output size equal to the possible number of actions and it uses linear units for activation, as the FNN approximates the Q-function, which may take any value in $\mathbb{R}$, see (2.28).

For DRQN, we use a two layers neural network whose structure can be seen in Figure 6.6. The first layer is an LSTM [104], whose output space has a dimensionality of 32 and it takes sequences of 4 time steps. The second layer has 24 neurons, an output size equal to the possible number of actions and it uses linear units for activation. Note that, although DRQN was proposed for working in partially observable environments, we include it here because it is able to work with non stationary policies, as it keeps information about the past. However, as we noted before, in this case the empirical results show that there is not a significant gain from using a non stationary policy.

For both DQN and DRQN, we choose Adam as optimizer [122], with parameters $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$. The loss function we use is the mean squared error (MSE). The maximum number of experiences stored is 10^4. After a new experience e^n has been obtained and added to the set E, we randomly pick a mini-batch of 128 experience elements and train the neural networks. We use an ε-greedy policy with variable ε: $\varepsilon_0 = 1$ and $\varepsilon_d = 0.9995$, with a minimum value $\varepsilon = 0.01$. We train the networks using $n_{epochs} = 2 \cdot 10^3$ episodes.

We simulate using the same CSS network as in the previous simulation, i.e., $I = 10$, $n_2 = 1$, $n_1 = 9$, $M = 4$, $q = 2$ and $g = 5.51$. Again, we test for $M_j = \{0,1,2\}$, using 51 equispaced values of P_c in the range $P_c \in [0,0.5]$, using $N = 5$ stages and using $u = 0$ and $u = 1$. We compare the empirical curves obtained averaging 100

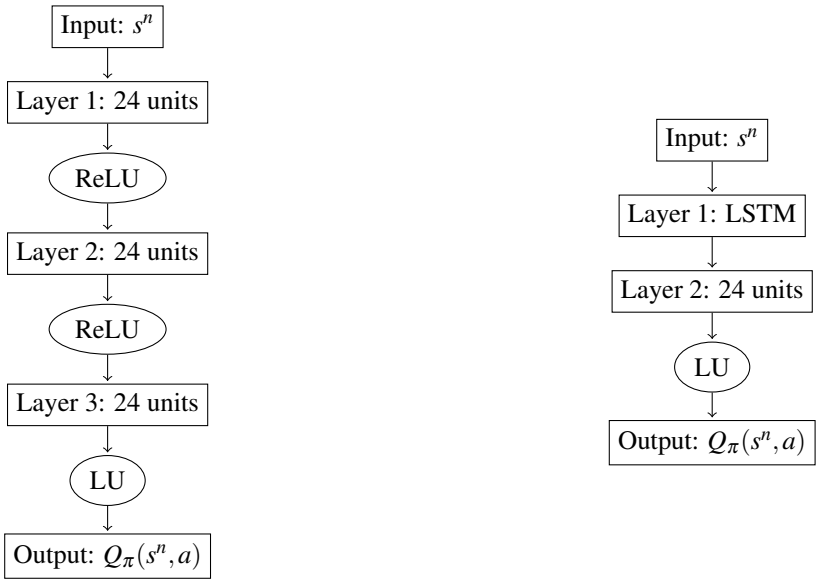
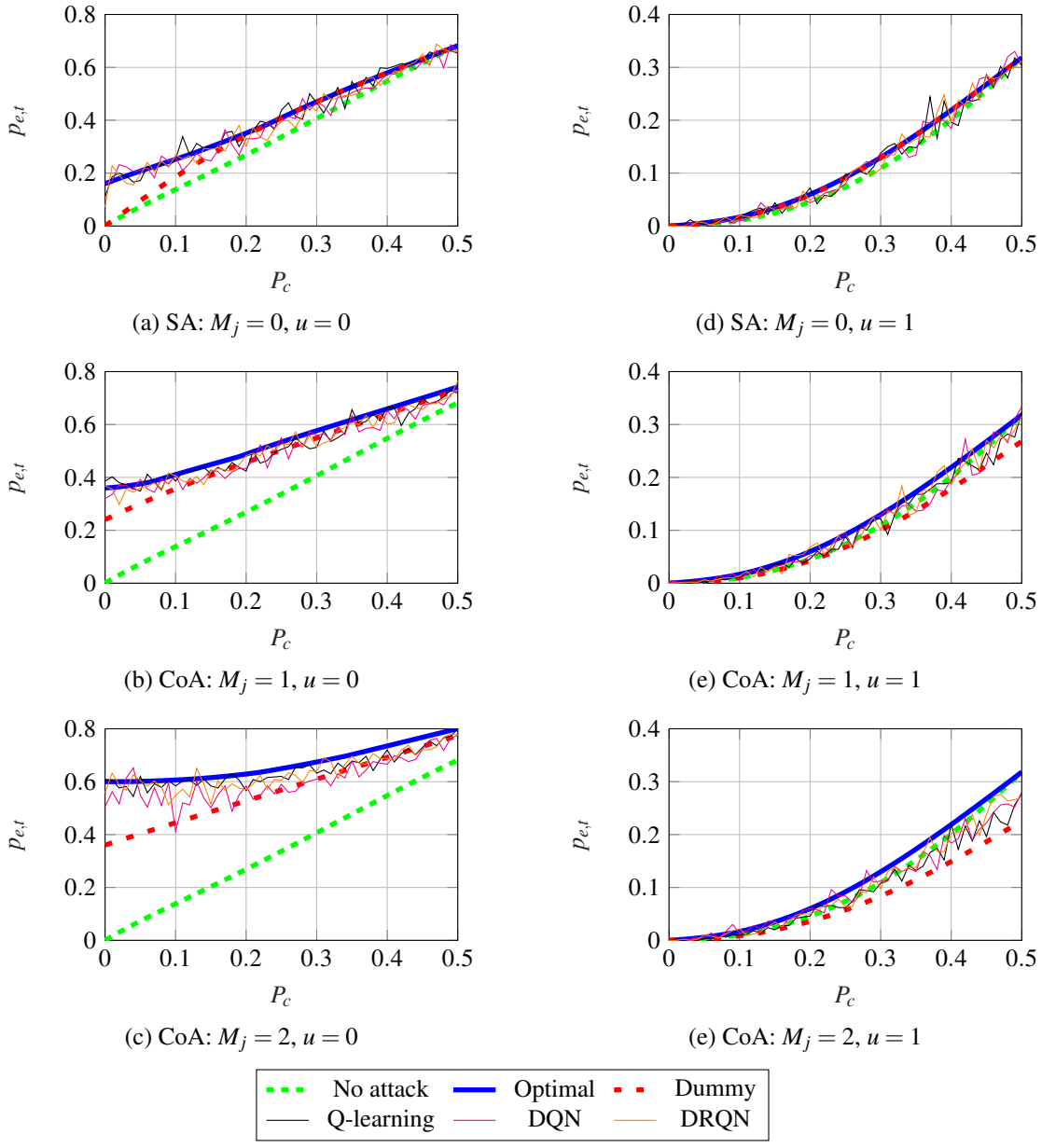


Fig. 6.6 DQN and DRQN structures chosen. For DQN (left), the three layers are fully connected and each of them has 24 units. ReLU is the rectified nonlinearity activation function $f(x) = \max(0,x)$ and LU is the linear activation function $f(x) = x$. For DRQN (right), the first layer is an LSTM with an output space dimensionality of 32, and the second is a dense layer. The input is the state s^n and the output is the estimation of $Q_\pi(s^n,a)$.

runs of the trained policies obtained using Q-learning, DQN and DRQN. We compare these values with the theoretical curves obtained for the optimal and naive attacks obtained in the previous simulation. The results can be observed in Figure 6.7. Note that Q-learning, DQN and DRQN provide very good results, with error curves similar to the optimal theoretical ones for all cases. Being the results quite similar, DQN and DRQN presents an advantage: they took less computation time to learn than Q-learning, around one magnitude order below, and DQN learned slightly faster than DRQN. This is mainly due to the use of experience replay: note that Q-learning trained using 10 times more epochs than DQN and DRQN. Another advantage of DQN and DRQN is that they do not need much memory to store the Q-function, although they require memory to store the E set. Note, however, that E has a size limited that we choose before training, but Q-value function entries number grows exponentially with the problem parameters, as shown in see Section 6.3.2. Hence, DQN and DRQN are more versatile solutions for high dimensionality problems, where Q-learning may not be practical due to the memory cost.



(a) SA: $M_j = 0$, $u = 0$

(d) SA: $M_j = 0$, $u = 1$

(b) CoA: $M_j = 1$, $u = 0$

(e) CoA: $M_j = 1$, $u = 1$

(c) CoA: $M_j = 2$, $u = 0$

(e) CoA: $M_j = 2$, $u = 1$

Fig. 6.7 Performance of optimal, naive and RL strategies against EWSZOT in terms of $p_{e,t}$ as a function of P_c. We compare the results of the three RL algorithms with the optimal and naive strategies theoretical values. Observe that all RL algorithms learn strategies that are quasi-optimal.

Discussion on attack strategies against EWSZOT

We compare the strategies used using the following points, that are summarized in Table 6.2:

- In terms of complexity to obtain the policy, naive strategies are simple since they are fixed beforehand. The optimal approach on the other hand is hard: we must know both the state space and the transitions, and that is costly. Q-learning requires knowing all possible states: these might be hard to compute. Finally, DQN and DRQN do not require knowing neither the states nor the transition probability function, and thus, the difficulty in this case is lower than using Q-learning; yet tuning the hyperparameters to obtain an adequate training might be tricky for this approach.

- In terms of computational resources, naive strategies have very small requirements both in computation and memory capacities. Optimal strategies need a large amount of both. Q-learning is not as computationally expensive but still needs a huge amount of memory. DQN and DRQN do not require as much computational capacity as the optimal strategies, and their memory requirements are lower and indeed, could be controlled by adjusting the size of E, although note that this would influence the training process.

- In terms of policy implementation, naive strategies are very easy to implement due to being fixed. The optimal and Q-learning strategies can be implemented as a search over a table that returns the prescribed action for a given state. DQN and DRQN require implementing a neural network, which causes it to be a bit harder.

- In terms of attack results, i.e., the total error probability, naive strategies give the worst results. The best possible result, by definition, correspond to the optimal strategies. And Q-learning, DQN and DRQN provide quasi-optimal solutions, similar to the optimal ones.

By comparing all strategies, we see that naive and optimal strategies do pose a strong trade-off between attack results and complexity. But this trade-off can be alleviated by using RL strategies, specially DQN and DRQN. Using neural networks, specially DQN, provides a good compromise between complexity and attack results: the algorithm is not too hard to implement, it is not excessively expensive computationally and finally, its results are quasi-optimal.

Note that our results show that current WSN defense mechanisms can be vulnerable to an intelligent attacker, specially to one based on Deep RL tools. As we just showed, such an attacker can obtain good attack results, i.e., it can degrade the defense mechanism performance, even if it does not know the concrete defense mechanism used. The computational cost for that attacker may be under the computational capacities of current hardware. Thus, we need to obtain defense mechanisms that are able to tackle with these intelligent attackers: this will be the topic of the next Chapter.

Hence, we have shown that an AS making use of our MDP approach could successfully exploit a defense mechanism in WSN. The main strengths of using our approach are:

- It is possible to obtain the theoretical results of different attack policies if the MDP is modeled, and even obtaining the optimal attack policies. Note that this means that different attacks can be compared in terms of their effects in a straightforward way. It also addresses the optimality problem: we can obtain the optimal attack and hence, have a bound on the defense mechanism performance.

- The use of RL tools allows us to obtain quasi-optimal attack results if the MDP cannot be modeled due to being unknown or if it is highly complex. We have shown that Deep RL tools are of special interest

	Naive	Optimal	Q-learning	DQN/DRQN
Policy obtention	Easy	Very hard	Hard	Medium
Computation	Low	Very high	High	Medium
Implementation	Very easy	Easy	Easy	Medium
Attack results	Poor	Optimal	Very good	Very good

Table 6.2 Comparison summarizing the different strategies used against EWSZOT.

here due to the balance between results achieved and their complexity. We remark that RL tools exploit the problem of ad hoc defense: minor changes in the attack mechanism, such as giving false reports selectively in case of EWSZOT, may cause a significant degradation in the defense mechanism. As the next Section shows, RL tools are able to tackle with high dimensional attack problems successfully.

Our approach also has some weaknesses:

- Modeling an MDP may be very hard: the state-action space could be prohibitively large or the transition function could be very complex to obtain. Hence, optimal solutions could be computationally very complex to obtain. Also, note that minimal changes in the MDP may significantly change the MDP definition and transition function. Note that this is a strength for RL methods: they can be adapted easily to such changes, as they work in incomplete information problems.
- Q-learning suffers also when the action-space is large.
- We have focused in the single agent case, i.e., only one AS, and we have also focused on discrete actions. However, in the incoming Section we will drop this assumption.
- We have also assumed that the ASs can observe the state of the system, which are the reputations in case of EWSZOT. This assumption need not hold in real-life systems, where only a noisy observation of the state could be available. Again, in next Section we will drop this assumption and assume an imperfect information setup.
- Deep RL methods results can be sensitive to hyperparameter tuning and also, to the reward scheme. That is, different reward schemes may cause the agent to learn different attacks. In our example, the reward definition was straightforward, i.e., the total error probability, but we assumed that the ASs knew the channel state, which needs not happen in real-life systems.

In next simulation, we show how these results generalize to other defense mechanism, where we make use of our more general DLA architecture.

6.5.3 Simulation 3: DLA attack results against the soft fusion CSS

Now, we change our experimental environment to attack the high dimensional, soft fusion CSS defense mechanism explained in Section 6.2.1 using our DLA architecture. We set a maximum number of 250 time steps for the SSDF attack. The WSN contains 10 GSs and additionally, $\{1,5,10\}$ ASs. We test four different setups: with communication and Neural Networks Mean Embedding (CNNME), with communication and Mean-based Mean Embedding (CMME), with communication but without using mean embeddings (C) and without any communication among ASs (NC). Note that in the C case, the input size of the policy network is equal to the dimension of o_i^n and thus increases with the number of ASs. Also, note that for CNNME, CMME and CNME, the reward that each agent maximizes is the sum of the rewards of all ASs. For each attack, the training procedure is the same. We train the policy network using 500 TRPO iterations; in each of these iterations, a batch with 2500 experience time steps is used. For each combination of number of ASs, DLA setup and attack type, we repeat the training using 10 different seeds and the results are obtained by averaging the best 5 seeds to avoid outliers.

Note that we do not know the optimal solution to the underlying POMDP that models the attack, which is an important difference regarding the EWSZOT case. In order to be able to evaluate the quality of the DLA agent, we compare the results obtained with two baselines policies:

- Random policy: this policy samples the actions uniformly from the action space. Such a policy has a low computational burden, but it provides no guarantee at all in terms of performance. Note that this is a particularly simple naive strategy.

- Black box optimization: this is another naive strategy, in which we take a parameter $\beta \in [0,1]$ that represents the normalized energy that the AS reports to the FC. We take ten equispaced β values in the interval $[0,1]$, and run 5 simulations of the environment with each of them. The β parameter that provides the best result is used. Note that this approach does not require knowledge of the state of the system, and hence, it requires the same information as our DLA. It is simpler than the DLA approach but, as shown by our experiments, it can not keep up with the complex NN policies learned by our agent.

