DECENTRALIZED COORDINATION IN MULTI-AGENT SYSTEMS

Mihail Mihaylov

July, 2012

VUBPRESS

Vrije Universiteit Brussel

Faculty of Science and Bio-engineering Sciences
Department of Computer Science
Computational Modeling Lab

Decentralized Coordination in Multi-Agent Systems

Mihail Mihaylov

Dissertation submitted for the degree of Doctor of Philosophy in Sciences

July, 2012

Supervisors: Prof. Dr. Ann Nowé
Prof. Dr. Karl Tuyls

Print: Silhouet, Maldegem

Cover design by Mihail Mihaylov

2012 Uitgeverij VUBPRESS Brussels University Press
VUBPRESS is an imprint of ASP nv (Academic and Scientific Publishers nv)
Ravensteingalerij 28
B-1000 Brussels
Tel. +32 (0)2 289 26 50
Fax +32 (0)2 289 26 59
E-mail: info@vubpress.be
www.vubpress.be

ISBN 978 90 5718 142 9
NUR 984 / 986
Legal deposit D/2012/11.161/078

To Marilyn, with love

Scientific committee members:

Supervisors:

Prof. Dr. Ann Nowé
Vrije Universiteit Brussel

Prof. Dr. Karl Tuyls
Maastricht University

Internal members:

Prof. Dr. Theo D'Hondt
Vrije Universiteit Brussel

Prof. Dr. Bernard Manderick
Vrije Universiteit Brussel

Prof. Dr. Kris Steenhaut
Vrije Universiteit Brussel

External members:

Dr. Anna Förster
University of Applied Sciences and Arts of Southern Switzerland

Prof. Dr. Matthew Taylor
Lafayette College

Abstract

Many computer systems are comprised of multiple entities (or agents) with common objectives. Though these systems can be made intelligent, using artificial intelligence techniques, individual agents are often restricted in their capabilities and have only limited knowledge of their environment. However, the group as a whole is capable of executing more complex tasks than a single agent can perform. Individual agents, therefore, need to coordinate their activities in order to meet the design objectives of the entire system. Implementing a centralized control for distributed computer systems is an expensive task due to the high computational costs, the communication overhead, the curse of dimensionality and the single point of failure problem. The complexity of centralized control can be reduced by addressing the problem from a multi-agent perspective. Moreover, many real-world problems are inherently decentralized, where individual agents are simply unable to fulfill their design objectives on their own. In multi-agent systems with no central control, agents need to efficiently coordinate their behavior in a decentralized and self-organizing way in order to achieve their common, but complex design objectives. Therefore it is the task of the system designer to implement efficient mechanisms that enable the decentralized coordination between highly constrained agents.

Our research on decentralized coordination is inspired by the challenging domain of wireless sensor networks (WSNs). The WSN problem requires resource-constrained sensor nodes to coordinate their actions, in order to improve message throughput, and at the same time to anti-coordinate, in order to reduce communication interference. Throughout this thesis we analyze this (anti-)coordination problem by studying its two building blocks separately so that we form a solid basis for understanding the more complex task of (anti-)coordination. We study pure

coordination in the problem of convention emergence and pure anti-coordination in dispersion games. We then study the full problem of (anti-)coordination in time, as seen in the WSN domain.

Our main contribution is to propose a simple decentralized reinforcement learning approach, called Win-Stay Lose-probabilistic-Shift (WSLpS), that allows highly constrained agents to efficiently coordinate their behavior imposing minimal system requirements and overhead. We demonstrate that global coordination can emerge from simple and local interactions without the need of central control or any form of explicit coordination. Despite its simplicity, WSLpS quickly achieves efficient collective behavior both in pure coordination games and in pure anti-coordination games. We use our approach in the design of an adaptive low-cost communication protocol, called DESYDE, which achieves efficient wake-up scheduling in wireless sensor networks. In this way we demonstrate how a simple and versatile approach achieves efficient decentralized coordination in real-world multi-agent systems.

Acknowledgments

First and foremost I would like to express my sincere gratitude to my supervisors Ann Nowé and Karl Tuyls. Ann, thank you for providing the opportunity to start my PhD and thank you, Karl, for the encouragement to actually take that opportunity. Thank you both for your extensive guidance throughout my research, for correcting all my texts, and for keeping me focused when I start to diverge.

I would also like to thank the members of the examination committee — Anna, Bernard, Kris, Matthew and Theo, who found the time to read this (verbose) thesis and provide insightful comments and constructive criticism.

A round of applause goes to my colleagues and friends, who have helped me in numerous ways throughout my PhD and with whom I have shared a working environment on a daily basis. Thank you Abdel, Allan, Bart, Bert, Cosmin, David C., David S., Frederik, Ha, Jonatan, Kevin, Kristof, Lara, Maarten D., Maarten P., Madalina, Marjon, Matteo, Peter, Ruben, Pasquale, Saba, Stijn, Sven, Steven, Tim, Yailen, Yann-Aël and Yann-Michaël. Thank you all from CoMo, ETRO and ARTI for the fruitful discussions and occasional distractions that have fueled my research.

Besides my colleagues and friends from the Vrije Universiteit Brussel, I would like to thank all those, who contributed to a fruitful collaboration within the DiCoMAS project. Here I gratefully acknowledge the research funding provided by the agency for Innovation by Science and Technology (IWT), project DiCoMAS (IWT60837). In addition, a big *dank(e)* goes to my friends from Maastricht University for all enjoyable moments on conferences worldwide.

Искам също така да благодаря на родителите си и брат ми за моралната подкрепа и мотивация, от които така се нуждаех. Благодаря също и на приятелите ми в Германия, Белгия и Холандия за веселите телефонни разговори и забавни моменти заедно.

Salamat, Marilyn, sa imong gugma ug suporta alang kanako.

Contents

Chapter 1

Introduction

The aim of this dissertation is to present the tools necessary to enable the efficient decentralized coordination between cooperative, but highly constrained entities (or agents) in a multi-agent system. Our work on decentralized coordination is inspired by the challenging domain of wireless sensor networks, where sensor nodes need to efficiently coordinate their activities in order to fulfill the complex objectives of the user. We apply techniques from Artificial Intelligence (AI) in order to make multi-agent systems intelligent, allowing individual agents to coordinate their behavior in a decentralized manner and thus accomplish their design objectives. We take the standpoint of cooperative game theory and develop simple learning approaches that allow individual agents to have efficient adaptive behavior and take distributed goal-oriented decisions. Below we provide an introduction to agents and motivate the need for decentralized coordination in multi-agent systems in general.

1.1 Agents

In the field of computer science, any entity that can autonomously act in its environment is called an agent. Though there is a widespread debate over the precise meaning of the term agent, the definition that is in line with our views is that of Jennings *et al.* [1998]:

Definition 1 (Agent)**.** *An agent is a computer system, situated in some environment, that is capable of flexible autonomous action in this environment in order to meet its design objectives.*

Due to the broad nature of this definition, we need to further elaborate on a number of important issues. First of all, Jennings *et al.* consider that an agent is a *computer system*, although the above definition may also apply to biological entities, such as ants, birds, or humans. Nevertheless, throughout this dissertation we will focus on computer agents, such as electronic devices and robots. Secondly, no specific *environment* is mentioned, as it refers to the wide range of settings, in which agents might find themselves. Agents need to be *autonomous*, so that they are able to operate without human intervention. Lastly, the agent's *design objectives* specify to a certain extent the purpose or goals of that agent. This definition does not reflect *how* agents can achieve their design objectives. Certain objectives are relatively simple and require purely reactive agents, such as surveillance cameras starting to record upon motion, or smoke detectors triggering an alarm at the first signs of fire. As design objectives become more complex, agents need to reason about their environment in order to meet those objectives. A team of robot vehicles, for example, needs to be able to navigate autonomously in an unfamiliar terrain without crushing into obstacles or into each other. Similarly, the microcontrollers of an aircraft need to take a large number of factors into account when flying autonomously. Agents need to execute (complex) autonomous actions in a goal-oriented manner and adapt to changes in the environment. Wooldridge & Jennings [1995] distinguish 3 characteristics that agents need to possess in order to satisfy their design objectives:

- *reactivity*: the ability to perceive their environment and respond in a timely fashion to changes that occur in it.
- *proactivity*: the ability to take initiative and exhibit goal-directed behavior.
- *social ability*: the ability to interact with other agents, including humans.

Designing a goal-directed agent to operate in a static environment is a relatively simple task, but when multiple agents act simultaneously in the same environment, they must be able to react to external changes, caused by the actions of other agents. A purely reactive agent, on the other hand, may be unable to meet its design objectives unless it takes initiative to pursue its goals. Thus, the challenge in designing agents is to find a good balance between *reactivity* and *proactivity*. Finally, the *social ability* allows agents to communicate with other agents that are situated in the same environment. Summarizing the characteristics above, an agent must be able to react timely to changes in its environment in an autonomous goal-directed manner and interact with other agents in the system in order to meet its design objectives.

1.2 Intelligent multi-agent systems

Many biological or computer systems are comprised of multiple agents with common objectives. Some examples from biology are insect colonies, animal herds, and human crowds. Other examples include computer networks, and robot swarms. Though computer systems can be made intelligent using artificial intelligence techniques, individual agents are often restricted in their capabilities and have only limited knowledge of their environment. However, the group as a whole is capable of executing more complex tasks than a single agent can perform. Agents, therefore, need to use their social ability in order to meet their often complex design objectives. For example, a single ant does not know the precise location of a food source, and is limited in the amount of food it can carry. A group of ants, on the other hand, through collective efforts, is able to gather food for the entire colony. Such multi-agent systems (MASs) are common in nature and are widely studied in computer science. Jennings *et al.* [1998] define a MAS as follows:

Definition 2 (Multi-Agent System)**.** *A multi-agent system is a loosely coupled network of agents that work together to solve problems that are beyond the capabilities or knowledge of individual agents.*

1.3 Decentralized coordination

There are numerous examples of single-agent problems, whose complexity can be reduced by addressing the problem from a multi-agent perspective. For example, traffic lights guiding vehicles through a city, or surveillance cameras tracking moving targets are typically implemented in a centralized manner. However, centralized adaptive behavior for all traffic lights or cameras in a city are expensive tasks due to the high computational costs, the curse of dimensionality and the single point of failure problem. Moreover, many complex problems are inherently decentralized. Central control is simply unavailable and costly to set up in problems such as computer devices communicating over a wireless medium, or robot vehicles exploring large unfamiliar terrains. In these settings individual agents are simply unable to fulfill their design objectives on their own. Another example of a decentralized problem is the energy trade in the smart grid, where having a central entity is undesirable, due to the monopoly it exercises on the energy market. In such multi-agent systems agents need to *coordinate* their behavior in a *decentralized* manner in order to solve complex problems and achieve their design objectives.

1.4 Motivation

Our work on decentralized coordination is inspired by the challenging domain of **wireless sensor networks** (WSNs). We will first describe the WSNs as a real-world example that motivates the need for decentralized coordination as well as anti-coordination (or (anti-)coordination for short) in multi-agent systems. As we will see, the (anti-)coordination problem that agents are facing is complex, considering the nodes' constrained abilities and the limited environmental feedback. We will explore the two components separately so that we form a solid basis for studying the more complex problem of (anti-)coordination. Moreover, the individual components are challenging by themselves and are already present in other real-world scenarios, as we will see in Chapters 3 and 4. We will study the pure coordination problem in the domain of **convention emergence**, followed by the pure anti-coordination task in **dispersion games**. Both these problems, when examined separately, present agents with a relatively simpler coordination task than the combined task of coordination and anti-coordination. Nevertheless, the limited feedback from the environment and the lack of central control make these problems still challenging.

1.4.1 Coordination in wireless sensor networks

A wireless sensor network is a collection of small autonomous devices (or nodes), which gather environmental data with the help of sensors. A more detailed description of WSNs can be found in Chapter 5. Some applications, such as habitat monitoring, or search and rescue, require that sensor nodes are *small* to be easily deployed and *inexpensive* so that they are disposable [Warneke *et al.*, 2001]. However, the limited resources of such sensor nodes make the design of a WSN application challenging. Application requirements, in terms of lifetime, latency, or data throughput, often conflict with the network capacity and energy resources.

1.4.1.1 Challenges in coordination

WSNs are an example of a multi-agent system, where highly constrained sensor nodes need to coordinate their behavior in a decentralized manner in order to fulfill the requirements of the WSN application. Here we list some of the main challenges in this domain, together with the design requirements for WSN applications:

- A message transmission by one node may cause **communication interference** with another, resulting in message loss. Therefore, the sender needs to

coordinate its transmissions not only with the receiver but also with other nodes within range.

- There is **no central control**, as the sensor nodes are typically scattered over a vast area. There is no single unit that can monitor and coordinate the behavior of all nodes. As a result, nodes need to coordinate their transmissions in a decentralized manner.
- **Communication is expensive** in terms of battery consumption, since the radio transmitter consumes the most energy. For this reason agents cannot coordinate explicitly using (energy-expensive) control messages, such as a node saying to all nodes in range *"I will transmit a message in* 5 *seconds, so everyone please stay silent"*.
- Due to the small transmission and sensing range, nodes have **only local information** and lack any global knowledge (e.g. of the network topology). Again, communicating such local information comes at a certain cost. Thus, nodes should be able to adapt their behavior based on local interactions alone.
- Nodes possess **limited memory and processing** capabilities and therefore cannot store large amounts of data, or reliably execute complex algorithms. The coordination behavior needs to be simple and have low memory requirements.
- Sensor nodes **cannot** directly **observe** the actions of others, but only the effect of their own actions. When a sensor node selects transmit and the message is not acknowledged by the recipient, the sender does not know if the receiver was itself transmitting, sleeping, or it was listening but encountered interference.

The design objectives of individual nodes are to forward their sensor measurements towards the sink in a timely fashion. As stated above, successful communication between two nodes requires good coordination with all nodes in range. When a node needs to *transmit* a message at a given time, the intended receiver must *listen* for messages. We refer to this type of coordination in time between a sender and a receiver as **synchronization**. The two nodes perform the same action at the same time, i.e. forward a message towards the sink. The sender and receiver nodes we call "communicators" for short, while all other nodes in range of the communicators we call "neighbors". In addition to communicators synchronizing, no other neighbors can forward a message at the same time, because their message will interfere with the transmission between the two communicating nodes. Therefore, the other

neighbors should *sleep* instead. This type of coordination in time between the communicators and neighbors we call **desynchronization**, since the two groups cannot perform the same action at the same time, i.e. they cannot forward a message when another message is being forwarded. They need to desynchronize their activities in time, so that transmissions do not occur simultaneously in close proximity.

In literature pure coordination is described as the problem where all agents need to select the same action to avoid conflict. Analogously, pure anti-coordination is the problem where neighboring agents need to select different actions. Little attention has however been given in literature to MASs where either pure coordination *or* pure anti-coordination of the system is impractical and/or undesirable. In many MASs, an optimal solution is intuitively found where sets of agents coordinate with one another, but anti-coordinate with others. Nodes communicating in a wireless sensor network are only one example. Other examples include traffic lights guiding vehicles through crossings in traffic control problems and jobs that have to be processed by the same machines at different times in job-scheduling problems. In such cases applying pure coordination or pure anti-coordination alone is not appropriate to address the problem (e.g. all traffic lights showing green, or complementary jobs processed at different times). In these systems, agents should logically organize themselves in groups, such that the actions of agents within the group are coordinated, while *at the same time* being anti-coordinated with the actions of agents in other groups. We refer to this concept for short as **(anti-)coordination**. An important characteristic of these systems is also that agents need to (anti-)coordinate their actions without the need of centralized control. Moreover, in such decentralized systems no explicit grouping is necessary. Rather, these groups emerge from the global objectives of the system, and agents learn by themselves to which groups they should belong (e.g. to maximize throughput in a routing problem).

We draw here a parallel to the literature on cooperation and defection in order to compare it to our subject of coordination and anti-coordination. In these terms, successful message forwarding requires cooperation between nodes, while the time constraints imply competition for the shared communication medium. Therefore, agents are faced with a challenging task. On the one hand individual agents are self-interested in the sense that they maximize their own payoff and "compete" for the medium. On the other hand agents are owned by the same user and therefore they are fully cooperative and have the same goal, i.e. to forward messages to the sink, only coordinating their behavior in a decentralized manner is hard. Therefore the system designer has the task to align the global system objective of efficient message forwarding with the individual agent objective of successful transmission

of messages, such that global coordination emerges from the self-interest of agents. For this reason we do not study the factors that promote cooperation, as agents belong to the same user. Instead, we explore approaches that align individual with global objectives and help agents coordinate in a decentralized manner under limited environmental feedback.

At each time step, each sensor node needs to both synchronize with its communicating partner and at the same time desynchronize with all other nodes in range. We refer to synchronization as coordination in time, while desynchronization stands for anti-coordination in time. When two or more agents "attune to each other" we speak of coordination, while when agents "avoid each other", we refer to it as anti-coordination. Although coordination and anti-coordination are studied separately in literature, in this thesis it becomes obvious that there is no fundamental difference between the two. We note that in coordination games a global solution always exists where all agents select the same action. However, a global solution in anti-coordination games, where neighboring agents select different actions, need not always exist. Provided there are solutions in both types of coordination problems, we will see that anti-coordination is merely another form of coordination, rather than its opposite. Throughout this dissertation, when we mention coordination in a more general context (e.g. as in the title of this thesis), we mean both coordination *and* anti-coordination. Sometimes we will write this as (anti-)coordination. In a more detailed context (e.g. when we analyze specific agent interactions) we make a distinction where necessary. However, both terms mean one and the same thing — that agents select the appropriate actions in order to avoid conflicts, based on the specification of the underlying game. When coordination (or anti-coordination) is not successful, we say that agents experience conflicts with each other. Moreover, from game-theoretic point of view, successful and unsuccessful coordination differ only in the feedback that agents receive from their interactions.

Below we show an example of the (anti-)coordination problem that sensor nodes are facing when forwarding data. In Chapter 5 we will examine that problem in more detail.

Example ((De)Synchronization in WSNs)**.** *Consider a number of wireless sensor nodes, arranged in an arbitrary topology. For a successful transmission between two nodes, the sender needs to put its radio in transmit mode, the intended receiver needs to listen to the channel, while all other nodes in range need to turn off their radios. In the absence of central control, how can all nodes in the wireless sensor network learn over time to (de)synchronize their activities, such that they successfully forward data to the sink?*

Although in this dissertation we will closely examine the decentralized coordination problem in the WSN domain, that problem is present in numerous other areas as well. For example traffic lights on neighboring intersections need to coordinate their cycles in order to efficiently route the traffic flow through the city. Another example is the coordination between robot units exploring an unfamiliar environment. The solutions we propose for decentralized coordination in WSNs are applicable in these domains as well. **Thus, the main question we as system designers are investigating in this thesis is the following: How can the designer of a decentralized system, imposing minimal system requirements and overhead, enable the efficient coordination of highly constrained agents, based only on local interactions and incomplete knowledge?**

1.4.1.2 Our method

In order to design a reliable methodology for WSN applications, one must enable the decentralized coordination between highly constrained sensor devices. Based on the above challenges, sensor nodes need to rely on simple decentralized coordination mechanisms that work with limited feedback and are based on local information. Moreover, coordination cannot be explicit in the form of additional control messages, due to the communication costs. Sensor nodes need to make efficient use of their limited resources while following their design objectives. In this thesis we develop simple learning approaches that allow agents to have efficient adaptive behavior and take distributed goal-oriented decisions. In Section 2.3 we present the approaches considered in this thesis.

We rely on the reinforcement learning (RL) framework to make individual agents optimize their own performance by considering the effect of their actions on other agents in the system. However, due to the distributed nature of the WSN domain and the limited information available, individual nodes cannot measure the global system performance in order to optimize their long-term behavior. Nevertheless, we show that maximizing immediate payoffs not only significantly reduces the learning duration, which is rather costly in the WSN domain, but also results in near-optimal[1] network performance by (de)synchronizing the activities of nodes. In Chapter 5 we present in more detail the problem of (de)synchronization in wireless sensor networks.

[1] the difference with optimal latency is in the matter of milliseconds to a few seconds

1.4.2 Coordination for convention emergence

As we saw above, the decentralized coordination problem in wireless sensor networks involves both coordination (in the form of synchronization) and anti-coordination (or desynchronization). Moreover, this (anti-)coordination has to be performed in time, i.e. at each time step agents need to (anti-)coordinate with their neighbors. In this thesis we will first split the WSN problem in several components and analyze each one individually. Only then will we approach the full problem of (anti-)coordination in WSNs.

Most generally, a convention in a MAS is a behavior that is common among agents, e.g. driving either on the right side or the left side of the road. In pure coordination games agents benefit from selecting the same action as others (see Section 3.4 for details). If all agents have learned to select the same action at every step in repeated pure coordination games, we say that they belong to a convention. Therefore, a convention can be seen as a solution to a pure coordination problem, where agents can realize mutual gains if they exhibit common behavior, i.e. if a convention emerges in the MAS.

In Chapter 3 we study how conventions can emerge as a solution to repeated decentralized coordination problems in large multi-agent systems. To illustrate the concept of conventions in WSNs, we present an example of a pure coordination problem, which we will study and elaborate on later in this thesis.

Example (WSN pure coordination)**.** *Consider an arbitrary network of nodes, which typically communicate on different frequencies (or channels) in order to avoid radio interference. Every so often, all nodes need to switch to the same channel, regardless which, in order to exchange control messages, e.g. to synchronize their clocks. In the absence of central control, how can all nodes in the wireless sensor network learn over time to select the same broadcast frequency?*

Here a channel cannot be decided in advance since the quality of some channels is worse than the quality of others due to external disturbances. Thus energy constrained sensor nodes need to quickly learn to select the same reliable frequency in repeated interactions under very limited feedback from the environment.

1.4.3 Anti-coordination in dispersion games

Besides synchronization, the WSN coordination problem involves desynchronization between nodes, or anti-coordination in time. The anti-coordination problem arises when multiple agents need to select actions, such that no two adjacent agents have

the same action. Vehicles arriving at an intersection is an example of an anti-coordination task, where agents should take different actions (e.g. yield or proceed) in order to avoid conflict.

Dispersion games [Grenager *et al.*, 2002] model the anti-coordination problem between agents in a fully connected network of arbitrary size, where the aim is to let agents maximally disperse over the set of actions. In WSNs, however, the anti-coordination problem is played on a graph and hence is more complex, since agents need to disperse their actions, taking into account the topological restrictions of the graph. Simply dispersion over the set of available actions will not necessarily result in good performance since locality now plays a role and neighboring agents on the graph may still experience conflicts.

In Chapter 4 we study the pure anti-coordination problem, as well as the combined problem of coordination and anti-coordination in single-stage repeated games. The combined (or (anti-)coordination) game resembles the (de)synchronization problem of nodes in a wireless sensor network, which we study in detail in Chapter 5. To study the problem of pure anti-coordination between nodes in a WSN, in Chapter 4 we elaborate on the following problem.

Example (WSN pure anti-coordination)**.** *Consider a wireless sensor network of an arbitrary topology, where sensor nodes need to forward large amounts of data. To allow for parallel transmissions, neighboring nodes need to select different frequencies (or channels) to send their data simultaneously. In the absence of central control, how can neighboring nodes in the wireless sensor network learn over time to transmit on different frequencies?*

The above example demonstrates the pure anti-coordination problem faced by highly constrained agents under limited environmental feedback. Individual sensor nodes need to rely on a simple decentralized approach that allows agents to anti-coordinate their actions through only local interactions.

1.5 Problem statement

In multi-agent systems with no central control, agents need to efficiently coordinate their behavior in a decentralized and self-organizing way in order to achieve their common, but complex design objectives. Therefore it is the task of the system designer to implement efficient mechanisms that enable the decentralized coordination between highly constrained agents. In this thesis we take the role of designers of decentralized systems and investigate the following problem, which motivates our

research:

> **How can the designer of a decentralized system, imposing minimal system requirements and overhead, enable the efficient coordination of highly constrained agents, based only on local interactions and incomplete knowledge?**

As outlined in Section 1.4 some decentralized systems require agents to both coordinate with some agents and at the same time anti-coordinate with others, which we term (anti-)coordination for short. To answer the above question and form a solid basis for studying (anti-)coordination games, we first split the decentralized coordination problem in its two components, namely pure coordination and pure anti-coordination and analyze the two components individually. To obtain a better understanding of each component, we pose the following research questions:

Q1: *How can conventions emerge in a decentralized manner in pure coordination games?*

Q2: *How can agents achieve pure anti-coordination in a decentralized manner in dispersion games?*

We propose a simple decentralized approach that allows agents to achieve efficient collective behavior in pure coordination games. We also show the performance of the same approach in pure anti-coordination games, as well as in the (anti-)coordination game. To study the (anti-)coordination game in time in the context of a real-world scenario, we pose the following question:

Q3: *How can highly constrained sensor nodes organize their communication schedules in a decentralized manner in a wireless sensor network?*

We use our approach in the design of several low-cost communication protocols for efficient (de)synchronization in wireless sensor networks. In this way we demonstrate how a simple and versatile approach achieves efficient decentralized coordination in real-world multi-agent systems.

Designing an intelligent decentralized system of agents that operate on limited resources is undoubtedly a challenging task. The challenges stem from the characteristics of the above problems, namely:

- multiple highly constrained agents act autonomously in the same environment;
- agents are fully cooperative and have the same goals, but have no mechanism

of coordination;

- the MAS has complex design objectives, beyond the capabilities of individual agents;
- there is no central control over the agents and they have no global knowledge.

In this thesis we study how one can overcome these challenges and achieve efficient decentralized coordination in multi-agent systems.

1.6 Summary and contributions

In Chapter 2 we make an extensive overview of game-theoretical concepts in order to have the necessary tools for modeling the strategic interactions between players, participating in a game. We describe the details of coordination and anti-coordination games, as well as combined (anti-)coordination game. We outline the theory behind the reinforcement learning (RL) framework and describe three common learning algorithms, which serve as the basis for our proposed approach. Lastly, we introduce the preliminaries of the theory of Markov chains, which allows us to examine the convergence properties of our learning approaches and describe how the behavior of agents changes over time.

In Chapter 3 we survey the first part of the (anti-)coordination game, namely pure coordination. We describe in detail the problem of convention emergence and the underlying interaction model of agents, comparing it to related literature. The main contributions of this chapter are the following:

- We propose Win-Stay Lose-probabilistic-Shift (WSLpS) — a decentralized approach, based on the RL framework, for fast convention emergence, and outline its advantages, compared to other algorithms, proposed in the literature on coordination games.
- We analytically study its properties using the theory of Markov chains and prove its convergence in pure coordination games;
- We perform an extensive empirical study analyzing the behavior of agents in a wide range of settings, and study how the type of feedback influences the rate of convention emergence.

We also explore the relation between two types of agent interactions (pairwise and multi-player) on different graphs and between different network densities in terms of convergence speed.

In Chapter 4 we present the rest of the (anti-)coordination problem, namely pure anti-coordination and the combined problem of coordination and anti-coordination. We show how the same WSLpS approach, presented in Chapter 3, can be applied in pure anti-coordination games to help agents self-organize based only on local interactions with limited feedback. We survey the literature on anti-coordination games and describe the details of several algorithms that bare resemblance to WSLpS. The main contributions of this chapter are the following:

- We compare the convergence rate of WSLpS to other approaches presented in literature on anti-coordination and demonstrate how WSLpS can be applied in a wide range of scenarios, in which other, sometimes more complex algorithms are not suitable.
- We study the difficulty that agents face in pure coordination problems, as compared to pure anti-coordination problems, illustrate the relationship between the two game types and show how the (anti-)coordination game involves characteristics of both.

We see that the convergence time of (anti-)coordination games that involve equal amount of coordination and anti-coordination, is much closer to that of pure anti-coordination than to pure coordination.

In Chapter 5 we show how the (anti-)coordination games studied in Chapters 3 and 4 map to the WSN (de)synchronization problem. We provide an overview of the decentralized coordination and anti-coordination challenges in the real-world domain of WSNs and study how WSLpS can be used by computationally bounded sensor nodes to organize their communication in an energy-efficient decentralized manner. The main contributions of this chapter are the following:

- We study the (de)synchronization problem in WSNs from two perspectives: as one multi-stage (anti-)coordination game in time, as well as a sequence of repeated single-stage graphical games at different time intervals, obtaining comparable results.
- We propose different adaptive communication protocols and demonstrate the importance of (anti-)coordination in WSNs, as opposed to pure coordination and pure anti-coordination.
- We argue that optimization of long-term goals is non-trivial and costly in WSNs and demonstrate that maximizing immediate payoffs still results in acceptable near-optimal behavior.

- We show that even without modeling the temporal relation between interactions at different time intervals in the WSN, agents are able to learn an efficient policy.

The WSN scenario clearly demonstrates the need for decentralized coordination in multi-agent systems. Our communication protocols are based on the simple WSLpS approach and therefore impose minimal system requirements and overhead. In this way the scheduling of the sensor nodes' behavior is a result of simple and local interactions without the need of central mediator or any form of explicit coordination. Therefore, our approach makes it possible that (anti-)coordination emerges in time rather than is agreed upon.

Chapter 2

Background

In this chapter we present the preliminaries of our work. We study some concepts from game theory that help us represent different games and determine the behavior of rational agents in these strategic interactions. We present the theory behind the reinforcement learning framework of agents and show how they can adapt their behavior in a dynamic environment by trial and error. Lastly, we study the theory of Markov chains, which allows us to examine the convergence properties of our learning algorithms and describe how the behavior of agents changes over time.