We use finite horizon episodes: i.e., in each episode the FC asks the sensors up to 250 times to send the energy level they measure. We consider that the duty cycle is 0.2, i.e., the probability that the channel is actually occupied by a primary transmitter is 0.2. If an AS is detected, the episode ends for this AS, since the FC does not ask it to send more reports. We implement the defense mechanism explained in Section 6.2.1, with $\eta = 1$, $\zeta = 1.6$ [229] and λ_{PHY} taking uniformly generated random values in the interval $[0.2, 0.8]$ for each episode. Hence, the ASs need to attack in such a way that they can exploit several λ_{PHY} values. If the reputation of a sensor t_{PHY} falls below λ_{PHY}, the sensor is detected as an AS and the episode ends. If the AS attacks indiscriminately, the episode will end early and its reward will be low. In each time step, the defense mechanism is invoked and the sensor reputation is updated.

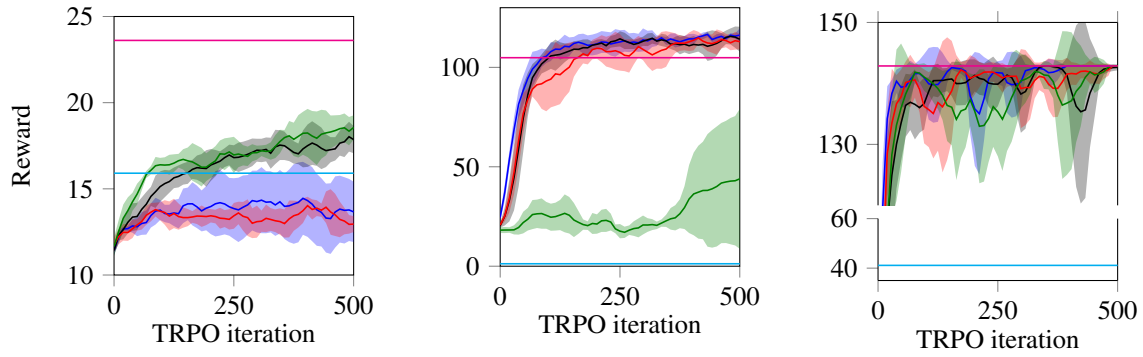
At the beginning of each episode, we pick the sensor distances to the FC, d_m, from a uniform random distribution in the range $[800, 1000]$ meters. We consider that the transmitter power is $P_{tx} = 23$ dBm and use the following path loss expression:

$$P_m = P_{tx} - (35 + 3 \cdot 10 \log_{10}(d_m)), \tag{6.17}$$

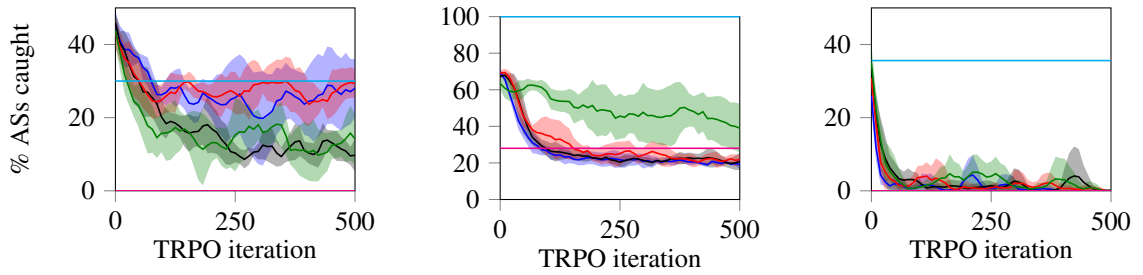
where P_m is the received power in the sensor m, in dBm. This expression allows to obtain $SNR_m = 10^{\frac{P_m - NP}{10}}$, where we consider the noise power to be $NP = -110$ dBm. We consider the time-bandwidth product $k = 5$.

Finally, we generate the E_m values sampling from the distributions in (6.1), depending on whether the primary is transmitting.

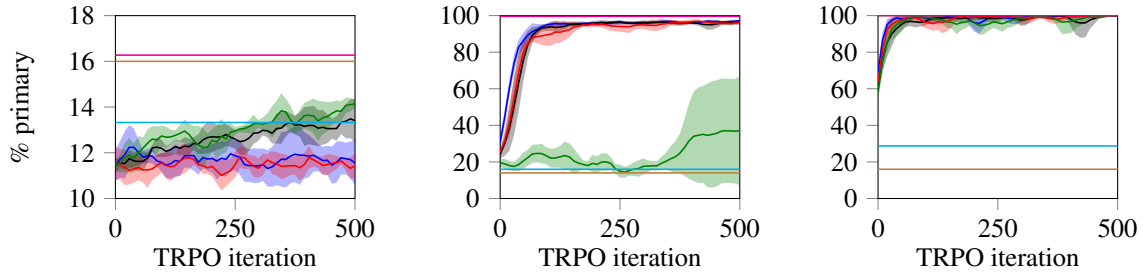
The results can be observed in Figure 6.8 and Tables 6.3 and 6.4. In Figure 6.8, we observe that ASs do learn to successfully exploit the defense mechanism. In the first TRPO iterations, the performance of the DLA is similar to the random policy, but as the number of iterations increases, it surpasses the performance of the random policy. Also, the random policy gives large reward variations depending on the number of ASs. Note, however, that for a single AS, the black box optimization approach provides a better reward. This is due to the ASs taking its decision without information about the actual energy level, thus, the attack is highly ineffective. However, the attack does overcome the defense mechanism when there are several ASs, as shown in Table 6.3: all proposed DLAs perform well. With 5 ASs cooperation is crucial while with 10 ASs, the attack is so easy that no cooperation is needed.



(a) Total reward. From left to right: $1/5/10$ ASs. Higher is better.



(b) Proportion of ASs discovered. From left to right: $1/5/10$ ASs. Lower is better.



(c) Primary detection proportion. Brown is the case in which there is no attack. From left to right: $1/5/10$ ASs. Higher is better.

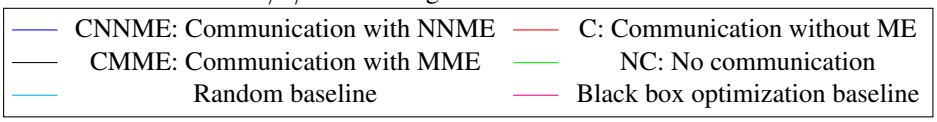


Fig. 6.8 Training results for the SSDF attack. In all figures, the horizontal axis correspond to the TRPO iteration. Note how DLAs are able to successfully exploit the defense mechanism: they send many false reports and a significant proportion of them remains undetected for the defense mechanism. As the primary transmits actually a 20 % of the time, note that with 5 and 10 AS, the DLA is able to blind the defense mechanism and cause that the FC always believes that a primary is transmitting.

In order to study how cooperation is crucial with 5 ASs, we represent the policies for the CNNME and NC cases in Figure 6.9, where we can observe that, with communication, the ASs learn to jointly report high energy levels, which causes that the defense mechanism detects a primary transmitting. Without communication, however, if some ASs deviate and report low energy levels, the defense mechanism detects the ASs. Thus, for 5 ASs, communication is crucial in order to overcome the defense mechanism.

6.5.4 Simulation 4: DLA attack results against the partial observation backoff attack

We now proceed to attack the partial osbervation backoff defense mechanism explained in Section 6.2.2 using our DLA architecture. We simulate the backoff attack for $5 \cdot 10^5$ μs in a WSN containing 10 GSs and additionally, $\{1, 5, 10\}$ ASs. We use the same four setups used in the soft fusion attack, that is, CNNME, CMME, C and NC. Again, for CNNME, CMME and CNME, the reward that each agent maximizes is the sum of the rewards of all ASs. We use the same training procedure as in the soft fusion case: we train the policy network using 500 TRPO iterations, using batches with 2500 experience time steps, we repeat each training case using 10 different seeds and the results are obtained by averaging the best 5 seeds to avoid outliers.

Again, we compare to two baselines. The random policy is one of the baselines again, but the black box optimization baseline changes: now, the parameter $\beta \in [0, 1]$ is the parameter of a Bernoulli random variable that says whether the AS transmits or not in the current time step. Again, we take ten equispaced β values in the interval $[0, 1]$, and run 5 simulations of the environment with each of them, choosing the β parameter with highest reward.

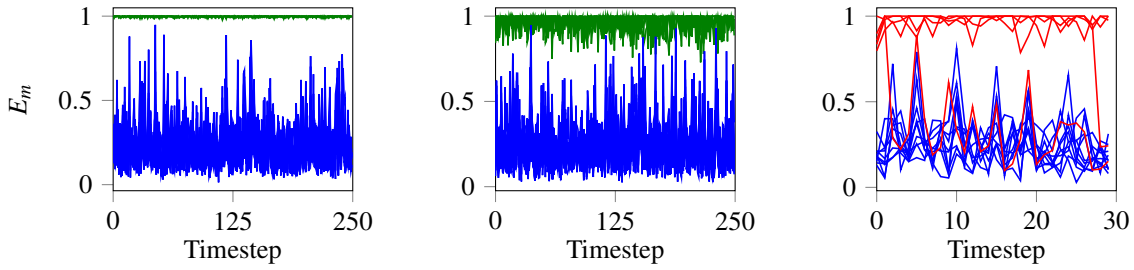
In each time step, an AS decides whether to start transmitting or stay idle. Hence, time steps are related to backoff steps, not to physical time, i.e., if an agent starts transmitting in time step n, time step $n+1$ will take place when that agent finishes transmitting. Note that we do not penalize collisions. The defense system explained in Section 6.2.2 is used, with $K = 5$ and $L = 1000$. If the reputation of a sensor t_{MAC} falls below λ_{MAC}, the episode ends for this sensor. Again, λ_{MAC} takes uniformly generated random values in the interval $[0.2, 0.8]$, which means that if the AS attacks indiscriminately, the episode ends early with a low reward: recall that a final reward is given to each AS that corresponds to the remaining reward of the episode, i.e., if the AS is caught, it does not have the opportunity any more to hinder the GSs to transmit, yielding a lower reward. We use the network parameters from Table 4.2 to simulate the backoff attack. The defense mechanism is run once every 5 time steps, in order to ease the computational load.

The results can be observed in Figures 6.10 and 6.11, and Tables 6.3 and 6.4. We observe that:

- In all setups, the DLA is able to find a successful attack to the backoff defense mechanism. Note that the use of mean embeddings proves to be specially consistent, giving the best attack policies in all cases. This result is due to the compromise they provide between having multiple observations - in this attack, coordination among ASs is crucial to be successful - and keeping the observation space from growing unbounded. Note that mean embeddings allow fusing information from many sensors while keeping the observation space dimension independent of the number of ASs.

- Having more ASs means that the GSs transmit less often. Also, from our simulations, we observed that the total amount of bits transmitted significantly decreased with the number of ASs. For instance, for the CNNME case, with 1 AS the average bits transmitted are 246 $kbits$, for 5 ASs is 136 $kbits$ and for 10 ASs, 65 $kbits$. The more AS present in the network, the higher the reduction in the network total bitrate.

- Finally, in Figure 6.11, we observe that the DLA is able to adapt to different values of λ_{MAC}. It is able to sacrifice some ASs, which are discovered, in order to prevent the rest of AS of being discovered and



(a) CNME. (b) NC. (c) NC.

Fig. 6.9 Examples of learned SSDF attack policies for the DLA, using CNNME and without communication (NC) with 5 ASs. For comparison purposes we set $\lambda_{PHY} = 0.5$. We plot the normalized energy that each sensor reports, where blue are the energies reported by GSs, red are the energies reported by discovered ASs and green are the energies reported by undiscovered ASs. In the CNNME case, the agents learn to transmit high levels of energy and not being discovered (a), whereas in NC case, there are times in which sensors are discovered due to their lack of cooperation (c). In general, in NC, ASs report energies lower than in the CNME case (compare (a) to (b)): cooperation helps obtaining a more aggressive policy which, at the same time, allows the ASs to camouflage.

		CNNME	CMME	C	NC
SSDF attack	1 AS	14.46 ± 8.15	$\mathbf{18.51 \pm 5.98}$	13.22 ± 8.19	$\mathbf{19.34 \pm 5.68}$
	5 ASs	$\mathbf{115.79 \pm 54.08}$	$\mathbf{113.06 \pm 55.0}$	$\mathbf{122.22 \pm 48.49}$	47.09 ± 46.51
	10 ASs	$\mathbf{142.88 \pm 0.0}$	$\mathbf{142.86 \pm 0.1}$	$\mathbf{142.88 \pm 0.0}$	$\mathbf{142.88 \pm 0.0}$
Backoff attack	1 AS	$\mathbf{-38.77 \pm 3.75}$	$\mathbf{-38.8 \pm 6.01}$	-40.55 ± 4.67	$\mathbf{-38.95 \pm 4.98}$
	5 ASs	$\mathbf{-20.81 \pm 5.5}$	$\mathbf{-20.46 \pm 5.18}$	-24.26 ± 5.13	$\mathbf{-20.96 \pm 4.13}$
	10 ASs	$\mathbf{-10.91 \pm 4.09}$	$\mathbf{-11.15 \pm 4.7}$	-13.97 ± 4.36	-13.61 ± 3.62

Table 6.3 Final rewards obtained for each combination of attack, number of ASs and setup. The values were obtained averaging 50 episodes for the best 5 seeds of each case. We show the mean final reward, $\pm$ one standard deviation. Bold entries are the largest mean reward using DLA, where a Welch test is used to detect whether means are significantly different for a significance level $\alpha = 0.01$. Higher is better.

hence, successfully overcoming the defense mechanism. Note that this result explains why in Figure 6.10 there are always some ASs discovered: some ASs need to be discovered in order to detect where is the threshold λ_{MAC} and hence, deciding how the rest of ASs should attack. This cooperation is exploited in communications schemes, and hence, it is expected that communication with mean embeddings DLAs consistently give the best result in this setup (see Table 6.3).

6.6 Conclusions

In this Chapter, we have modeled several defense mechanisms using MDP tools and then, we have used DP and RL tools in order to obtain successful attack strategies against such defense mechanisms. We have considered three different attacks, two of them against a CSS procedure and the third against the backoff mechanism. We have seen that the MDP framework allows even obtaining analytical results, however, they come at the cost of obtaining the probability transition function, which in general may not be easy or even possible if the defense mechanism is unknown. Also, we have noted that, even if we have the probability transition function,

		Random	Black box optimization
SSDF attack	1 AS	15.91	**23.61**
	5 ASs	1.23	104.85
	10 ASs	41.12	**142.87**
Backoff attack	1 AS	−75.42	−75.30
	5 ASs	−68.79	−63.22
	10 ASs	−66.83	−67.67

Table 6.4 Mean final rewards obtained for the two baselines. The values were obtained averaging 50 episodes. In bold, we show when a baseline provides an equal or better reward value than the best DLA. Higher is better.

it is possible that the curse of dimensionality appears and hence, obtaining the optimal policy may become computationally intractable.

In order to overcome these problems, we have made use of Deep RL tools, as they can approximate successfully the optimal policy for an MDP without needing to know the probability transition function, i.e., in incomplete information settings. Moreover, they are successful even in high dimensionality problems with imperfect information. Hence, in this Chapter, we show that a Deep RL based attacker is a threat against current WSN defense mechanisms, as they are able to exploit them simply by interacting with them: we propose a DLA architecture that is able to coordinate several ASs in order to exploit a defense mechanism with partial observability. The main problems that may arise using our approach are the following:

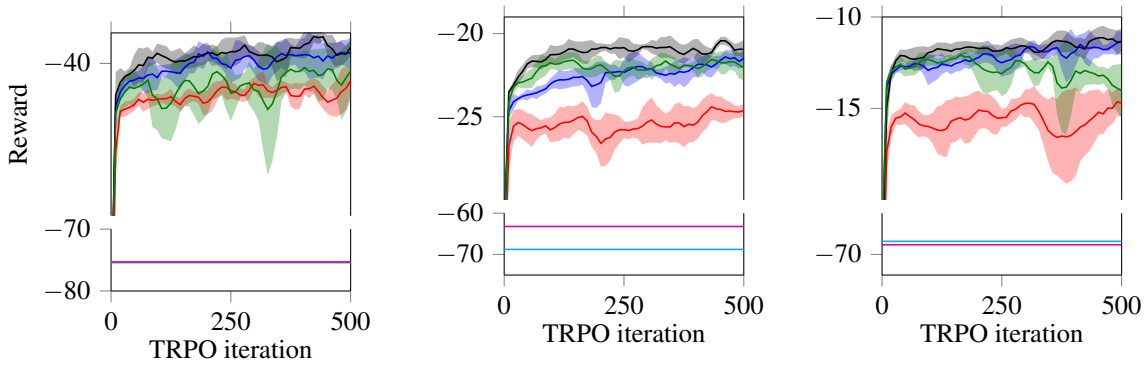
1. The reward scheme has a strong influence on the attack that is learned. There is a tradeoff between instantaneous and final rewards that conditions the learning, and also the reward scheme must reflect the desired attack outcome. Thus, modifying the reward scheme will cause that the agents learn a different attack strategy.

2. It is difficult to evaluate how efficient the learned attacks are when the problem dimension grows. In the hard fusion SSDF attack, we were able to compare to the analytical solutions, but in the soft fusion SSDF attack and the partially observable backoff attack we could only compare with a random and a black box optimization policies, where we note that the results obtained by our agent surpassed these baselines in most cases. However, in general, we do not know the optimal attack strategy and hence, we can not measure how far our attack strategy lies from the optimal, even though the results from the hard fusion SSDF attack suggest that they might not be too far from the optimal in many cases.

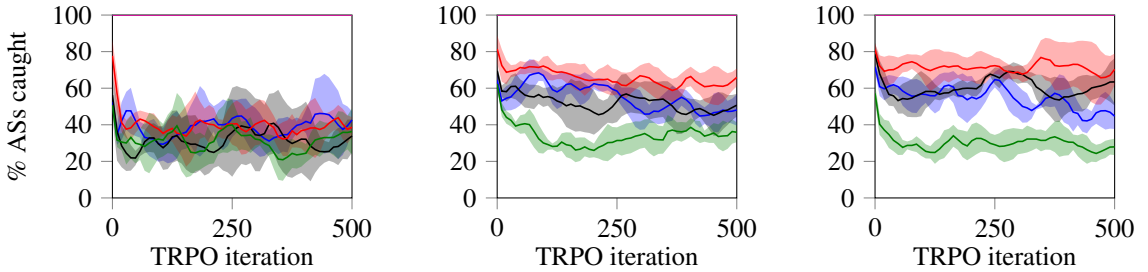
And using a Deep RL attacker presents several strong advantages:

1. It works with imperfect information: we do not need to know the state of the system, as we may rely only on partial observations and still obtain good results, specially when the ASs are able to communicate their observations. Observe that in Table 6.3, the results using communication consistently rank among the best. However, even when this communication is not considered, i.e., the NC setup in the last two simulations, the results are still good. Note that even though the underlying model of the attack is a POMDP, our agent learns to attack having only a limited amount of past observations.

2. It works with incomplete information: since a Deep RL attacker does not need to know a priori which defense mechanism it is facing, it is a very flexible approach. Thus, it could be the base for a universal attacker, successful in exploiting many defense mechanisms. Moreover, we do not provide any a priori information about the defense mechanism: the AS does not know which are the concrete parameters of



(a) Total reward. From left to right: $1/5/10$ ASs. Higher is better.



(b) Proportion of ASs discovered. From left to right: $1/5/10$ ASs. Lower is better.

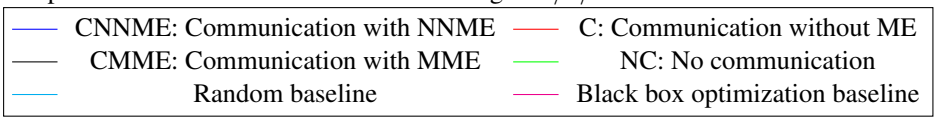


Fig. 6.10 Training results for the MAC attack. In all figures, the horizontal axis correspond to the TRPO iteration. Note how DLAs are able to exploit the MAC mechanism: each AS transmits more bits than a NS while not a significant portion of ASs is not detected by the defense mechanism.

the defense mechanism, and it can even adapt to variations in the parameters of the defense mechanism: note that we have trained our agent in the last two simulations using a variable threshold in the defense mechanisms and it has been able to adapt to them, as Figure 6.11 shows.

3. It is a method with balanced computational requirements. The training process is the most computationally expensive part of the system. However, most of this cost was used in generating samples from the defense mechanisms: training the neural network was fast. Note that this low neural network training cost appears because we use a simple neural network, which however is enough to exploit the defense mechanisms. Once the neural network is trained, the policy is fast to execute. This has been observed in all the simulations of this Chapter.

4. We remark that we have used the same set of hyper-parameters for our last two simulations: this has been done on purpose in order to provide an architecture as general as possible in the sense that it may adapt to different defense mechanisms. We have done no fine-tuning of these hyper-parameters, and thus, our approach may suit very different attack situations with minimal tuning. Equivalently, the results obtained could be improved by doing a fine-tuning for each situation.

5. We have tested several agents, and we can observe, from Tables 6.3 and 6.4, that they do consistently provide better results than other baselines, and in Figure 6.7, they are very close to the optimal values.

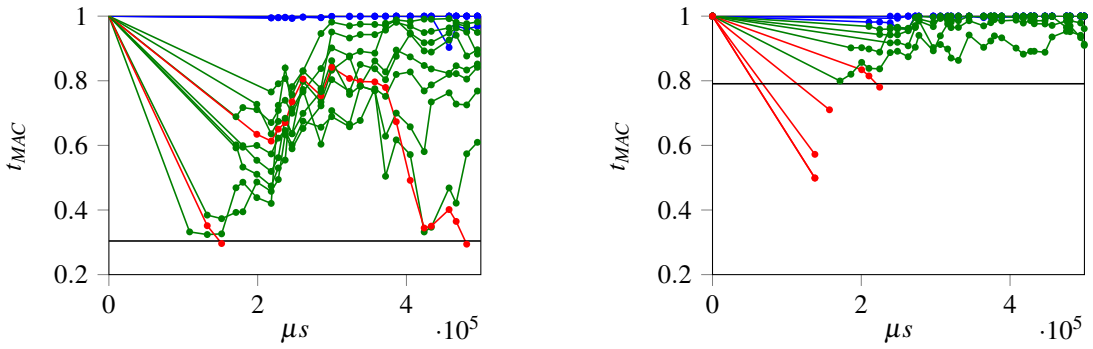


Fig. 6.11 Examples of learned backoff attack policies for the DLA, using CNNME with 10 ASs. The colored lines are the t_{MAC} values, and each dot indicates that the defense mechanism has been invoked. Blue is for GSs, green for ASs not discovered and red for discovered ASs. The black line is λ_{MAC}. Note how the ASs are able to adapt to the different values of λ_{MAC}.

Specifically, when having more than one AS, we recommend using communication and mean embeddings: this approach consistently gives the best results.

Thus, the attacking approach that we propose in this Chapter presents strong challenges to current WSN defense mechanisms. First, because of the growing computational capabilities of current hardware, there could be soon, if not already, sensors with enough computational resources to implement such an attacker [181]. Second, because our approach is adaptive and flexible, not requiring an a priori modeling of the defense mechanism nor knowledge of their parameters, it can learn to exploit a wide range of defense mechanisms, and also note that our attacker only uses partial observations. In other words, is an attacker suitable for imperfect and incomplete information situations. Thus, it is of capital importance researching on defense mechanisms against such attack mechanisms, in order to minimize the threat they pose. A promising defense mechanism could be one in which the defense mechanism also learns how to defend, which means entering the field of Multi Agent Competitive Learning [97], which until today poses strong challenges. A different one consists in using IRL tools in order to model the behavior expected from a GS in order to detect ASs: this is the idea that we develop in Chapter 7.

Chapter 7

Intelligent defense mechanisms against intelligent attackers

7.1 Introduction

In order to successfully address the intelligent attackers presented in Chapters 5 and 6, we could think of using Game Theory tools in order to model the ASs and the defense mechanism as players with different targets: we now assume that the defense mechanism is also dynamic, while in Chapters 5 and 6 it was considered static. Note that we should not use the RGs framework as we did in Chapter 4, as the attackers make use of states or observations. Hence, we could think of modeling our defense mechanism using the SG or the POSG framework. Note that in an SG, each player solves a control problem coupled to the actions of the rest of players, and in a POSG, each player solves a POMDP coupled to the actions of the rest of players. Note, however, that SGs require perfect and complete information, while POSGs require complete information only, but the problems introduced in Chapter 6 were games of imperfect and incomplete information. For instance, in the partial observability backoff attack of Chapter 6, we have imperfect information because the ASs does not observe the reputation that the defense mechanism assigns to it, and the defense mechanism does not directly observes whether a sensor follows or not the binary backoff procedure. And we have incomplete information because the ASs do not have a priori knowledge of the defense mechanism, and the defense mechanism does not know which sensor is an AS and which sensor is a GS. A comparison of the situation described by this Chapter with respect to Chapters 4-6 can be seen in Table 7.1.

In incomplete information games, the concept of Bayesian Equilibrium is used to provide a solution to the game. This is an extension of the NE concept that conditions on the belief of each player about the type of the rest of the players. In other words, each player tries to obtain an equilibrium taking into account what she believes, is the type of the rest of the players. This concept applies to static and dynamic games [76], although the dynamic case has complexities derived from the fact that, as the dynamic game advances, the beliefs of the players change. Thus, having incomplete information is an additional difficulty to solve a POSG: as we saw in Chapter 2, in general, POSG cannot be solved efficiently in a way to be used in defense mechanisms.

However, it is possible to proceed differently. In this Chapter, we focus only on the partial observability backoff attack described in Chapter 6 because our simulations showed that it was an attack more complex to detect than the soft fusion SSDF one. With respect to the other versions of the backoff attack shown in Chapters 4 and 5, note that we are interested in the detection problem as in Chapter 5: we want to detect which sensor is

Chapter	CSMA/CA	CSS	Player	Information	Observation (A/S)	Behavior
4	Yes	No	Attack Defense	Complete Complete	Mixed / - Mixed / -	Static Static
5	Yes	Yes	Attack Defense	Complete Incomplete	- / State Realization / -	Dynamic Static
6	Yes	Yes	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / -	Dynamic Static
7	Yes	No	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / Observation	Dynamic Dynamic

Table 7.1 Table comparing the different setups used in Chapters 4-7. CSMA/CA. i.e., the backoff attack, and CSS, i.e., the SSDF attack, denote whether each of these setups is used in the Chapter. Information denotes whether each player knows the target of the other player (Complete) or not (Incomplete). Observation refers to what each agent observes with respect to the actions / states of the other players: regarding actions, they observe the mixed actions or the actions realizations, and regarding states, they observe the state or an observation of the rest of players: this is related to having perfect or imperfect information. Behavior refers to whether the player adapts its behavior with time or not.

an AS and which is a GS. However, as the attacker is intelligent, it can learn to exploit our defense mechanism, even if it is unknown, as we have shown in Chapters 5 and 6. We assume an intelligent attacker based on RL tools, and hence, it has a reward that its behavior optimizes. Our main idea for a defense mechanism against such attackers is based on IRL: we try to discriminate between ASs and GSs by finding the reward function that the behavior of a sensor is optimizing, and in case that it significantly differs from the expected reward of a GS, the sensor is considered an AS. This simple idea turns out to be an efficient detector.

7.2 Intelligent defense mechanism description

Again, we consider that we have a WSN with one or more ASs, while the rest of sensors are GSs. All sensors try to communicate with a central node using a CSMA/CA procedure: the ASs try to obtain as much transmission resources as possible. The central node has a defense mechanism which tries to enforce that the network resources are evenly distributed, and this defense mechanism can observe what each sensor does but does not know a priori whether each sensor is an AS or GS.

As we have already mentioned, there are many proposed defense mechanisms to detect ASs and ban them from the sensor network. The literature on this field is extensive, with many papers ranging from general works, as [220], [72] or [199], to works specialized in concrete attacks, such as byzantine attacks [257], jamming situations [156] or backoff attacks [229]. However, this solution, as we have noted, results in ad hoc defense mechanisms that need to have a priori knowledge about the attack. In order to effectively deal with our intelligent attacker, we need to make as few assumptions as possible: in our case, we assume that the attacker is using control theory tools in order to exploit the defense mechanism. We propose using IRL tools to optimize such a defense mechanism. We first present our approach and then discuss the assumptions on which our approach relies.

As noted, we first assume that the interaction between each sensor and the central node can be modeled using the MDP framework introduced in Chapter 2. This means that, in each timestep n, each sensor observes the state s^n, selects an action a^n and receives a reward r^n; then the system transitions to a different state s^{n+1} and the process is repeated. Note that the concrete definitions of the state and action spaces, as well as the reward

and transition probability functions, depend on the concrete problem we are trying to solve. Finally, for the sake of simplicity, we work using the MDP conventions, but as we know, most real life problems, including ours, is described by the POMDP framework. As noted in Chapter 2, a possible way to approximate a POMDP is by including in the observation past information, in order to solve it using MDP tools. This is the approximation we used in Chapter 6, and we use it again in this Chapter. Hence, everything we say about MDPs can be adapted to POMDPs by replacing the states s^n by the extended observation to include past information.

The main idea behind our method consists in using IRL on GSs in order to obtain the reward function $r(s,a)$ that explains the behavior of the GSs. Since ASs will have a different reward function, the histograms of instantaneous rewards produced by a GSs and an ASs will be different and hence, it will be possible to detect the ASs. In order to obtain the reward function, we use GAIL [101]. As we explained in Chapter 2, GAIL trains a GAN using sequences of states and actions in order to obtain a reward function in the discriminator and a policy in the generator. Thus, in our setup, we use GAIL with sequences of states and actions produced by GSs, and then use the discriminator, i.e., the reward function, in order to detect ASs.

There are several reasons that motivate our choice of GAIL. First, we propose using an IRL method because obtaining the reward function for a given environment may not be an easy task. In our case, we study a transmission procedure so well-known as the backoff method used in 802.11 standard [111], which is designed to minimize the probability of collisions among wireless nodes communicating to a central node. In this scheme, we want to avoid collisions and transmit as often as possible, but there are many potential reward functions that may suit these conditions. By using IRL, we do not need to model the reward function explicitly, as our algorithm will obtain it by itself. Note that, as we are using sequences of states and actions to train GAIL, the reward function obtained will correspond to the behavior exhibited in such trajectories.

Second, we propose using GAIL because it is a model-free method, that is, we do not need to model the transition probability function. This is an advantage for two main reasons. First, the transition probability function may be very large if the states and/or action spaces are large. And second, it may be hard to obtain an analytical model for this function, as the EWSZOT model in Chapter 6 shows. GAIL does not need to know the probability transition function and only needs access to a simulator of the dynamical system, which suits well problems of incomplete information.

And third, we choose GAIL because, as we have shown in Chapter 2, it is an IRL method which is both efficient and accurate. Note that the reward function is approximated using an NN, which means that it is able to approximate any reward function to any degree of accuracy provided that the NN is large enough [105].