2.1 Game theory concepts

Game theory (GT) is an economic theory that models the strategic interactions between a set of players, participating in a game. To emphasize the strategic aspects of player interaction, GT defines two specifications of a game, namely normal form and extensive form. The main difference between the two is the way agents select their actions at each interaction. In normal form games agents select their actions simultaneously (e.g. in the game of Rock-Paper-Scissors), while in the extensive form games (such as chess) — consecutively. However, either game specification can be used to model repeated interactions. The latter form is not of interest for the current research, since we consider simultaneous moves, such as those in a slotted wireless communication protocol (cf. Chapter 5). For more on extensive form games, the interested reader is referred to Peters [2008]. The normal (or strategic) form game is defined as follows:

Definition 3 (Normal form game)**.** *A normal form game is a tuple* $(N, \mathbb{A}, P_{i \in N})$*, where:*

- $N = \{1, \ldots, \mathcal{N}\}$ *is a set of* $\mathcal{N}$ *players, or agents.*
- $\mathbb{A} = A_1 \times \cdots \times A_{\mathcal{N}}$ *is the space of all possible joint actions, where* $A_i = \{a_i^1, \ldots, a_i^{k_i}\}$ *is the individual (finite) set of* k_i *actions available to agent* $i \in N$*.*
- $P_i : \mathbb{A} \to \mathbb{R}$ *is the individual payoff function of agent* $i \in N$*.*

In a normal form game, each agent $i \in N$ independently selects action $a_i \in A_i$ in a given time step and receives a payoff $P_i(\vec{a})$ based on the joint action $\vec{a}$. The *joint action* (or *action profile*) $\vec{a} \in \mathbb{A}$ is the combination of actions of all agents in that time step.

A normal form game can be represented by a $k_1 \times \cdots \times k_{\mathcal{N}}$-dimensional payoff matrix M. An example of a 2-player normal form game is the Stag hunt (SH) game, first suggested by Jean-Jacques Rousseau 1754. The game's payoff matrix can be seen in Table 2.1. The first player chooses rows and the second — columns. Each entry in the payoff matrix consists of two values. The first value represents the payoff that the row player receives, while the second shows the payoff of the column player.

Example 1 (Stag hunt)**.** *Two hunters can choose to either hunt a stag or a hare. The stag is larger, but requires both hunters to coordinate well, while the hare can be hunted individually, but it is a smaller meal.*

	stag	*hare*
stag	$(2, 2)$	$(0, 1)$
hare	$(0, 1)$	$(1, 1)$

Table 2.1: Payoff matrix of the 2-player Stag hunt game.

The behavior of each agent in a given game can be captured by the agent's *strategy*. A strategy $s_i : A_i \to [0, 1], s_i \in S_i$ of agent i is a probability distribution over the set of i's available actions A_i. A strategy s_i that assigns probability 1 to a given action $a \in A_i$ and 0 to all other actions in A_i is called *pure strategy* (or deterministic strategy). A *mixed strategy*, on the other hand, prescribes probability $s_i(a) < 1$, where $\sum_a s_i(a) = 1$ for all $a \in A_i$. The combination of all strategies $\vec{s} = (s_1, \ldots, s_{\mathcal{N}})$, where each agent $i \in N$ plays strategy $s_i \in A_i$, is termed *strategy profile*. If all agents are playing pure strategies, the strategy profile $\vec{s}$ corresponds

to a joint action $\vec{a}$. Lastly, the expected payoff $P_i(\vec{s})$ that agent i receives based on the strategy profile $\vec{s}$ is:

$$P_i(\vec{s}) = \sum_{\vec{a} \in \mathbb{A}} \prod_{j=1}^{\mathcal{N}} \vec{s}_j(a_j) P_i(\vec{a})$$

As stated earlier, game theory studies how agents will behave in a given (normal form) game. There are several solution concepts used to model the strategic interactions between players. We will briefly overview the most commonly used.

It is often convenient to define a strategy profile that does not include the strategy of a given agent. We define $\vec{s}_{-i} = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_{\mathcal{N}})$ as the strategy profile excluding strategy s_i of agent i. We will use this notation to define the best response behavior of agents.

Definition 4 (Best response)**.** *Strategy s_i of agent i is a best response to the strategy profile $\vec{s}_{-i}$ iff $s_i \in \arg\max_{s'_i \in S_i} P_i(\vec{s}_{-i}, s'_i)$*

We use $(\vec{s}_{-i}, s'_i)$ to denote the strategy profile $\vec{s}$, where agent i is using strategy s'_i. When all players, participating in a normal form game, select the (pure or mixed) strategy that is the best response to the others' strategies, we say that the agents are playing a Nash equilibrium of the game.

Definition 5 (Nash equilibrium)**.** *Strategy profile $\vec{s}$ is a Nash equilibrium if for each agent i, s_i is a best response of i to the strategy profile $\vec{s}_{-i}$.*
In terms of the payoff function we say that a strategy profile $\vec{s}$ is a Nash equilibrium iff

$$P_i(\vec{s}) \geq P_i(\vec{s}_{-i}, s'_i) \quad \forall i \in N, \quad s'_i \in S_i$$

That is, in a Nash equilibrium no agent has an incentive to unilaterally deviate from the chosen strategy. Put differently, no player can strictly improve its payoff by changing its strategy, while the strategies of others remain fixed. If the equilibrium strategy profile $\vec{s}$ contains only pure strategies, then we speak of pure Nash equilibrium, otherwise the Nash equilibrium is mixed. Nash [1950] proved that every n-player finite game has at least one Nash equilibrium. A Nash equilibrium, however, does not necessarily imply an optimal outcome for all players. Consider the well-known Prisoner's dilemma (PD) game, originally introduced by Merrill Flood and Melvin Dresher in 1950 and later popularized by Axelrod [1984]. The game's payoff matrix is shown in Table 2.2.

Example 2 (Prisoner's dilemma)**.** *Two suspects, accused of a crime, are separately interrogated by the police. Each suspect can either deny his involvement in the crime,*

or betray his partner. Neither suspect knows what choice the other suspect will make. If only one betrays (defects), he goes free and the other suspect receives a 10*-year sentence. If both deny their involvement (cooperate), each gets* 1*-year sentence, otherwise if both betray each other, they are imprisoned for* 5 *years.*

	deny	*betray*
deny	$(-1, -1)$	$(-10, 0)$
betray	$(0, -10)$	$(-5, -5)$

Table 2.2: Payoff matrix of the Prisoner's dilemma game.

The pure Nash equilibrium of the above game is (*betray*, *betray*), since no agent can obtain higher payoff by unilaterally changing his action. However, if both agents pick the joint action (*deny*, *deny*) they will receive a higher payoff. Despite its popularity, the Nash equilibrium does not guarantee that players will get the highest possible payoff, as illustrated in the PD game. In addition, if several pure Nash equilibria exist in a game, the Nash solution concept is not sufficient to explain which equilibrium the players will select. For example, in the SH game in Table 2.1 there are two pure Nash equilibria, namely (*stag*, *stag*) and (*hare*, *hare*), the former of which yields higher payoff for both agents, than the latter. This suggests the idea behind the Pareto dominance and Pareto optimality solution concepts, introduced by Vilfredo Pareto.

Definition 6 (Pareto dominance)**.** *A strategy profile $\vec{s'}$ is strictly Pareto dominated by another strategy profile $\vec{s}$, if in $\vec{s}$ all agents receive at least the same payoff as in $\vec{s'}$ and at least one agent receives a strictly higher payoff.*
A strategy profile $\vec{s'}$ is weakly Pareto dominated by another strategy profile $\vec{s}$, if in $\vec{s}$ all agents receive at least the same payoff as in $\vec{s'}$.

Definition 7 (Pareto optimality)**.** *A strategy profile $\vec{s}$ is Pareto optimal (or Pareto efficient) if it is not strictly Pareto dominated by another strategy profile $\vec{s'}$.*

In a Pareto optimal outcome no agent could be made better off without making some other agent worse off. We note that a Pareto optimal solution need not be a Nash equilibrium and similarly, a Nash equilibrium need not be Pareto optimal. It is easy to see that the strategy profile (*deny*, *deny*) is a Pareto optimal solution in the PD game.[1] Neither player can receive a higher payoff by changing his strategy,

[1] In fact, all strategy profiles in PD are Pareto optimal, except (*betray*, *betray*)

without another player receiving a lower payoff. Although this strategy profile is strongly preferred by both agents, it is not a Nash equilibrium and therefore the players are likely to deviate from their strategies and adopt the Pareto dominated solution (*betray*, *betray*). In the SH game the Nash equilibrium (*stag*, *stag*) Pareto dominates the other Nash equilibrium (*hare*, *hare*) and therefore agents would benefit more from selecting the former.

One disadvantage of using the Pareto concepts is that they do not guarantee a "fair" solution for both agents. One equilibrium may favor one agent, while the other agent may prefer a different equilibrium. Consider for example the Battle of the sexes (BS) game, whose payoff matrix is shown in Table 2.3.

Example 3 (Battle of the sexes)**.** *A husband and a wife have agreed to attend an entertainment event together, but neither one recalls precisely which event — a boxing match or a pop concert. The man (row player) prefers to visit the boxing match, while the wife (column player) favors the concert, yet they would like to visit the same event together. The players are in different parts of the city with no means of communication and therefore have to make their decisions independently.*[2]

	boxing	*concert*
boxing	$(3, 2)$	$(1, 1)$
concert	$(0, 0)$	$(2, 3)$

Table 2.3: Payoff matrix of the Battle of the sexes game.

The above game has two pure and one mixed Nash equilibria. Each of the two pure equilibria (*boxing*, *boxing*) and (*concert*, *concert*) is Pareto optimal, but not fair. One agent will receive an expected payoff of 3 and the other gets 2. On the other hand, the mixed Nash, where each agent selects their preferred event with probability $\frac{3}{4}$, is fair but inefficient. The expected payoff of both agents is 1.5. In this case, a more efficient and fair outcome can be achieved by the correlated equilibrium, introduced by Aumann [1974].

Definition 8 (Correlated equilibrium)**.** *A trusted mediator samples a probability distribution π over the set S of all pure strategy profiles in the game and makes non-binding confidential recommendations to each player. With probability $\pi(\vec{s})$ the*

[2] Here we chose to distinguish the case where each player visits his or her own preferred event (payoff of 1 to each player) from the case where each player visits the other's preferred event (payoff of 0 to each player). This decision is only cosmetic and does not change the underlying structure of the game.

mediator selects $\vec{s} \in S$ *and recommends the* s_i *component of* $\vec{s}$ *to agent* i*. Then,* π *is a correlated equilibrium if no agent has an incentive to deviate from the pure strategy recommended to it by the mediator, i.e.:*

$$\sum_{\vec{s} \in S} \pi(\vec{s}) P_i(\vec{s}) \geq \sum_{\vec{s} \in S} \pi(\vec{s}) P_i(\vec{s}_{-i}, s_i') \qquad \forall s_i' \in S_i$$

There are two pure strategy Nash equilibria in the BS game. A trusted mediator can flip a fair coin and select the profile (*boxing*, *boxing*) if Heads and (*concert*, *concert*) if Tails. Then, the mediator will recommend the selected pure strategy to each agent. Since the probability of selecting any of the two pure Nash equilibria is equal, the result is fair for both players with expected payoff of 2.5, which is larger than the expected payoff of the mixed Nash. Once agent i receives a recommendation for $s_i(boxing)= 1$, for example, it has no incentive to select *concert*. Selecting *concert* would result in lower payoff, due to miscoordination with the other agent, who is also recommended *boxing*. Therefore, the probability distribution $\pi((boxing, boxing))= \pi((concert, concert))= \frac{1}{2}$ is a correlated equilibrium.

One disadvantage of implementing correlated equilibria is that a central entity is required to recommend strategies. Alternatively, a correlated equilibrium can be implemented using a public (random) signal from the environment, instead of private recommendations by a mediator. The agents, then, can learn or have a prior agreement that when they see signal **A** occurring in the environment, they should select *boxing*, and similarly if signal **B**, then *concert*. One real-world example of public signal as a form of centralized coordination is the traffic light at road intersections. The signal is public, since it is visible to all drivers approaching the intersection. The traffic law states that when a driver sees the red signal, he should select the action *stop*, while a green signal implies *go*. No one has any incentives to ignore this public signal and take a different action than the "recommended" one (e.g. running a red light). Thus, coordination is achieved using a centralized entity that shows a public signal to all agents. However, in this thesis we are interested in decentralized coordination in the absence of a central mediator. Cigler & Faltings [2011], for example, show how a public signal in WSNs can be implemented in a decentralized way.

2.2 Overview of games

In the previous section we discussed the outcomes of strategic interactions between two agents. We analyzed the actions that rational agents will select, given a specific

game. In this section we will focus on the **types of games**, based on the goals that agents have, and how those games can be **formally represented**.

2.2.1 Game types

	action1	*action2*
action1	(a, w)	(c, y)
action2	(b, x)	(d, z)

Table 2.4: General form of the payoff matrix for a two-player two-action game.

In Section 2.1 we showed a number of two-player two-action games. Here table 2.4 shows the general form of the payoff matrix for such games. As stated earlier, the first player chooses rows and the second — columns. Each entry in the payoff matrix consists of two values. The first value represents the payoff that the row player receives, while the second shows the payoff of the column player. The relation between the payoffs determines whether the game is **coordination**, **anti-coordination** or a zero-sum game. In zero-sum games the sum of the payoffs of all players for a given outcome is, intuitively, 0. The gain of one agent comes at the expense of another and therefore these games are also called competitive. However, the focus of this thesis is on aligning the goals of individual agents with the goal of the multi-agent system as a whole, using learning mechanisms. We, as system designers, are interested in helping agents (anti-)coordinate with each other in order to optimize the behavior of the entire system. For this reason we will not discuss zero-sum games.

2.2.1.1 Coordination games

Coordination games often occur in multi-agent systems and are commonly studied in literature [Lewis, 1969; Axelrod, 1986; Shoham & Tennenholtz, 1993]. Lewis [1969] describes the coordination problem as a game in which agents can realize mutual gains by selecting the same action in the presence of several alternatives. In a coordination game, the relation between the payoffs for the row agent in Table 2.4 are the following: $a > b$ and $d > c$. Similarly, for the column player it has to hold that $w > y$ and $z > x$. In *common interest* (or pure) coordination games, players have the same preferences over the different coordination outcomes in the sense that agents care little on which of the available actions they will coordinate,

as long as all agents select the same action [Schelling, 1960]. Thus, to show that the interest of players coincide, we add the following requirements: $a \geq d$ and $w \geq z$. Even though the preferences of agents coincide, coordinating their actions is not a trivial task, due to the distributed nature of the multi-agent system. In *conflicting interest* games, selecting the same action is still mutually beneficial, but agents have different preferences over the actions. So the additional payoff relations are: $a > d$ and $z > w$. An example of a conflicting interest coordination game is Battle of the Sexes where agents would like to visit the same event together, but each has its own preferred choice (see Example 3). A typical example of a common interest coordination problem given in literature is the game where agents have to decide on which side of a two-lane road to drive provided there are no *a priori* traffic laws.

Example 4 (Two-lane road)**.** *Two drivers are traveling in opposite directions on the same two-lane road. In the absence of traffic laws, it matters little to anyone on which side of the road they drive, as long as both drivers do the same. However, if one of them drives on the left in one direction and the other chooses right in the opposite direction, they will end up in the same lane and therefore collide.*

	left	*right*
left	$(1, 1)$	$(0, 0)$
right	$(0, 0)$	$(1, 1)$

Table 2.5: Payoff matrix of the Two-lane road game.

The game in Table 2.5 has two pure strategy Nash equilibria, both of which are Pareto optimal with expected payoff of 1 for each player. The mixed strategy equilibrium, where each player selects *left* with probability $\frac{1}{2}$, gives an expected payoff of 0.5 for each player. The actual problem that the agents face is coordinating on the two pure strategies. Coordination games can be easily extended to more than two agents or two actions. For instance, the above game can be played on a four-lane road between all inhabitants in a given city. Although the game remains the same, the payoff tables are expanded, and so is the number of pure strategies that agents need to coordinate on.

2.2.1.2 Anti-coordination games

Similarly to the above type of games, in anti-coordination games agents need to coordinate the choice of their strategies in order to obtain positive feedback. However, here agents coordinate on choosing different actions. In the two-agent case, a

coordination game can be transformed into an anti-coordination game by renaming one player's action labels. An anti-coordination game [Bramoullé *et al.*, 2004] has the following payoff relations for the row player: $b > a$ and $c > d$; and for the column player: $x > z$ and $y > w$ (cf. Table 2.4). Here too the game can be common interest or conflicting interest anti-coordination game. In common interest we have $b \geq c$ and $x \geq y$, while in conflicting interest: $b > c$ and $y > x$. For example, multi-channel wireless communication is a common interest anti-coordination problem. Provided the quality of all channels is the same, wireless nodes care little on which channel they transmit, as long as neighboring nodes send on different channels. An example of 2-player common interest anti-coordination game is the dropped call game.

Example 5 (Dropped call)**.** *A telephone call between two participants gets unexpectedly dropped. Each one has the option to either call back immediately, or to wait for the other participant to call. If both decide to call the line will be busy, while if both wait, the call will not take place.*

	call	*wait*
call	$(0, 0)$	$(1, 1)$
wait	$(1, 1)$	$(0, 0)$

Table 2.6: Payoff matrix of the Dropped call game.

Provided calling is free, there are two pure Pareto optimal Nash equilibria, where one player selects *call* and the other *wait*s. According to Table 2.6 the expected payoff for *(call,wait)* is 1 for each agent, however neither of them knows which of the two Pareto optimal equilibria the other agent will select. There is a third equilibrium in mixed strategy, where each player selects *call* (or *wait*) with probability $\frac{1}{2}$. The expected payoff to both players is 0.5. Thus, in this game it is better for agents to coordinate on the choice of pure strategies, rather than implement mixed strategies, which result in a lower expected payoff.

A generalization of anti-coordination games for arbitrary number of agents and actions are *dispersion games* (DGs), studied by Grenager *et al.* [2002]. In DGs agents attempt to be maximally dispersed over the set of available actions. An example of dispersion games is the load balancing problem in wireless sensor networks [Tewfik, 2012]. Nodes try to spread the message load over different network paths, in order to avoid traffic congestion. We will study dispersion games in more detail in Chapter 4. A typical example that illustrates conflicting interest anti-coordination games for more than 2 players is the famous El Farol Bar problem, first introduced by Arthur

[1994]:

Example 6 (El Farol Bar problem)**.** *Every Thursday evening* 100 *individuals decide simultaneously but independently whether to attend a bar or stay at home. The capacity of that bar is limited to* 60 *persons, so if more people decide to go, the bar will be overcrowded and therefore less enjoyable, than staying at home. However, if at most* 60 *persons attend, they will have a better time than if they remained home.*

In this problem agents need to anti-coordinate their choice of attendance in order to have an enjoyable evening. However, Arthur shows that no pure strategy exists that performs optimally. The game is conflicting interest and attending the bar is always preferred to staying at home. A small modification to the above problem can make the game common interest and also allow for pure strategies to be successful. For example, instead of having only one bar, the individuals can choose among several bars with smaller, but in total sufficient capacity. In this case, provided the bars do not differ much, it is of common interest for agents to attend different bars. The latter problem bares resemblance to the topic of grid computing [Galstyan *et al.*, 2005], where agents have a common interest of spreading their jobs on different processors, so as to minimize execution time.

The El Farol Bar problem has inspired a class of games, known as Minority games [Challet & Zhang, 1997]. An odd number of agents choose between 2 actions at each round of the game and those in the minority win. The strategies that successfully predict the winning action have a higher probability to be adopted by other agents. We refer the interested reader to Challet *et al.* [2005] for learning successful strategies in Minority games.

2.2.1.3 Coordination and anti-coordination

In the above two sections it becomes apparent that coordination and anti-coordination are in fact two sides of the same coin. In both game types agents need to coordinate their strategies, i.e. select the appropriate actions, in order to avoid conflicts. Selecting the same action in coordination games, or choosing different action in dispersion games results in positive feedback for all agents. We can transform a 2-player coordination game into an anti-coordination game by swapping the action labels of one agent. Analogously, a dispersion game can be transformed into a coordination game in which agents coordinate on a maximally dispersed assignment of actions to agents [Grenager *et al.*, 2002]. However, such transformations require a unique ordering of each agent's actions, which is not realistic in large multi-agent systems.

In this thesis we are interested in developing an approach that is applicable both in coordination as well as in anti-coordination games. We study the relation between these two game types and the difficulty of the combined (anti-)coordination problem. Many real situations require both coordination and anti-coordination for agents to perform efficiently. Furthermore, the behavior of agents is influenced by the underlying game topology. As we mentioned in Section 1.4.1, in wireless sensor networks agents are involved in a game that is neither pure coordination, nor pure anti-coordination. Depending on the network topology, sensor nodes need to coordinate with some neighbors in order to forward messages and at the same time anti-coordinate with others in order to avoid interference. In addition, real-world scenarios may display both types of coordination. A variant of the El Farol Bar problem from Example 6 states that the evening is less enjoyable not only if too many people show up, but also if too few attend, since the bar will be too boring. This variant presents a different dimension to the synergy between coordination and anti-coordination. Here agents need to coordinate up to a certain level (e.g. of attendance) and then anti-coordinate. This type of (anti-)coordination differs from the one in WSNs, where agents always coordinate with specific nodes and anti-coordinate with others. Nevertheless, the goal of learning in these repeated games is the same — achieving successful (anti-)coordination of agents. In Section 2.3 we will examine several learning algorithms that help agents (anti-)coordinate.

2.2.2 Game representations

Strategic interactions can be formally represented in a number of different ways, based on the characteristics of the underlying game. For one-shot games, where players choose their actions simultaneously, we typically use the **normal form** representation. When players are engaged in a one-shot game, which is followed by another (or several others), we can consider the sequence of these games as one multi-stage game. A repeated multi-stage game, then, is called a **stochastic game** (or Markov game) [Shapley, 1953]. A stochastic game with only one state reduces to a repeated normal form game.

One assumption in normal form games is that the payoff of an agent depends on the actions of all agents in the game. When agent interaction is bounded by an underlying interaction graph, the payoffs to agents depend only on their (immediate) neighbors in the graph. A more suitable representation that captures payoff independence between agents is that of **graphical games** (or network games) [Kearns *et al.*, 2001; Galeotti *et al.*, 2010]. When the underlying interaction graph is fully connected, the graphical game reduces to a normal form game.

2.2.2.1 Normal form game

The formal notation of a normal form game (NFG) has been presented in Definition 3. For consistency, we will repeat it here:

Definition (Normal form game)**.** *A normal form game is a tuple* $(N, \mathbb{A}, P_{i \in N})$*, where:*

- $N = \{1, \ldots, \mathcal{N}\}$ *is a set of* $\mathcal{N}$ *players, or agents.*
- $\mathbb{A} = A_1 \times \cdots \times A_{\mathcal{N}}$ *is the space of all possible joint actions, where* $A_i = \{a_i^1, \ldots, a_i^{k_i}\}$ *is the individual (finite) set of* k_i *actions available to agent* $i \in N$*.*
- $P_i : \mathbb{A} \to \mathbb{R}$ *is the individual payoff function of agent* $i \in N$*.*

In a NFG, all agents select their actions simultaneously and receive feedback based on the actions of all agents. One limitation of NFGs is that they do not capture the underlying structure of the strategic interactions. For example, in WSNs the payoff of one node can be considered independent from the action of another node on the other side of the network. Another limitation is that NFGs cannot capture complex dynamic play that unfolds over time. For example, the game played between wireless nodes at one point in time may significantly differ from the game played by the same nodes at another time step.

2.2.2.2 Stochastic game

When several consecutive single-stage games can be represented as one multi-stage game and played repeatedly, we use the stochastic game representation (also called Markov Game or MG). For example, consider a 2-stage game where in the first stage agents attempt to coordinate by playing the Two-lane road game from Example 4 and in the second stage they play a Four-lane road game. Clearly in the second stage agents can condition their actions based on the outcome of the first stage. Stochastic games [Owen, 1995] thus model the strategic interactions in games composed of multiple stages.

Definition 9 (Stochastic game)**.** *A stochastic game is a tuple* $(N, \mathbb{A}, P_{i \in N}, S, T)$*, where:*

- $N = \{1, \ldots, \mathcal{N}\}$ *is a set of* $\mathcal{N}$ *agents.*
- $\mathbb{A} = A_1 \times \cdots \times A_{\mathcal{N}}$ *is the space of all possible joint actions, where* $A_i = \{a_i^1, \ldots, a_i^{k_i}\}$ *is the individual (finite) set of* k_i *actions available to agent* $i \in N$*.*
- $P_i : S \times \mathbb{A} \to \mathbb{R}$ *is the individual payoff function of agent* $i \in N$*.*

- $S = \{s^1, \ldots, s^M\}$ *is a finite set of system states.*
- $T : S \times \mathbb{A} \rightarrow \pi(S)$ *is the transition function.*

Stochastic games extend repeated normal form games to multiple states. $A_i(s^m)$ is now agent i's action set in state $s^m \in S$, where $m : 1, \ldots, M$. The transition function $T(s^m, \vec{a}^m, s^n)$ specifies the probability with which the system will transition from state s^m to state s^n under the joint action $\vec{a}^m$ in state s^m, where $\vec{a}^m = (a_1^m, \ldots, a_{\mathcal{N}}^m)$ with $a_i^m \in A_i(s^m)$. The individual payoff function $P_i(s^m, \vec{a}^m, s^n)$ of agent i now depends on the current state s^m, the next state s^n and the joint action $\vec{a}^m$ in state s^m. A special form of stochastic games are Multi-agent Markov Decision Processes (MMDPs) [Boutilier, 1996; Claus & Boutilier, 1998] where agents are fully cooperative and share the same payoff function. Although fully cooperative, in our games agents do not share the same payoff function.

One limitation of stochastic games is that agents are assumed to be aware of the complete system state, i.e. agents have a view of the entire system. This is certainly a disadvantage from multi-agent perspective, where we assume that central control is not available. A more suitable framework, in which agents have only local information, is that of **Decentralized Markov games** (DEC-MGs) [Aras *et al.*, 2004].

Definition 10 (Decentralized Markov game)**.** *A decentralized Markov game is a tuple* $(N, \mathbb{A}, P_{i \in N}, S, T, \Omega, O)$*, where:*

- $N = \{1, \ldots, \mathcal{N}\}$ *is a set of* $\mathcal{N}$ *agents.*
- $\mathbb{A} = A_1 \times \cdots \times A_{\mathcal{N}}$ *is the space of all possible joint actions, where* $A_i = \{a_i^1, \ldots, a_i^{k_i}\}$ *is the individual (finite) set of* k_i *actions available to agent* $i \in N$*.*
- $P_i : S \times \mathbb{A} \times S \rightarrow \mathbb{R}$ *is the individual payoff function of agent* $i \in N$*.*
- $S = S_1 \times \cdots \times S_{\mathcal{N}}$ *is a finite set of system states, where* S_i *is the set of local states of agent* i*.*
- $T : S \times \mathbb{A} \rightarrow \pi(S)$ *is the transition function.*
- $\Omega = \Omega_1 \times \cdots \times \Omega_{\mathcal{N}}$ *is a finite set of joint observations, where* Ω_i *is the set of observations of agent* i*.*
- $O : S \times \mathbb{A} \times S \times \Omega \rightarrow \mathbb{R}$ *is the observation function.* $O(o_b | s^m, \vec{a}^m, s^n)$ *is the probability of making observation* $o_b \in O$ *when taking joint action* $\vec{a}^m$ *in state* s^m *and transitioning to state* s^n *as a result.*

Here each system state $\vec{s} = (s_1, \ldots, s_{\mathcal{N}})$, with $s_i \in S_i$, contains all information about the current local state of agents. However, DEC-MGs assume information exchange between agents in order to study how agents can learn to cooperate if communication were possible. Our assumption in this thesis is that communication is costly and therefore agents are not allowed to exchange any information regarding their local states.

2.2.2.3 *Graphical game*

Graphical models offer the tools to study games, which impose restrictions on the strategic interactions between agents. For example, the underlying network structure in WSNs specifies direct payoff influences between neighboring agents and payoff independence between distant nodes.

Definition 11 (Graphical game)**.** *A graphical game is a tuple* $((N, E), \mathbb{A}, P_{i \in N})$*, where:*

- (N, E) *is an undirected graph, where* $N = \{1, \ldots, \mathcal{N}\}$ *is a set of* $\mathcal{N}$ *nodes and* E *is the set of edges. Here* $n_i = \{j | j \in N, e_{i,j} \in E\}$ *is the set of all neighbors* j *of agent* i*, for which there is an edge* $e_{i,j} \in E$ *between* i *and* j*.*
- $\mathbb{A} = A_1 \times \cdots \times A_{\mathcal{N}}$ *is the space of all possible joint actions, where* $A_i = \{a_i^1, \ldots, a_i^{k_i}\}$ *is the individual (finite) set of* k_i *actions available to agent* $i \in N$*.*
- $P_i : \times_{j \in n_i \cup \{i\}} A_j \to \mathbb{R}$ *is the individual payoff function of agent* $i \in N$*.*

A graphical game is a special case of a normal form game where an agent i's payoff function P_i is defined over the joint actions of its neighborhood $\times_{j \in n_i \cup \{i\}} A_j$ rather than over the entire joint action set $\mathbb{A}$. Each graphical game $((N, E), \mathbb{A}, P_{i \in N})$ represents a normal form game $(N, \mathbb{A}, P'_{i \in N})$ where:

- $\mathbb{A} = A_1 \times \ldots \times A_{\mathcal{N}}$, is the joint action set with A_i the action set of player i, identical in both games.
- the payoff function $P'_i : \mathbb{A} \to \mathbb{R}$ is defined as $P'_i(\vec{a}) = P_i(\vec{a}\,|_{n_i \cup \{i\}}), \forall \vec{a} \in \mathbb{A}$, where $\vec{a}\,|_S$ denotes the actions in $\vec{a}$ restricted to the agents in set S.