7.2.1 Offline defense mechanism

There are at least two possible ways to use GAIL in our problem. We start by describing what we denote the offline defense mechanism. Before starting the interactions, we train GAIL on a set of trajectories of a WSN in which all sensors are GSs, that is, we use a controlled environment to train GAIL. Once that we have trained GAIL and have a reward function, we can use it to detect ASs. The key idea here is noting that the reward function estimated by GAIL will maximize the reward of the GSs, and hence, ASs will provide a lower reward. Thus, if we are given a set of state-action pairs to test, we can obtain its rewards using the reward function computed by GAIL using GSs and compare it to the histogram of rewards given by the original state-action pairs. We propose using a simple test to decide whether a set of state-action pairs comes from a GSs or not. First, we use a threshold η on the reward value which corresponds to the $\alpha \in [0,1]$ proportion of the Cumulative

Distribution Function (CDF) of the rewards of GSs. Mathematically, η is:

$$\begin{aligned} \eta = & \arg\min_{r(s,a)} CDF\left(r(s,a)\right) \\ & s.t. \quad CDF\left(r(s,a)\right) \geq \alpha \end{aligned} . \tag{7.1}$$

Thus, we first obtain the empirical CDF of the rewards of the GSs state-action pairs, and the threshold η is the minimum reward that makes the empirical CDF equal or larger than α. Note that this means that α controls the tradeoff between false alarm probability and power of the test. The η value, hence, is obtained after GAIL has been trained, as it needs to use the same state-action pairs set that we used to train GAIL.

Second, during execution, we receive a set of j state-action pairs from a sensor, and we want to test whether this sensor is a GS or not. In order to do so, we first obtain the reward for each of the j state-action pairs of the sensor using the $r(s,a)$ NN returned by GAIL. Thus, we have the estimated reward that a GS would get by playing action a in state s for each of the j state-action pairs given. Then, we compute $i \leq j$, which is the number of reward values that satisfy $r(s,a) \leq \eta$. In other words, i is the number of reward samples that are low for a GS. A low i could indicate that the samples come from a GS, while a large i would provide evidence that the state-action samples did not come from a GS.

In order to decide whether i indicates a GS or not, we use the following approximation: we assume that the $r(s,a)$ samples are independent and identically distributed. Note that this assumption needs not be true, as consecutive state-action pairs are correlated due to the transition probability function. However, we follow this assumption because it simplifies our model, allows using simpler calculations and does not need to know the probability transition function for the correlation. We can use a binomial test in order to check whether i is large or not using the following ρ threshold:

$$\begin{aligned} \rho = & \arg\min_{k} B(j,k,\alpha) \\ s.t. \quad & B(j,k,\alpha) = \binom{j}{k} \alpha^k (1-\alpha)^{j-k} \\ & B(j,k,\alpha) \geq 1-\alpha \\ & k \in 0,1,...,j \end{aligned} . \tag{7.2}$$

In (7.2), ρ is obtained using a Binomial distribution. The equation that we are minimizing is the Binomial probability for j experiments with k successes and α probability of success. Note that k is the number of successes, and hence, is equivalent to i: the number of times that $r(s,a) \leq \eta$, where η was obtained in (7.1) as the reward value with probability α. Thus, we are modeling the probability of obtaining k values of reward below η. The inequality constraint is the threshold of our test: we use the same α value, although a different value could be used, in order to set ρ as the threshold to decide that lower values of i are generated by GSs, but higher ones are not, with a confidence level of $1-\alpha$. Thus, our decision rule is:

$$\begin{cases} \text{Sensor is GS} & \text{if} \quad i < \rho \\ \text{Sensor is AS} & \text{if} \quad i \geq \rho \end{cases} , \tag{7.3}$$

where a flow diagram for both the training and classification stages can be observed in Figures 7.1 and 7.2.

Hence, for the offline defense case, we first have a training stage, in which we start by getting a set of state-action pairs for GSs in our concrete problem. This set is obtained using simulations or real data. Note that this set of state-action pairs collects the expected behavior of GSs. Then, we use GAIL to train an NN

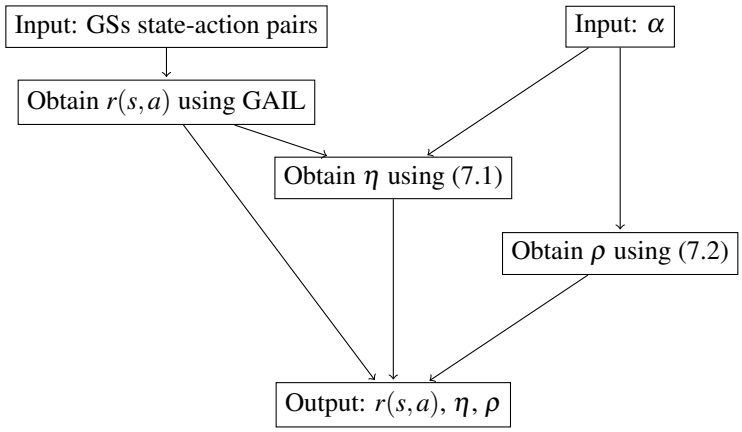


Fig. 7.1 Flow diagram for the training stage of the proposed defense mechanism, both for online and offline cases.

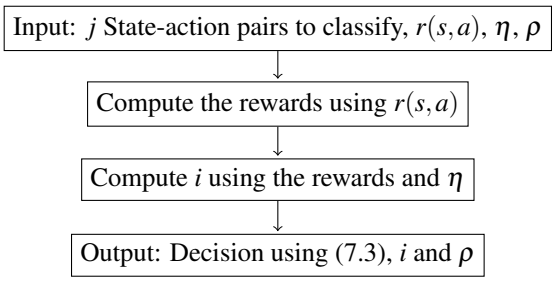


Fig. 7.2 Flow diagram for the classification stage of our proposed defense mechanism, for both online and offline cases.

as the reward function $r(s,a)$ of the GSs. This NN, together with the state-action pairs of GSs used to train GAIL, is also used to obtain the threshold η on the rewards by using (7.1) and also the threshold ρ (7.2). A flow diagram for this stage can be observed in Figure 7.1. Then, there is a second stage, in which the reward function and the threshold η are used to classify sequences of state-action pairs as follows. First, the reward function for each state-action pair is obtained using $r(s,a)$. Then i, the number of rewards such that $r(s,a) \leq \eta$, is obtained, and a decision is made using (7.3), as shown in Figure 7.2. Note that we train GAIL only once, but the classification stage may be run more than once, as shown in Figure 7.3.

Note that the decision method we propose, based on η and ρ, could be replaced by other decision methods, such as measuring the Kullback-Leibler divergence between the distribution of rewards using trajectories from GSs and the rewards using trajectories from a sensor with unknown type. Here, we only focus on using η and ρ because it is a simple and computationally fast method which, nonetheless, provides good results, as we will see in short.

7.2.2 Online defense mechanism

The offline defense mechanism already described presents a problem derived from the fact that the behavior of GSs could be influenced by the actions of the ASs. As GAIL is trained offline, using samples from a network in which there are only GSs, the same GSs in presence of ASs may present a different behavior as a consequence of the actions of the ASs: note that the behavior of GSs and ASs is coupled, as they affect each other. This

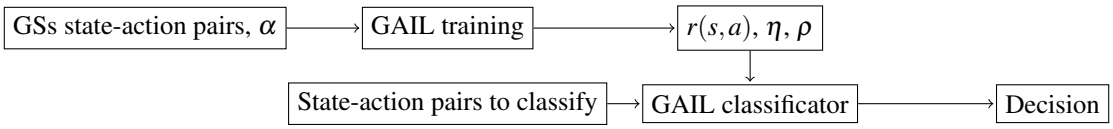


Fig. 7.3 Flow diagram for the offline defense mechanism, where the training stage is explained in Figure 7.1 and the classification stage is explained in Figure 7.2. Note that GAIL is trained once and offline, while there might be multiple decision: the thresholds obtained by GAIL are used each time that a decision is made.

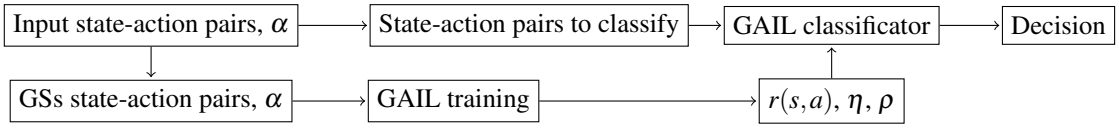


Fig. 7.4 Flow diagram for the online defense mechanism, where the training stage is explained in Figure 7.1 and the classification stage is explained in Figure 7.2. Note that the main difference with respect to the offline case in Figure 7.3 is that now GAIL is trained more than once, using state-action pairs collected from trusted GSs. Thus, the input state-action pairs contain both GSs state-action pairs to train GAIL and state-action pairs to classify. In this case, again, there might be multiple decisions; note, however that the thresholds obtained by GAIL are updated every time that GAIL is updated, whereas in the offline case the thresholds were fixed.

could cause that some GSs are detected as ASs. Hence, we could think of continuously training our defense mechanism by running the training phase using state-action pairs from trusted GSs when there are ASs: this means that the GAIL classifier is updated continuously taking into account the effect of the ASs over the GSs. Note that the trusted GSs used to train GAIL must be known a priori, and their behavior is used to classify the rest of sensors. This is what we denote as online defense mechanism, and a flow diagram describing it can be seen in Figure 7.4. Note that the main difference with respect to the offline case is that now, GAIL is updated during the execution phase using state-action pairs from the trusted GSs. We note that the online defense mechanism has a higher computational complexity, as now GAIL is updated several times; but it also explicitly takes into account the effect of the ASs over the GSs and hence, it should potentially provide better results.

7.2.3 Assumptions of our defense mechanisms

Both of our proposed defense mechanisms rely on the following three assumptions:

1. The interaction between each sensor and the central node can be modeled using the MDP / POMDP framework, where sequences of states / observations and actions can be obtained. We have access to a simulator of the system.
2. GAIL can be used to obtain a solution to the IRL problem of the GSs.
3. The ASs behavior will be different from the GSs behavior in terms of the IRL cost obtained.

Note that our three keys assumptions are fairly general, and hence, they allow dealing in a very general way with unknown ASs. The third assumption is the key one in order to detect ASs, and it is related to the fact that GAIL searches for a reward function that explains the state-action pairs. An AS would present a different behavior, and hence, the reward function obtained by GAIL can be used to discriminate between GSs and ASs.

Our approaches also present several weaknesses. The first one is that we do not take into account additional information about the ASs, as ours are general defense mechanisms. Obviously, the more we know about what

the attackers can do, the more ad hoc defense mechanism can be used in order to address a concrete attack. However, as Chapter 6 shows, ad hoc defense mechanisms could be exploited by minor attack variations. We try to present general defense mechanisms against intelligent attackers, valid against a broad range of attackers. Note that our defense mechanisms could also be used together with other ad hoc defense mechanisms in order to incorporate more information about the attacker. In our simulations, we will use our proposed defense mechanisms together with the backoff defense mechanism introduced in Chapter 6.

A second weakness is derived from the computational complexity associated to our methods. Even though GAIL is an efficient IRL method, nonetheless it has a significant resources consumption. Note that this is a problem that affects specially to the online defense mechanism, as it needs to train GAIL several times. Hence, there is a trade-off between a defense mechanism general enough and the computational load required. A final weakness is related to the online method. If the trusted GSs used to train GAIL are compromised, then GAIL may be exploited as well and our defense mechanism may fail.

However, our methods present several advantages. The first one is that they are very general methods, which require very little knowledge about the concrete setup. The second is that we train only using GSs, so our methods are able to detect, potentially, any AS that presents a different behavior from GSs. And finally, observe that we need not modeling the transition probability function, as GAIL only need access to a simulator of the system we want to defend.

7.3 Empirical results: the partially observable backoff attack

In order to validate our approach, we use the backoff attack described in Chapter 6, as it was a complex attack successfully learned by our DLA. Again, we consider a WSN in which a CSMA/CA access mechanism is used in the MAC layer. GSs follow the binary exponential backoff mechanism, while an AS may not respect such procedure: as Chapter 4 shows, this causes that the network throughput is not fairly distributed among all the sensors of the network. In Chapter 6, we presented a defense mechanism used against such an attack based on a statistical test [229], which however, is vulnerable against intelligent attackers that make use of a Deep Reinforcement Learning algorithm.

We proceed to test our defense mechanism in this environment. We test in three different scenarios, for $\{1,5,10\}$ ASs and 10 GSs, that is, in total we have $\{11,15,20\}$ sensors in our WSN. Again, we use the defense mechanism for the backoff attack proposed in [229] and described in Section 6.2.2, with a threshold $\lambda = 0.5$, and we set the rest of parameters to the same values as in Chapter 6. The backoff mechanism implementation follows the values of the 802.11 IEEE standard [111], where again, we simulate the backoff environment for $5 \cdot 10^5$ μs.

Remember from Chapter 6 that the ASs are solving a POMDP. Each timestep is a backoff time unit, in which the AS can transmit or not. Hence, each AS has two possible actions, which are transmitting in this time step or not. The local information of each sensor is the normalized time difference between the current time step and the last 5 transmissions, as well as whether these transmissions were successful or there were collisions. We add also a flag indicating whether the sensor has been discovered by the defense mechanism or not. We assume that each sensor is able to obtain the local information of the other sensors, and hence, we consider that the observation vector for each ASs is formed by the concatenation of the mean of the local information of GSs, the mean of the local information of the ASs and the local information of the AS. Thus, the ASs are intelligent and may communicate among them, following the attacker model proposed in Chapter 6, where we use the Mean-based Mean Embedding (CMME), as in Chapter 6, because it was shown to achieve good attack results. We train the ASs as shown in the Chapter 6 using 200 TRPO iterations. Finally, note that,

again, each AS has a reward of -1 if a GS transmits, and 0 otherwise. This causes that the ASs try to minimize the transmission rates of GSs.

We incorporate our proposed defense mechanism as an additional security layer to this backoff attack. The observation of the security mechanism is composed by the transmission times of a sensor. This observation, together with the action of the sensor, is the input to the reward NN estimator that is trained using GAIL. We use an NN with two hidden layers with 256 neurons each and hyperbolic tangent activation function. Note that the defense mechanism is also running a POMDP and approximating the observation by using only the last 5 transmissions. These observations take the place of the state in our previous exposition of our security mechanism, since as we indicated, this is a possible way to approximate a POMDP.

In the offline defense mechanism, for the training phase of GAIL, we use the trajectories from 100 episodes of $5 \cdot 10^5$ μs, where each trajectory contains between $200-300$ time steps each, that is, around $200-300$ state-action pairs. In each of these episodes, we consider that there are $\{11, 15, 20\}$ GSs, that is, sensors that respect the backoff mechanism. Then, we use GAIL to obtain an estimation of the reward function. We use 10 GAIL iterations, and in each of them, we update 3 times the generator and once the discriminator. We use 10 iterations as there was no further improvement by increasing the training time. We use the open code implementation of GAIL found in [59], with the rest of parameters set as default. Then, by using $\alpha = 0.05$, we obtained η and ρ using (7.1) and (7.2) respectively.

In the online defense mechanism, we use the same classifier as in the offline case, but the training phase differs. We use the state-action pairs of 5 trusted GSs and train GAIL at the end of each TRPO iteration of the attackers. Hence, each time that the ASs update their policy, GAIL updates the reward estimator. Note that the number of state-action pairs used by the online defense mechanisms is variable, as it depends on the ASs policy. Each time that we train GAIL, we use 10 GAIL iterations, and in each of them, we update 3 times the generator and once the discriminator.

We run our proposed defense mechanism together with the statistical test described in Chapter 6 for three cases: the baseline, in which only the statistical test is used, and the cases in which the statistical test is combined with the online and offline defense mechanisms respectively. Our defense mechanism classifier is run every time that there are 5 new state-actions pairs per sensor, and if a sensor is detected as an AS, is banned from the network. Note that a sensor could be banned from the network by either defense mechanism. Finally, for each value of the ASs, we simulate using 10 different seeds, and we use a discount factor $\gamma = 0.995$.