Graphical games (GGs) are most appropriate for games with sparse interactions between players. While the normal form game representation requires parameters exponential in the number of players, the parameters of GGs are exponential only in the size of the largest local neighborhood [Kearns, 2007]. The payoff to each player depends only on its actions and on the actions of its direct neighbors, rather than on the actions of the entire population. Thus, the representational benefits of GGs

are much greater when there is a small number of strong influences between agents. Most literature on graphical games, however, studies only two-action games. It offers algorithms for the computation of Nash equilibria and analyzes their complexity. Galeotti *et al.* [2010] propose a framework similar to GGs, which they name network games (NGs). NGs focus more on the structure of equilibria and its interaction with the underlying topology of the game. They study the relationship between the network topology and the behavior of agents. In this thesis we are interested in the way coordination can be achieved when agents are interacting on a graph and have only local knowledge. Although we are not focusing on computing the equilibria or examining their dependence on the game topology, we will use the notion of graphical games to explain agent interactions. Still, we study the effect of the topology on the convergence rate of agents.

game	agents		knowledge		states		payoff	
representation	single	multiple	global	local	single	multiple	common	individual
MDP	✓		✓			✓	✓	
DEC-MDP		✓		✓		✓	✓	
MMDP		✓	✓			✓	✓	
MG		✓	✓			✓		✓
DEC-MG		✓		✓		✓		✓
NFG		✓	✓		✓			✓
GG		✓		✓	✓			✓

Table 2.7: Comparison between different game representations.

We compare the characteristics of the different game representations in Table 2.7. For consistency and comparison we add here the Markov Decision Process (MDP) and its decentralized version (DEC-MDP). The MDP is a model for sequential decision making of a single agent in multi-stage games. An extension of MDPs for multiple agents is the Decentralized Markov Decision Process where the agents take decisions based on local information and obtain a common payoff. The games we study in this thesis are most related to DEC-MGs, since they model multi-agent multi-stage games where agents have only local information and receive individual payoffs. However this game representation assumes that agents communicate to share local state information, while in our games agents learn only based on their local observations.

2.3 Reinforcement learning

Game theory tells us what rational strategies are in a given strategic interaction. It does so by analyzing the payoff matrix of the game from a global perspective (i.e. by looking at the entire payoff matrix) and computing the strategies, for which agents will maximize their expected payoff. However, from the perspective of individual agents in a decentralized multi-agent system, such computations might be impossible, due to the limited information available to them. Furthermore, in Section 2.1 we saw that even if agents are somehow aware of the equilibrium strategies, they might still have a hard time choosing among the different Nash equilibria. This equilibrium selection problem is difficult by itself [Harsanyi & Selten, 1988; Boutilier, 1996], since there is no central entity that can instruct agents what the "correct" actions are. Therefore, in order to successfully (anti-)coordinate in repeated games, agents need to evaluate the expected payoff of their strategies by trial and error and learn which actions to take in which situations. We are interested here in implementing algorithms that help agents (anti-)coordinate in a decentralized manner. In Section 1.4.1.2 we motivated the need for learning in dynamic environments, populated by highly constrained agents. Here we will describe different learning approaches that align the objectives of individual agents with the global system objective. By maximizing the individual's welfare, our algorithms aim to help agents achieve successful (anti-)coordination as a group.

Reinforcement Learning (RL) is a machine-learning technique that allows an agent to learn to select optimal actions in an unknown dynamic environment by trial and error [Sutton & Barto, 1998]. The agent performs actions in its environment and as a result acquires feedback, which shows the effect of its actions. This feedback signal is called reinforcement or reward, and hence the name of this field. Reinforcement learning was originally introduced as a single-agent framework and only later extended to multi-agent systems. Since in this thesis we are interested in systems comprised of multiple agents, in the following description we will assume the perspective of multi-agent systems.

Two main categories of RL techniques exist. The model-based techniques assume some form of knowledge of the transition and reward functions. Agents have (or learn) an explicit model of the dynamics of the system and compute an optimal behavior given that model. Model-free techniques, on the other hand, do not require explicit model of the environment. Agents learn the quality of their actions using the reinforcements obtained by interacting with the system. Using these reinforcements, the goal of each agent is to learn to select actions that result in positive feedback more often and to avoid actions with negative outcome. In this thesis we will

consider only model-free methods, since in our WSN domain we cannot assume that sensor nodes possess (accurate) global, or even local, information on the dynamics of the system. Although many types of multi-agent learners have been proposed, in the context of this thesis we distinguish between two main types of multi-agent learners — independent learners (ILs) and joint-action learners (JALs). ILs are agents who apply their learning algorithm while not explicitly modeling the actions of other agents in the system. They learn simply the effect of their own actions in the environment, as if they are acting independently. Joint-action learners, in contrast, observe the joint actions in order to learn the effect of their own actions in conjunction with those of other agents. Different types of JALs exist, based on the number of other agents considered. Note that in our application domain of WSNs agents cannot observe directly the strategies of others, but only the effect of their own actions (see Section 1.4.1.1). The only information coming from the environment is the reward signal. Since any additional information comes at communication costs, in this thesis we will consider only independent learners. Despite the fact that JALs use more information (i.e. the actions of others) during learning, Claus & Boutilier [1998] have shown that their performance does not significantly differ from that of ILs.

In some scenarios, rewards are given only after a sequence of actions. These *delayed rewards* make the learning problem more difficult, since agents need to learn to take correct decisions, based on a payoff that can take place arbitrary far in the future. In addition, the rewards may be stochastic, such that the same action may yield different payoffs at different time steps. The transition function may also be stochastic where an action in a given state may lead the agent to one of multiple next states with a certain probability. Another issue the agent needs to consider is the *exploration-exploitation trade-off*. On the one hand, agents need to explore their environment in order to gather more information on the quality of their actions. On the other hand, they need to exploit desirable actions and avoid unsuccessful ones. Several learning algorithms exist that can help agents cope with the above challenges. Here we will present some of the most popular ones — **Q-learning**, **Learning automaton** and **Win-stay lose-shift**.

2.3.1 Q-learning

The Q-learning algorithm [Watkins, 1989] estimates the quality of agent's actions in each state in order to derive an optimal policy. In the literature on reinforcement learning the **policy** specifies the action that the agent should take in every perceivable state of the system. This definition coincides with the term strategy that we

use in game theory (cf. Section 2.1). Although in other domains distinction can be made, in the context of this thesis we use the terms policy and strategy interchangeably. A **value function** helps the agent keep track of the performance of its actions. The quality (also called Q-value) of an action in a given state indicates how good (or bad) the action is in that particular state. A separate **action selection mechanism** is then applied to decide which action the agent should pick in the current state. Once the agent executes an action in a given state, it updates the Q-value of that action, as shown in Definition 12. When agents are involved in a multi-stage game, the goal of Q-learning is to approximate the optimal state-action values without having an actual model of the world in the form of transition and payoff functions (which themselves may be stochastic).

Definition 12 (Q-value update)**.** *The Q-value $Q(s,a)$ is the agent's current estimate of the expected discounted payoff of taking action a in state s. The Q-value of each state-action pair is updated based on the current Q-value $Q(s,a)$ and the immediate payoff p after taking action a in s and arriving in s':*

$$Q(s,a) \leftarrow (1-\lambda)Q(s,a) + \lambda\Big[p + \gamma \max_{a'} Q(s',a')\Big]$$

where $\lambda \in (0,1]$ is the learning rate, $\gamma \in [0,1]$ is the discount factor and $\max_{a'} Q(s',a')$ is the optimal state-action value that can be obtained in the next state s' based on the current estimates.

Although conventionally α is used for the learning rate, here we have intentionally replaced it with λ in order to avoid ambiguity with the parameter α of our algorithm, introduced in Chapter 3. Similarly, for consistency throughout this thesis we use p for the reward signal (or payoff), although typically r is written instead.

2.3.1.1 Learning rate and discount factor

As time progresses, these estimates become more accurate. The Q-values are computed based on the previous estimates in a process known as bootstrapping. The starting Q-values can be initialized in a number of ways, depending on the problem at hand. Some typically used initialization methods are random, pessimistic, optimistic, or based on domain knowledge. The learning rate $\lambda \in (0,1]$ controls the weight of recent experience as compared to past experience. A value of 0 will make the agent discard any recently obtained rewards and therefore it will not learn anything. A value of 1, on the other hand, tells the agent to discard any previous experience and consider only the immediate effect of its actions. Clearly, λ affects the rate of convergence. The learning rate should be set large enough to overcome

any initial conditions, and yet small enough to assure that the policy will eventually converge to the optimal one. It is also possible to vary λ through time, starting with a larger value and gradually decreasing it. However, in the nonstationary environments that we consider in this thesis the agent should be able to constantly adapt to changes and therefore the learning rate should never become 0. Another possibility is to vary λ according to the obtained payoff, as done by Bowling & Veloso [2002]. In stationary environments having the Markov property (see Definition 13), under the assumption that all state-action values are updated infinitely often using a suitable learning rate, Tsitsiklis [1994] has proven that the Q-values will always converge to the optimal values. The discount factor $\gamma \in [0, 1]$ weights the importance of short-term reinforcements, as compared to distant future reinforcements. A value of 0 makes the agent consider only immediate rewards, disregarding what comes ahead, while a value of 1 puts more weight on future expected rewards. The value of the discount factor needs to be carefully considered, as it is illustrated in the following example.

Example 7 (Robot in a maze)**.** *A robot has to repeatedly find its way out of a given maze. The decision (or action) at each turn (or state) in the maze gives a negative reward to the agent, since the robot spends energy. Only the last turn that leads to the goal, i.e. exiting the maze, provides a large positive reinforcement. Thus, the robot needs to learn to navigate out of the maze, spending the least amount of energy.*

Since the agent is faced with a *delayed reward* at the end of the maze, it has to put more weight on future expected rewards, rather than on immediate payoffs. Moreover, the robot needs to propagate the positive reinforcement to earlier states, so that in the next runs it can take better decisions and exit the maze faster. This propagation of rewards is the effect of the bootstrapping process described earlier.

2.3.1.2 Single-stage vs. multi-stage

The Q-value update rule in Definition 12 shows how agents maintain an estimate of the payoff of each action at each state in a multi-stage game, such as the maze example above. Here we make an important distinction between agent (or local) state and system (or global) state. By agent state we mean the information that is available to the agent when making its decisions. In this section we use agent states to explain how the learning algorithm helps agents use environmental feedback to improve their behavior. A system state, on the other hand, contains the collection of the information available to each agent at a given time step. In multi-stage games the

system transitions between states as a result of agents' actions. However, as stated in Section 1.4.1.2 agents have no global knowledge of the system state. They are aware only of their local agent state. Furthermore, in some scenarios (as we will see in Chapters 3 and 4) the information available to agents is insufficient to make any distinction between system states. Note that dependency between system states does exist, but the agent has no means of knowing when state transitions occur. In these settings the learning algorithm of agents assumes there is no dependency between time steps and regards the game as a repeated normal form game, rather than a multi-stage game. In a repeated normal form game, the notion of different agent states (and hence transition function) is no longer relevant, since the agent is always in the same state. This setting is called non-associative learning [Sutton & Barto, 1998] — the feedback signal is the only information that the agent receives from its environment. The agent needs to learn the most favorable action given that feedback. However, the reinforcement signal may change over time as a result of the changing system state, which makes the learning problem challenging. When the multi-agent system consists of only one state, we say that the system is single-state (or stateless). An example of a single-agent stateless system is the k-armed bandit problem, originally introduced by Robbins [1952].

Example 8 (k-armed bandit)**.** *A gambler has to decide which arm of a k-armed slot machine to pull in order to maximize his total payoff in a series of trials. The reward of each lever is drawn from a distribution associated to that specific arm, but is unknown to the player.*

In this example we consider fixed distributions, although in more general settings the expected payoff of each arm may vary over time [Koulouriotis & Xanthopoulos, 2008]. Unknown to the agent, these distributions may also change as a result of its actions, e.g. the expected payoff of a given lever may drop as a result of a large win. If the agent would know of this relation, it could use a multi-stage algorithm to optimize its behavior. Since the agent is unaware of such dependencies, it regards the problem as single-state. The expected payoff of each action is independent of the previous action. Thus, the Q-values become estimates of the actual payoff of each action, rather than of each state-action pair. The state-action values $Q(s, a)$ in Definition 12 reduce to only $Q(a)$ and since the future state is always the same, there is no need for the discount factor γ. The Q-value update rule then simplifies to:

$$Q(a) \leftarrow (1 - \lambda)Q(a) + \lambda p$$

The agent needs to learn the effect of each arm in order to maximize its payoff over

some time period.

2.3.1.3 Action selection mechanisms

Starting with no prior information about the reward distribution of each lever, the gambler needs to explore his actions in order to gain more information on the expected payoff of each arm. At the same time, the agent wants to select the arm with the highest expected reward, so as to maximize his earnings. This example clearly illustrates the *exploration-exploitation trade-off*, as the agent is faced with the decision whether to gather more information, or optimally use the current information. The action selection function helps the agent balance this trade-off. If the agent is using the currently best action, he is applying a greedy action selection. However, always selecting the best action may lead to suboptimal performance, since the agent does not have accurate information on the expected payoffs of each arm. To gain more information, while still performing optimally based on the current estimates, the agent may apply the ϵ-greedy mechanism. This action selection rule lets the agent use the currently best action most of the time, while with a small probability ϵ the agent will select a uniformly random action, independent of the current Q-values. Thus, the parameter ϵ controls the exploration probability, while with probability $(1 - \epsilon)$ the agent will exploit its knowledge.

One major drawback of ϵ-greedy is that it explores actions using a uniform probability distribution. The probability of exploring the second-best action is the same as that of selecting the worst action. An alternative action selection rule is softmax, which overcomes this drawback. Softmax selects actions based on a probability distribution derived from the current estimates, rather than a uniform probability distribution. In other words, the probability $\pi(a)$ of selecting action a out of k available actions is based on the current estimate $Q(a)$ according to the Boltzmann distribution:

$$\pi(a) = \frac{e^{\frac{Q(a)}{\tau}}}{\sum_{b=1}^{k} e^{\frac{Q(b)}{\tau}}}$$

where $\tau \in (0, \infty)$ is the temperature parameter controlling how greedily the agent chooses its actions. High temperature causes actions to be all (nearly) equiprobable. Low values of τ, on the other hand, make actions with higher estimates to be selected more often than those with lower estimates. Thus, as $\tau \to 0$ softmax behaves more like the greedy action selection rule.

In multi-agent settings, such as the ones considered in this thesis, agents cannot assume that the environment is static. A good action at one point in time may become bad later on, due to the activities of other agents in the system. Therefore

neither ϵ (when using ϵ-greedy), nor τ (when using softmax) should approach 0. Each agent needs to constantly explore for better alternatives and update its estimates based on recent information.

2.3.2 Learning automaton

Similarly to Q-learning, the learning automaton (LA) algorithm helps the agent use feedback from the environment to increase the performance of its behavior through time. Actions are drawn according to a probability distribution that is adjusted based on their relative success. LA uses a **learning scheme** to update the probabilities of selecting each action, without maintaining an estimate of the expected payoff. According to the law of effect, the learning scheme increases the selection probability of good actions and decreases that of unfavorable actions. For comparison, Q-learning uses a value function to update the estimates of the actual payoff of each action, and a separate action selection mechanism to decide how the agent should pick its actions. While in Q-learning the policy is derived from the current estimates, in LA the probability distribution *is* the policy.

Several learning schemes have been proposed in the past, the general form of which is given below. The probability $\pi(a) \in [0, 1]$ of selecting action $a \in A$ out of k available discrete actions is updated based on its current probability and the obtained reward $p \in [0, 1]$:

$$\pi(a) \leftarrow \pi(a) + \lambda p(1 - \pi(a)) - \mu(1 - p)\pi(a)$$

where $\lambda \in [0, 1]$ is the reward parameter and $\mu \in [0, 1]$ is the penalty parameter. Although typically β is used for the penalty parameter, here we use μ to avoid ambiguity with the parameter β of our approach, presented in Chapter 3. The selection probability $\pi(b) \in [0, 1]$ of all other discrete actions $b \in A, b \neq a$ is updated in a similar manner:

$$\pi(b) \leftarrow \pi(b) - \lambda p\pi(b) + \mu(1 - p)\left(\frac{1}{k - 1} - \pi(b)\right)$$

In both equations the parameters λ and μ can be set according to three common learning schemes proposed in literature [Hilgard, 1948]:

- Linear reward-inaction (L_{R-I}) where $\mu = 0$
- Linear reward-penalty (L_{R-P}) where $\mu = \lambda$
- Linear reward-ϵ-penalty ($L_{R-\epsilon P}$) where $\mu \ll \lambda$

In this thesis we shall not compare the behavior of the learning agent in the above schemes. We refer the interested reader to Peeters [2008]. For simplicity here we will always use the L_{R-I} scheme, for which the update function simplifies to:

$$\pi(a) \leftarrow \pi(a) + \lambda p(1 - \pi(a)) \tag{2.1}$$

$$\pi(b) \leftarrow \pi(b) - \lambda p \pi(b) \qquad \forall b \neq a \tag{2.2}$$

In this case λ is sometimes also called the learning rate, as in the Q-learning algorithm. The LA algorithm can also be extended to solve multi-stage problems. When agents are involved in a stochastic game, they keep a probability distribution of actions for each state and update the distribution related to the corresponding state they visit [Peeters, 2008; Vrancx, 2010]. Moreover, in the above description we assume that actions are drawn from a discrete set of available actions. A generalization of LA to continuous actions is introduced by Santharam *et al.* [1994].

2.3.3 Win-Stay Lose-Shift

Another learning algorithm studied in literature is the Win-Stay Lose-Shift (WSLS). It is a simple, yet powerful learning rule that can be applied in virtually any type of repeated decision problems. WSLS (also called Pavlov strategy) is a widespread rule in biology [Thorndike, 1911] and as a consequence has been widely studied in computer science [Robbins, 1952]. It was first presented as win-stay lose-change by Kelley *et al.* [1962] and later analyzed by Nowak & Sigmund [1993] in the iterated Prisoner's Dilemma (IPD) game (see Example 2). The latter authors showed that WSLS outperforms another simple rule — tit-for-tat. The basic idea of the WSLS algorithm is that the agent will keep on selecting the same action, as long as its payoff is above a certain threshold level (also called aspiration level), and will change its action when the payoff drops below that level. It resembles greedy Q-learning (see Section 2.3.1) in single-state environments with learning rate $\lambda = 1$. As such, WSLS requires no parameter that needs to be tuned, nor a separate action selection mechanism. Another positive aspect of this rule is that agents require only minimal information when updating their actions. Unlike tit-for-tat, which requires information on the action of others, WSLS reacts only to the own action and payoff. This is very beneficial in domains where agents are not able to freely observe the actions of other agents. For example, in grid computing agents cannot be assumed to see where other agents submit their jobs [Galstyan *et al.*, 2005]. A disadvantage, however, is that WSLS makes no difference between losing and losing big [Kraines & Kraines, 1995]. Nevertheless, as we will see in Chapter 3, it performs well in games

with binary feedback. In addition, WSLS is successful in environments where exploration is costly, such as in WSNs (see Chapter 5). It quickly finds a good solution and does not necessarily look for the optimal. This behavior is indeed satisfactory in real-world scenarios where near-optimal solutions are well tolerated.

WSLS selects an action based on the outcome of the last selected action and therefore requires no memory at all. This is sometimes referred to as memory of one, since the agent needs to "remember" its current action. In this thesis the term memory stands for the history of past plays and not for the current play, therefore we say that WSLS is memoryless. This memoryless behavior can be appealing for highly constrained agents, who lack the ability to (reliably) store information. However, due to this property, WSLS has a low performance in stochastic environments. Posch [1999] studied WSLS and the impact of noise on the behavior of agents, i.e. when players sometimes make errors in the implementation of their actions. He extended the algorithm to include a memory of past interactions, which determines the aspiration levels of actions. A related approach is introduced by Barrett & Zollman [2009]. They presented it in the context of the evolution of language and named it Win-Stay Lose-Randomize (WSLR). Agents stick to any successful action in the past and choose an action at random if unsuccessful.

To this day, WSLS has been studied mostly in the context of iterated Prisoner's Dilemma as a rule that promotes cooperation, as opposed to defection. However, in this thesis we are not looking at games where agents care only of their individual payoff and where defection is (individually) preferred to cooperation. In our games agents are fully cooperative in the sense that they have the same goal, only reaching it is hard. WSLS allows agents to quickly (anti-)coordinate in a decentralized manner even under very limited feedback from the environment.

2.4 Markov chains

Throughout Section 2.3 we outlined how the learning algorithm helps agents use local environmental feedback to improve their individual behavior. In this section we describe a framework that will help us examine the global dynamics of the system. To study the convergence properties of our learning algorithms and describe how policies change over time, we use the theory of Markov chains (MCs). The latter theory relies on the Markov property, which we shall define first.

Definition 13 (Markov property)**.** *A stochastic process involving a random variable* $\{X(t)\}_{t\geq 0}$ *possesses the Markov property if the conditional distribution of* $X(t+1)$

depends only on $X(t)$ and not on previous values:

$$Pr[X(t{+}1) = y_{t+1}|X(t) = y_t, X(t{-}1) = y_{t-1}, \ldots, X(0) = y_0] = Pr[X(t{+}1) = y_{t+1}|X(t) = y_t]$$

Definition 14 (Markov chain)**.** *A Markov chain is a stochastic process in which a sequence of random variables $\{X(t)\}_{t\geq 0}$ takes values in a set S. The transition probabilities $Pr[X(t+1)|X(t)]$ need to satisfy the Markov property.*

In the context of the reinforcement learning framework, the variable $X(t)$ represents the current system state $s \in S$. A system state represents the actions of all agents at a given time step, together with any information available to them. At each "step" the process moves from one state to another with a given probability. The Markov property states that the transition probabilities between system states in S are independent of past states given the current state. The transition probability $Pr[X(t+n) = s_j|X(t) = s_i]$ of going from state s_i to state s_j in n steps is written as $\pi_{ij}^{(n)}$ for short. With probability $\pi_{ii}^{(1)}$ (or simply π_{ii}) the process remains in state s_i in the next step. It is useful here to introduce some related definitions.

Definition 15 (Accessible state)**.** *State s_j is accessible (or reachable) from state s_i if $\pi_{ij}^{(n)} > 0$ for a given $n \in \mathbb{N}$.*

Definition 16 (Ergodic chain)**.** *A Markov chain is ergodic (or irreducible), if any state is accessible from any other state (not necessarily in one step).*

Definition 17 (Absorbing state)**.** *A state s_i is called an absorbing state if $\pi_{ii} = 1$ and consequently $\pi_{ij} = 0$ for $i \neq j$.*

Definition 18 (Absorbing chain)**.** *A Markov chain is absorbing if it has at least one absorbing state, and if from every state it is possible to go to an absorbing state (not necessarily in one step).*

Definition 19 (Transient state)**.** *In an absorbing Markov chain a state is called transient if it is not absorbing.*

The Markov chain framework will allow us to calculate the probability that agents will (anti-)coordinate their actions in different settings and the expected number of time steps to convergence. It is important to note that agents are not able to compute the expected convergence duration by themselves. We, as system designers, take a global view on the system in order to calculate the probability of and time to convergence of the behavior of agents. We will use Markov chains as a tool for analyzing the learning process of agents.

2.5 Summary

In this chapter we presented the tools that we need to represent and solve decentralized coordination problems in large multi-agent systems. We made an overview of several relevant game-theoretic concepts that allow us to determine the outcomes of strategic interactions between agents. We then separately studied coordination and anti-coordination games, as well as the link between the two game types. In addition, we learned different formal representations of these games together with their advantages and disadvantages. We also presented the reinforcement learning framework that helps individual agents (anti-)coordinate their actions in a decentralized and self-organizing way. The theory of Markov chains, on the other hand, allows us to examine the global behavior of the system and study the convergence properties of our learning algorithms.

Chapter 3

Pure coordination: convention emergence

In this chapter we will describe one aspect from the (anti-)coordination problem in wireless sensor networks (WSNs), namely pure coordination. Recall that nodes in the WSN are involved in a complex game where each node needs to both synchronize with some agents in order to forward messages and at the same time desynchronize with others so that interference is minimized. Here we study only the pure coordination problem of agents and apply it in the context of convention emergence (explained below). Thus, in this chapter we depart from the WSN domain and study general (abstract) pure coordination games as done in literature. Nevertheless, all our choices and examples are motivated from the WSN perspective. We investigate the answer to the following question:

Q1: *How can conventions emerge in a decentralized manner in pure coordination games?*

We investigate how conventions can emerge as a solution to repeated decentralized pure coordination problems in large multi-agent systems, in the absence of central control. Moreover, we consider limited environmental feedback and highly constrained agents lacking any global information, as is the case in WSNs.

The main contributions of this chapter are that we propose an approach for emergent coordination between agents in the absence of a central entity and perform

extensive theoretical and empirical studies. Our approach is called Win-Stay Lose-probabilistic-Shift (WSLpS) and is related to two well-known strategies in game theory, that have been applied in other domains. Using WSLpS, agents engaged in a repeated pure coordination game can all learn to select the same action through only local interactions. Our approach achieves 100% convergence, scales in the number of agents and requires no memory of previous interactions, given the last play. Through extensive theoretical and empirical studies we investigate the speed of convergence of agents with respect to both different topological configurations and different interaction models. We explore the convergence duration in ring, scale-free and fully connected topologies where agents may have 2, 3 or 5 available actions. We study also the behavior of agents in random 2-player interactions with binary payoff and in multi-player interactions with a more informative feedback signal. Finally, we study the effect of local observation on the convergence rate and show how all agents can always reach mutually beneficial outcome based only on local interactions and limited feedback.

3.1 Introduction

Easley & Kleinberg [2010] identify informational effects and network effects as the two main reasons why individuals might prefer to imitate the behavior of others. Informational effects are a result of the belief that the behavior of other people conveys information about what they know. Therefore observing this behavior and copying it might be a rational decision. Network effects, on the other hand, capture the notion that for some kinds of decisions individuals incur an explicit benefit when they align their behavior with that of others. For this reason the network effects are also called direct-benefit effects. (De)synchronization in our WSN scenario displays both effects, which motivate the need for coordination in the topic of convention emergence.

Common interest and conflicting interest coordination games were described in detail in Section 2.2.1.1. In each of these two games agents can realize mutual gains by selecting the same action in the presence of several alternatives. Although nodes in a WSN may compete for the wireless medium, we assume that they belong to the same user and share the common interest of forwarding their messages towards the sink. Therefore, in this chapter we will consider only pure (or common interest) coordination games to study how highly constrained agents can reach a common solution, i.e. how conventions can emerge, in a decentralized setting with limited environmental feedback.

3.1.1 Conventions

Recall that in pure coordination games agents benefit from selecting the same action as others out of several alternatives (see Section 2.2.1.1 for details). In this thesis we see conventions as solutions to decentralized coordination problems. Lewis [1969] defines a convention as a regularity in behavior of agents.

Definition 20 (Convention)**.** *A regularity R in the behavior of members of a population P when they are agents in a recurrent situation S is a convention if and only if, in any instance of S among members of P:*

- *everyone conforms to R;*
- *everyone expects everyone else to conform to R;*
- *everyone prefers to conform to R on condition that the others do, since S is a coordination problem and uniform conformity to R is a proper coordination equilibrium in S.*

An important question then, addressed by **Q1**, is how this regularity can become established in a population, when agents have the same preferences and a number of alternatives to choose from. One way in which a convention can come into existence is when one action is agreed upon in advance, i.e. agent behavior can be designed or programmed off-line by a central entity, before agents are involved in the coordination game [Shoham & Tennenholtz, 1995]. Traffic laws are an example of such pre-defined conventions where, for example, all vehicles must stop at red light. Alternatively, coordination could emerge on-line as a result of a central authority that regulates behavior (e.g. through sanctions) or computes an outcome based on common choice (e.g. using voting mechanisms). However quite often such a central control might be unavailable, or costly to set up, as it is the case with WSNs. Wireless nodes scattered in a large and dynamic environment simply cannot rely on a pre-programmed behavior or centralized control. Similarly, when a telephone call between two persons gets unexpectedly cut off, there is no central authority to select who should call back. In these settings, agents will simply rely on a set of "unwritten laws" or customs that have worked well in the past and have become conventions. Note that in this thesis we will not investigate the behavior of human agents.

In the context of this dissertation we say that a convention is a regularity of agent behavior emerged as a result of repeated interactions. If all agents in a repeated pure coordination game have learned to select the same action at every iteration, we say that they have formed a convention. Therefore, a convention can be seen as

a solution to a pure coordination problem, where agents can realize mutual gains if they exhibit common behavior, i.e. if a convention emerges in the MAS.

3.1.2 Aim

In this chapter we study how conventions can emerge as a solution to repeated decentralized coordination problems in large multi-agent systems, where agents are arranged in different interaction graphs (or topologies). We examine the coordination game through repeated local interactions between members of a society in the absence of central authority. We propose an approach that guides agents in selecting their actions in order to reach a mutually beneficial outcome in as few time steps as possible. In particular, we are interested in the average number of repeated interactions until *all* agents arrive at a convention when using our on-line learning approach in different topological configurations. Note that the purpose of our approach is to be applied in engineering applications where agents have no individual preferences. For this reason we are not investigating the behavior of human agents and their individual welfare.

To illustrate the concept of conventions in WSNs, we present an example of a pure coordination problem, which we will study and elaborate on later in this chapter.

Example 9 (WSN pure coordination)**.** *Consider an arbitrary network of nodes, which typically communicate on different frequencies (or channels) in order to avoid radio interference. Every so often, all nodes need to switch to the same channel, regardless which, in order to exchange control messages, e.g. to synchronize their clocks. In the absence of central control, how can all nodes in the wireless sensor network learn over time to select the same broadcast frequency?*

Here energy constrained sensor nodes need to quickly learn to select the same frequency in repeated interactions under very limited feedback from the environment. In this scenario the longer this learning process takes, the more costly it becomes for agents. Note that we do not distinguish between channels of different interference levels (or quality), but only between high and low interference. In this example we are concerned with finding a channel with sufficient quality to allow for proper communication, and not necessarily the best channel.

In the next section we situate our work on emergent conventions in the context of related work and then outline our contributions in Section 3.3. We study in detail the coordination game that our agents are involved in and the underlying interaction models in Sections 3.4 and 3.5, respectively. We describe our approach in Section 3.6

and investigate its performance in different settings in Section 3.7. We present our conclusions in Section 3.9 and provide some directions for future work.

3.2 Related work

In this section we will examine the related work by grouping it according to a number of characteristics. Some of the main features we explore are topology type, memory size, interaction model and convergence threshold. These characteristics will allow us to compare the different settings used in the literature of convention emergence.

Most related work presented below considers populations of artificial agents as well as human populations, in which players are not self-interested and all aim towards the same goal. We use the same assumption of altruistic agents in this thesis, but we restrict our attention to computer agents.

One of the earliest and most influential works on the study of conventions is that of Lewis [1969]. He explores the emergence of conventions and the evolution of language in signaling games. Later, Axelrod [1986] investigates the factors that speed up convention emergence and the conditions under which a convention remains stable. Young [1993] studies stochastically stable equilibria in coordination games where agents can occasionally explore, or make mistakes. These authors, as well as others [De Vylder, 2008] consider only fully connected **network** of players where each agent can potentially interact with any other agent with the same probability. However, nodes in a sensor network are usually spread over a vast area, forming a specific topology. We cannot assume that the network is fully connected. In that regard, the work of Kittock [1993] is the first to explore the influence of a restrictive interaction model on the evolution of the system. He investigates the performance of agents in different interaction graphs. A similar study is the one of Delgado *et al.* [2003] who investigates the speed of convention emergence in scale-free networks. Restrictive topologies are indeed more general than fully connected topologies and often represent more realistic scenarios, such as nodes in a WSN. Therefore, in this chapter we also study convention emergence in a number of interaction graphs, including a fully connected one. These different models allow us to study the effect of both global interactions (everyone can interact with everyone else) as well as local ones (agents can only play with their immediate neighbors).

A large body of research has studied convention emergence as one-to-one **interactions** between randomly selected individuals of the population [Kittock, 1993; Shoham & Tennenholtz, 1997; Barrett & Zollman, 2009]. With the rise of virtual interaction environments, such as social networks, weblogs and forums, this model

seems to less accurately resemble the evolution of behavior in these societies. Moreover, nodes in a WSN broadcast messages to all nodes in range, thereby affecting the behavior of several others at the same time. Little work has focused on multi-player interactions where an agent can meet a variable number of other players in a single game [Delgado *et al.*, 2003; Villatoro *et al.*, 2011b]. In this chapter we examine both pairwise interactions (agents play 2-player coordination games) as well as multi-player ones (n-player coordination games). In addition, most research considers only 2 actions per agent, while we will explore settings with 3 and 5 actions in order to study the scalability of our approach.

Shoham & Tennenholtz [1993] introduced the Highest Cumulative Reward (HCR) update rule that lets agents select the most successful action in the last l time steps. Thus, the parameter l is the number of previous interactions of the agent, or its **memory**. In the latter work, the authors assume that agents have access to either the entire history of plays, or just a limited memory window. The assumption of unlimited history of interactions is unrealistic in WSNs, where nodes have limited memory capabilities. Wireless nodes also need their memory to store incoming packets and sensor measurements. Moreover, selecting the most successful action in the last l time steps introduces what Villatoro *et al.* refer to as "the frontier effect". A number of agents at the border between two groups, who each has converged to a different action, experience conflicts, but cannot resolve it, since the most successful action for each agent is reinforced by its neighbors. We will see in Section 3.7 that such scenarios do not occur when using our approach. Here we relax the memory assumption altogether and say that agents are memoryless, or that they keep no history of previous interactions, except the current one. Note that this setting can be also viewed as having a memory of one, since the agent keeps its current action in memory. Here we refer to the memory as the history of past interactions and therefore we say that our agents are memoryless. After each play, agents decide to alter or keep their current action, based on the immediate payoff, hence no memory of previous interactions. In fact, Barrett & Zollman [2009] investigate how forgetting past interactions may help agents reach conventions faster in the evolution of language. They conclude that forgetting helps move agents away from suboptimal equilibria and therefore increases the probability of evolving an optimal language in signaling games.

Similarly, the experimental results of Villatoro *et al.* [2011b] also suggest that convergence is faster for smaller memory sizes, but they do not attempt to abandon the use of memory altogether. They introduce a new reward metric which determines the payoffs of agents based on the history of their interactions and they measure the

time steps until full convergence. However, due to their stochastic action-selection policy (ϵ-Greedy), even if 100% of the population select the same action at some moment in time, agents may still escape the convention in the next step and, over time, form new one. In contrast, our action selection approach ensures that agents will **never escape** a convention. It instructs agents to keep their successful actions and to probabilistically select a different one if they encounter conflicts. We name this approach Win-Stay Lose-probabilistic-Shift (WSLpS) and it shares characteristics with two related techniques proposed in literature. Barrett & Zollman [2009] introduced the Win-Stay Lose-Randomize (WSLR) algorithm in the context of the evolution of language. WSLR is maximally forgetful, in the sense that it retains no history of past interactions, and outperforms two different reinforcement learning algorithms that use memory. The authors draw an analogy to a similar algorithm, named Win-Stay Lose-Shift (WSLS) (see Section 2.3.3). It was first presented as win-stay lose-change by Kelley *et al.* [1962] and later analyzed by Nowak & Sigmund [1993] in the iterated Prisoner's Dilemma (IPD) game (see Example 2). As we will see in Section 3.6, our action selection approach resembles both WSLR and WSLS, but differs in the way agents select their action upon failure.

Most of the literature on convention emergence assumes that a convention is reached when at least 90% of the population select the same action [Kittock, 1993; Delgado *et al.*, 2003] (or even 85% by Shoham & Tennenholtz [1997]). However, Villatoro *et al.* [2011b] argue that a **threshold** of 90% is not sufficient to say that agents' behavior has converged. Unless 100% of the population select the same action, there is always the possibility that the majority agents may learn, over time, to select a different action and therefore arrive at a different convention. Moreover, if any group of agents arrive at a different (sub-)convention than the rest of the population, the agents on the border between the different groups will experience conflicts and thus incur costs. In a WSN, for example, if some nodes are in conflict with others, they will deplete their batteries faster than the rest of the network, drastically lowering the overall network lifetime. We require that in order for a coordination problem to be solved there may not exist any sub-coalitions, i.e. groups of agents whose actions differ from those of other groups. For the rest of this chapter we say that the population has reached a convention, or that the individual's behavior has converged, if and only if *all* agents have learned to select the same action, regardless which. We recall that agents have no individual preferences.

In Table 3.1 we summarize the contributions of other researchers according to the features presented in this Section and situate our work in this domain. One important difference between our experimental setting and the one studied by some

Author(s)	Approach	Memory size	Topology type	Interaction model	Number actions	Convergence threshold
Kittock	HCR	limited, none	full, restrictive	2-player	2	90%
Young	Adaptive Play	global limited	full	n-player	2, 3	not reported
Shoham & Tennenholtz	HCR	full, limited	full	2-player	2	85% – 95%
Delgado *et al.*	HCR, GSM	limited	restrictive	2-player, n-player	2	90%
Barrett & Zollman	ARP, smoothed-RL, WSLR	full, limited, none	full	2-player	2,3	99%
Villatoro *et al.*	Q-Learning	limited	full, restrictive	2-player, n-player	2, 3, 4	100%
this chapter	WSLpS	none	full, restrictive	2-player, n-player	2, 3, 5	100%

Table 3.1: Summary of related work.

authors is that we use **synchronous** action selection. At every time step all agents select their actions at the same time. This behavior is common in artificial societies, such as wireless sensor networks, where agents are required to communicate using specific protocols, such as TDMA (see Section 5.2). In other scenarios (e.g. in human populations) synchronous behavior cannot be enforced and therefore different agents may change their actions with different frequencies. A more accurate model then is the stochastic social games, studied by Shoham & Tennenholtz [1997] where agents asynchronously select their actions. Asynchronous action selection implies that some agents may change their action more often than others. Therefore, further study needs to be conducted applying WSLpS in games where agents interact asynchronously in order to compare with previous work.

Tan [1993] investigates whether cooperative agents can outperform agents who learn **independently**. The author concludes that sharing learned policies helps cooperative RL agents to learn faster than independent agents. However, sharing

knowledge comes with a communication cost and requires larger state space, since agents are allowed to observe the state of other agents. Indeed, communicating information between nodes in a WSN is costly in terms of energy consumption. Moreover, it is not always possible — a node cannot communicate with another that is listening on a different channel. Therefore we assume in our work that agents cannot communicate to solve the decentralized coordination problem. Another crucial aspect of our coordination games is the limited environmental feedback and the lack of information on the actions of others. Sen *et al.* [1994] investigate the setting where two agents learn independently without sharing any problem-solving knowledge. In contrast to the above work, they conclude that global coordination can emerge between agents without explicit information sharing. Moreover, in their simulations the authors demonstrate that two agents can learn coordinated actions when neither of them knows that there is another agent in the system. In support of the latter findings, Claus & Boutilier [1998] have shown that despite the use of more information by joint-action learners (JALs), their performance does not significantly differ from that of independent learners (ILs). Claus & Boutilier point out another drawback of JALs besides their larger memory requirements to store the state space. Since JALs maintain beliefs about the strategy of the other agents, beliefs based on a lot of experience require a considerable amount of contrary experience to be overcome. ILs, in contrast, do not consider other agents in the system and therefore can easily adapt to changes in the environment. Due to the limitations of JALs especially in the WSN setting, in our work we use only independent learners.

It is important to note here the relationship between our domain of convention emergence and that of emergence of **cooperation** [de Jong *et al.*, 2008; Segbroeck *et al.*, 2009]. While both fields study how cooperation/coordination can emerge from local interactions, in our coordination games agents are fully cooperative in the sense that they have the same goal, only reaching it is hard. We are not looking at competitive games where agents care only of their individual payoff and where defection is (individually) preferred to cooperation. For this reason we are not studying the factors that promote cooperation. Instead, we are exploring algorithms that help agents coordinate in a decentralized manner under limited environmental feedback. We are interested in the *rate* at which conventions emerge when individual agents learn in different settings.

3.3 Summary of contributions

Many real-world scenarios involve agents participating in a coordination game. In this chapter we study pure coordination games played by computer agents, but we will use the WSN domain as a real-world scenario to motivate our design choices. In the previous chapters we mentioned several other examples, such as robotics, virtual environments and social networking websites. In most settings agents need to reach a convention in the absence of a central authority. Moreover, due to the implied cost of miscoordination, e.g. between battery-powered wireless nodes, conventions need to emerge in as few interactions as possible.

Our main contributions are (1) to propose a decentralized approach for on-line convention emergence, (2) to analytically study its properties and (3) to perform an extensive empirical study analyzing the behavior of agents in a wide range of settings. Our approach is called Win-Stay Lose-probabilistic-Shift, or WSLpS for short. We introduce a parameter that defines the probability with which agents will change their action upon miscoordination (discussed further in Section 3.6). In the limit of this parameter setting, WSLpS can behave as the well-known game-theoretic strategy Win-Stay Lose-Shift (see Section 2.3.3) or the algorithm Win-Stay Lose-Randomize, proposed by Barrett & Zollman [2009]. As such, WSLpS is a generalization of both algorithms. Typically, highly constrained sensor nodes have limited memory capabilities that nodes use to store incoming packets and sensor measurements. Therefore, we require that our learning approach imposes minimal requirements on the memory of agents. Our action selection approach is unique in that it requires *no memory* of previous interactions, given the current one, and drives agents to *full convergence*, i.e. 100% of the agents reach a convention. Moreover, once convention is reached, agents will never change their actions and thus never escape the convention.

We investigate what factors can speed up the convergence process of agents arranged in different *topological configurations*, such as ring, scale-free and fully connected topologies. Since interactions between individuals often incur certain cost (e.g. time, computational resources, etc.), it is necessary that agents reach a convention in the least number of interactions. We study how the convergence speed is affected by the amount of information agents receive from the environment. Due to the limited capabilities of nodes, the wireless agents can detect the result of their actions only as "success" or "failure", lacking any details about the cause of each outcome. Thus, they interpret each result as a (binary) payoff signal from the environment. We also propose a *local observation* strategy that significantly enhances the rate of convergence in some settings. This technique resembles the one

proposed by Villatoro *et al.* [2011a]. However, in our work agents can only observe the current actions of their immediate neighbors and not past actions or agents further in the network. We show how information on the actions of neighbors, if available, can be incorporated in the decision process to help agents reach consensus faster. To reflect the limited environmental feedback of nodes in a WSN, we examine here the convergence duration under a binary (less informative) and multi-valued (more informative) feedback from the environment. We also study how our approach scales for different population sizes and for games where agents have more than 2 actions.

Forwarding a message in WSNs requires the coordination of at least two nodes. A transmission, however, may affect all nodes in range. In addition to *pairwise* interactions, we study the emergence of conventions as *multi-player* games where each agent interacts with possibly many other neighbors at the same time and obtains a single feedback from that interaction. This type of one-to-many encounters is rarely studied in literature, but often occurs in artificial societies, such as WSNs. In such real-life settings, each agent can interact with any number of players simultaneously. In a given coordination game an agent can meet *one* other member, or interact with all neighbors at the same time. A directed unicast signal, for example, will be received by only one agent, while a broadcast message will reach all members in range. Thus, we investigate the performance of agents under two interaction models based on the topological configurations and the information that is available to them. Put differently, we provide insights in the conditions under which conventions emerge faster both in pairwise and in multi-player interactions.

We list here the contributions of this chapter in short:

- We propose WSLpS – a decentralized approach for fast convention emergence that achieves full convergence without requiring the history of previous interactions.
- We analyze the theoretical convergence properties of our algorithm using the theory of Markov chains.
- We investigate how the topological configuration of agents affects the speed of convergence and examine the scalability of our approach in the number of agents and actions.
- We study the behavior of agents in pairwise interactions under a binary (less informative) feedback from the environment and in multi-player interactions with multi-valued (more informative) feedback.

- We show how observation of the actions of neighbors can be used to further speed up the convergence rate.

3.4 The coordination game

In Sections 2.2.1.1 and 3.1 we introduced the coordination game and gave two typical examples of coordination problems. Here we will present in more detail the coordination game that we use in our analysis and simulations. We are interested in pure coordination games, where agents have the same preferences over the different solutions of the coordination problem. Selecting the same action as others yields higher payoff to each agent, than if their actions differ. In this chapter we investigate *how fast* all agents can arrive at the same decision via repeated interactions.

A common assumption in the literature on coordination games is that agents are involved in symmetric 2-player interactions with random individuals from the population. Thus, the same one-on-one coordination game is played throughout the whole network between randomly selected pairs of agents. In some settings it is reasonable to assume that pairs of agents are involved in the same symmetric game where each one receives a payoff based on the joint action. However, quite often in practice the coordination game differs among agents based on the number of players involved in an interaction and their current actions. Moreover, some agents might be unaffected by the outcome of a given game, or simply unaware that they are involved in a game at all. Imagine for example a node A trying to send a message to another node B. If the radio of the latter is switched off, damaged, or currently in communication with another node, the intended receiver cannot detect that someone is trying to transmit information (and hence will receive no payoff from that game). Node A, on the other hand, will receive a negative payoff from the environment (or the wireless medium) due to the energy spent to transmit a message that was not delivered. In our example, the first node started the game by attempting to coordinate with the second, while the latter was unaware of its involvement in that game. Only when both agents select the same action, they will be aware of each other's involvement in the game. However, if not only the initiator, but both agents use this information, i.e. if both agents receive payoff and update their strategy, they will reinforce the chosen action, and consequently make it difficult to coordinate with their other neighbors who might choose different actions. This model of one-sided interactions is also used by Bramoullé *et al.* [2004]. Similarly, Villatoro *et al.* [2011b] experiment with the setting where only the "first" agent updates its strategy, while the "second" agent ignores the obtained payoff from the game. This implementation

resembles our setting where only the initiator of the interaction is aware of the game and updates its strategy.

We model agent interaction as a one-shot game between $n = 2$ agents. This model will be extended for $n > 2$ in the second half this chapter where agents are involved in multi-player interactions. All agents in the network simultaneously select an action and then pairs of agents meet in pairwise coordination games. When two agents "interact", "meet" or "play", only the initiator of the game receives payoff, based on the joint action of these agents. The payoff to the other agent is determined by the game that it itself initiates. Each payoff is determined based on the combination of the actions of the two agents participating in that game, which is also called their joint action. The term "joint action" is used here in the context of the individual coordination games induced by the network topology and not as a way to describe the actions of all agents in the population. As mentioned above, in some cases not all agents can be aware of their involvement in a game. A similar setting is used by Villatoro *et al.* [2011b] where in every interaction one of the two interacting agents is selected at random to update its action, while the other agent "discards" the obtained payoff. In our work all agents simultaneously initiate exactly one game at every iteration and therefore each agent obtains exactly one payoff per time step. Section 3.5 describes in more detail the model of agent interaction.

A classical game-theoretic assumption is that agents select actions in order to maximize their payoff. For this reason, rational players must choose an action based on their expectation of the play of others. The game for each agent i is represented as a two-dimensional payoff matrix M_i. For example, if an agent i initiates a coordination game with another agent j, i's two-dimensional payoff matrix can be seen in Table 3.2, where $\{a_i^1, a_i^2, \ldots, a_i^k\}$ is i's action set, and similarly, $\{a_j^1, a_j^2, \ldots, a_j^k\}$ is the action set of agent j.

	a_j^1	a_j^2	$\cdots$	a_j^k
a_i^1	1	0	$\cdots$	0
a_i^2	0	1	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
a_i^k	0	0	$\cdots$	1

Table 3.2: Payoff matrix of the row agent i involved in a 2-player k-action pure coordination game.

Each cell of M_i contains the payoff that agent i (the row agent) receives for the joint action. Recall that our agents cannot observe the joint action. According to the

above description of a game, we say that the column player does not receive a payoff for the encounter shown in Table 3.2, since it did not initiate the game. We assume here without loss of generality, that the number of actions k is the same for all players, though in certain coordination problems agents may have different number of available actions. Furthermore, it is generally assumed that the actions of agents are not labeled or ordered in any specific way, for one could design a simple rule saying that all agents should pick the first action. For example, if the WSN designer programs all nodes to communicate on the first wireless channel and it becomes unavailable due to interference, nodes could spend a lot of energy attempting to communicate on that frequency. To avoid this trivial and often unrealistic setting, we require that the behavior of agents should be independent of action names and their order. This assumption is known as action symmetry. Lastly, as shown in Table 3.2 we have selected the value of 1 for each pair of matching actions and 0 otherwise. These values can be chosen arbitrarily, as long as the payoff of successful coordination is the same for all pairs of matching actions and strictly higher than the payoff of any alternative joint action (cf. Section 2.2.1.1). Thus, the payoffs in the game matrix capture the incentive for agents to reach a convention.

As mentioned earlier, the coordination problem is "solved" if all agents select the same action, or in other words, if they all reach convention. In that setting we say that their joint action is a pure-strategy Nash equilibrium if no agent has an incentive to unilaterally deviate from the joint action. Put differently, no player can strictly improve its payoff by acting differently, while the others keep their actions unchanged.

Another related notion we will use in this Chapter is that of Pareto optimality or Pareto efficiency (see Definition 6). To recall, a joint action is Pareto optimal if and only if there is no other combination of actions with a strictly higher payoff for at least one agent and weakly higher payoffs for all others. In a Pareto efficient solution no agent can obtain higher payoff without decreasing the payoff to another. For example, in the 2-player coordination game described above, all pure-strategy Nash equilibria are Pareto optimal. Since agents have no individual preferences, in our games no NE is strictly dominated (or more preferred than another). Therefore we say that agents are indifferent between the Pareto optimal Nash equilibria of the game. In general it holds that all solutions of a given pure coordination problem (or all conventions) are Nash equilibria. However, the reverse is not always true. Certain joint strategies might result in a sub-optimal (or Pareto dominated) Nash equilibrium that is not a convention. For example, agents arranged in the topology, shown in Figure 3.1, can reach an equilibrium that is not a convention, if agents A,

B, C and D select actions a_a^1, a_b^1, a_c^2 and a_d^2, respectively. Agents A and D each get a payoff of 1, since their immediate neighbors select the same action. Agents B and C, on the other hand, are in conflict with each other, but not in conflict with their respective neighbors A and D. In this way no single agent can change its action to improve its payoff and therefore the outcome is a Nash equilibrium, that is not a solution of the given coordination problem. Nevertheless, as we mentioned earlier and will see in Section 3.7, our individual learners can escape such suboptimal outcomes and achieve full convergence in only a few iterations.

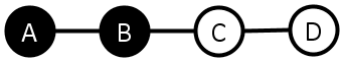


Figure 3.1: A sample topology of 4 agents playing a pure coordination game with two available actions. Agents A and B have selected the same action (black) and agents C and D have selected the other (white). Agents B and C will have a payoff of 0, since they are in a conflict, but none of them can unilaterally change its action and improve its payoff.

The general form of our pure coordination games is that agents need to coordinate on the choice between several Pareto optimal Nash equilibria of the game, without a central authority. The final outcome matters little to any player, as long as all players select the same action. Due to the implied cost of miscoordination, we would like to make agents, arranged in an arbitrary topology, reach a convention in as few interactions as possible. The main challenge in such repeated games is to design an action selection rule that will be adopted by individual agents and will lead them to global successful coordination. Before we present how our approach guides agents into a convention, we describe the details of the interaction model.

3.5 The interaction model

We consider a fixed population of N agents, arranged in a static connected interaction graph (or topology). Vertices represent agents and a direct interaction between agents is allowed only if they are connected by an edge in the graph. Players who share an edge are called neighbors and all neighbors of a given player constitute its neighborhood. We assume fixed topology where agents cannot change their connections to others, i.e., there is no rewiring between vertices. We note here that agents are unaware of the identity (or names) of others. Thus, players cannot condition their action selection on agent names. This assumption, called agent symmetry, stems from a similar requirement we stated earlier — agents are affected only by

what others are playing, i.e., their actions, and not by *whom* they are playing with, or their identity. The behavior of an agent should remain the same if we are to replace the identity of one of its neighbors with another. For example, when a node in a WSN runs out of battery, it can be replaced by another node with a different ID. The neighbors of that node should still learn to coordinate in the same way as with the node that was depleted. The intuition behind this principle is that we cannot anticipate in advance which particular individuals will be involved in a coordination game. Even though information on the identities of other agents might be quite informative in some settings (e.g. a backbone node in a heterogeneous WSN), the role of agent identities is out of the scope of this chapter, as it is often done in the literature of coordination games [Shoham & Tennenholtz, 1997; De Vylder, 2008; Villatoro *et al.*, 2011a].

We study the iterated abstract coordination game in a simulation that proceeds as follows. At every discrete time step (or iteration), agents play the pure coordination game, outlined in Section 3.4. At every time step all agents independently and simultaneously select an action and then each agent meets one of its neighbors at random. All agents use the same action in all individual games within one iteration. Each player may be part of many games, but *initiates* exactly one coordination game per iteration and receives a single payoff from the environment that is computed based on its own action and that of the agent with whom it interacts. Unless stated otherwise, we assume that agents cannot observe each other's action before, during or after they meet. We say that the environment determines the payoff to the initiator (i.e., the row agent), based on the joint action. At the end of the iteration, agents use their action selection mechanism and the obtained payoff to synchronously pick their actions, which will be used in the next iteration. After that, the new iteration begins. Note that all agents have the same set of available actions. This repeated coordination game is played until all agents learn to select the same action, i.e. until agents reach a convention, or until T_{max} time steps have passed. Our performance criterion here is the number of iterations until full convergence. This simulation process is detailed in Algorithm 1.

At first the interaction graph is created (function *initTopology*) based on the number of agents N and the topology type S. The action of each agent is initialized at random (function *selectRandomActions*) after which agents repeatedly meet (lines 4-9) until full convergence or until the maximum number of iterations T_{max} is reached. We set T_{max} high enough to allow for a sufficient number of attempts to reach convention. Each agent selects one of its neighbors in the graph g (function *selectNeighbors*, Algorithm 2). Then, each agent initiates a game with its selected

Algorithm 1 Main simulation process for the pure coordination problem

Input: $N \leftarrow$ number of agents,
$S \leftarrow$ type of topology,
$T_{max} \leftarrow$ maximum iterations
Output: time steps t until full convergence or T_{max}

1: $g \leftarrow initTopology(N, S)$
2: $a \leftarrow selectRandomActions$
3: $t \leftarrow 0$
4: **repeat**
5: $r \leftarrow selectNeighbors(g)$
6: $p \leftarrow getPayoffsFromGames(a, r, g)$
7: $a \leftarrow selectActions(p)$
8: $t \leftarrow t + 1$
9: **until** $conventionReached(a)$ **OR** $t \geq T_{max}$
10: **return** t

neighbor and the environment determines the payoff to that agent, based on the joint action in each coordination game (function *getPayoffsFromGames*). Using that payoff, all agents will simultaneously pick their actions for the next iteration accordingly (function *selectActions*). The process then determines if all agents have selected the same action (function *conventionReached*). If they all belong to a convention, the simulation will stop and return the number of iterations until full convergence (line 10). Otherwise in the next iteration agents will select new neighbor(s) to interact with and the process will repeat (line 4). Note that we stop the simulation process once convention has been reached, since we are interested in the number of time steps until convergence. Agents themselves are not aware of the global behavior of the population, i.e. the fact that a convention has been reached, and therefore they will continue playing the pure coordination game. However, their action selection process must ensure that they do not leave the state of convention. We will see in Section 3.6.2 how our action selection algorithm ensures just that.

At every time step in the pairwise interaction model each agent meets one randomly selected neighbor and plays a pure coordination game. This model of stochastic interactions is typically studied in literature [Kittock, 1993; Barrett & Zollman, 2009] and often occurs in practice. An example of scenarios involving pairwise coordination between players is peer-to-peer communication in computer networks [Lewis, 1969]. In the second half of this chapter we study the multi-player interaction model, where each agent meets all its neighbors in a single coordination game

at every iteration. This one-to-many interaction is rarely studied in literature, but is inherently present in a vast number of real-life settings. Coordination in multi-player interaction can occur between robots deciding on a meeting location or even in the evolution of language. Other scenarios where several agents interact simultaneously to coordinate are in radio communication and in on-line environments. We will elaborate on this model in Section 3.8.

The pseudo-code of the function *selectNeighbors* from Algorithm 1 is given in Algorithm 2 for the pairwise interaction model.

Algorithm 2 function *selectNeighbors* for the pairwise interaction model

Input: game topology g

Output: a vector r with elements r_i indicating the interaction partner of each agent i {assuming pairwise interaction model}

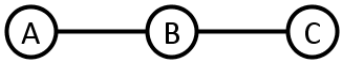
1: **for all** agents i **do**
2: $b \leftarrow getNeighbors(i, g)$
3: $j \leftarrow selectRandomNeighbor(b)$
4: $r_i \leftarrow assignPlayers(i, j)$ {r_i is element i from vector r}
5: **end for**
6: **return** r

A notable difference between our pairwise interaction model and the model studied in literature is that we allow only the initiator of a coordination game to receive a payoff signal, and not the second player. As mentioned in Section 3.4, agents might be unaware of their involvement in a game and therefore obtain no feedback from an interaction that other agents initiate. At every time step t each agent i selects one of its neighbors at random, say j, and plays a k-action coordination game, where k is fixed at the beginning of the simulation. Based on the joint action of i and j, the environment determines the payoff $p_i \in \{0, 1\}$ to agent i (in function *getPayoffsFromGames*, Algorithm 1). Agent j does not receive a payoff from this interaction, since it did not initiate that game. Its payoff will be determined from the game that it initiates with a randomly selected neighbor (possibly with agent i). In other words, the direction of the encounter matters. Formally, the payoff p_i to agent i is computed based on its action a_i and the action a_j of its neighbor j in the following manner:

$$p_i = \begin{cases} 1 & \text{if } a_i = a_j, \\ 0 & \text{if } a_i \neq a_j. \end{cases} \tag{3.1}$$

After all agents have obtained the payoff from their respective game, each agent uses our Win-Stay Lose-probabilistic-Shift approach to independently decide whether to

keep its action unchanged in the next iteration, or to select a different one at random. In Section 3.6 we will elaborate in more detail how this action selection is performed.