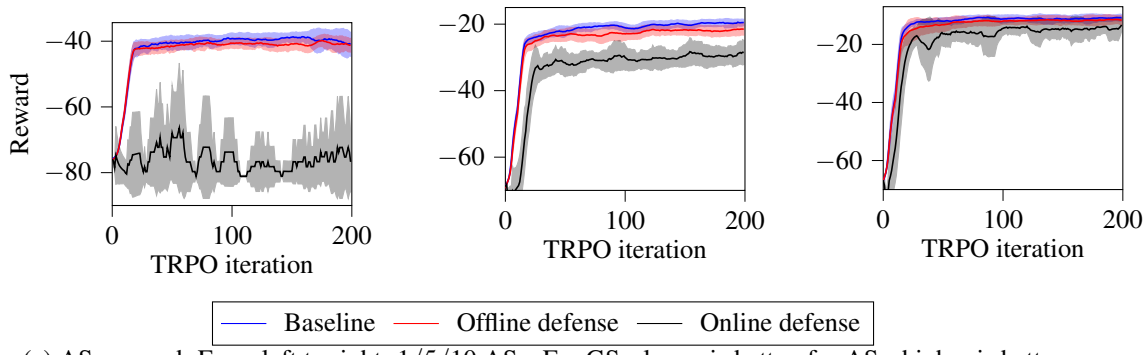
The simulation results averaged on the best 5 seeds for the defense mechanism can be observed in Figure 7.5 and Table 7.2, where we show the total reward, the proportion of sensors banned and the proportion of bits transmitted by each sensor, for all the cases tested. We can observe that the online and the offline cases outperform the baseline, specially the online defense mechanism: the differences are remarked in terms of proportion of agents discovered and proportion of bits transmitted. Note that, using our defense mechanisms, we are able to increase the detection of ASs, at the cost of increasing also the false positives on GSs. This increase in the detection capabilities of ASs has an impact on how the resources are distributed among sensors: the proportion of bits transmitted by ASs and GSs become similar when our defense mechanisms are used, specially when using the online method. Note that the improvement is very significant when having 1 and 5 ASs, while it is not in the 10 ASs case: as noted in Chapter 6, in the latter case, the proportion of ASs is high enough to overtake the WSN.

Note that the main drawback of our methods is the significant increase in the false positive probability: by using our defense mechanisms, there is a significant increase in the number of GSs which are considered ASs. This effect is expected, as the detection threshold is related to the behavior of GSs, and thus, it will detect GSs

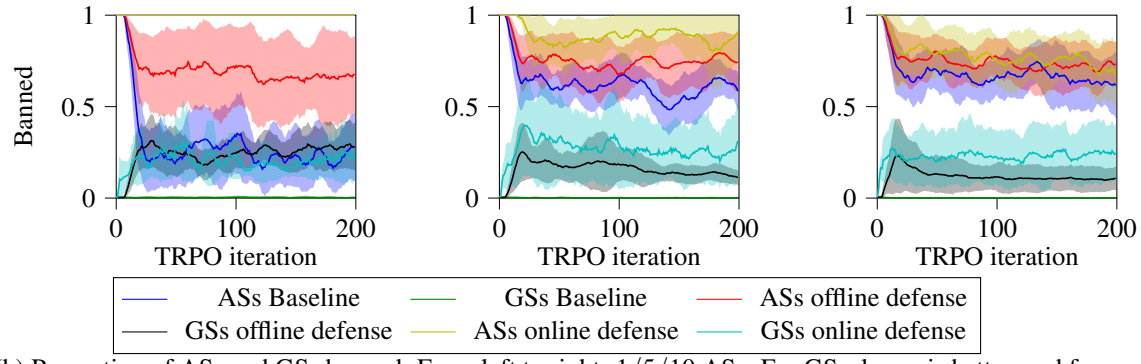
		1 AS	5 ASs	10 ASs
Reward	Baseline	-40.64 ± 8.44	-19.59 ± 3.73	-11.12 ± 3.06
	Offline	-41.34 ± 6.60	$\mathbf{-21.83 \pm 4.44}$	$\mathbf{-12.05 \pm 3.84}$
	Online	$\mathbf{-81.19 \pm 2.86}$	$\mathbf{-28.57 \pm 4.44}$	$\mathbf{-14.00 \pm 4.42}$
Proportion ASs banned	Baseline	0.19 ± 0.39	0.60 ± 0.26	0.59 ± 0.29
	Offline	$\mathbf{0.64 \pm 0.48}$	$\mathbf{0.73 \pm 0.27}$	$\mathbf{0.74 \pm 0.23}$
	Online	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.86 \pm 0.19}$	$\mathbf{0.73 \pm 0.28}$
Proportion GSs banned	Baseline	$\mathbf{0.002 \pm 0.01}$	$\mathbf{0.001 \pm 0.01}$	$\mathbf{0.001 \pm 0.01}$
	Offline	0.27 ± 0.23	0.17 ± 0.14	0.11 ± 0.12
	Online	0.17 ± 0.12	0.08 ± 0.12	0.18 ± 0.15
Proportion bits AS	Baseline	0.16 ± 0.05	0.11 ± 0.02	0.073 ± 0.01
	Offline	0.12 ± 0.05	$\mathbf{0.10 \pm 0.02}$	$\mathbf{0.070 \pm 0.01}$
	Online	$\mathbf{0.04 \pm 0.01}$	$\mathbf{0.07 \pm 0.02}$	$\mathbf{0.066 \pm 0.01}$
Proportion bits GS	Baseline	0.08 ± 0.01	0.04 ± 0.01	0.027 ± 0.01
	Offline	$\mathbf{0.09 \pm 0.01}$	$\mathbf{0.05 \pm 0.01}$	$\mathbf{0.030 \pm 0.01}$
	Online	$\mathbf{0.10 \pm 0.01}$	$\mathbf{0.06 \pm 0.01}$	$\mathbf{0.033 \pm 0.01}$

Table 7.2 Final results obtained for each number of ASs. The values were obtained averaging 100 episodes for each of the best 5 seeds after training. We show the mean final value, $\pm$ one standard deviation. Bold entries are the values with best mean, where a Welch test is used to detect whether means are significantly different for a significance level 0.01 with respect to the baseline. In case or total reward of the attacker, proportion of GSs banned and proportion of bits transmitted by ASs, lower is better. In case of proportion of ASs banned and proportion of bits transmitted by GSs, higher is better.

erroneously. We observe that our proposed mechanism based on GAIL does produce an improvement in the defense mechanism against an intelligent backoff attack, even though it is a generic defense mechanism.



(a) ASs reward. From left to right: $1/5/10$ ASs. For GSs, lower is better, for ASs, higher is better.



(b) Proportion of ASs and GSs banned. From left to right: $1/5/10$ ASs. For GSs, lower is better, and for ASs, higher is better.

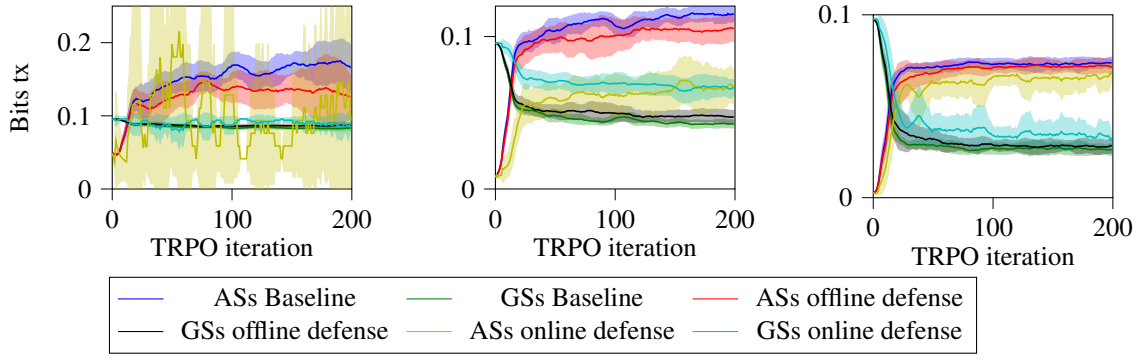


Fig. 7.5 Results evolution during training for the proposed backoff attack setup. In all figures, the horizontal axis correspond to the TRPO iteration. Note how both defense mechanisms improve in all measures the baseline, except for the increase in false alarm, i.e., the probability of banning GSs.

7.3.1 Analysis of our proposed defense mechanisms

We now study with more detail the reward distributions that arise in our backoff problem. In Figure 7.6, we plot an example of the empirical histogram of rewards obtained for each number of ASs during one episode. We show three distributions: the distribution of rewards with only GSs that is obtained after training GAIL, the distribution of GSs under attack and the distribution of the ASs after they have been trained. There are several interesting points to note:

- The distribution of the rewards of the GSs may change when there are ASs. This idea was our motivation to use the online defense mechanism, and we can observe that the GSs distribution is actually affected by the presence of ASs, as expected.
- The intelligent attacker tries to mimic the reward distributions of the GSs in order not to be detected. This means that its behavior is similar to the GSs behavior, which explains why the statistical test used as baseline is exploited by such an intelligent attacker. Note, however, that our proposed defense mechanisms, specially the online one, is effective against the attack, except for the 10 ASs case as explained before.
- The decision threshold has a strong dependency on the GAIL training dataset, as for different seeds of the same setup, we obtained different threshold values. This means that the reward function $r(s,a)$ learned for different seeds is different due to the GAIL training datasets being different. Thus, further research is needed in order to improve this step, which may include testing for different classifier methods.

7.4 Conclusions

In this Chapter, we have given a step forward towards intelligent defense mechanisms that are able to cope with the intelligent attackers presented in Chapter 6. Namely, we have proposed a defense mechanism based on IRL methods that is able to detect unknown attacks, i.e., it operates under incomplete (and imperfect) information settings. Our main idea consists in using IRL in order to learn the behavior of GSs and use that information to detect ASs, as the behavior of ASs differs from the behavior of GSs. We have validated our idea in the backoff attack of Chapter 6, and we have seen that our proposed method increases the false alarm probability but also significantly increases the detection of ASs and enforces an even distribution of the network throughput. We remark that the main contribution of our defense mechanism is that it does not need a priori knowledge about the attack characteristics. Nowadays, most defense mechanisms are ad hoc designed against concrete attacks, but the advances in the field of machine learning, and specifically in RL, have brought attacker architectures that are able to learn to exploit the vulnerabilities of a network. In order to face such intelligent attacks, the mechanism we propose is a step forward towards defense mechanisms able to cope with such attackers.

As we noted in the introduction of this Chapter, what we propose here is a method to detect intelligent attackers. As we have shown in Chapters 5 and 6, intelligent attackers are able to overcome static defense mechanisms, that is, defense mechanisms whose behavior does not evolve with time, by means of solving or approximating a control problem. In this Chapter, we make use of control techniques as well in the defense mechanism, which, as our simulations show, are effective detecting such intelligent attackers. Specifically, our defense mechanism allows that the WSN resources are evenly distributed among sensors, while the ASs in this setup try to concentrate the network resources.

In the first version of our backoff problem, presented in Chapter 4, we assumed that the defense mechanism was able to observe the mixed actions of the ASs. Chapter 5 showed the implications of what would happen if the

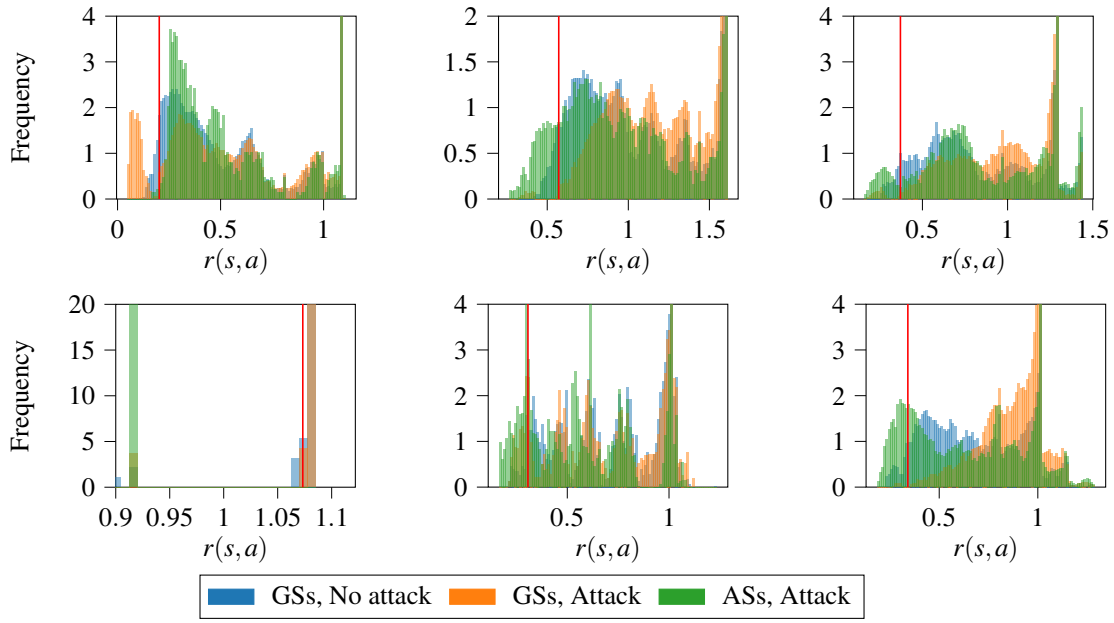


Fig. 7.6 Histogram of rewards compared for GSs and ASs, for one seed. From left to right: $1/5/10$ ASs. Top are for offline defense, bottom for online. Peaks have been cut off for clarity. Blue is the reward histogram when only GSs are present, that is, for training, orange is for GSs under attack and green is for ASs during attack. The red line is the decision threshold obtained during training. Note how sometimes, the ASs are able to behave in such a way that they mimic the reward shape of the GSs, but other times they do not. Also, note how the GSs distribution changes if there is an attack: this explains why the online defense mechanism performs significantly better.

defense mechanism only observed the actions realizations, while Chapter 6 and this show some consequences of observing only partially the actions of the ASs. Note that there has been an increase in the environment complexity as we have included more realistic assumptions in our problems. Also, observe that in Chapter 4, the defense mechanism had an optimal action against the ASs actions, that is, we focused on obtaining the best responses against the ASs actions, but in this Chapter, we have focused on detecting the ASs and banning them. As mentioned in the introduction to this Chapter, if we want to obtain a more complex policy for the defense mechanism against intelligent ASs, we should use incomplete information dynamic game theory tools, which is left for future research. As shown in [97], there is a significant amount of work towards achieving efficient algorithms to learn complex games: this research direction, as this work emphasizes, is of utmost importance to the WSN field.

Chapter 8

Conclusions and future research

8.1 Conclusions

Now, it is moment to look back to the work developed in this thesis in order to draw some conclusions in this Section, and propose ways in which our research could be extended in the next Section. Our work has dealt with security setups in WSNs in which the agents made decisions sequentially, and they may have imperfect and incomplete information. We have considered several variations of two main problems, the backoff and the SSDF attacks, and we have studied for different conditions of the attackers and the defense mechanism: the main points are contained in Table 8.1, which we repeat once more for the sake of completeness.

In Chapter 3, we have shown that current algorithms for learning repeated games do not satisfactorily address the discounted payoff case, in spite of being a problem of special interest in practical terms. We have noted that the discounted payoff presents at least two particularities that introduce significant differences with respect to the average payoff, namely, that the payoff variance depends on the discount factor and the total payoff is not evenly assigned in all stages. The latter condition means that, in practical cases, we need learning algorithms that learn fast, otherwise, a bad strategy in the first stages may cause that the player obtains a poor total payoff regardless of its behavior in the subsequent stages. Another way to approach this problem is enforcing security conditions, by which the player is conservative and avoids actions that provide her with a payoff lower than a certain threshold. Our LEWIS algorithm is based on this idea, it is designed to work in incomplete information settings and its empirical performance is good in self-play, against other learning algorithms and even against a minmax player, according to the results of Chapter 3. We also propose an algorithm to negotiate repeated games equilibria, which we denote as CA. It is a fully distributed, incomplete information algorithm that is based on several players sampling the action spaces, proposing candidate equilibrium points and distributedly selecting a Pareto-efficient one. As we have shown, it may take advantage of the Folk Theorem and obtains good payoffs both for NE and CE concepts.

Chapter 4 thoroughly studies the consequences of the backoff attacks and proposes a CSMA/CA game, which we solve using static and dynamic game theory tools. Our simulations show that there is a gain in terms of payoffs by using repeated games, as the Folk Theorem does help finding better equilibrium points, although this comes at the cost of a higher computational complexity. This Chapter is a first approach to our security situations, in which we make several assumptions in order to facilitate the analysis, such as perfect and complete information for all players. When these assumptions fail, as often happens in realistic environments, we need to use more advanced tools. Hence, although repeated games can be solved in a reasonable time, they

Chapter	CSMA/CA	CSS	Player	Information	Observation (A/S)	Behavior
4	Yes	No	Attack Defense	Complete Complete	Mixed / - Mixed / -	Static Static
5	Yes	Yes	Attack Defense	Complete Incomplete	- / State Realization / -	Dynamic Static
6	Yes	Yes	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / -	Dynamic Static
7	Yes	No	Attack Defense	Incomplete Incomplete	Realization / Observation Realization / Observation	Dynamic Dynamic

Table 8.1 Table comparing the different setups used in Chapters 4-7. CSMA/CA. i.e., the backoff attack, and CSS, i.e., the SSDF attack, denote whether each of these setups is used in the Chapter. Information denotes whether each player knows the target of the other player (Complete) or not (Incomplete). Observation refers to what each agent observes with respect to the actions / states of the other players: regarding actions, they observe the mixed actions or the actions realizations, and regarding states, they observe the state or an observation of the rest of players: this is related to having perfect or imperfect information. Behavior refers to whether the player adapts its behavior with time or not.

present limitations against complex, realistic attacks, as Chapters 5 and 6 show. Given the fact that the hardware of current attackers allows them to use complex, dynamic behavior, it is important highlighting that repeated games may not be the best tool against such attackers. Also, note that if attackers present a dynamic behavior, they can exploit a static behavior defense mechanism. Chapter 5 has been devoted to this point, where we also assumed that the defense mechanism had incomplete and imperfect information. We present an optimal attack against an mechanism, which is a static mechanism, and to counter this situation, we have developed a defense mechanism based on an OCSVM to detect intelligent attackers. Also in Chapter 5, we have derived a very efficient sequential test which makes use of prior information. Our results show that this test is fast in deciding and presents a lower error than the counting rule and SPRT, and hence, the use of prior information is of interest when it comes to testing in WSN setups.

A key Chapter in our development is Chapter 6, as it shows the capabilities that Deep RL algorithms have when it comes to obtaining attack strategies against defense mechanisms, even when the attackers have imperfect and incomplete information of the defense mechanism. Note that the main disadvantage of Deep RL methods, today, is that they are sample inefficient, and hence, they need many samples to learn. In real life problems, an attacker should learn as fast as possible, and hence, increasing the sample efficiency is important in order to obtain better attackers. Note that this is already an active area or research in the Deep RL field [232].

A crucial idea in our work has been the asymmetry: Chapters 5 and 6 deal with the case in which the attacker is able to adapt dynamically, but the defense mechanism is not. Note that this makes our proposed RL attackers such a threat: most current defense mechanisms are static. Hence, the dynamic defense mechanism of Chapter 7, based on IRL tools, is but a first step towards defense mechanisms able to deal with such attackers.

8.2 Future research

Now, we proceed to present some of the most interesting future research lines that arise from our work in this thesis. First, let us focus on LEWIS algorithm. We have considered that the actions are chosen attending to the rewards that these actions yielded to the agent in the past, as shown by the action selection block described in Chapter 3.3.1. However, there are other action selection strategies that could be followed in order to take

advantage of different situations, such as the ones explained in [185]. Another promising approach that has appeared recently is described in [17]: it is based on using gradient methods in order to solve a differential game for N_p players, and these ideas may be also an interesting alternative action selection block to explore. As the Folk Theorem shows, in discounted repeated games, there might be payoffs which are better for all players [146], and LEWIS allows obtaining them when they satisfy certain security conditions. Another research line possible consists in investigating different security condition definitions and whether they facilitate reaching a better payoff for all players or not. For instance, as we have noted, a similar work using a different security condition is [50]; hence, it may be interesting comparing this security condition to ours in a variety of setups and check which advantages and disadvantages has each of them.

Our CA algorithm can also be extended in several ways. A first one would be improving the sampling method: as we point out in Chapter 3, the performance of CA strongly depends on the sampling method chosen. In this work, we have compared an equispaced sampling with a random sampling and an intelligent sampling method based on a non-convex optimization algorithm. Of all these methods, the non-convex optimization algorithm provided the best performance, hence, it might be reasonable to test other sampling schemes based on different non-convex optimization methods such as [222] or [120]. Also, it would be interesting extending it to different strategies than UNR, such as grim trigger, tit-for-that or forgiving strategies [146], [103]: observe that the payoff regions depend on the chosen strategy and thus, some strategies may provide higher payoffs than others. And finally, another point of interest would be extending CA to work in the case in which different strategies are used by each player: this increases the complexity of finding an equilibrium point, but also may provide each player with a strategy adapted to its computational capabilities, and thus, allow more complex strategies which provide a better payoff on agents with large computational capabilities, and less complex strategies on agents with restricted computational capabilities. Note that this would be an asymmetric situation, which has been a case of special interest on this thesis.

In Chapter 5, we have introduced a very efficient sequential test which made use of prior information by means of using Beta priors, as they can be updated in a very efficient way using properties of the Gamma function. We note that this is a significant advance, as one of the problems related to Bayesian updates is the complex update calculations when there are no closed form expressions [119]. A point of interest that arises is deepening into the approximation capabilities of the Beta functions: note that we use them as basis to approximate possibly complex prior functions. Hence, an important question to pose is whether Beta distributions, when used as basis functions, are able to approximate any distribution in the range $[0, 1]$ or not, and to which degree of accuracy. Also, a related question to this one is how to efficiently obtain the weights and parameters of the linear approximation using Beta basis functions that best approximates a given distribution. And finally, we also observe that, in dynamic games, Bayesian equilibrium requires that, at each stage, a belief over the types of the rest of the players is updated [76]. The algorithm that we have used to update the Bayes Factor sequential test may be an interesting choice to update the beliefs in an incomplete information game, and hence, it may be the basis for an efficient algorithm to obtain Bayes equilibria.

We conclude with our intelligent defense mechanism from Chapter 7: there are several possible ways in which this mechanism could be extended. One of them would consist in using a different classifier, other than the Binomial test that we have used. There are several possible candidates, one of them being a threshold on the Kullback-Leibler divergence, in a similar fashion to the concept of trust region used in TRPO [197], but while TRPO uses the KL divergence to avoid having policies too different, we would use it to discriminate between too different reward histograms. Another possible option would be using an OCSVM, introduced in Chapter 5, as they are able to detect samples different from the ones with which they were trained. Also, note that the parameters used in the classifier have an impact on the defense mechanism performance: this is

another point that needs more assessment in order to achieve a good tradeoff between the detection capabilities of the system and its computational requirements. But maybe the most interesting line of research against intelligent attackers as the one presented in Chapter 6 is the use of dynamic game theory tools. As we noted in Chapter 2, in the general case, our game is of incomplete and imperfect information, and hence, intractable computationally, although there is work ongoing in that direction [97]. However, as Chapters 6 and 7 show, it is possible that the recent advances in Deep RL may help in finding strategies good enough for all the players involved in such games. Note that researching in this direction is highly promising, as it may allow using the game theory framework, which is specially suitable to security problems, and also may allow obtaining solutions computationally tractable which are a good approximation for a certain security setup. What is clear from our work is that current Deep RL tools pose a significant threat to many WSN defense mechanism, the question is, will RL tools also be the solution to this threat?

References

[1] Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM.

[2] Abdallah, S. and Kaisers, M. (2016). Addressing environment non-stationarity by repeating q-learning updates. *The Journal of Machine Learning Research*, 17(1):1582–1612.

[3] Abdallah, S. and Lesser, V. (2008). A multiagent reinforcement learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research*, 33:521–549.

[4] Abreu, D. (1988). On the theory of infinitely repeated games with discounting. *Econometrica: Journal of the Econometric Society*, pages 383–396.

[5] Agah, A. and Das, S. K. (2007). Preventing dos attacks in wireless sensor networks: A repeated game theory approach. *IJ Network Security*, 5(2):145–153.

[6] Akchurina, N. (2010). *Multi-agent reinforcement learning algorithms.* PhD thesis, University of Paderborn.

[7] Akkarajitsakul, K., Hossain, E., Niyato, D., and Kim, D. I. (2011). Game theoretic approaches for multiple access in wireless networks: A survey. *IEEE Communications Surveys & Tutorials*, 13(3):372–395.

[8] Alpcan, T. and Başar, T. (2010). *Network security: A decision and game-theoretic approach*. Cambridge University Press.

[9] Alsheikh, M. A., Lin, S., Niyato, D., and Tan, H.-P. (2014). Machine learning in wireless sensor networks: Algorithms, strategies, and applications. *IEEE Communications Surveys & Tutorials*, 16(4):1996–2018.

[10] Anderson, T. W. (1962). On the distribution of the two-sample cramer-von mises criterion. *The Annals of Mathematical Statistics*, pages 1148–1159.

[11] Arasteh, H., Hosseinnezhad, V., Loia, V., Tommasetti, A., Troisi, O., Shafie-Khah, M., and Siano, P. (2016). Iot-based smart cities: a survey. In *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*, pages 1–6. IEEE.

[12] Aref, M. A., Jayaweera, S. K., and Machuzak, S. (2017). Multi-agent reinforcement learning based cognitive anti-jamming. In *Wireless Communications and Networking Conference (WCNC), 2017 IEEE*, pages 1–6. IEEE.

[13] Aumann, R. J. (1974). Subjectivity and correlation in randomized strategies. *Journal of mathematical Economics*, 1(1):67–96.

[14] Aumann, R. J. and Hart, S. (1992). *Handbook of game theory with economic applications*, volume 2. Elsevier.

[15] Aumann, R. J., Maschler, M., and Stearns, R. E. (1995). *Repeated games with incomplete information*. MIT press.

[16] Avis, D., Rosenberg, G. D., Savani, R., and Von Stengel, B. (2010). Enumeration of nash equilibria for two-player games. *Economic Theory*, 42(1):9–37.

[17] Balduzzi, D., Racaniere, S., Martens, J., Foerster, J., Tuyls, K., and Graepel, T. (2018). The mechanics of n-player differentiable games. *arXiv preprint arXiv:1802.05642*.

[18] Banerjee, B. and Peng, J. (2004). Performance bounded reinforcement learning in strategic interactions. In *Proceedings of the 19th national conference on Artifical intelligence*, pages 2–7.

[19] Basar, T. and Olsder, G. J. (1999). *Dynamic noncooperative game theory*, volume 23. SIAM.

[20] Basseville, M., Nikiforov, I. V., et al. (1993). *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs.

[21] Bayraktaroglu, E., King, C., Liu, X., Noubir, G., Rajaraman, R., and Thapa, B. (2013). Performance of ieee 802.11 under jamming. *Mobile Networks and Applications*, 18(5):678–696.

[22] Begum, K. and Dixit, S. (2016). Industrial wsn using iot: A survey. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 499–504. IEEE.

[23] Benedetto, F., Tedeschi, A., Giunta, G., and Coronas, P. (2016). Performance improvements of reputation-based cooperative spectrum sensing. In *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on*, pages 1–6. IEEE.

[24] Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840.

[25] Bertsekas, D. P. (2005). *Dynamic programming and optimal control*, volume 1. Athena Scientific.

[26] Bertsekas, D. P. (2007). *Dynamic programming and optimal control*, volume 2. Athena Scientific.

[27] Bhadre, P. and Gothawal, D. (2014). Detection and blocking of spammers using spot detection algorithm. In *Networks & Soft Computing (ICNSC), 2014 First International Conference on*, pages 97–101. IEEE.

[28] Bianchi, G. (2000). Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal on selected areas in communications*, 18(3):535–547.

[29] Bizanis, N. and Kuipers, F. A. (2016). Sdn and virtualization solutions for the internet of things: A survey. *IEEE Access*, 4:5591–5606.

[30] Bloem, M. and Bambos, N. (2014). Infinite time horizon maximum causal entropy inverse reinforcement learning. In *53rd IEEE Conference on Decision and Control*, pages 4911–4916. IEEE.

[31] Bloembergen, D., Kaisers, M., and Tuyls, K. (2010). Lenient frequency adjusted q-learning. In *Proc. of 22nd Belgium-Netherlands Conf. on Artif. Intel.*

[32] Bowling, M. (2005). Convergence and no-regret in multiagent learning. In *Advances in neural information processing systems*, pages 209–216.

[33] Bowling, M. and Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250.

[34] Brown, G. W. (1951). Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1):374–376.

[35] Brown, N. and Sandholm, T. (2017). Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, page eaao1733.

[36] Buehrer, R. M. (2006). *Synthesis Lectures on Communications*, volume 1, chapter Code division multiple access (CDMA), pages 1–192. Morgan & Claypool Publishers.

[37] Cagalj, M., Ganeriwal, S., Aad, I., and Hubaux, J.-P. (2005). On selfish behavior in CSMA/CA networks. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 4, pages 2513–2524. IEEE.

[38] Cannady, J. (2000). Next generation intrusion detection: Autonomous reinforcement learning of network attacks. In *Proceedings of the 23rd national information systems security conference*, pages 1–12.

[39] Chakraborty, D. and Stone, P. (2014). Multiagent learning in the presence of memory-bounded agents. *Autonomous agents and multi-agent systems*, 28(2):182–213.

[40] Chatterjee, K., Chmelík, M., and Tracol, M. (2016). What is decidable about partially observable markov decision processes with ω-regular objectives. *Journal of Computer and System Sciences*, 82(5):878–911.

[41] Chen, R., Park, J.-M., and Bian, K. (2008). Robust distributed spectrum sensing in cognitive radio networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1876–1884. IEEE.

[42] Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103.

[43] Cichoń, K., Kliks, A., and Bogucka, H. (2016). Energy-efficient cooperative spectrum sensing: A survey. *IEEE Communications Surveys & Tutorials*, 18(3):1861–1886.

[44] Ciuonzo, D., De Maio, A., and Rossi, P. S. (2015). A systematic framework for composite hypothesis testing of independent bernoulli trials. *IEEE Signal Processing Letters*, 22(9):1249–1253.

[45] Ciuonzo, D. and Rossi, P. S. (2014). Decision fusion with unknown sensor detection probability. *IEEE Signal Processing Letters*, 21(2):208–212.

[46] Ciuonzo, D. and Rossi, P. S. (2018). Dechade: Detecting slight changes with hard decisions in wireless sensor networks. *International Journal of General Systems*, 47(5):535–548.

[47] Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998:746–752.

[48] Conitzer, V. and Sandholm, T. (2007). 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.

[49] Crandall, J. W. (2012). Just add pepper: extending learning algorithms for repeated matrix games to repeated markov games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 399–406. International Foundation for Autonomous Agents and Multiagent Systems.