(a) Game topology with 3 agents.

	a_b^1	a_b^2
a_a^1	1	0
a_a^2	0	1

(b) Payoff matrix of agent A.

	a_a^1	a_a^2
a_b^1	1	0
a_b^2	0	1

(c) Payoff matrix of agent B against A.

	a_c^1	a_c^2
a_b^1	1	0
a_b^2	0	1

(d) Payoff matrix of agent B against C.

	a_b^1	a_b^2
a_c^1	1	0
a_c^2	0	1

(e) Payoff matrix of agent C.

Figure 3.2: A sample coordination problem with 2 actions and 3 agents and their corresponding payoff matrices in the pairwise interaction model with binary payoffs.

To illustrate the pairwise interaction model, we will now present a very small example of a coordination problem with 3 agents and two actions ($N = 3$, $k = 2$). Consider the topology shown in Figure 3.2a, where agent A has B as neighbor and similarly C has an edge to B. Thus, agent B is connected with both A and C. At every iteration agent A meets B (as its only neighbor) and receives a payoff from that encounter (see Table 3.2b). The same goes for agent C, who always interacts with B. Its payoff matrix is shown in Table 3.2e. Agent B, on the other hand, at each iteration selects one of its two neighbors at random with equal probability and has a one-on-one encounter with that agent, followed by a payoff from the environment. Thus, at each time step, B's payoff table would have either A or C as the column player, as shown in Tables 3.2c and 3.2d respectively. To summarize, at each iteration agent B will participate in three games — two games, initiated by each of its two neighbors and one game initiated by itself. Agents A and C will each participate in either one or two games, depending on the interaction partner that B chooses each time step. As stated in Section 3.5, all agents use the same action in all individual games during one iteration and receive a payoff only from the game that they themselves initiate. Although we assume that agents are not aware of their involvement in the game, if two agents select the same action, the partner of the initiator will become aware of the game that is taking place. As indicated earlier, the second player will not use this information, since reinforcing its action will make it difficult to adapt to the rest of the network. Nevertheless, further analysis is necessary to confirm this claim.

Notice how the game topology restricts agent interaction: the payoff of agent A is independent of the action of C and analogously, C is not influenced by the play of A. Still, agents are not aware of the game topology, but only of their immediate neighbors. In general, in the pairwise interaction model with k actions the payoff table of each player is always 2-dimensional containing k^2 binary values – one for each joint action. An agent gets 1 if its action is the same as its opponent and 0 otherwise. Although we consider here abstract pure coordination games, this interaction model is observed in real-world pairwise interactions, such as those between a wireless transmitter and a receiver. If two wireless devices select different frequencies, they will not be able to communicate. In this setting we say that the transmitter obtains a payoff of 0 for that encounter. We point out that the feedback is determined by the environment since we assume that agents are not allowed to see each other's selected actions at any time, unless stated otherwise. In the above example the intended receiver is unaware of the interaction initiated by the transmitter, since the former is listening on another channel.

3.6 Win-Stay Lose-probabilistic-Shift approach

The main question we are investigating in this chapter is how agents involved in a repeated pure coordination game can reach a mutually beneficial outcome on-line without a central mediator. In the presence of several alternatives and no individual preferences agents need to rely only on repeated local interactions to reach a convention. After each round of interactions agents use their payoff signal to decide whether to select a different action (at random) or keep their action unchanged in the next iteration. Our action selection approach resembles two well-known algorithms in game theory, namely Win-Stay Lose-Shift (WSLS) and Win-Stay Lose-Randomize (WSLR). The WSLS strategy was studied in the repeated Prisoners' Dilemma game [Nowak & Sigmund, 1993], while WSLR was applied in signaling games [Barrett & Zollman, 2009]. Our approach, however, differs from the classic versions of WSLS and WSLR in a probabilistic component that we have introduced (see Sections 3.6 and 3.8.2), and therefore is more general than both. Moreover, as we will see in Section 3.7 none of the above two algorithms can outperform ours in pure coordination games.

Intuitively, if an agent receives the maximum payoff from an interaction ("win"), it means that the other player in that given encounter has selected the same action. In that case it is reasonable that the agent will select the same action in the next time period ("stay"). A low payoff, on the other hand ("lose"), indicates that the

selected neighbor has picked different action and therefore the initiator needs to possibly change its action in the next interaction ("probabilistic shift"). Whereas in the classic WSLS and WSLR agents will always change their action upon "lose", in our version we introduce a probability for "shift". This stochasticity is necessary to ensure that agents with conflicting actions will not constantly alternate their choices and thus will reach a convention faster. Due to this probabilistic component, we name our action selection approach Win-Stay Lose-probabilistic-Shift (WSLpS).

The pseudo-code of our action selection approach (function *selectActions* from Algorithm 1) is presented in Algorithm 3 for the pairwise interaction model. Each agent will keep its last action unchanged in the next iteration if it obtained a payoff of 1 ("win-stay"). If $p_i = 0$, on the other hand, the agent will select a different action at random with probability α ("lose-probabilistic-shift"); with probability $1 - \alpha$ the agent will keep its action unchanged. In other words, α is the *shift probability* upon conflict in the pairwise interaction model. This parameter gives the probabilistic component of our WSLpS in pairwise interactions. A value of α close to 1 drives agents to change their actions more often, while a value close to 0 makes them more "stubborn". Note that if $\alpha = 1$, our approach resembles the classic Win-Stay Lose-Shift. Agents will always change their actions when they obtain a payoff of 0, which results in constant oscillation of actions especially in 2-action games. In Win-Stay Lose-Randomize, on the other hand, upon conflict agents randomly select an action. This behavior may cause the agent to select the same action as in the last time step with probability $1/k$, where k is the number of actions. Therefore, WSLR resembles WSLpS when $\alpha = \frac{k-1}{k}$. If $\alpha = 0$, on the other hand, agents will never select a different action and therefore never reach a convention. For this reason we require that α lies in the open interval $(0, 1)$. The value of α is the same for all agents and it is fixed at the beginning of the simulation. Section 3.7 will study in detail the best values for this parameter.

Formally, agent i will select a different action at random in the next iteration with probability $\Pi_i \in [0, 1]$:

$$\Pi_i = \begin{cases} \alpha & \text{if } p_i = 0 \\ 0 & \text{if } p_i = 1 \end{cases} \tag{3.2}$$

upon receiving a payoff of $p_i \in \{0, 1\}$ in the current iteration, and using parameter $\alpha \in (0, 1)$. With probability $1 - \Pi_i$ the agent will keep its action unchanged in the next iteration, even if its last payoff was 0.

To summarize, at each time step our WSLpS approach allows for 2 possible action selection outcomes depending on the algorithm parameters and the obtained payoff from the latest interaction. Each agent will select an action, based on the

Algorithm 3 function *selectActions* for the pairwise interaction model

Input: payoff $p_i \in \{0, 1\}$ for each agent i from the latest interaction

Output: a vector a indicating the new action of each agent

```
1: for all agents i do
2:     a_i ← getLastAction(i)
3:     rnd ← generateRandomNumber(0, 1)
4:     Π_i ← max(α − p_i, 0)
5:     if rnd < Π_i then
6:         a_i ← selectNewRandomAction(a_i)
7:     end if
8: end for
9: return a
```

following probabilities:

- With probability Π_i the agent will select in the next iteration a different action at random from a uniform distribution. This probability is based on the obtained payoff p_i and the parameter α. Note that in this case the agent does not observe and therefore does not know the action of its neighbor.
- With probability $\hat{\Pi}_i = 1 - \Pi_i$ the agent will select in the next iteration the same action that it selected in the current one. Here too, the agent is unaware of the action that its neighbor selected.

3.6.1 Properties of WSLpS

One important advantage of our WSLpS approach is that it is fully decentralized. The algorithm is run independently by each agent and updates the agent's action only based on local interactions. Agents need not be aware of distant players or their payoffs. Propagating such information in large networks can be costly or unreliable. Communication in wireless sensor networks, for example, is costly in terms of energy consumption and can also be unreliable due to external interferences. Our decentralized approach can be implemented in both synchronous and asynchronous environments. In this chapter we investigate the behavior of agents using synchronous action selection, since it resembles the interactions of wireless nodes that use a slotted communication protocol.

Another positive property is that agents do not need to keep a history of (recent) interactions. Most coordination algorithms proposed in the literature base agent's action selection on the history of past interactions [Young, 1993; Villatoro *et al.*,

2011b]. In our case each agent selects an action based only on the current interaction and therefore requires no memory, apart from the one necessary to store the algorithm itself.

Yet another desirable aspect of our action selection mechanism is that a convention can be reached in a finite number of time steps (see Theorem 2). Moreover, once a convention is reached agents will not change their actions. Put differently, agents will never escape the Pareto optimal Nash equilibria of the coordination game. It is important to note that agents cannot realize that a convention has been reached since they have no global view and do not share any information with others. Nevertheless, agents need not be aware of the convention at all. From the individual's point of view, the agent receives maximum payoff and therefore has no incentive to change its action. After a convention has been reached, all agents will still be running the WSLpS algorithm, but will always select the same action. If a new agent joins the coordination game with a random action, the whole population will converge again to a Pareto optimal Nash equilibrium.

3.6.2 Markov chain analysis

In order to study the expected convergence time and the parameter α of our algorithm, we can represent our system as a Markov chain (MC) (see Section 2.4). In a network of N agents with k actions a state s is an N-tuple that contains the action of each agent at a given time step. The set of all N-tuples (or states) constitutes the state space $S = \{s_1, s_2, \ldots, s_{k^N}\}$. We are interested in the probability π_{s_x,s_y} (or $\pi_{x,y}$ for short) of going from state s_x to s_y in one step. Thus, $\forall s_x, s_y \in S$, $\pi_{x,y}$ constitute the elements of the row-stochastic transition matrix $\mathcal{P}$. The transition matrix is able to tell us the probability of transitioning between any two states in a single time step. To compute the probability of going to a given state in any number of time steps from any starting state, we use the following theorem.

Theorem 1. *Given a probability vector $\vec{u}$ representing the distribution of starting states and a transition matrix $\mathcal{P}$, the final probabilities $\vec{u}^{(t)}$ of arriving at each state after t time steps is:*

$$\vec{u}^{(t)} = \vec{u}\mathcal{P}^t \tag{3.3}$$

Thus, the yth entry in the vector $\vec{u}^{(t)}$ shows the probability that the MC is in state s_y after t time steps, starting from an initial distribution $\vec{u}$. We are interested in arriving at those states s_y which are conventions, i.e. in which all agents select the same action. We use the Equation 3.3 to determine the probability of arriving at those states in a given number of time steps starting from any initial state.

To compute each element in $\mathcal{P}$, we need to determine the transitioning probability between each pair of system states for a single step. To aid the discussion, we will represent each index of $\mathcal{P}$ (i.e. each state in S) as an N-digit number with base k. The first digit shows the action of the first agent, the second digit — the action of the second one and so on. Furthermore, we define the operator $\ominus$ that shows the binary "difference" between two states s_x and s_y, having a functionality similar to the equivalence operator[1] on binary numbers. Thus, $s_x \ominus s_y$ gives a vector $\vec{b}_{x,y}$ with N elements. The ith element of $\vec{b}_{x,y}$ is 0 if the ith digit of s_x differs from the ith digit of s_y and 1 otherwise. In other words, we apply $\ominus$ to two states in order to see which agents changed their actions (resulting in a value of 0) and which agents kept (resulting in a value of 1). We then use the binary vector $\vec{b}_{x,y}$ to compute a vector $\vec{v}_{x,y}$ containing the agents' individual probabilities of shifting or keeping between states s_x and s_y:

$$\vec{v}_{x,y} = (\vec{1} - \vec{b}_{x,y}) \times \frac{\vec{c}_x}{k-1} + \vec{b}_{x,y} \times (\vec{1} - \vec{c}_x) \tag{3.4}$$

where $\vec{1}$ is the vector of size N containing all ones, $\times$ denotes the element-wise vector multiplication and $\vec{c}_x$ contains the probability for each agent to change its action, depending on the actions of its neighbors in state s_x and on the network topology. Each element $\vec{c}_x[i]$ of the vector $\vec{c}_x$ is computed in the following way:

$$\vec{c}_x[i] = \frac{(n_i - n_i|_{a_j=a_i})\alpha}{n_i} \tag{3.5}$$

where n_i is the number of neighbors of agent i, $n_i|_{a_j=a_i}$ is the number of neighbors, whose action a_j is the same as that of agent i and α is the shift probability parameter of WSLpS. In essence, $\vec{c}_x$ contains the probability for each agent i to change its action when in state s_x. Each element is computed as the probability of not agreeing with a randomly selected neighbor, times α. The vector $\vec{v}_{x,y}$ computes the individual probability for each agent to change its action when the system goes from state s_x to state s_y. Finally, the total probability of the system transitioning from state s_x to state s_y is the product of all individual probabilities:

$$\pi_{x,y} = \prod_{i=1}^{N} \vec{v}_{x,y}[i] \qquad \forall x, y \in S \tag{3.6}$$

Using Equation 3.6 we calculate each entry in the transitioning matrix $\mathcal{P}$.

Note that this Markov chain analysis is generic enough and can be used to analyze the behavior of arbitrary number of agents and actions in any topology. If

[1] or NOT XOR

we assume that all initial states are equally likely, we can set all elements of $\vec{u}$ to $1/k^N$, since there are k^N possible states. We can compute the probability of arriving at any absorbing state after t time steps using Equation 3.3. The probability $\Pi^{\text{conv},(t)}$ that a network with N agents and k actions will converge in t time steps is:

$$\Pi^{\text{conv},(t)} = \sum_{i \in \hat{S}} \vec{u}^{(t)}[i] \tag{3.7}$$

where $\hat{S} \subseteq S$ is the set of all goal states in S, and $\vec{u}^{(t)}[i]$ is the ith element of $\vec{u}^{(t)}$. Here we simply sum the probabilities of arriving at all k goal states.

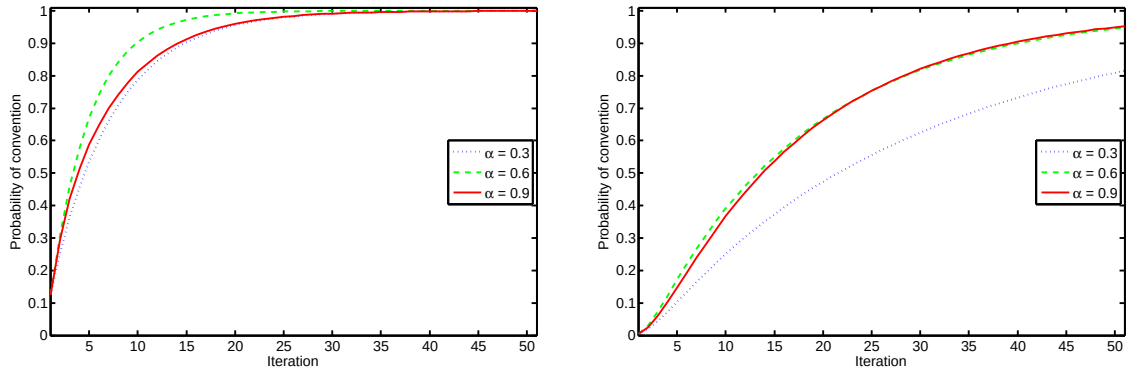
Lastly, we arrive at our main analytical result, expressed in the following theorem.

Theorem 2. *The Markov chain of WSLpS is absorbing and therefore agents using WSLpS have a non-zero probability to reach convention in a finite number of time steps.*

Proof. Let $A = \{1, 2, \ldots, N\}$ be a finite set of N agents and $K = \{1, 2, ..., k\}$ be a finite set of k actions, where a_i^t represents the action of agent i at time step t. Further, let $C_m^t = \{i | i \in A, a_i^t = m\}$ be the set of all agents i whose action a_i^t at time step t is $m \in K$. Thus, the sets C_m^t are a partitioning of the set of agents A. A goal state (or a state of convention) in our Markov chain is a state where $C_m^t = A$ for a given action $m \in K$ and $C_n^t = \emptyset \ \forall n \neq m$. Since in each goal state there are no conflicts between agents, the probability that any agent i will select a different action in the next time step is 0. Therefore all goal states are absorbing (cf. Definition 17). In any other non-goal state there is a conflict between at least two neighboring agents. Since $\alpha > 0$ there is a non-zero probability that the system will transition to a different state and therefore these are called transient states (cf. Definition 19). Thus there are no absorbing states other than the goal states.

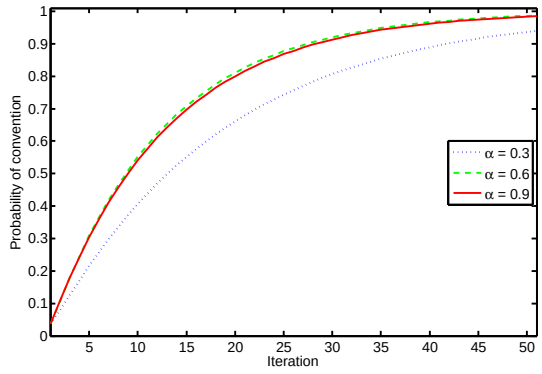
A transient state at time step t implies that there is a conflict between at least two neighboring agents. Therefore, $\exists i, j \in A$ such that i and j are neighbors in the graph and $i \in C_m^t$ and $j \in C_n^t$ for $m, n \in K, m \neq n$. Thus, at time step $t + 1$ there is a non-zero probability that $C_m^{t+1} = \{j\} \cup C_m^t \ \forall j$ neighbors of i. Similarly, at time step $t + 2$ the probability that at all neighbors l of agent j will select j's action is larger than 0. In a connected network with diameter d (the longest shortest path between any two agents), there is a non-zero probability that at time step $t + d$ all neighbors of i, all neighbors of j and so on will select action a_m^{t+d} and therefore $C_m^{t+d} = A$. Thus, an absorbing state can be reached from any transient state (not necessarily in one step) and since the network has a finite diameter d, agents using the WSLpS algorithm are able to converge in finite number of time steps. □

The above theorem says that for $t < \infty$, $\Pi^{\text{conv},(t)} > 0$, i.e. there is a non-zero probability that agents will reach convention in a finite number of time steps. Following from the properties of absorbing Markov chains [Kemeny & Snell, 1969] we have that as $t \to \infty$, $\Pi^{\text{conv},(t)} \to 1$.

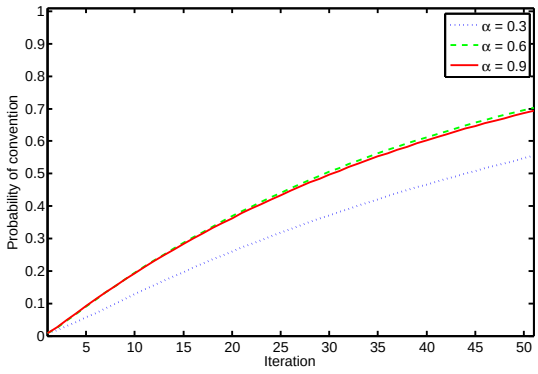


(a) $N = 4$, $k = 2$.

(b) $N = 9$, $k = 2$.



(c) $N = 4$, $k = 3$.



(d) $N = 4$, $k = 5$.

Figure 3.3: Probabilities for N agents with k actions to reach convention within the first $t = 1, \ldots, 50$ iterations in a ring topology for different values of α.

Note that the absorption probability $\Pi^{\text{conv},(t)}$ can be computed for arbitrary number of agents and actions and for any network topology, based on the shift parameter α. We show in Figure 3.3 the probability of convergence for different number of agents and actions in a ring topology. We see that for larger networks higher α increases the probability of convergence. We also see that the higher the number of agents or available actions, the lower the probability of convergence. The latter observation will be confirmed by our simulation studies in Section 3.7.

Besides the probability of convergence, one can also compute the expected con-

vergence time of agents, i.e. the average number of iterations necessary until all agents select the same action. Let $\mathcal{Q}$ be the matrix generated by taking $\mathcal{P}$ and removing all rows and columns that contain a probability of 1. In other words we remove all k rows that contain the probabilities of going from an absorbing state to any state and all k columns with probabilities of reaching an absorbing state from any state. In this way $\mathcal{Q}$ contains the probabilities of transitioning between any pair of transient states in a single step. We then compute the fundamental matrix $\mathcal{N}$ to obtain the expected number of times the process is in each transient state:

$$\mathcal{N} = (\mathcal{I} - \mathcal{Q})^{-1} \tag{3.8}$$

where $\mathcal{I}$ is the identity matrix. Using the fundamental matrix $\mathcal{N}$ we can compute the row vector $\vec{e}$ that gives us for each starting state the expected number of time steps until the chain is absorbed:

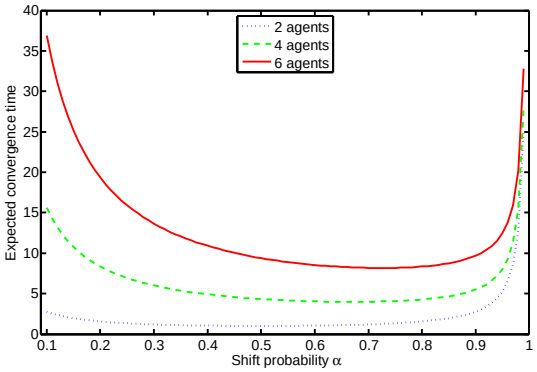
$$\vec{e} = \mathcal{N}\vec{1} \tag{3.9}$$

where $\vec{1}$ is a column vector of all ones. Thus the element $\vec{e}[x]$ shows us the number of time steps before the chain is absorbed when starting in state s_x. Finally, we compute the expected convergence time E based on the initial distribution of states:

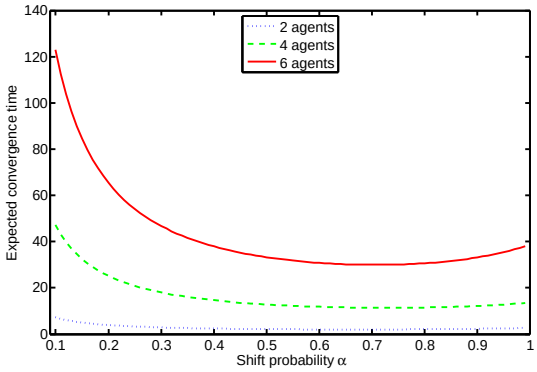
$$E = \vec{u}'\vec{e} \tag{3.10}$$

where $\vec{u}'$ is the transposed vector representing the distribution of starting states. In this way the Markov chain allows us to study the effect of the parameter α on the convergence time of agents. We show in Figure 3.4 the expected time of agents to reach convention in the ring topology for different values of α.

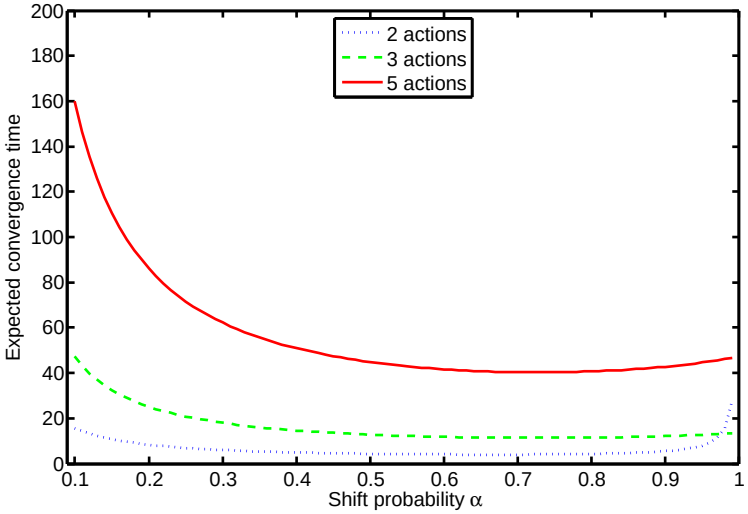
In Figure 3.5 we show the best value for α, i.e. the parameter that achieves the fastest expected convergence time, in ring topologies of different sizes. For games with 2 available actions the best value of our shift probability saturates rather quickly for larger networks. Thus, the MC model suggests that in these networks our parameter α should be no lower than 0.8. We will see in Section 3.7 that this result will be confirmed in our simulation studies in networks of 100, 200 and 500 agents. Games with more than 2 actions, however, seem to converge faster with $\alpha < 0.8$. We will study this setting empirically in Section 3.8.4. Since the state space of the Markov chain grows exponentially in the number of agents, we are not able to illustrate the theoretical properties of our system in large networks especially with more than two actions. Therefore, for large networks we will show results based on empirical data from extensive simulation studies to determine the best α and to estimate the average number of time steps until convergence.



(a) $k = 2$ actions.



(b) $k = 3$ actions.



(c) $N = 4$ agents.

Figure 3.4: Expected convergence time of agents in a ring topology for different values of α.

3.7 Results

We explore the rate of convergence of our Win-Stay Lose-probabilistic-Shift approach in different settings. We follow here the same reasoning as in Villatoro *et al.* [2011b] and set the threshold to 100%. We measure the time steps until *all* agents learn to select the same action, regardless which. Recall that during one time step each agent may participate in several coordination games (with the same action), but receives exactly one payoff signal from the game that it initiates. In the following sections we will study the effect of various system parameters on the convergence time.

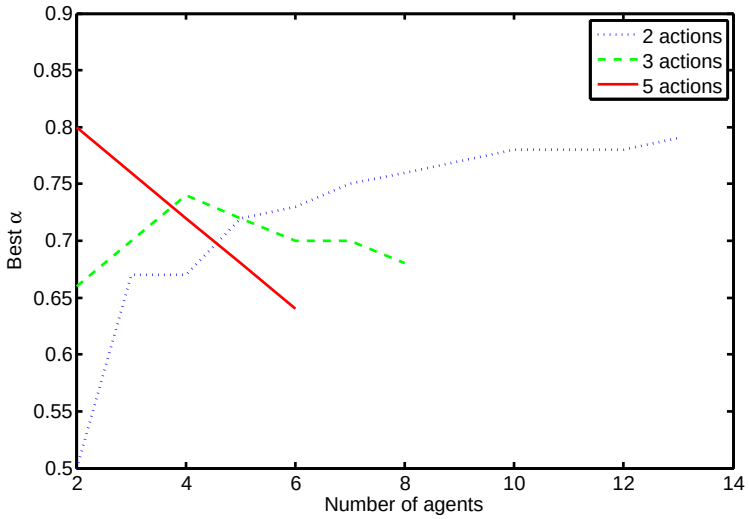
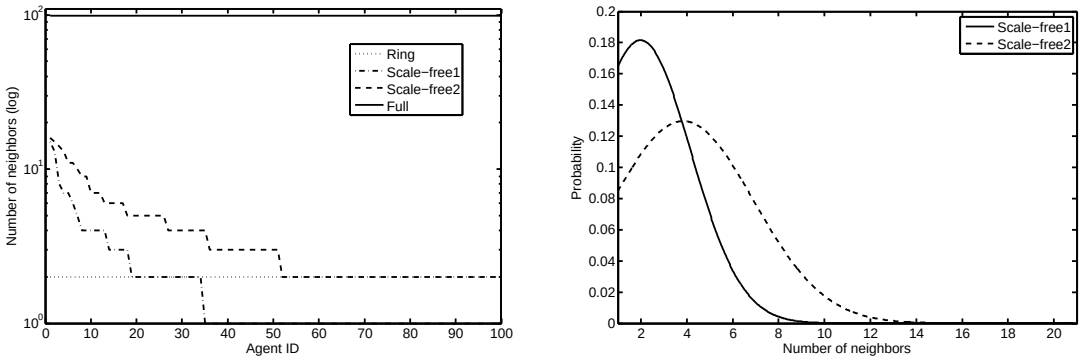


Figure 3.5: The best value of α for different number of agents and actions in the ring topology.

We will study 3 different **topology** types: ring, scale-free and fully connected. The *ring* topology resembles a common scenario in computer networks where each node has exactly 2 neighbors. It poses an interesting challenge for convention emergence, due to the sparse connectivity of agents and the high network diameter (the longest shortest path between any two agents). Next, we explore two different *scale-free* networks — one sparsely connected and one denser, in order to understand how densities affect convergence time. These networks represent also the connectivity between agents in a social network. Our scale-free networks are generated using the preferential attachment algorithm of Barabasi *et al.* [1999], where the number of neighbors per node follows a power-law distribution. The first scale-free network has $N-1$ edges, while the denser network has twice the number of edges, i.e., $2(N-1)$, where N is the number of agents in the network. Note that the number of edges in the sparse scale-free network is the same as in the ring topology, but their distribution is not. Lastly, we measure the convergence process in *fully connected* networks, as it is often done in the literature of coordination games. This extreme case resembles the interconnectivity in some artificial systems where everybody can interact with everybody. The number of neighbors for each agent in the above four topologies is displayed in Figure 3.6, where "Scale-free1" stands for the sparse topology and "Scale-free2" — for the dense. Note that for clarity in Figure 3.6a we show only one particular instance for each scale-free network, since these networks are generated by a stochastic process. In our experiments, however, we generate a different scale-free network in each sample run.



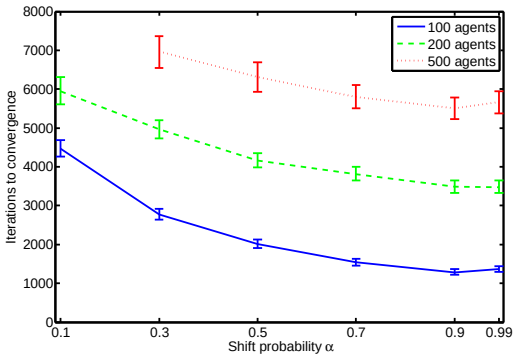
(a) Number of neighbors per agent.

(b) Probability of neighbors per agent.

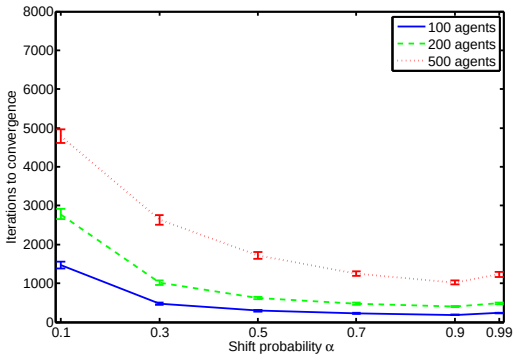
Figure 3.6: Distribution of neighbors per agent in different topologies.

In the above topologies we vary the **number of agents** and **number of actions** available to those agents in order to investigate the scalability of our approach. We explore networks of 100, 200 and 500 players where agents can have 2, 3 or 5 actions. The convergence times for each parameter configuration are averaged over 1000 runs of Algorithm 1 in MATLAB. 1000 runs with a given parameter configuration we call a sample. Each run ends either when a convention emerges, or when a maximum of 10000 iterations is reached ($T_{max} = 10000$), in which case the run is not counted towards the mean of the sample. The sample is considered only if at least 60% of the runs have finished within T_{max} time steps. Each sample approximates a Gaussian distribution, whose standard deviation is rather large, due to the probabilistic component of our approach. We measured empirically that samples with larger means have also larger standard deviation. This observation is not surprising, since our data is time dependent and the approach is stochastic. The more iterations (or time) it takes for convention emergence, the lower the predictability of the data. Conversely, samples with lower means (i.e., shorter convergence duration) are closely centered around the reported mean. For clarity of exposition we chose to report the statistical significance of the data mean, instead of its spread, which is rather large and obscures the plots. In all reported results the error bars indicate the 95% confidence interval of the reported mean. The action for each agent is initialized uniformly random from the available actions. As stated above, the performance measure of the system is the number of iterations until the actions of all agents converge. We study here the parameter $\alpha \in (0, 1)$, or the shift probability upon conflict.

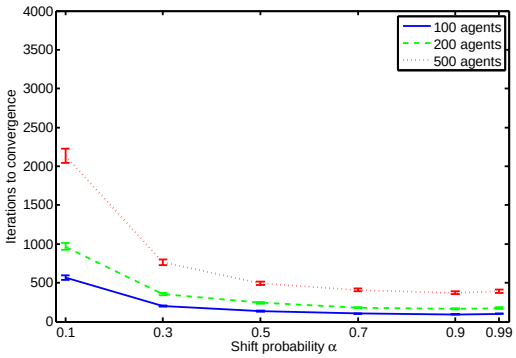
Figure 3.7 shows the convergence duration of agents arranged in different topologies under the pairwise interaction model with binary payoffs where agents have 2



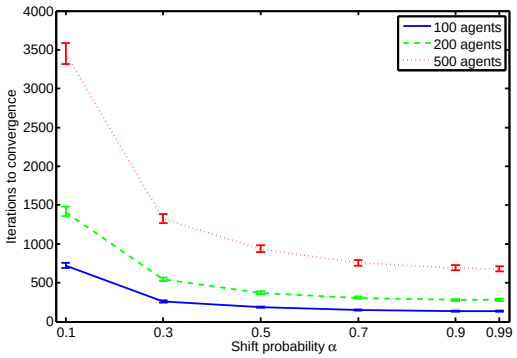
(a) Ring topology.



(b) sparse Scale-free topology.



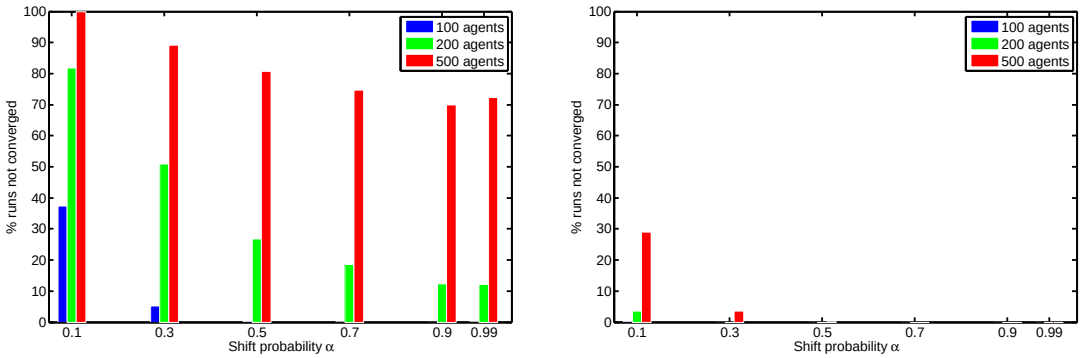
(c) dense Scale-free topology.



(d) Fully connected topology.

Figure 3.7: Convergence time for different topologies under pairwise interactions with 2 actions per agent. Error bars show the 95% confidence interval of the mean.

available actions to choose from. This is also the classic experimental setting reported in literature (see Section 3.2). Figure 3.7a has a missing value for 500 agents, since all runs with $\alpha = 0.1$ required more than $T_{max} = 10000$ iterations to converge and therefore were cut off before a convention has emerged. Figures 3.8a and 3.8b show the percentage of runs that did not converge within this limit for the ring and sparse scale-free topologies respectively. All runs converged in the other two topologies. We notice that the performance of the approach is relatively sensitive to the shift probability. Nevertheless it is consistent in all topologies and network sizes. We can conclude that in all four topologies a value of $\alpha = 0.9$ gives the lowest convergence time of all values tested, regardless of the network size. Recall that a large α increases the probability that an agent will change its actions when in conflict with another. Thus, we observe that high shift probability leads to faster convention emergence. However, when $\alpha \to 1$, the convergence time slightly increases. We



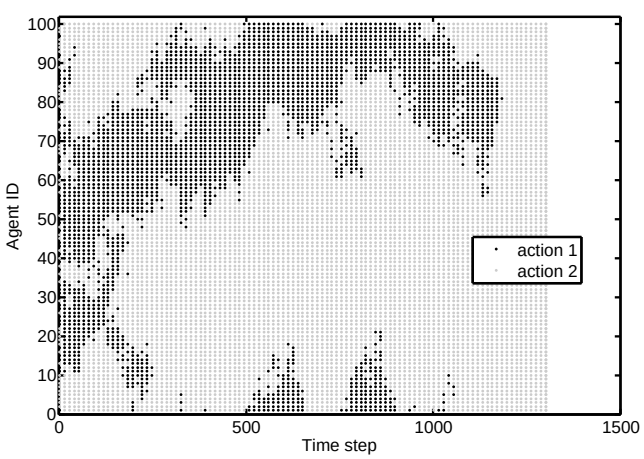
(a) Ring topology.

(b) sparse Scale-free topology.

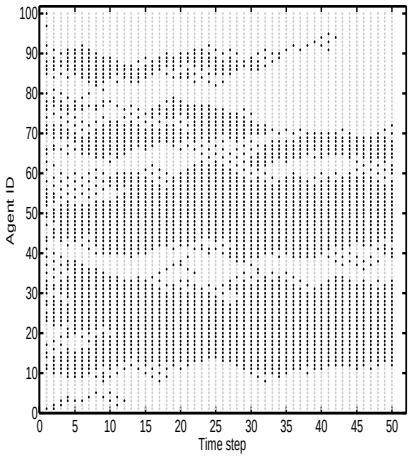
Figure 3.8: Percentage of runs that did not converge within T_{max} iterations from Figure 3.7.

observe also the scalability of our approach with respect to the number of agents — convergence time increases linearly with population size. Another intriguing observation is that densely connected networks (Figures 3.7c and 3.7d) converge on average faster than networks with sparse connectivity (Figures 3.7a and 3.7b). The reason for this behavior is that agents in denser networks have pairwise interactions with a large number of different agents and thus conventions spread faster than in sparsely connected networks. Analogously, agents in sparse networks interact with only a limited set of other agents and therefore reinforce their neighbors' action, which may differ from the action of other groups.

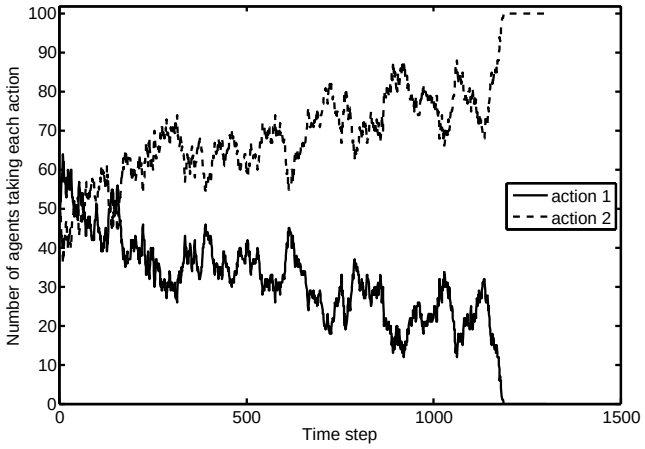
To have a better understanding of the convergence process at a finer scale, we present in Figure 3.9 the behavior of agents in a typical simulation run of Algorithm 1. Figure 3.9 displays the action of agents during learning in the ring topology with 100 agents and 2 available actions. According to Figure 3.7a we set the shift probability α to 0.9 in the pairwise interaction model. The latter figure indicates that the mean convergence time with this parameter configuration is a little more than 1000 iterations, which is what we observe in our simulation run in Figure 3.9. Each value on the vertical axis of Figure 3.9a represents a different agent. In other words, *Agent ID* is not a continuous variable, but simply shows the individual agents. Each dot in the latter figure stands for the action of each agent at the corresponding time step. A black dot represents action 1 and a gray dot — action 2. As mentioned in Section 3.5, these actions are initialized at random. In Figure 3.9b we show a detailed view of the first 50 time steps. It can be observed that at time step 1 agents are randomly assigned actions 1 and 2. Since agents are arranged in a ring topology, consecutive agent IDs in Figure 3.9a are also neighbors in the network.



(a) The action of each agent at each tenth time step.



(b) A detail of the first 50 time steps.



(c) The number of agents taking each action at each time step.

Figure 3.9: Results from a single (typical) simulation run of Algorithm 1 in the ring topology with 100 agents and 2 available actions. Pairwise interaction model with $\alpha = 0.9$.

An interesting observation is that shortly after the start, large contiguous clusters of neighboring agents tend to select the same action. Only agents on the border between different sections experience conflicts and therefore change actions. This behavior demonstrates the essence of our WSLpS approach — agents with "successful" actions will keep selecting the same action, while those who experience conflicts have a non-zero probability to change.

While Figure 3.9a shows the action of each agent at every time step, Figure 3.9c reports the number of agents taking each action at every time step during the same

simulation run. One can observe that at time step 1 equal number of agents select each action, as stated earlier. We notice that although in the first 50 time steps action 1 is dominant, later on the majority of the agents learn to select action 2. We see large fluctuations in the number of agents who select the same action. The reason for these fluctuations is the probabilistic component of our WSLpS. Large shift probability α leads to larger changes in the selected actions from one time step to another, while small α results in a smoother behavior, but requires more time steps to convergence. We see that around time step 1200 all agents learn to select action 2 and thus a convention has emerged. We ran the simulation for 100 steps more in order to illustrate that agents continue to select the same action and therefore do not escape the convention.

Figure 3.10 compares the convergence duration of agents in the four topologies for different values of our parameter α. We have fixed here the network size to 100 agents with 2 actions per agent. Direct comparison with algorithms, reported in literature is difficult, since our interaction model and experimental settings are not the same as in the related work. Therefore, more detailed studies need to be performed in the future. In Section 3.6 we pointed out that our WSLpS approach resembles both Win-Stay Lose-Shift (WSLS) when $\alpha = 1$ and Win-Stay Lose-Randomize (WSLR) when $\alpha = \frac{k-1}{k}$. In our experiments the number of available actions k is 2. As we see from Figure 3.7 $\alpha = 0.5$ results in slower convention emergence than what we obtain with $\alpha = 0.9$. Moreover, for $\alpha = 1$ the algorithm is not guaranteed to converge in the ring topology with 2 actions. If each agent is in conflict with both its neighbors, all agent will change change their action and thus remain in conflict. Therefore none of the two algorithms can outperform WSLpS in pairwise interactions.

It is evident from our experiments that denser networks converge on average faster than sparser ones. This observation can be explained by the following. Agents who interact with more neighbors have a larger sample of the most common action locally in the network, but have also higher probability for conflicts. In contrast, an agent with only two neighbors, for example, receives feedback based on the actions of agents in a very small portion of the network and therefore its locally most common action varies faster than in denser networks. Players in denser networks, on the other hand, behave in response to the actions of larger groups of neighboring agents and therefore have better chance to arrive at a mutually beneficial outcome. Moreover, denser networks have shorter average path length (average shortest distance between all pairs of nodes), which helps conventions spread faster through the network. The longer the average distance between agents, the more interactions it takes to propagate successful actions. However, we see that the fully connected network

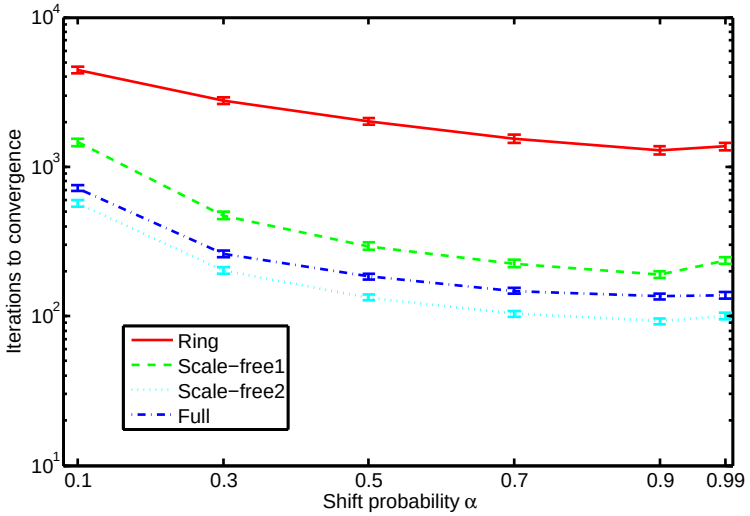


Figure 3.10: Convergence time of the pairwise interaction model under different topologies with 100 agents and 2 actions per agent. Error bars show the 95% confidence interval of the mean.

converges on average slightly slower than the dense scale-free topology, since in the former topology agents have more neighbors and thus experience more conflicts. A detailed study needs to be conducted in order to determine the precise relationship between average path length and convergence time.

3.8 Multi-player interactions

In the first half of this chapter we studied pairwise interactions where each agent selects one neighbor at random and plays a pure coordination game. In this section we will extend this model to multi-player interactions — each agent interacts with all its neighbors. This type of one-to-many encounters is rarely studied in literature, but often occurs in practice. For example, when a wireless node broadcasts a message, all nodes in range are affected. We would like to investigate in this section how multi-player interactions can affect the convergence time of agents.

3.8.1 The interaction model

In the multi-player interaction model each agent plays a pure coordination game with all its neighbors. Thus, at time step t each agent i is engaged in an n_i-player game, where n_i is the size of i's neighborhood. The pseudo-code for selecting neighbors is shown in Algorithm 4.

Algorithm 4 function *selectNeighbors* for the multi-player interaction model

Input: game topology g

Output: a vector r indicating the interaction partner or partners of each agent

1: **for all** agents i **do**
2: $b \leftarrow getNeighbors(i, g)$
3: $r_i \leftarrow assignPlayers(i, b)$ {r_i is element i from vector r}
4: **end for**
5: **return** r

Similarly to the pairwise model, only the initiator i of the game receives payoff $p_i \in [0, 1]$ from the environment, determined by the joint action of all participating agents. Nevertheless, since all agents initiate a game in each iteration, each agent obtains exactly one payoff signal and updates its strategy accordingly. The payoff matrices of agents are n_i-dimensional, where n_i may be different for each agent i. Here too we require that only the initiator of the game obtains payoff while the other agents may be unaware of the game at all. This requirement stems from the limitations in the WSN domain. When a node broadcasts a message on one channel, neighbors listening on another channel cannot know that the game is taking place. Consecutively, the initiator cannot directly know the actions of its neighbors. For the same reasons nodes cannot exchange information that will help them solve the coordination problem.

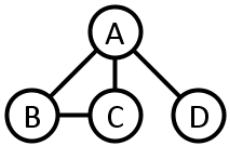
In the pairwise model we assumed that the payoff agents receive from the environment is binary (or less informative). The outcome for a given player is 1 if the agent it meets selects the same action, and 0 otherwise. This setting, however, is much harder in the multi-player model, but it is also rather unrealistic. It would mean that the agent's payoff will be 0 if only a single neighbor selects a different action, regardless of how many other neighbors have the same. In addition, in some settings these problems can be reduced to multiple sequential pairwise interactions with binary rewards. For these reasons, we will not investigate multi-player encounters with binary payoffs in this chapter. Instead, we will consider a multi-valued (or more informative) payoff signal when using the multi-player interaction model (see example below). A similar model is adopted by Bramoullé [2007], where agents play a 2-player game with each of their neighbors and obtain the sum of these bilateral games' payoffs. In a broad range of applications, the environmental feedback contains information on the number of neighbors who have selected the same action (but not what others have selected). Formally, in the multi-player interaction model the payoff p_i to agent i is computed based on its action a_i and the action a_j of each

neighbor j in i's neighborhood with size n_i in the following manner:

$$p_i = \frac{n_i|_{a_j=a_i}}{n_i} \tag{3.11}$$

Here $n_i|_{a_j=a_i}$ is the number of neighbors of i, whose action a_j is the same as that of agent i.

Consider the sensor network coordination problem in Example 9. A given node A broadcasts a signal on channel $c^1 \in \mathcal{C}$ where $k = |\mathcal{C}|$. All nodes within range of A that listen on channel c^1 receive and acknowledge its message, as required by the communication protocol. The rest of A's neighbors that listen on a different channel $c^j \in \mathcal{C}$, $j \neq 1$, are unaware of the transmissions on c^1 and thus send no acknowledgment to A. Given that node A knows the total number of its neighbors, it is able to determine, based on the feedback it receives what percentage of its neighborhood selected the same action. Here A is the initiator of the coordination game and therefore only it receives feedback. The payoffs of A's neighbors are computed in a similar fashion based on the coordination games that they themselves initiate. Note that A has no information on the actions of its neighbors who did not select c^1, if the number of alternatives k is larger than two. For $k = 2$, we have $\mathcal{C} = \{c^1, c^2\}$, thus node A can simply deduce the action of its "non-conforming" neighbors. Still, deducing those actions does not simplify the problem — the agents still need to find a way to "agree" on one of the actions.



(a) Game topology with 4 agents.

	a_b^1,a_c^1,a_d^1	a_b^1,a_c^1,a_d^2	a_b^1,a_c^2,a_d^1	a_b^2,a_c^1,a_d^1	a_b^1,a_c^2,a_d^2	a_b^2,a_c^1,a_d^2	a_b^2,a_c^2,a_d^1	a_b^2,a_c^2,a_d^2
a_a^1	1	$2/3$	$2/3$	$2/3$	$1/3$	$1/3$	$1/3$	0
a_a^2	0	$1/3$	$1/3$	$1/3$	$2/3$	$2/3$	$2/3$	1

(b) Payoff matrix of agent A.

Figure 3.11: A sample coordination problem with 2 actions and 4 agents together with the payoff matrix of agent A in the multi-player interaction model with multi-valued payoffs.

For an illustration of the multi-player interaction model with informative feedback signal, consider the following pure coordination problem. $N = 4$ agents are

arranged in the topology shown in Figure 3.11a where each has $k = 2$ available actions to choose from. Agent A is involved in a 4-player game, B and C each plays a 3-player game, while agent D is engaged in a 2-player game. Table 3.11b shows the 4-dimensional payoff matrix of agent A. The matrices of the other agents can be obtained in a similar way, but we omit them for brevity. In contrast to the binary model, here the payoff of the agent is in fact the ratio of its neighbors who select the same action as itself. This feedback is indeed more informative than the binary 0-1 payoff, since it provides a measure of how far agents are from an equilibrium. Note that agents with only one neighbor, such as agent D in Figure 3.11b, play a 2-player game and therefore obtain only binary feedback. In other words, an n-player game with multi-valued feedback reduces to a 2-player game with binary feedback if the agent has only one neighbor.

3.8.2 WSLpS for multi-player interactions

In the multi-player interaction model the payoff agents receive from the environment contains information on the fraction of neighbors that have selected the same action as the initiator of the interaction. Agents use this information in their action selection algorithm to determine whether to select a new action in the next iteration or to keep their action unchanged. The pseudo-code of this action selection algorithm (function *selectActions* from Algorithm 1) is presented in Algorithm 5 for the multi-player interaction model.

Algorithm 5 function *selectActions* for the multi-player interaction model

Input: payoff $p_i \in [0, 1]$ for each agent i from the latest interaction
Output: a vector a indicating the new action of each agent

for all agents i **do**
    $a_i \leftarrow getLastAction(i)$
    $rnd \leftarrow generateRandomNumber(0, 1)$
    $\Pi_i \leftarrow \max(1 - p_i^{I_n} - \beta, 0)$
    if $rnd < \Pi_i$ **then**
        $a_i \leftarrow selectNewRandomAction(a_i)$
    end if
end for
return a

Similarly to the pairwise model, agents with high payoff will keep their last action unchanged in the next iteration. Otherwise, if $p_i < 1 - \beta$, the agent will select a

different action at random in the next iteration with probability $1 - p_i - \beta$, and with probability $p_i + \beta$ it will keep its action unchanged. In other words, in the next iteration each agent i will select the same action as in the last iteration with probability equal to its payoff p_i from that interaction, plus a constant β. Thus, the higher the payoff of the agent, the more likely it will keep its action unchanged for the next iteration. For $p_i = 1$ for example, the probability of keeping the same action is 1, therefore the agent will never change a "successful" action. For $p_i = 0$ on the other hand, the agent still has a probability of $0 + \beta = \beta$ to select the same action in the next iteration. Therefore, we name β the *keep probability* for "unsuccessful" actions. The role of our parameter β is similar to the parameter α from Section 3.6 — it ensures that agents will not constantly alternate their actions when in conflict with all their neighbors. In the multi-player model β gives the probabilistic component of WSLpS. A too small β will have only little effect on the performance of agents, while a large value will slow down convergence.

Formally, agent i will change its action in the next iteration with probability $\Pi_i \in [0, 1]$ when it obtained a payoff of $p_i \in [0, 1]$ in the current iteration. This probability for the multi-player interaction model is:

$$\Pi_i = \begin{cases} 1 - p_i - \beta & \text{if } p_i < 1 - \beta \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

where $\beta \in (0, 1)$ is the parameter of our approach. With probability $1 - \Pi_i$ the agent will keep its action unchanged in the next iteration.

3.8.3 Local observation

In certain scenarios involving multi-player interactions it is reasonable to assume that agents are able to occasionally observe the actions of their immediate neighbors. What we will investigate in Section 3.8.4 is whether and how such information can help agents reach convention faster. In Section 3.1 we presented a real-world example of a pure coordination game. We will show here how local observation can be incorporated in this scenario.

Example 10 (WSN pure coordination with observation)**.** *Consider an arbitrary network of nodes, which typically communicate on different frequencies (or channels) in order to avoid radio interference. Every so often, all nodes need to switch to the same channel, regardless which, in order to exchange control messages, e.g. to synchronize their clocks. Each node will only hear neighbors on the same channel and will be unaware of the channels that its other neighbors have selected. However, certain models of wireless sensor nodes possess the ability to perform a "channel*

sweep". The latter consists in listening on each channel for a brief amount of time to determine whether any neighboring nodes have selected that channel for communication. After performing a sweep (i.e., local observation), the node has information on the action that each of its neighbors have selected. In the absence of central control, how can nodes use this information to converge faster to the same broadcast frequency?

The above example illustrates how local observation can be incorporated in a repeated coordination game in order for agents to gain information on the actions of others. However, in some cases gaining such information comes at a certain cost. A channel sweep for example requires energy, which is a valuable resource in wireless networks. We acknowledge here that the cost of observation will inevitably influence agent behavior and therefore the duration until convergence. To keep our analysis simple, in this chapter we will not study this cost. Instead, we will simply assume that local observation incurs a cost larger than 0, such that agent should not always observe. We are interested more in *how* agents can use this information, rather than *when* they should observe. In contrast, De Hauwere [2011] studies when agents should observe information from other agents.

Intuitively, if an agent knows what the majority of its neighbors are playing, it will select the most-played action in the next iteration. In this way, the agent increases its chance to obtain higher payoff. However, observing and selecting the majority action at every time step will not guarantee success. Actions are initialized at random and each agent has different neighbors. Thus, each agent may observe different majority action and therefore never reach convention. For this reason, and due to the implied cost, agents need to use local observation carefully.

To study the role of observation, in our WSLpS approach we introduce a *local observation* parameter $\gamma \in [0, 1]$. In the multi-player model, this parameter indicates the probability with which an agent will observe the action of all its immediate neighbors. After observation, in the next iteration that agent will select the majority action of the current iteration within its neighborhood. If there is a tie for the most common action, the agent selects one of the majority actions at random. With probability $1 - \gamma$ the agent will select its action as outlined in Algorithm 5 respectively. In this thesis we assume homogeneous topologies and thus all agents have equal observation probability. In heterogeneous settings one may consider the effect of local observation when only certain nodes (e.g. hubs) have this ability. However, sparse observation may lead to more conflicts, since different parts of the network (i.e. where observing hubs are) may converge to different actions. In Section 3.8.4 we study the local observation parameter γ in homogeneous networks in more detail.

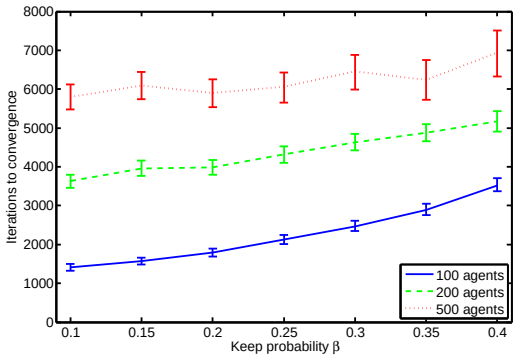
To summarize, at each time step our WSLpS approach allows for 3 possible action selection outcomes depending on the algorithm parameters and in some cases the obtained payoff from the latest interaction. In the multi-player interaction model each agent will select an action, based on the following probabilities:

- With probability $\Pi_i^{\text{obsr}} = \gamma$ the agent will observe the action of the neighbors with whom it is involved in a game. In the next iteration, the agent will select the most played action among its interaction partners. One can notice that the probability Π_i^{obsr} is independent of the payoff p_i the agent receives in the current interaction.
- With probability $\Pi_i^{\text{chng}} = \max\{(1-\gamma)(1-p_i-\beta), 0\}$ the agent will select in the next iteration a different action at random from a uniform distribution. Note that in this case the agent does not observe and therefore does not know the action of its neighbors.
- With probability $\Pi_i^{\text{keep}} = \max\{(1-\gamma)(p_i+\beta), 0\}$ the agent will select in the next iteration the same action that it selected in the current one. Here too, the agent is unaware of the action that its neighbors selected.

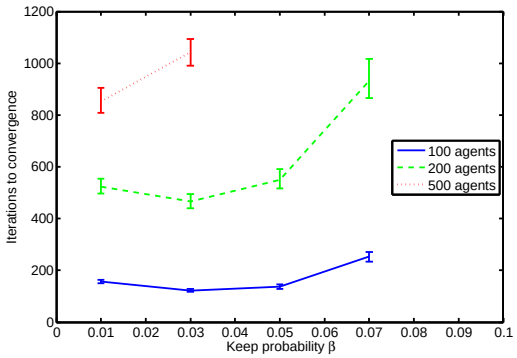
3.8.4 Results from the multi-player interaction model

In the multi-player interaction model the probabilistic shift of our WSLpS approach is only partially determined by the parameter β. The payoff from each interaction also influences the probability of changing the action. In Figure 3.12 we study how β affects the convergence time of agents in different topologies.