[50] Crandall, J. W. and Goodrich, M. A. (2011). Learning to compete, coordinate, and cooperate in repeated games using reinforcement learning. *Machine Learning*, 82(3):281–314.

[51] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

[52] Damer, S. and Gini, M. (2017). Safely using predictions in general-sum normal form games. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 924–932. International Foundation for Autonomous Agents and Multiagent Systems.

[53] Das, S. K. and Ho, J.-W. (2011). A synopsis on node compromise detection in wireless sensor networks using sequential analysis (invited review article). *Computer Communications*, 34(17):2003–2012.

[54] Daskalakis, C., Goldberg, P. W., and Papadimitriou, C. H. (2009). The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259.

[55] De Cote, E. M., Lazaric, A., and Restelli, M. (2006). Learning to cooperate in multi-agent social dilemmas. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 783–785. ACM.

[56] De Cote, E. M. and Littman, M. L. (2008). A polynomial-time nash equilibrium algorithm for repeated stochastic games. In *Proceedings of the 24th Conference Annual Conference on Uncertainty in Artificial Intelligence*, pages 419–426.

[57] Demirkol, I., Ersoy, C., and Alagoz, F. (2006). Mac protocols for wireless sensor networks: a survey. *IEEE Communications Magazine*, 44(4):115–121.

[58] Dermed, M. and Charles, L. (2013). *Value methods for efficiently solving stochastic games of complete and incomplete information*. PhD thesis, Georgia Institute of Technology.

[59] Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). Openai baselines. https://github.com/openai/baselines.

[60] Diaconis, P. and Ylvisaker, D. (1979). Conjugate priors for exponential families. *The Annals of statistics*, 7(2):269–281.

[61] Dibangoye, J. S., Amato, C., Buffet, O., and Charpillet, F. (2016). Optimally solving dec-pomdps as continuous-state mdps. *Journal of Artificial Intelligence Research*, 55:443–497.

[62] Dong, P., Du, X., Zhang, H., and Xu, T. (2016). A detection method for a novel ddos attack against sdn controllers by vast new low-traffic flows. In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–6. IEEE.

[63] Dougherty, E. R. (1999). *Random processes for image and signal processing*. SPIE Optical Engineering Press.

[64] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338.

[65] Elidrisi, M., Johnson, N., Gini, M., and Crandall, J. (2014). Fast adaptive learning in repeated stochastic games by game abstraction. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1141–1148. International Foundation for Autonomous Agents and Multiagent Systems.

[66] Entacher, K. (1998). Bad subsequences of well-known linear congruential pseudorandom number generators. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):61–70.

[67] Enz, C. C., El-Hoiydi, A., Decotignie, J.-D., and Peiris, V. (2004). Wisenet: an ultralow-power wireless sensor network solution. *Computer*, 37(8):62–70.

[68] Etesami, S. R. and Başar, T. (2019). Dynamic games in cyber-physical security: An overview. *Dynamic Games and Applications*, pages 1–30.

[69] Fernandez, M. F. and Aridgides, T. (2003). Measures for evaluating sea mine identification processing performance and the enhancements provided by fusing multisensor/multiprocess data via an m-out-of-n voting scheme. In *Detection and Remediation Technologies for Mines and Minelike Targets VIII*, volume 5089, pages 425–437. International Society for Optics and Photonics.

[70] Filar, J. and Vrieze, K. (2012). *Competitive Markov decision processes*. Springer Science & Business Media.

[71] Finn, C., Christiano, P., Abbeel, P., and Levine, S. (2016). A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*.

[72] Fragkiadakis, A. G., Tragos, E. Z., and Askoxylakis, I. G. (2013). A survey on security threats and detection techniques in cognitive radio networks. *IEEE Communications Surveys & Tutorials*, 15(1):428–445.

[73] Friesz, T. L. (2010). *Dynamic optimization and differential games*, volume 135. Springer Science & Business Media.

[74] Fudenberg, D. and Maskin, E. (1986). The folk theorem in repeated games with discounting or with incomplete information. *Econometrica: Journal of the Econometric Society*, pages 533–554.

[75] Fudenberg, D. and Maskin, E. (1991). On the dispensability of public randomization in discounted repeated games. *Journal of Economic Theory*, 53(2):428–438.

[76] Fudenberg, D. and Tirole, J. (1991). *Game theory*. MIT press Cambridge, MA.

[77] Gara, F., Saad, L. B., and Ayed, R. B. (2017). An intrusion detection system for selective forwarding attack in ipv6-based mobile wsns. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 276–281. IEEE.

[78] Ghazvini, M., Movahedinia, N., Jamshidi, K., and Moghim, N. (2013). Game theory applications in CSMA methods. *IEEE Communications Surveys & Tutorials*, 15(3):1062–1087.

[79] Gilboa, I. and Zemel, E. (1989). Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93.

[80] Goldberg, P. W. and Papadimitriou, C. H. (2006). Reducibility among equilibrium problems. *Proceedings of the 38th annual ACM symposium on Theory of computing*, pages 61–70.

[81] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

[82] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[83] Goyal, D. and Tripathy, M. R. (2012). Routing protocols in wireless sensor networks: a survey. *Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on*, pages 474–480.

[84] Greenwald, A., Hall, K., and Serrano, R. (2003). Correlated q-learning. In *ICML*, volume 3, pages 242–249.

[85] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2017). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232.

[86] Gwon, Y., Dastangoo, S., Fossa, C., and Kung, H. (2013). Competing mobile network game: Embracing antijamming and jamming strategies with reinforcement learning. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 28–36. IEEE.

[87] Han, G., Xiao, L., and Poor, H. V. (2017). Two-dimensional anti-jamming communication based on deep reinforcement learning. In *Proceedings of the 42nd IEEE International Conference on Acoustics, Speech and Signal Processing,*.

[88] Hansen, E. A., Bernstein, D. S., and Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, pages 709–715.

[89] Hansen, K. A., Ibsen-Jensen, R., and Miltersen, P. B. (2014). The complexity of solving reachability games using value and strategy iteration. *Theory of Computing Systems*, 55(2):380–403.

[90] Hart, S. and Mas-Colell, A. (2000). A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150.

[91] Hart, S. and Mas-Colell, A. (2013). *Simple adaptive strategies: from regret-matching to uncoupled dynamics*, volume 4. World Scientific.

[92] Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *Proc. of Conf. on Artificial Intelligence, AAAI, 2015.*

[93] Hauskrecht, M. (1997). Incremental methods for computing bounds in partially observable markov decision processes. In *AAAI/IAAI*, pages 734–739. Citeseer.

[94] Hauskrecht, M. (2000). Value-function approximations for partially observable markov decision processes. *Journal of artificial intelligence research*, 13:33–94.

[95] Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

[96] Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier.

[97] Hernandez-Leal, P., Kaisers, M., Baarslag, T., and de Cote, E. M. (2017a). A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*.

[98] Hernandez-Leal, P., Munoz de Cote, E., and Sucar, L. E. (2014). A framework for learning and planning against switching strategies in repeated games. *Connection Science*, 26(2):103–122.

[99] Hernandez-Leal, P., Zhan, Y., Taylor, M. E., Sucar, L. E., and de Cote, E. M. (2017b). Efficiently detecting switches against non-stationary opponents. *Autonomous Agents and Multi-Agent Systems*, 31(4):767–789.

[100] Hernandez-Leal, P., Zhan, Y., Taylor, M. E., Sucar, L. E., and de Cote, E. M. (2017c). An exploration strategy for non-stationary opponents. *Autonomous Agents and Multi-Agent Systems*, 31(5):971–1002.

[101] Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573.

[102] Ho, J.-W., Wright, M., and Das, S. K. (2011). Fast detection of mobile replica node attacks in wireless sensor networks using sequential hypothesis testing. *IEEE transactions on mobile computing*, 10(6):767–782.

[103] Hoang, D. T., Lu, X., Niyato, D., Wang, P., Kim, D. I., and Han, Z. (2015). Applications of repeated games in wireless networks: A survey. *IEEE Communications Surveys & Tutorials*, 17(4):2102–2135.

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

[105] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

[106] Hu, Y., Gao, Y., and An, B. (2015). Multiagent reinforcement learning with unshared value functions. *IEEE transactions on cybernetics*, 45(4):647–662.

[107] Hu, Z., Zhang, J., and Wang, X. A. (2016). Intrusion detection for wsn based on kernel fisher discriminant and svm. In *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 197–208. Springer.

[108] Hull, T. E. and Dobell, A. R. (1962). Random number generators. *SIAM review*, 4(3):230–254.

[109] Hunter, J. K. and Nachtergaele, B. (2001). *Applied analysis*. World Scientific Publishing.

[110] Hüttenrauch, M., Šošić, A., and Neumann, G. (2019). Deep reinforcement learning for swarm systems. *Journal of Machine Learning Research*, 20(54):1–31.

[111] IEEE (2016). IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. pages 1–3534.

[112] Jaynes, E. T. (1957). Information theory and statistical mechanics. *Physical review*, 106(4):620–630.

[113] Jeffreys, H. (1935). Some tests of significance, treated by the theory of probability. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 31, pages 203–222. Cambridge University Press.

[114] Jeffreys, H. (1961). *Theory of Probability*. Oxford University Press.

[115] Kailkhura, B., Brahma, S., and Varshney, P. K. (2014). On the performance analysis of data fusion schemes with byzantines. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 7411–7415. IEEE.

[116] Kaisers, M. and Tuyls, K. (2010). Frequency adjusted multi-agent q-learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 309–316. International Foundation for Autonomous Agents and Multiagent Systems.

[117] Kalai, E. (1977). Proportional solutions to bargaining situations: interpersonal utility comparisons. *Econometrica: Journal of the Econometric Society*, pages 1623–1630.

[118] Kalai, E. and Smorodinsky, M. (1975). Other solutions to nash's bargaining problem. *Econometrica: Journal of the Econometric Society*, pages 513–518.

[119] Kass, R. E. and Raftery, A. E. (1995). Bayes factors. *Journal of the american statistical association*, 90(430):773–795.

[120] Kawaguchi, K., Kaelbling, L. P., and Lozano-Pérez, T. (2015). Bayesian optimization with exponential convergence. In *Advances in neural information processing systems*, pages 2809–2817.

[121] Kay, S. M. (1993). *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[122] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR*.

[123] Konorski, J. (2006). A game-theoretic study of csma/ca under a backoff attack. *IEEE/ACM Transactions on Networking (TON)*, 14(6):1167–1178.

[124] Kramer, G. (1998). *Directed information for channels with feedback*. Hartung-Gorre.

[125] Lai, T. L. (1988). Nearly optimal sequential tests of composite hypotheses. *The Annals of Statistics*, 16(2):856–886.

[126] Lai, T. L. (2001). Sequential analysis: Some classical problems and new challenges. *Statistica Sinica*, 11(2):303–351.

[127] Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Perolat, J., Silver, D., Graepel, T., et al. (2017). A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4193–4206.

[128] Le Treust, M. and Lasaulce, S. (2010). A repeated game formulation of energy-efficient decentralized power control. *IEEE Transactions on Wireless Communications*, 9(9):2860–2869.

[129] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[130] L'Ecuyer, P. and Simard, R. (1999). Beware of linear congruential generators with multipliers of the form a=±2 q±2 r. *ACM Transactions on Mathematical Software (TOMS)*, 25(3):367–374.

[131] Li, G., Liu, X., and Wang, C. (2010). A sequential mesh test based selective forwarding attack detection scheme in wireless sensor networks. In *Networking, Sensing and Control (ICNSC), 2010 International Conference on*, pages 554–558. IEEE.

[132] Li, J., Feng, Z., Wei, Z., Feng, Z., and Zhang, P. (2014). Security management based on trust determination in cognitive radio networks. *EURASIP Journal on Advances in Signal Processing*, 2014(1):48.

[133] Li, S., Da Xu, L., and Zhao, S. (2015). The internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259.

[134] Li, Y., Quevedo, D. E., Dey, S., and Shi, L. (2017). Sinr-based dos attack on remote state estimation: A game-theoretic approach. *IEEE Transactions on Control of Network Systems*, 4(3):632–642.

[135] Liang, X. and Xiao, Y. (2012). Game theory for network security. *IEEE Communications Surveys & Tutorials*, 15(1):472–486.

[136] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[137] Lin, P., Qiao, C., and Wang, X. (2004). Medium access control with a dynamic duty cycle for sensor networks. *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, 3:1534–1539.

[138] Lin, Y., Chen, B., and Varshney, P. K. (2005). Decision fusion rules in multi-hop wireless sensor networks. *IEEE Transactions on Aerospace and Electronic Systems*, 41(2):475–488.

[139] Lisman, J. and Zuylen, M. v. (1972). Note on the generation of most probable frequency distributions. *Statistica Neerlandica*, 26(1):19–23.

[140] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Elsevier.

[141] Littman, M. L. and Stone, P. (2005). A polynomial-time nash equilibrium algorithm for repeated games. *Decision Support Systems*, 39(1):55–66.

[142] Littman, M. L. and Sutton, R. S. (2002). Predictive representations of state. *Advances in neural information processing systems*, pages 1555–1561.

[143] Liu, S., Liu, Q., Gao, J., and Guan, J. (2011). Attacker-exclusion scheme for cooperative spectrum sensing against ssdf attacks based on accumulated suspicious level. In *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2011 IEEE International Conference on*, pages 239–243. IEEE.

[144] Lovejoy, W. S. (1991). Computationally feasible bounds for partially observed markov decision processes. *Operations research*, 39(1):162–175.

[145] Luo, Y., Szidarovszky, F., Al-Nashif, Y., and Hariri, S. (2010). Game theory based network security. *Journal of Information Security*, pages 41–44.

[146] Mailath, G. J. and Samuelson, L. (2006). *Repeated games and reputations: long-run relationships*. Oxford university press.

[147] Malone, D., Duffy, K., and Leith, D. (2007). Modeling the 802.11 distributed coordination function in nonsaturated heterogeneous conditions. *IEEE/ACM Transactions on networking*, 15(1):159–172.

[148] Manshaei, M. H., Zhu, Q., Alpcan, T., Bacşar, T., and Hubaux, J.-P. (2013). Game theory meets network security and privacy. *ACM Computing Surveys (CSUR)*, 45(3):25.

[149] McKelvey, R. D. and McLennan, A. (1996). Computation of equilibria in finite games. *Handbook of computational economics*, 1:87–142.

[150] Mertens, J.-F., Sorin, S., and Zamir, S. (2015). *Repeated games*. Cambridge University Press.

[151] Min, A. W., Shin, K. G., and Hu, X. (2009). Attack-tolerant distributed sensing for dynamic spectrum access networks. In *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on*, pages 294–303. IEEE.

[152] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

[153] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

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

[155] Mohammad, F. R., Ciuonzo, D., and Mohammed, Z. A. K. (2018). Mean-based blind hard decision fusion rules. *IEEE Signal Processing Letters*, 25(5):630–634.

[156] Mpitziopoulos, A., Gavalas, D., Konstantopoulos, C., and Pantziou, G. (2009). A survey on jamming attacks and countermeasures in wsns. *IEEE Communications Surveys & Tutorials*, 11(4).

[157] Munos, R. (2011). Optimistic optimization of a deterministic function without the knowledge of its smoothness. In *Neural Information Processing Systems*, pages 783–791.

[158] Murray, C. and Gordon, G. (2007). *Finding correlated equilibria in general sum stochastic games*. Carnegie Mellon University.

[159] Nachbar, J. H. and Zame, W. R. (1996). Non-computable strategies and discounted repeated games. *Economic theory*, 8(1):103–122.

[160] Nash, J. F. (1950a). The bargaining problem. *Econometrica: Journal of the Econometric Society*, pages 155–162.