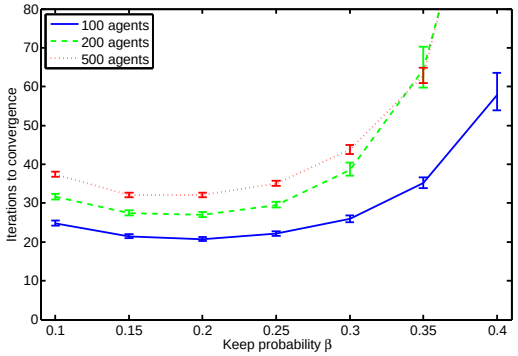
Similarly to the results from the pairwise model, convergence time here increases linearly with population size and therefore our approach scales well in the number of agents. Here too dense connectivity results in faster convergence (Figures 3.12c and 3.12d), while sparse networks learn slower (Figures 3.12a and 3.12b). In Figure 3.13 we display the percentage of runs that did not converge within $T_{max} = 10000$ iterations for the latter two topologies. The peculiarities in Figures 3.12b and 3.12c are a result of the irregular structure of the two scale-free topologies, where the number of neighbors is different for each agent. We see that a value between 0.1 and 0.25 is acceptable in all topologies, except the sparse scale-free one, where a value of $\beta > 0.01$ leads to slower convergence for large networks. Therefore we will use $\beta = 0.01$ for all topologies when studying the observation parameter γ. Although this choice will ultimately affect the convergence times in all topologies, our aim here is to study the influence of each parameter separately, rather than look for the optimal configuration.



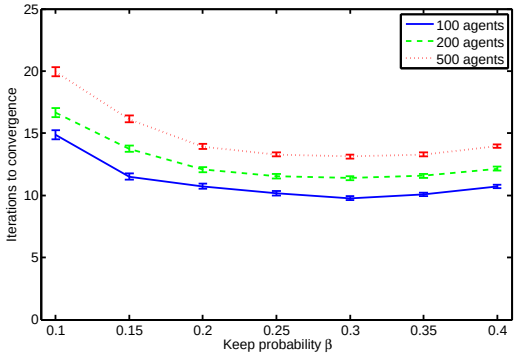
(a) Ring topology.



(b) Sparse scale-free topology.



(c) Dense scale-free topology.



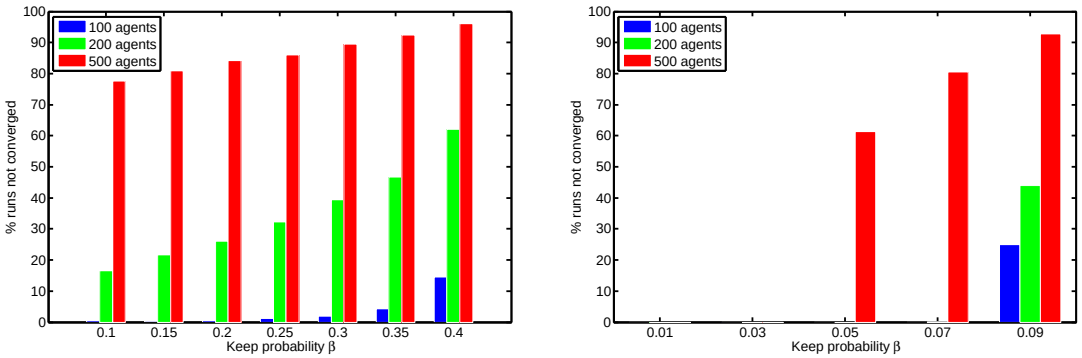
(d) Fully connected topology.

Figure 3.12: Parameter study in different topologies under multi-player interactions with 2 actions per agent. Error bars show the 95% confidence interval of the mean. Observation probability $\gamma = 0$.

Figure 3.14 shows the convergence duration of agents arranged in different topologies under the multi-player interaction model with multi-valued feedback where agents have 2 available actions to choose from. We study here the observation parameter γ, or how additional information on the actions of neighbors can help agents converge faster.

One can notice that there is no best value of γ for all topologies. Recall that observation makes agents select the majority action in their neighborhood. In the ring topology each agent has only 2 neighbors and therefore observation has only little effect — the majority action is simply one of the two neighbors' actions. Similarly, in the first scale-free network due to the sparse connectivity, observation lets agents reinforce the action of only small groups, sparsely connected with others, and therefore this behavior results in more conflicts as γ increases.

In denser networks, in contrast, observation is beneficial, as can be determined



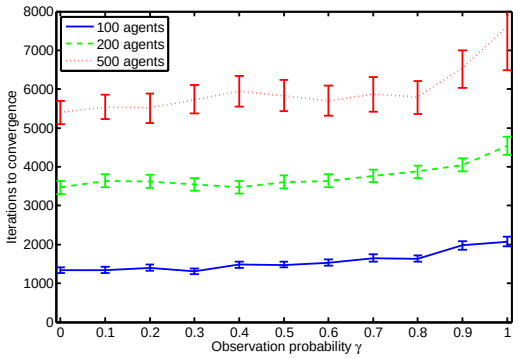
(a) Ring topology.

(b) Sparse scale-free topology.

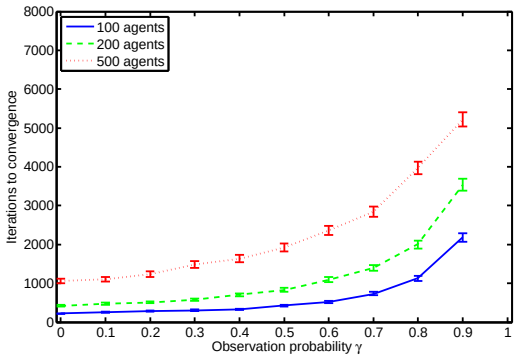
Figure 3.13: Percentage of runs that did not converge within T_{max} iterations from Figure 3.12.

from Figures 3.14c and 3.14d. In the latter two topologies agents have more neighbors and observation quickly spreads the most common action through the network. In the limit, when $\gamma = 1$, agents in the fully connected network need only two iterations to converge. In the first iteration actions are initialized randomly, each agent i meets all others and receives payoff p_i based on the joint action. In the second iteration each agent observes the actions of all others with probability $\gamma = 1$ and selects the most common one. Since all agents observe the same majority action, they will all select the same action and thus reach convention in the second iteration. However, in rare cases, if exactly half of the population is initialized with one action and the other half — with another, a convention will never emerge, since agents will constantly alternate between the two choices. This phenomenon can be observed in Figure 3.15, which displays the percentage of runs that did not converge within $T_{max} = 10000$ iterations for all four topologies. The same phenomenon is particularly visible in the ring topology (see Figure 3.15a), where agents may find themselves constantly switching between two majority actions. Also, Figure 3.14b has missing values for $\gamma = 1$, since agents cannot converge when they always observe their neighbors. Figure 3.15b confirms this result, showing that all runs in this setting exceeded T_{max} iterations. Therefore, a value of $\gamma = 1$ is generally not advisable in these topologies, i.e. agents should not select the majority action at every time step.

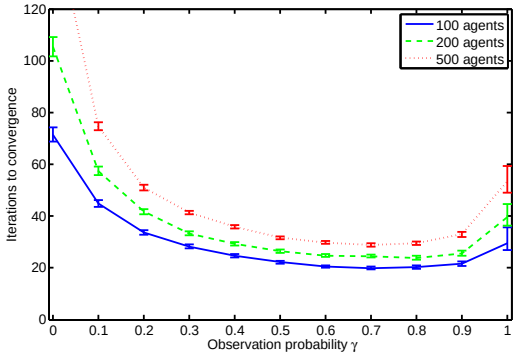
Lastly, we study the scalability of WSLpS with respect to the number of actions. Figure 3.16 displays the number of iterations necessary for convention emergence when agents have 2, 3, or 5 available actions in the fully connected topology. Note that the y-axis is logarithmic and thus the convergence time with 5 available actions



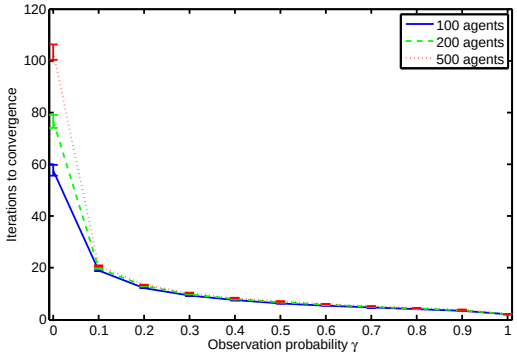
(a) Ring topology.



(b) Sparse scale-free topology.



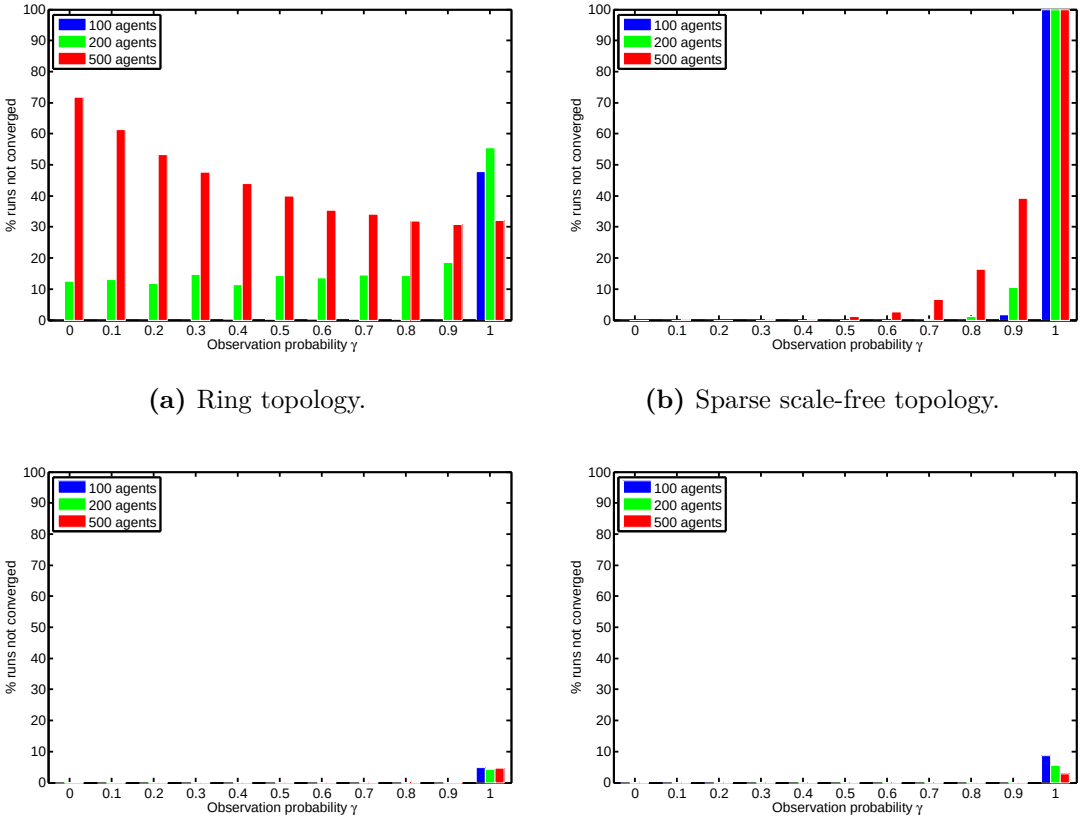
(c) Dense scale-free topology.



(d) Fully connected topology.

Figure 3.14: Convergence time for different topologies under multi-player interactions with 2 actions per agent. Error bars show the 95% confidence interval of the mean. Keep probability $\beta = 0.01$.

increases exponentially in the number of agents. In Figure 3.17 we study again the observation probability, but for 2, 3, and 5 actions. Missing values mean that all runs in the sample with the corresponding parameter configuration were cut off, because they exceeded the limit of 10000 iterations. Similarly, the more runs did not finish, the larger the confidence interval of the mean. One can notice that when agents have more than 2 available actions and use low observation probability, the network almost never converges within that limit. Agents in the ring topology, for instance, need more than 10000 iterations to reach a convention in a game with only 3 actions (cf. Figure 3.17a). However, an intriguing result is that agents with 3 available actions in the sparse scale-free network are able to reach convention in 4000 iterations without observation (cf. Figure 3.17b). Again, the network density plays an important role. That scale-free network is neither too sparse, such that conventions spread too slow, nor too dense, such that agents often experience conflicts.



(a) Ring topology.

(b) Sparse scale-free topology.

(c) Dense scale-free topology.

(d) Fully connected topology.

Figure 3.15: Percentage of runs that did not converge within T_{max} iterations from Figure 3.14.

The latter case is indeed what is preventing the denser networks to reach convention using low observation probability. In all networks except the ring, larger observation probability improves convergence time for games with more than 2 actions. We see the effect of local observation also in Figure 3.18, which displays the percentage of runs that did not converge within 10000 iterations. Again, higher γ enables agents to reach convergence with more than 2 actions. However, $\gamma = 1$ occasionally leads to cycles where agents constantly switch between majority actions and therefore never converge.

We show in Figure 3.19 results from a single (typical) simulation run of Algorithm 1 in the dense scale-free topology with 100 agents and 3 available actions. We use the multi-player interaction model with keep probability $\beta = 0.01$ and local observation probability $\gamma = 0.8$. According to Figure 3.17c the mean convergence time with this parameter configuration is 21 iterations, which is the case for the simulation run presented in Figure 3.19. We ran the simulation for 10 steps more to

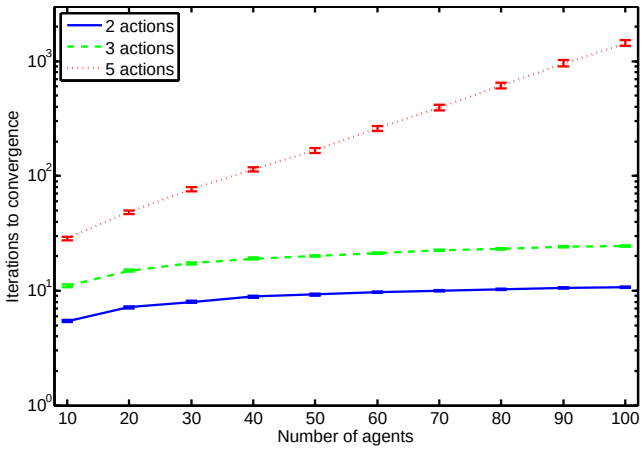
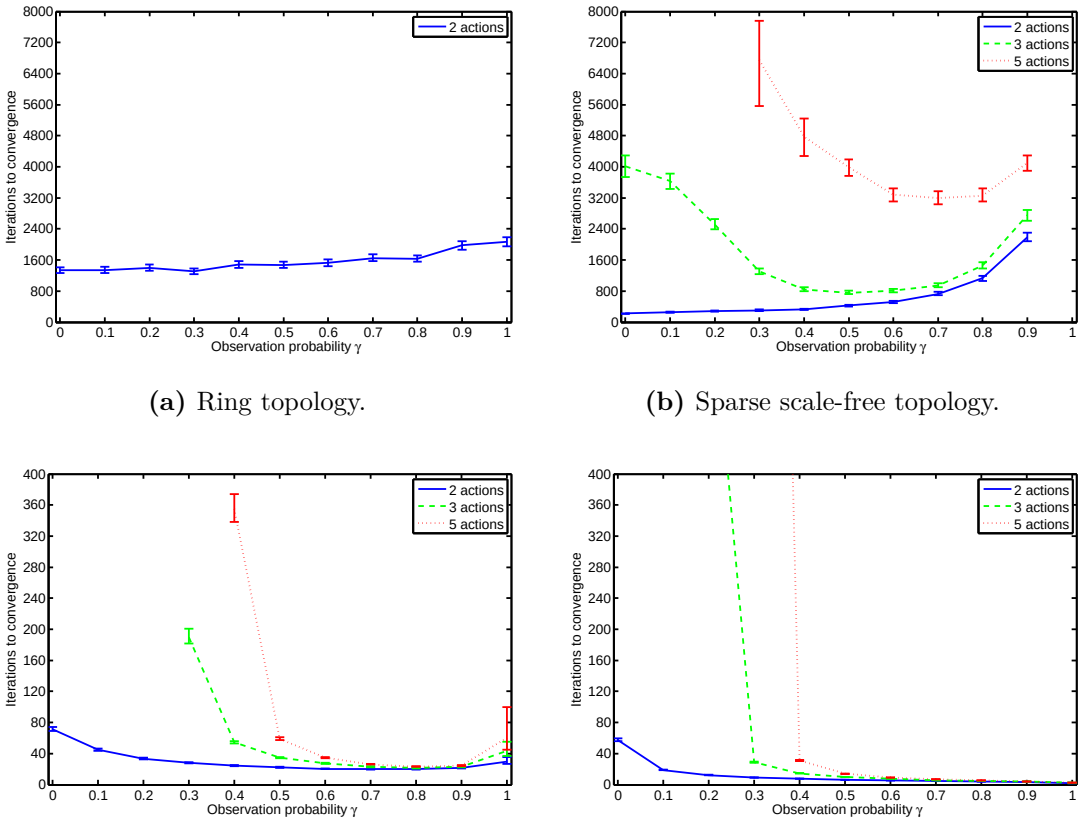


Figure 3.16: Convergence time in the fully connected topology for different number of agents and actions. Multi-player interaction model with $\beta = 0.3$ and $\gamma = 0$.

demonstrate that agents continue to select the same action after a convention has been reached. Once again, in Figure 3.19a *Agent ID* shows the individual agents and each dot represents the action of a given agent at one particular time step. At time step 1 agents are randomly assigned actions 1, 2 or 3, which are represented with a black, gray and white dot, respectively. Neighboring Agent IDs are not necessarily neighbors in the network due to the stochastic algorithm for construction of the scale-free topology. Therefore, contrary to Figure 3.9a, clusters of agents cannot be directly observed and thus the dots in the plot appear random. Nevertheless, Figure 3.19b shows the gradual increase of the number of agents selecting action 1.

Adding observation in multi-player model significantly improves convergence time in the dense scale-free and full topologies. We determined empirically that in the ring and the sparse scale-free network, observation slows down convention emergence under the multi-player model and therefore γ should be set to 0. The slower convergence comes from the fact that observation makes the agent select the most common action. As explained above, in sparse topologies agents have an insufficient sample of the best action and therefore their observations (and hence actions) vary significantly. In the denser topologies, on the other hand, the best value for γ is between 0.8 and 0.9, which results in more frequent observation. The effect of local observation is even more profound when agents have more than 2 available actions. In nearly all topologies setting the observation probability larger than 0.2 dramatically reduces the convergence time. With $\gamma \leq 0.2$ the behavior of agents with more than 2 actions cannot converge within 10000 iterations. Therefore, further studies



(a) Ring topology.

(b) Sparse scale-free topology.

(c) Dense scale-free topology.

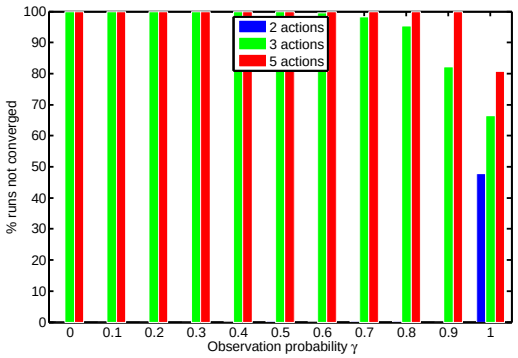
(d) Fully connected topology.

Figure 3.17: Convergence time for different topologies under multi-player interactions with 100 agents. Error bars show the 95% confidence interval of the mean. Keep probability $\beta = 0.01$.

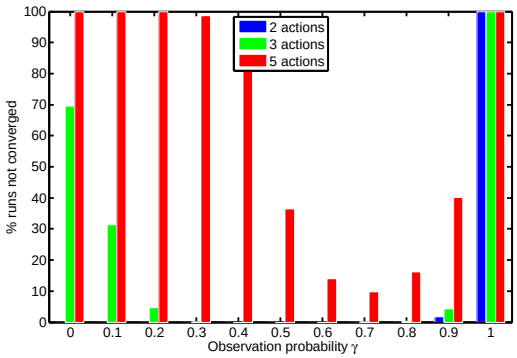
are required to find a good trade-off between the rate with which observation speeds up convergence and the cost incurred by agents due to observation.

3.8.5 Comparison with pairwise interactions

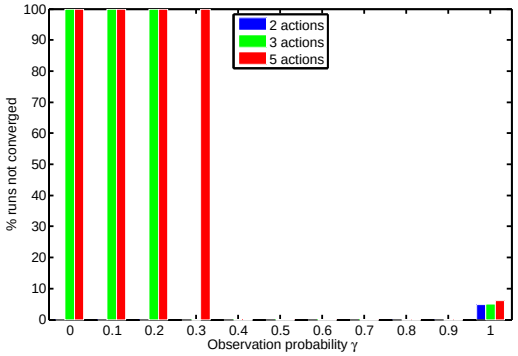
In sparse topologies, the convergence time of agents under the multi-player interaction model is comparable to that of the pairwise model. When the network is sparsely connected, conventions emerge equally fast when at every time step agents interact with only one random neighbor or if they interact with all neighbors at the same time. In denser networks without local observation ($\gamma = 0$) multi-player interactions only slightly outperform the pairwise model. This result is somewhat surprising, since agents in the multi-player model receive a more informative feedback signal. We point out here that the interaction model is a property of the co-



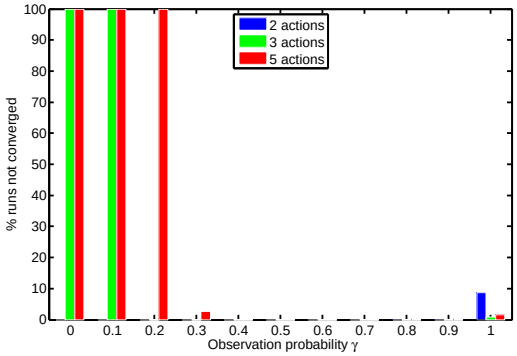
(a) Ring topology.



(b) Sparse scale-free topology.



(c) Dense scale-free topology.



(d) Fully connected topology.

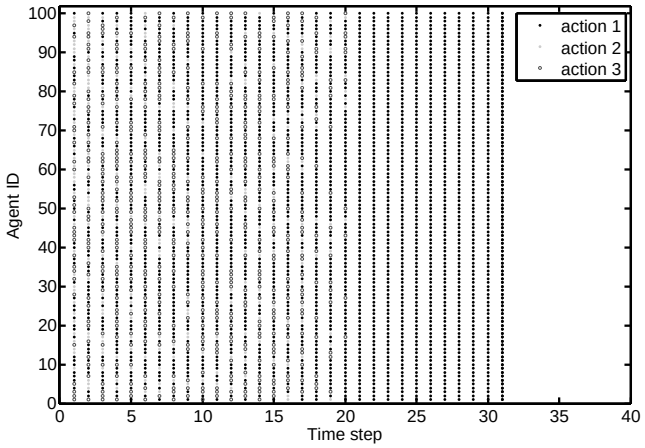
Figure 3.18: Percentage of runs that did not converge within T_{max} iterations from Figure 3.17.

ordination game that agents play and not a parameter that one can set in advance. We can conclude that our approach can be successfully applied in any topology, regardless whether agents interact with one other agent or many agents at the same time.

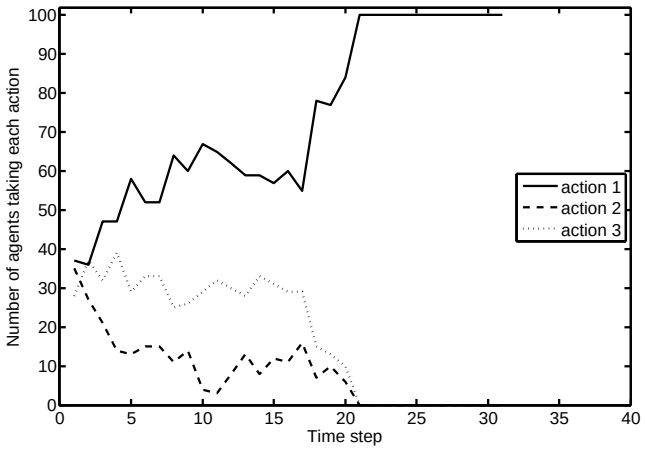
We also notice that for 2-player interactions the fastest convergence is achieved with a high shift probability, while in the multi-player model best results are obtained with a low keep probability. In other words, in both interaction models conventions emerge faster when agents have a large probability to select a different action upon conflict.

3.9 Conclusions

Our main objectives in this chapter were to propose a decentralized approach for fast on-line convention emergence in multi-agent systems, to analyze its convergence



(a) The action of each agent at each time step.



(b) The number of agents taking each action at each time step.

Figure 3.19: Results from a single (typical) simulation run of Algorithm 1 in the dense scale-free topology with 100 agents and 3 available actions. Multi-player interaction model with $\beta = 0.01$ and $\gamma = 0.8$.

properties and to evaluate the behavior of agents through an extensive simulation study. Our approach is called Win-Stay Lose-probabilistic-Shift (WSLpS), generalizing two well-known strategies in game theory — Win-Stay Lose-Shift (WSLS) and Win-Stay Lose-Randomize (WSLR). The probabilistic component of our approach, however, allows for a whole spectrum of strategies, two of which are WSLS and WSLR. Our empirical results suggest that for certain values of this probabilistic component WSLpS yields strategies that outperform both WSLS and WSLR. Concerning our research question **Q1**, we showed that using WSLpS, within only a short

number of time steps agents involved in a repeated pure coordination game are able to reach a mutually beneficial outcome on-line without a central mediator. Using the theory of Markov chains we proved that our WSLpS approach always converges in finite number of time steps to a pure coordination outcome. Our empirical evidence also suggests that agents applying WSLpS can reach convention based on only local interactions and limited feedback. Another desirable property of our approach is that conventions become absorbing states of the system, so that once all agents learn to select the same action, they will no longer change actions and escape the convention. Nevertheless, if the convention is somehow externally disrupted, agents will still converge to a (possibly different) convention.

We studied the behavior of players in different topological configurations and conclude that densely connected agents reach a convention on average faster than agents in sparser networks. We investigated empirically the convergence duration of our approach under both pairwise interactions with binary payoffs and multi-player interactions with multi-valued feedback. In both models we observe that conventions emerge faster when agents have a large probability to change their action upon conflict. The results also indicate that WSLpS performs equally well in both interaction models and therefore can be successfully applied in such domains. Adding local observation in the multi-player model further lowers the convergence duration. The latter improvement is even more pronounced in games with more than 2 available actions. However, information on the actions of others does not always lead to significant improvements, as we observed in the pairwise model. Thus, adding local observation to WSLpS is only useful in dense networks where agents are involved in multi-player coordination games.

One line of future work we are considering is to apply our WSLpS in an asynchronous setting, where agents may select their actions with different frequencies in each of the two interaction models. Another important aspect that needs further study is to investigate in a more detailed manner the relationship between the average path length of the network and the convergence time of agents.

Chapter 4

(Anti-)Coordination: dispersion games

In the previous chapter we studied in detail the pure coordination problem faced by highly constrained agents, such as nodes in a wireless sensor network (WSN). We proposed a simple decentralized approach that when adopted by individual agents leads to global successful coordination. In this chapter we will examine the rest of the (anti-)coordination problem, namely pure anti-coordination and the combined problem of coordination and anti-coordination. Similarly to the previous chapter, here too we are concerned with the abstract problem of (anti-)coordination, but all our choices and examples are motivated from the WSN perspective. To guide our research on pure anti-coordination, we pose the following question:

Q2: *How can agents achieve pure anti-coordination in a decentralized manner in dispersion games?*

We then examine the combined problem of coordination and anti-coordination in dispersion games. We argued in Section 2.2.1 that coordination and anti-coordination are inherently related and that the goal of agents in both games is the same — learning to select the appropriate actions, in order to avoid conflicts. In fact, the main difference between these games is the way the payoff signal is defined. Therefore, the same win-stay lose-probabilistic-shift (WSLpS) approach can be applied without any modification in these settings as well. We see in this chapter that WSLpS works as well in pure anti-coordination games as it does in pure coordination games from

the previous chapter, using the same limited environmental feedback and only local interactions. Moreover, we will show how the same approach can perform well in games that involve both coordination and anti-coordination.

We then compare the performance of WSLpS to several algorithms, proposed in the literature of anti-coordination games. We show that WSLpS outperforms these algorithms in different topologies and for different number of agents and actions. In addition, WSLpS can be applied in a wide range of scenarios, in which other algorithms are not suitable. Lastly, we compare the speed of convergence between coordination, anti-coordination and the combined game and show how the former two game types relate to each other and how the (anti-)coordination game involves characteristics of both.

4.1 Introduction

In this chapter we study the behavior of agents in dispersion games, introduced by Grenager *et al.* [2002]. In dispersion games the aim is to let agents anti-coordinate by maximally dispersing over the set of available actions. Grenager *et al.* consider only games played on a fully connected graph, but here we study other topologies as well. Due to the topological restrictions, however, in the kind of dispersion games we consider, we require that each agent selects an action different from those of all its neighbors. This requirement stems from the communication constraints of WSNs, where neighboring transmissions can interfere and therefore neighbors should select different channels.

The pure coordination problem, studied in Chapter 3 manifests itself in WSNs when sensor nodes attempt to communicate. For example, two neighbors need to coordinate on selecting the same time slot for forwarding a message, or selecting the same channel for communication. Similarly, the pure anti-coordination problem explored in this chapter is present in WSNs as well. Two neighboring nodes attempting to forward different messages need to select either different time slots for transmission, or different channels, otherwise their messages will interfere. Yet nodes themselves have no individual preferences on who goes first[1], as long as all messages are successfully transmitted. Clearly, the number of failed trials has a huge impact on the lifetime of the system and therefore agents need to learn to (anti-)coordinate in as few time steps as possible. In addition, the limited information and resources available to sensor nodes do not allow them to execute complex algorithms that require large memory. We present here an example of the pure

[1] Assuming no specific quality of service requirements.

anti-coordination problem faced by energy constrained sensor nodes under limited environmental feedback.

Example 11 (WSN pure anti-coordination)**.** *Consider a wireless sensor network of an arbitrary topology, where sensor nodes need to forward large amounts of data. To allow for parallel transmissions, neighboring nodes need to select different frequencies (or channels) to send their data simultaneously. In the absence of central control, how can neighboring nodes in the wireless sensor network learn over time to transmit on different frequencies?*

The challenge for the designer of such a decentralized system is to engineer an approach that will allow the individual nodes to anti-coordinate their choices in only few interactions using minimal resources. In Phung *et al.* [2012] we report on a WSN communication protocol in a similar multi-channel anti-coordination scenario as the one presented in Example 11.

In the next section we present related literature on the anti-coordination problem and then define that problem in Section 4.3. In Section 4.4 we describe several algorithms for anti-coordination, presented in literature, as well as our own WSLpS. We compare the performance of these algorithms in Section 4.5. We then study the full problem of (anti-)coordination in Section 4.6 before we conclude in Section 4.7.

4.2 Related work

In contrast to the extensive literature on coordination games, little work has focused on anti-coordination games. Despite the close relationship between the two types of games, they differ in one key aspect, namely the solutions (or equilibria) of the games. While agents in a coordination game can always arrive at a solution, e.g. by all selecting the same action, a solution need not always exist in anti-coordination games. Bramoullé (2001, 2007) has shown how the underlying interaction graph affects the equilibria of the latter games. The author shows that in 2-action games agents can anti-coordinate with all their partners only when the network is bipartite. A bipartite network is a graph where the set of vertices (or agents) can be partitioned in two disjoint subsets such that no link connects two vertices in the same subset. For example, 3 agents in a fully connected network cannot arrive at a solution when having only two available actions, since the graph is not bipartite. In contrast, in a coordination setting the 3 agents can always converge on one of the two actions. Generally, successful anti-coordination with k actions is possible if the network is k-partite.

Anti-coordination games were originally studied for two agents and two actions. Bramoullé [2001] studies these games with multiple agents, arranged in a fixed topology and calls them *complementarity games*. Although he investigates only 2-action games, his findings naturally extend to games with more than two actions. In that regard, Grenager *et al.* [2002] generalize the anti-coordination games to arbitrary number of agents and actions and call it *dispersion games*. Dispersion games can naturally model the load balancing problem and the class of games, known as minority games. The authors evaluate the convergence times of several learning strategies that agents can use in dispersion games. Each of these algorithms requires different amounts of information (see Table 1 in Grenager *et al.* [2002]). In this chapter we will compare the two algorithms that rely only on local information to our WSLpS in terms of convergence time.

't Hoen & Bohte [2003] enhance the collective intelligence framework of Wolpert & Tumer [2002] to improve the convergence results in dispersion games. However, their algorithm requires global knowledge and additional communication between agents. Namatame [2006] proposes the Give-and-Take (GaT) behavioral rule and evaluates it in minority games. The rule instructs agents to yield to others if they gain, and otherwise randomize their actions. In this way agents take turns being in the minority. This simple rule bares resemblance to WSLpS and therefore we will compare it to our approach. A drawback of GaT is that it is defined only for two actions and as a consequence can only be applied in bipartite graphs. In addition, in anti-coordination games this turn-taking behavior leads to oscillations — once agents successfully anti-coordinate, they will keep switching between the two goal states.

The anti-coordination problem studied in this chapter is closely related to the problem of graph coloring [Jensen & Toft, 1995]. In graph coloring we need to find an assignment of colors to vertices, such that no two adjacent vertices share the same color. However, our domain differs from graph coloring, due to the additional constraints of dispersion games in the context of WSNs. In particular, in our setting algorithms for anti-coordination must be decentralized, rely only on limited local information and use no additional communication between agents. Moreover, in our context, agents interact simultaneously, while graph coloring maintains no particular notion of agent interaction. Nevertheless, decentralized graph coloring algorithms that obey these restrictions may also be used by agents in dispersion games. Analogously, the algorithms we describe in this chapter can also be applied to graph coloring problems. The problem of graph coloring is related to the framework of distributed constraint optimization (DCOP). While DCOP is limited to

planning problems using complete information, the work of Taylor *et al.* [2011] extends this framework to address real-world problems, such as optimization in WSNs. The authors propose distributed coordination algorithms balancing exploration and exploitation in order to maximize the on-line, rather than the final, reward.

Another problem that focuses on the on-line performance and the exploration-exploitation trade-off is the multi-armed bandit (MAB) problem [Auer *et al.*, 2002]. MAB problems typically assume that the payoffs of each arm are drawn from some random distribution with given parameters that are unique for each arm, but unknown to the agent. Single-agent algorithms attempt to minimize the regret with respect to the optimal arm. In non-stationary settings the agent must continue to explore, since the payoff distribution parameters may change, leading to a different optimal arm. Recently MAB approaches have been proposed in a multi-agent setting, where the non-stationarity of the environment comes from the behavior of other agents in the system. Liu & Zhao [2010] have implemented a decentralized multi-agent MAB algorithm in a particular game setting, in the context of cognitive radios. Agents choose actions (or wireless channels) with unknown payoff distributions related to the quality of the channels and attempt to minimize regret with respect to the best channel. However, if two agents select the same channel, they will experience interference and therefore will receive no payoff. While this setting resembles our problem of anti-coordination, in dispersion games we have no notion of a best action, i.e. all alternatives are equally good (or equally bad). In addition, MAB action selection policies typically explore actions that have not been selected "often enough" in the recent past and therefore the anti-coordination states are not absorbing. We will come back to this problem in Section 4.4.2.

4.3 The Anti-coordination Game

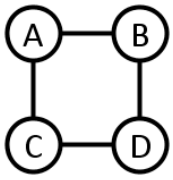
In this chapter we use the same game model and assumptions presented in Chapter 3. These include the requirements for agent and action symmetry, as well as the assumption that agents have no individual preferences. Again we have N agents arranged in a static connected interaction graph where agents that share an edge are called neighbors. Throughout this chapter, and as done in the literature of anti-coordination games, we adopt the multi-player interaction model with informative feedback, presented in Section 3.8.1. Each agent interacts with all its neighbors and receives a payoff based on the number of neighbors that choose a different action. The multi-player interaction model is also motivated from the sensor network domain where transmissions are omnidirectional and affect all nodes within range. In

our anti-coordination games agents know only how many neighbors have selected a different action and not what action they have selected. Formally, the payoff p_i to each agent i is:

$$p_i = \frac{n_i - n_i|_{a_j=a_i}}{n_i} = \frac{n_i|_{a_j \neq a_i}}{n_i}$$

where n_i is the number of neighbors of i, $n_i|_{a_j=a_i}$ are those who select the same action and analogously $n_i|_{a_j \neq a_i}$ are the neighbors with different action. As motivated by the WSN domain, here too we require that only the initiator of each game may receive payoff and that agents use the same action in all games they participate at a given time step. This model of one-sided multi-player interactions is also adopted by Bramoullé *et al.* [2004] in the context of human players with no individual preferences. A solution (or a Pareto optimal Nash equilibrium) of the anti-coordination game is where each agent has selected an action unlike that of its neighbors.

Figure 4.1 illustrates the anti-coordination game (or dispersion game) using a small example. We show in Figure 4.1b the payoff table of agent A in the topology of Figure 4.1a. Here A chooses rows, B chooses columns and C chooses tables. Notice that agent D is not affecting the payoff of A, since the two agents are not neighbors.



(a) Grid topology with 4 agents.

a_c^1	a_b^1	a_b^2
a_a^1	0	1/2
a_a^2	1	1/2

a_c^2	a_b^1	a_b^2
a_a^1	1/2	1
a_a^2	1/2	0

(b) Payoff matrix of agent A.

Figure 4.1: A sample topology of 4 agents with 2 actions together with the payoff matrix of agent A for the pure anti-coordination game.

Since the graph in Figure 4.1a is bipartite, agents can reach an equilibrium using only 2 actions. In this chapter we are interested in the speed of convergence of agents in large networks with different amount of actions. We study three topologies in particular, namely ring, grid and fully connected. In this chapter we omit the scale-free topology, since the probabilistic element involved in the generation of the network makes the analysis more complex. Nevertheless, anti-coordination games with k actions can be played on scale-free topologies, as long as the degree of any vertex does not exceed k. While the grid topology and the ring (with even number of agents) are bipartite, this is not the case with the fully connected one. Note that in dispersion games Grenager *et al.* require only that agents in a fully

connected network are maximally dispersed over the set of available actions without any restrictions on the number of actions k. However, for nodes in a fully connected WSN, the setting where $k < N$ implies that the messages of some nodes interfere with those of others, leading to inefficient network performance. Therefore, in the fully connected topology we require that $k = N$, i.e. the number of actions should be the same as the number of agents. Similar requirement can be applied for studying scale-free networks.

4.4 Algorithms for anti-coordination

In this section we will address **Q2** and outline the algorithms that we use to solve the decentralized anti-coordination problem in dispersion games. Although possibly many algorithms can be applied in this setting, we chose to compare WSLpS only against other algorithms proposed in the literature on anti-coordination. We will start with our **Win-stay Lose-probabilistic-Shift** algorithm that we presented in Chapter 3. Then, we will present the **Q-Learning** and **Freeze** algorithms, studied by Grenager *et al.* [2002]. All other algorithms tested by these authors require more information on the actions of others and hence cannot be used by agents with limited local knowledge, such as wireless sensor nodes. Lastly, we apply the **Give-and-Take** algorithm, used by Namatame [2006] in the local minority game, where each agent plays an anti-coordination game with its nearest neighbors.

We study the iterated pure anti-coordination game in a simulation process, similar to that in the chapter on pure coordination games using multi-player interactions. As outlined earlier, we will not study pairwise interactions, as the literature on anti-coordination is concerned primarily with multi-player interactions, which are also observed in WSN communication. At every discrete time step (or iteration), each agent meets all its neighbors and receives a payoff that indicates the ratio of neighbors that selected a different action (but not which action). This is the only information that agents receive from the environment. Thereafter, agents use their action selection mechanism and the obtained payoff to synchronously pick their (new) actions, which will be used in the next iteration. After that, the new iteration begins. This repeated anti-coordination game is played until the action of each agent differs from the action of its neighbors, or until $T_{max} = 10000$ time steps have passed. Our performance criterion here is the number of iterations until convergence. We use the same simulation process, detailed in Algorithm 6, for each of the above algorithms. However, each algorithm has a separate implementation of the function *selectAction* (line 11). That function specifies how action probabilities

Algorithm 6 Main simulation process for the pure anti-coordination problem

Input: $N \leftarrow$ number of agents,
$S \leftarrow$ type of topology,
$T_{max} \leftarrow$ maximum iterations
Output: time steps t until full convergence or T_{max}

```
1: t ← 0
2: g ← initTopology(N, S)
3: for all agents i do
4:     a_i ← selectRandomAction
5: end for
6: repeat
7:     for all agents i do
8:         p_i ← getPayoffFromGame(a_i, i, g)
9:     end for
10:    for all agents i do
11:        a_i ← selectAction(p_i, a_i)
12:    end for
13:    t ← t + 1
14: until anticoordinationReached(a) OR t ≥ T_max
15: return t
```

are updated based on the payoff and how actions are selected based on these probabilities. We will now define this function for each algorithm, as implemented by the individual agents. Note that each agent implements the same code.

4.4.1 Win-Stay Lose-probabilistic-Shift

We use here the same WSLpS algorithm that we applied to pure coordination games in Section 3.8. The algorithm uses the payoff p_i and parameter $\beta \in (0, 1)$, which is the *keep probability* upon conflict and it is the same for all agents. In the next iteration each agent i will select the same action as in the last iteration with probability Π_i^{keep} where

$$\Pi_i^{\text{keep}} = \begin{cases} p_i + \beta & \text{if } p_i < 1 - \beta \\ 1 & \text{otherwise} \end{cases} \tag{4.1}$$

Thus, the probability with which an agent will keep its action depends on the number of neighbors with whom it agrees. With probability $1 - \Pi_i^{\text{keep}}$ the agent will select a different action uniformly random.

In Section 4.5.2 we will show how β can be set. Algorithm 7 shows the pseudo-code of WSLpS that will be implemented by each agent. Since agents will never leave successful anti-coordination, we count the number of iterations until the first time they anti-coordinate.

Algorithm 7 function *selectAction* for WSLpS

Input: payoff $p_i \in [0, 1]$ from the latest interaction
current action a_i

Output: the new action a_i of the agent

$rnd \leftarrow generateUniformlyRandomNumber(0, 1)$
if $rnd > (\beta + p_i)$ **then**
    $a_i \leftarrow selectDifferentUniformlyRandomAction(a_i)$
else
    // *keep action* a_i
end if
return a_i

Algorithm 8 function *selectAction* for QL

Input: payoff $p_i \in [0, 1]$ from the latest interaction
current action a_i

Output: the new action a_i of the agent

$q[a_i] \leftarrow (1 - \lambda) \cdot q(a_i) + \lambda \cdot p_i$
for all available actions m **do**
    $\pi[m] \leftarrow \frac{e^{\frac{q[m]}{\tau}}}{\sum_{b=1}^{k} e^{\frac{q[b]}{\tau}}}$ // *map the q-values to the Boltzmann distribution*
end for
$a_i \leftarrow selectActionAccordingToDistribution(\pi)$
return a_i

4.4.2 Q-Learning

Next we implement the Q-Learning algorithm (QL), used by Grenager *et al.* [2002] in pure anti-coordination games. Note that we apply the QL algorithm, as outlined and implemented by the authors. Nevertheless, the algorithm bares resemblance to algorithms implemented for non-stationary multi-armed bandit (MAB) problems, where an agent learns the expected reward of each arm and selects arms so as to minimize regret with respect to the best one. For example, Koulouriotis & Xanthopoulos

[2008] use an exponentially-weighted sample average (as in line 1 of Algorithm 8) to determine action-value estimates of a non-stationary single-agent MAB problem. They also apply the softmax actions selection (as in line 5 of Algorithm 8) to select the best arm, according to these estimates. However, in a game setting payoffs are determined based on the actions of other agents and thus there is no notion of a best arm. Liu & Zhao [2010] implement a decentralized MAB approach in a multi-agent game setting, where payoffs are unknown and independent of actions of others, except when some agents select the same action. MAB approaches are suitable when the maximum reward is not known and alternatives score differently. When the expected payoff of each action is the same, as in the games considered in this chapter, MAB approaches will have problems learning a good anti-coordination outcome.

In the QL algorithm agents learn the expected payoff of performing each action and apply the softmax action selection mechanism using the Boltzmann distribution with temperature parameter $\tau \in \mathbb{R}$. QL stores a quality value (or q-value) for each action and updates the value of the selected action at every time step based on the payoff p_i from the last interaction and a learning rate parameter $\lambda \in (0, 1]$ (cf. Definition 12). As outlined in Section 2.3.1, low τ makes the action selection algorithm more greedy, while high τ makes it more random. We are interested in a more greedy behavior, so that agents who successfully anti-coordinate with their neighbors can keep playing the same action and thus allow others to find conflict-free actions. The learning rate parameter, on the other hand, needs to be relatively high to give more weight on recent payoffs, rather than on past plays, in order to quickly find actions that are not selected by neighbors. Note that QL requires the tuning of two parameters and is sensitive to the selection of initial q-values. Grenager *et al.* [2002] do not specify the exact values for the two parameters, nor the initial q-values. In Section 4.5.2 we study how λ and τ affect the convergence time of the system when the q-values are initialized to 0.5, which is half way between the worst and the best q-value. The pseudo-code for the Q-Learning algorithm (QL) is displayed in Algorithm 8. Note that due to the exploration policy, agents may still escape a successful anti-coordination outcome. Nevertheless, for a fairer comparison, we count the number of time steps until the first time all agents anti-coordinate. A softmax action selection mechanism with a decreasing temperature could eventually lead to steady behavior where agents do not escape the anti-coordination outcome. However, in our experiments we implement the QL algorithm, as described by Grenager *et al.* [2002].

4.4.3 Freeze

Grenager *et al.* [2002] also apply the Freeze algorithm, which instructs each agent to choose actions randomly until the first time it differs from the actions of all its neighbors. Thereafter the agent continues to play that action, regardless of whether its neighbors select the same or different action. This strategy requires no parameter, uses only local information and imposes minimal system requirements. Algorithm 9, shows the pseudo code of the Freeze strategy, where the local variable *frozen* is initialized to *false* for each agent. Once pure anti-coordination is achieved, all agents will have their action "frozen" and therefore never leave the anti-coordination outcome. We count the number of iterations until the first time all agents anti-coordinate.

Algorithm 9 function *selectAction* for Freeze

Input: payoff $p_i \in [0, 1]$ from the latest interaction
current action a_i

Output: the new action a_i of the agent

```
1: if p_i == 1 then
2:     frozen ← true
3: end if
4: if not frozen then
5:     a_i ← selectUniformlyRandomAction
6: end if
7: return a_i
```

4.4.4 Give-and-Take

Another algorithm that uses only local information and imposes minimal system requirements is the Give-and-Take rule (GaT), proposed by Namatame [2006]. He applies it in games where each agent is involved in a local El Farol Bar problem (see Example 6) with their nearest neighbors on a grid topology. We remind the reader that in the games we study agents are not selfish, but collectively aim to improve the performance of the system. GaT makes agents yield to others if they gain, and otherwise randomize their actions. In this way agents take turns being in the minority, instead of selfishly aiming to *stay* in the minority. Since the rule is defined for only two actions, we can use GaT only in two-action pure anti-coordination games, played on bipartite graphs. If GaT would be applied in k-action games for $k > 2$, there could be multiple minorities and majorities and thus it is not clear how agents will select a minority and how they will yield to others.

Algorithm 10 function *selectAction* for GaT

Input: payoff $p_i \in [0, 1]$ from the latest interaction
current action a_i

Output: the new action a_i of the agent

1: **if** $a_i == 1$ **then**
2: $ratio \leftarrow 1 - p_i$
3: **else**
4: $ratio \leftarrow p_i$
5: **end if**
6: **if** $ratio \leq \theta$ **and** $a_i == 1$ **then**
7: $a_i \leftarrow 2$
8: **else if** $ratio > \theta$ **and** $a_i == 2$ **then**
9: $a_i \leftarrow 1$
10: **else**
11: $a_i \leftarrow selectRandomAction$
12: **end if**
13: **return** a_i

The author defines θ as the capacity of the bar in the El Farol Bar problem. Without loss of generality, in our anti-coordination games, we define "visiting the bar" as action 1 and "staying home" as action 2. Thus, an agent i is in the minority when it visits the bar ($a_i = 1$) and the bar is below its capacity ($1 - p_i \leq \theta$), or stays at home ($a_i = 2$) and the bar is overcrowded ($p_i > \theta$).[2] The GaT rule says that when the ratio of attendance (or ratio of neighbors with action 1) is weakly below the capacity θ, an agent visiting the bar is in the minority and therefore in the next time step it will not visit the bar, i.e. it will yield to others. As a result, once successful anti-coordination is reached, each agent will constantly change between the two available actions (i.e. win at one time step and yield in the next) and hence agents constantly switch between the two anti-coordination outcomes. Since agents will never escape the anti-coordination outcomes, we count the number of iterations until the first time pure anti-coordination is achieved. Namatame sets θ to 0.6 to resemble the classical El Farol Bar problem, where the capacity of the bar is 60% of the population. He assumes a grid topology in a torus shape where each agent has exactly 4 neighbors (i.e. all agents on one edge of the gird are connected with those on the opposite edge). In a WSN scenario, however, we cannot make this

[2] Note that the payoff p_i defines the number of neighbors choosing a *different* action from that of agent i and not the number of neighbors choosing action 1.

assumption and therefore we apply GaT on a standard grid topology, where some agents on the edges have less than 4 neighbors. In our experimental setting the best value for θ is 0.3 for all agents, so that agents on the edges of the grid can also find the minority action.

4.5 Results from pure anti-coordination games

4.5.1 Experimental settings

We investigate the pure anti-coordination problem in 3 different topologies. We study a ring topology with 20 agents, a 5-by-5 grid topology with 25 agents and four fully connected topologies with 20, 30, 40 and 50 agents. The ring and grid topologies are bipartite and therefore agents can successfully anti-coordinate with only 2 available actions. We compare the rate of convergence of QL, GaT and WSLpS algorithms in the latter two topologies. To study the scalability of QL and WSLpS algorithms, we examine the rate of convergence in bipartite graphs for up to 5 available actions. Since GaT is defined for only two actions, we cannot include it in the comparison in games with more than two actions. Similarly, the Freeze algorithm is designed for the full topology and thus it is not useful to apply it in ring and grid topologies. When some agents "freeze" their action, due to the topological configuration of the networks, other agents may not find a feasible outcome. For example, in Figure 4.1a (on page 96) if agent **A** freezes to a_a^1 and agent **D** freezes to a_d^2, agents **B** and **C** cannot find conflict-free actions when the number of actions k is 2.

Lastly, in the fully connected topologies agents cannot achieve successful anti-coordination with less actions than there are agents. Therefore, we set the number of available actions in the four fully connected topologies to 20, 30, 40 and 50, respectively. In this way we can study the scalability of our WSLpS approach for larger networks and with more available actions. We compare it to the Freeze algorithm, which is designed for the full topology. QL, on the other hand, needs to allow for sufficient exploration, in order to find an action that no other agent has selected and at the same time a sufficiently greedy behavior, in order to stick to it, so that other agents find a conflict-free action. All sample runs with QL in the full topology took more than our limit of 10000 iterations for the parameter configurations we studied and therefore we conclude that QL does not perform well in the fully connected topology. Although all settings were tested with the above algorithms, not all algorithms were able to converge. Table 4.1 gives an overview of the algorithms we are comparing and the corresponding experimental settings that

work well for the respective algorithm. Note that WSLpS is the only approach that is applicable in all these settings.

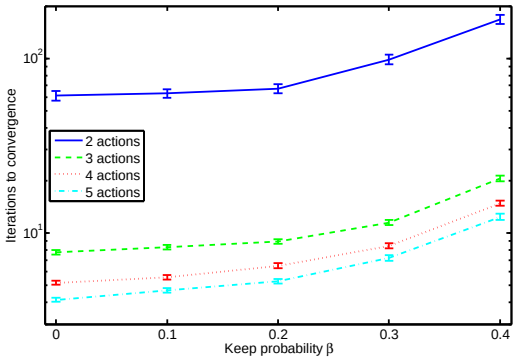
	topology:	ring				grid				full			
algorithm	actions:	2	3	4	5	2	3	4	5	20	30	40	50
WSLpS		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
QL		✓	✓	✓	✓	✓	✓	✓	✓				
Freeze										✓	✓	✓	✓
GaT		✓				✓							

Table 4.1: Overview of the algorithms and the corresponding experimental settings that work well.

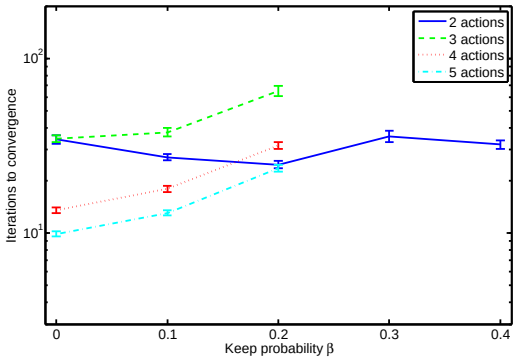
In all reported results we follow the same principles as outlined in Section 3.7. Results are averaged over 1000 runs, which constitute a sample. This number of runs was enough to draw statistically significant results, as the narrow confidence intervals of our plots indicate. Missing values indicate that all runs in the sample did not complete within 10000 iterations. The action for each agent is initialized uniformly random from the available actions. The performance measure of the system is the number of iterations until the action of each agent differs from that of all its neighbors. Note that in some graphs the y-axis is in logarithmic scale.

4.5.2 Parameter study

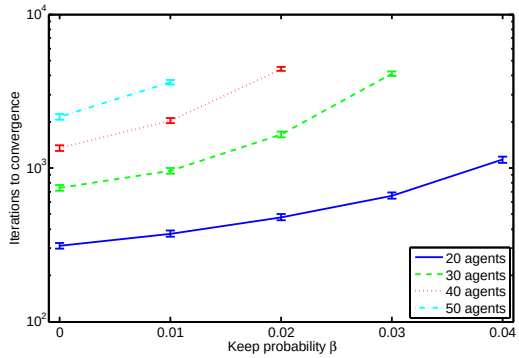
Before we compare the different algorithms, we will present a study of the parameters in WSLpS and QL. Figure 4.2 shows how the keep probability β of WSLpS affects the convergence time of agents in different topologies. We explain here the limit values for this parameter. We see that the larger the parameter, the slower the convergence time. However, if $\beta = 0$, agents in the bipartite graphs (i.e. ring and grid) cannot always reach anti-coordination with 2 actions (cf. Figures 4.3a and 4.3b). This is because agents, who are in conflict with each other and receive a payoff of 0 will all shift to the other action with probability 1 and thus still remain in conflict. Similarly, if $\beta \geq 0.5$ in the ring topology, an agent with one conflict will have a payoff of $p_i = 0.5$ and since $p_i \geq 1 - \beta$ its keep probability will be $\Pi_i^{\text{keep}} = 1$ (cf. Equation 4.1) and thus will not change its action. Therefore we do not test the settings where $\beta \geq 0.5$. In a similar fashion, in the grid topology with $\beta \geq 0.25$ an agent with four neighbors and only one conflict obtains a payoff of $3/4$ and will always keep its action (i.e. $\Pi_i^{\text{keep}} = 1$), since $p_i \geq 1 - \beta$ and therefore the network



(a) Ring topology with $N = 20$ agents.



(b) Grid topology with $N = 25$ agents.

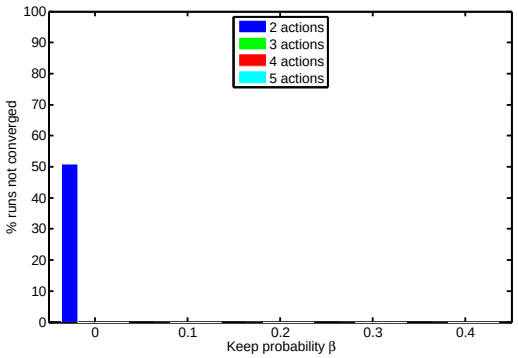


(c) Fully connected topology with $k = N$ actions.

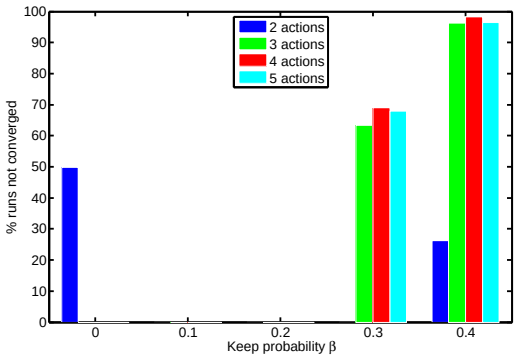
Figure 4.2: Convergence time of WSLpS in different topologies for different values of the keep probability β. Error bars show the 95% confidence interval of the mean.

will not always converge. The latter result is confirmed by Figure 4.3b, showing that for $\beta > 0.2$ not all runs converge in the grid topology. In addition we observe that an anti-coordination game with 2 actions, although having only 2 solutions, converges faster than a game with 3 actions, which has much more solutions. We explain the reason behind this phenomenon in Section 4.5.3 below. We determine from Figure 4.2 that the best value, among those tested, for ring and grid is $\beta = 0.1$, while in full topology $\beta = 0$ gives the fastest convergence time. Thus, in the latter topology agents keep their action with probability equal to the payoff they obtain. Note that here the range of "good" values for β is comparable to the "good" values of the keep probability in the pure coordination games from Section 3.8.4.

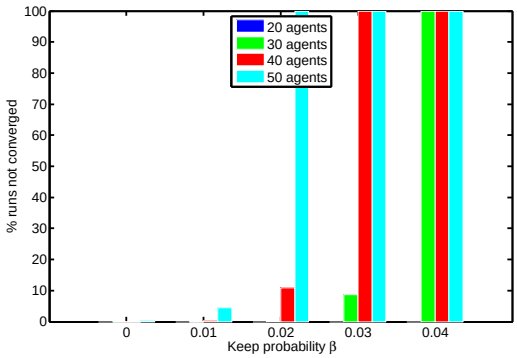
The effect of the learning rate and temperature parameters of QL are displayed in Figure 4.4. As we predicted in Section 4.4.2, the best convergence times are achieved with a relatively high learning rate, combined with a low temperature.



(a) Ring topology with $N = 20$ agents.



(b) Grid topology with $N = 25$ agents.



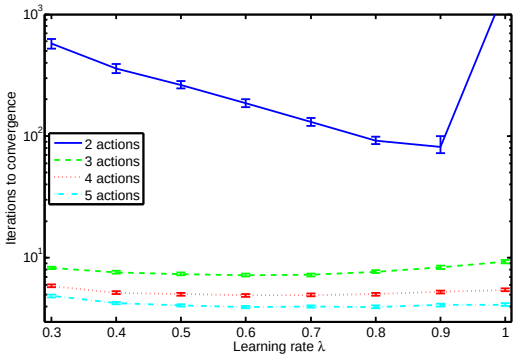
(c) Fully connected topology with $k = N$ actions.

Figure 4.3: Percentage of runs that did not converge within T_{max} iterations from Figure 4.2.