[161] Nash, J. F. (1950b). Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36:48–49.

[162] Ndiaye, M., Hancke, G. P., and Abu-Mahfouz, A. M. (2017). Software defined networking for improved wireless sensor network management: A survey. *Sensors*, 17(5):1031.

[163] Neyman, J. and Pearson, E. S. (1933). Ix. on the problem of the most efficient tests of statistical hypotheses. *Phil. Trans. R. Soc. Lond. A*, 231(694-706):289–337.

[164] Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *ICML*, volume 1, page 2.

[165] Ngu, A. H., Gutierrez, M., Metsis, V., Nepal, S., and Sheng, Q. Z. (2016). Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*, 4(1):1–20.

[166] Nguyen, K. T., Laurent, M., and Oualha, N. (2015). Survey on secure communication protocols for the internet of things. *Ad Hoc Networks*, 32:17–31.

[167] Nguyen-Thanh, N. and Koo, I. (2009). An enhanced cooperative spectrum sensing scheme based on evidence theory and reliability source evaluation in cognitive radio context. *IEEE Communications Letters*, 13(7).

[168] Niu, R., Chen, B., and Varshney, P. K. (2006). Fusion of decisions transmitted over rayleigh fading channels in wireless sensor networks. *IEEE Transactions on signal processing*, 54(3):1018–1027.

[169] Niu, R. and Varshney, P. K. (2005). Decision fusion in a wireless sensor network with a random number of sensors. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on*, volume 4, pages iv–861. IEEE.

[170] Niu, R. and Varshney, P. K. (2008). Performance analysis of distributed detection in a random sensor field. *IEEE Transactions on Signal Processing*, 56(1):339–349.

[171] Niyato, D. and Hossain, E. (2008). Competitive pricing for spectrum sharing in cognitive radio networks: Dynamic game, inefficiency of nash equilibrium, and collusion. *IEEE journal on selected areas in communications*, 26(1):192–202.

[172] Noon, E. and Li, H. (2010). Defending against hit-and-run attackers in collaborative spectrum sensing of cognitive radio networks: A point system. In *Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st*, pages 1–5. IEEE.

[173] Oliehoek, F. A., Spaan, M. T., and Vlassis, N. (2008). Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353.

[174] Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450.

[175] Parras, J. and Zazo, S. (2018). Wireless networks under a backoff attack: A game theoretical perspective. *Sensors*, 18(2):404.

[176] Parras, J. and Zazo, S. (2019a). Learning attack mechanisms in wireless sensor networks using markov decision processes. *Expert Systems with Applications*, 122:376–387.

[177] Parras, J. and Zazo, S. (2019b). Repeated game analysis of a csma/ca network under a backoff attack. *Sensors*, 19(24):5393.

[178] Parras, J. and Zazo, S. (2019c). Sequential bayes factor testing: A new framework for decision fusion. In *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE.

[179] Parras, J. and Zazo, S. (2019d). Using one class svm to counter intelligent attacks against an sprt defense mechanism. *Ad Hoc Networks*, 94:101946.

[180] Parras, J. and Zazo, S. (2020). A distributed algorithm to obtain repeated games equilibria with discounting. *Applied Mathematics and Computation*, 367:124785.

[181] Payal, A., Rai, C. S., and Reddy, B. R. (2015). Analysis of some feedforward artificial neural network training algorithms for developing localization framework in wireless sensor networks. *Wireless Personal Communications*, 82(4):2519–2536.

[182] Perc, M., Jordan, J. J., Rand, D. G., Wang, Z., Boccaletti, S., and Szolnoki, A. (2017). Statistical physics of human cooperation. *Physics Reports*, 687:1–51.

[183] Perc, M. and Szolnoki, A. (2010). Coevolutionary games—a mini review. *BioSystems*, 99(2):109–125.

[184] Pęski, M. (2014). Repeated games with incomplete information and discounting. *Theoretical Economics*, 9(3):651–694.

[185] Peysakhovich, A. and Lerer, A. (2018). Prosocial learning agents solve generalized stag hunts better than selfish ones. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2043–2044. International Foundation for Autonomous Agents and Multiagent Systems.

[186] Pineau, J., Gordon, G., Thrun, S., et al. (2003). Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, volume 3, pages 1025–1032.

[187] Poor, H. V. and Hadjiliadis, O. (2009). *Quickest detection*, volume 40. Cambridge University Press Cambridge.

[188] Powers, R., Shoham, Y., and Vu, T. (2007). A general criterion and an algorithmic framework for learning in multi-agent systems. *Machine Learning*, 67(1-2):45–76.

[189] Rahaman, M. F. and Khan, M. Z. A. (2018). Low-complexity optimal hard decision fusion under the neyman–pearson criterion. *IEEE Signal Processing Letters*, 25(3):353–357.

[190] Rahman, A. and Gburzynski, P. (2006). Hidden problems with the hidden node problem. In *23rd Biennial Symposium on Communications*, pages 270–273. IEEE.

[191] Rawat, P., Singh, K. D., Chaouchi, H., and Bonnin, J. M. (2014). Wireless sensor networks: a survey on recent developments and potential synergies. *The Journal of supercomputing*, 68(1):1–48.

[192] Roy, N. and Gordon, G. J. (2003). Exponential family pca for belief compression in pomdps. In *Advances in Neural Information Processing Systems*, pages 1667–1674.

[193] Roy, S., Ellis, C., Shiva, S., Dasgupta, D., Shandilya, V., and Wu, Q. (2010). A survey of game theory as applied to network security. In *2010 43rd Hawaii International Conference on System Sciences*, pages 1–10. IEEE.

[194] Sampath, A., Dai, H., Zheng, H., and Zhao, B. Y. (2007). Multi-channel jamming attacks using cognitive radios. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 352–357. IEEE.

[195] Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., and Platt, J. C. (2000). Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588.

[196] Schönbrodt, F. D., Wagenmakers, E.-J., Zehetleitner, M., and Perugini, M. (2017). Sequential hypothesis testing with bayes factors: Efficiently testing mean differences. *Psychological Methods*, 22(2):322.

[197] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

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

[199] Sengar, P. and Bhardwaj, N. (2017). A survey on security and various attacks in wireless sensor network. *International Journal of Computer Sciences and Engineering*, 5(4):78–84.

[200] Shapley, L. S. (1953). Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100.

[201] Shei, Y. and Su, Y. T. (2008). A sequential test based cooperative spectrum sensing scheme for cognitive radios. In *Personal, Indoor and Mobile Radio Communications, 2008. PIMRC 2008. IEEE 19th International Symposium on*, pages 1–5. IEEE.

[202] Shi, Y., Sagduyu, Y. E., Erpek, T., Davaslioglu, K., Lu, Z., and Li, J. H. (2018). Adversarial deep learning for cognitive radio security: jamming attack and defense strategies. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE.

[203] Shnidman, D. A. (1998). Binary integration for swerling target fluctuations. *IEEE Transactions on Aerospace and Electronic systems*, 34(3):1043–1053.

[204] Shoham, Y., Powers, R., and Grenager, T. (2007). If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377.

[205] Siegelmann, H. T. and Sontag, E. D. (1992). On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449. ACM.

[206] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China. PMLR.

[207] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354.

[208] Singh, S. and Singh, N. (2015). Internet of things (iot): Security challenges, business opportunities & reference architecture for e-commerce. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 1577–1581. IEEE.

[209] Smola, A., Gretton, A., Song, L., and Schölkopf, B. (2007). A hilbert space embedding for distributions. In *International Conference on Algorithmic Learning Theory*, pages 13–31. Springer.

[210] Sokullu, R., Dagdeviren, O., and Korkmaz, I. (2008). On the ieee 802.15. 4 mac layer attacks: Gts attack. In *Sensor Technologies and Applications, 2008. SENSORCOMM'08. Second International Conference on*, pages 673–678. IEEE.

[211] Šošić, A., KhudaBukhsh, W. R., Zoubir, A. M., and Koeppl, H. (2017). Inverse reinforcement learning in swarm systems. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 1413–1421. International Foundation for Autonomous Agents and Multiagent Systems.

[212] Stimpson, J. L., Goodrich, M. A., and Walters, L. C. (2001). Satisficing and learning cooperation in the prisoner s dilemma. In *IJCAI*, volume 1, pages 535–540.

[213] Stone, P. (2007). Multiagent learning is not the answer. it is the question. *Artificial Intelligence*, 171(7):402–405.

[214] Sutskever, I. (2013). Training recurrent neural networks. *University of Toronto, Toronto, Ont., Canada*.

[215] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press Cambridge.

[216] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

[217] Szabó, G. and Fath, G. (2007). Evolutionary games on graphs. *Physics reports*, 446(4-6):97–216.

[218] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics*. MIT press.

[219] Toledo, A. L. and Wang, X. (2007). Robust detection of selfish misbehavior in wireless networks. *IEEE journal on selected areas in communications*, 25(6):1124–1134.

[220] Tomić, I. and McCann, J. A. (2017). A survey of potential security issues in existing wireless sensor network protocols. *IEEE Internet of Things Journal*, 4(6):1910–1923.

[221] Urkowitz, H. (1967). Energy detection of unknown deterministic signals. *Proceedings of the IEEE*, 55(4):523–531.

[222] Valko, M., Carpentier, A., and Munos, R. (2013). Stochastic simultaneous optimistic optimization. In *International Conference on Machine Learning*, pages 19–27.

[223] Vamsi, P. R. and Kant, K. (2014). A lightweight sybil attack detection framework for wireless sensor networks. In *Contemporary computing (IC3), 2014 Seventh International conference on*, pages 387–393. IEEE.

[224] Van Dam, T. and Langendoen, K. (2003). An adaptive energy-efficient mac protocol for wireless sensor networks. *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180.

[225] Varshney, P. K. (2012). *Distributed detection and data fusion*. Springer Science & Business Media.

[226] Von Stengel, B. (2002). Computing equilibria for two-person games. *Handbook of game theory with economic applications*, 3:1723–1759.

[227] Wald, A. (1945). Statistical decision functions which minimize the maximum risk. *Annals of Mathematics*, pages 265–280.

[228] Wald, A. (1973). *Sequential analysis*. Courier Corporation.

[229] Wang, W., Sun, Y., Li, H., and Han, Z. (2010). Cross-layer attack and defense in cognitive radio networks. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–6. IEEE.

[230] Wang, X., Gao, L., Mao, S., and Pandey, S. (2015a). Deepfi: Deep learning for indoor fingerprinting using channel state information. In *Wireless Communications and Networking Conference (WCNC)*, pages 1666–1671. IEEE.

[231] Wang, Y., Wong, J., and Miner, A. (2004). Anomaly intrusion detection using one class svm. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, pages 358–364. IEEE.

[232] Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2016). Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*.

[233] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2015b). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

[234] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

[235] Werbos, P. J. et al. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[236] Wiering, M. and Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, 12:51.

[237] Wolfson, O. and Segall, A. (1991). The communication complexity of atomic commitment and of gossiping. *SIAM Journal on Computing*, 20(3):423–450.

[238] Wu, J., Song, T., Yu, Y., Wang, C., and Hu, J. (2018a). Sequential cooperative spectrum sensing in the presence of dynamic byzantine attack for mobile networks. *PloS one*, 13(7):e0199546.

[239] Wu, J., Yu, Y., Song, T., and Hu, J. (2018b). Sequential 0/1 for cooperative spectrum sensing in the presence of strategic byzantine attack. *IEEE Wireless Communications Letters*.

[240] Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5279–5288.

[241] Wunder, M., Yaros, J. R., Kaisers, M., and Littman, M. (2012). A framework for modeling population strategies by depth of reasoning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 947–954. International Foundation for Autonomous Agents and Multiagent Systems.

[242] Xiao, L., Li, Y., Huang, X., and Du, X. (2017). Cloud-based malware detection game for mobile devices with offloading. *IEEE Transactions on Mobile Computing*, 16(10):2742–2750.

[243] Xiao, L., Li, Y., Liu, G., Li, Q., and Zhuang, W. (2015). Spoofing detection with reinforcement learning in wireless networks. In *Global Communications Conference (GLOBECOM), 2015 IEEE*, pages 1–5. IEEE.

[244] Xiao, L., Wan, X., Lu, X., Zhang, Y., and Wu, D. (2018). Iot security techniques based on machine learning. *arXiv preprint arXiv:1801.06275*.

[245] Xiao, L., Xie, C., Chen, T., Dai, H., and Poor, H. V. (2016). A mobile offloading game against smart attacks. *IEEE Access*, 4:2281–2291.

[246] Xiao, Y., Park, J., and Van Der Schaar, M. (2012). Repeated games with intervention: Theory and applications in communications. *IEEE Transactions on Communications*, 60(10):3123–3132.

[247] Xing, Z., Pei, J., and Keogh, E. (2010). A brief survey on sequence classification. *ACM Sigkdd Explorations Newsletter*, 12(1):40–48.

[248] Xu, L., Collier, R., and O'Hare, G. M. (2017). A survey of clustering techniques in wsns and consideration of the challenges of applying such to 5g iot scenarios. *IEEE Internet of Things Journal*, 4(5):1229–1249.

[249] Yadav, R., Varma, S., Malaviya, N., et al. (2009). A survey of mac protocols for wireless sensor networks. *UbiCC journal*, 4(3):827–833.

[250] Yan, Q., Li, M., Jiang, T., Lou, W., and Hou, Y. T. (2012). Vulnerability and protection for distributed consensus-based spectrum sensing in cognitive radio networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 900–908. IEEE.

[251] Yang, K. (2014). *Wireless sensor networks*. Springer.

[252] Yang, L., Lu, Y., Xiong, L., Tao, Y., and Zhong, Y. (2017a). A game theoretic approach for balancing energy consumption in clustered wireless sensor networks. *Sensors*, 17(11):2654.

[253] Yang, Y., Wu, L., Yin, G., Li, L., and Zhao, H. (2017b). A survey on security and privacy issues in internet-of-things. *IEEE Internet of Things Journal*, 4(5):1250–1258.

[254] Yassen, M. B., Aljawaerneh, S., and Abdulraziq, R. (2016). Secure low energy adaptive clustering hierarchal based on internet of things for wireless sensor network (wsn): Survey. In *2016 International Conference on Engineering & MIS (ICEMIS)*, pages 1–9. IEEE.

[255] Ye, W., Heidemann, J., and Estrin, D. (2004). Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking (ToN)*, 12(3):493–506.

[256] Yu, F. R., Tang, H., Huang, M., Li, Z., and Mason, P. C. (2009). Defense against spectrum sensing data falsification attacks in mobile ad hoc networks with cognitive radios. In *Military Communications Conference, 2009. MILCOM 2009. IEEE*, pages 1–7. IEEE.

[257] Zhang, L., Ding, G., Wu, Q., Zou, Y., Han, Z., and Wang, J. (2015). Byzantine attack and defense in cognitive radio networks: A survey. *IEEE Communications Surveys & Tutorials*, 17(3):1342–1363.

[258] Zhao, B., Feng, J., Wu, X., and Yan, S. (2017). A survey on deep learning-based fine-grained object classification and semantic segmentation. *International Journal of Automation and Computing*, 14(2):119–135.

[259] Zhou, Z., Bloem, M., and Bambos, N. (2017). Infinite time horizon maximum causal entropy inverse reinforcement learning. *IEEE Transactions on Automatic Control*, 63(9):2787–2802.

[260] Zhu, F. and Seo, S.-W. (2009). Enhanced robust cooperative spectrum sensing in cognitive radio. *Journal of Communications and Networks*, 11(2):122–133.

[261] Ziebart, B. D., Bagnell, J. A., and Dey, A. K. (2010). Modeling interaction via the principle of maximum causal entropy. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 1255–1262. Omnipress.

[262] Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA.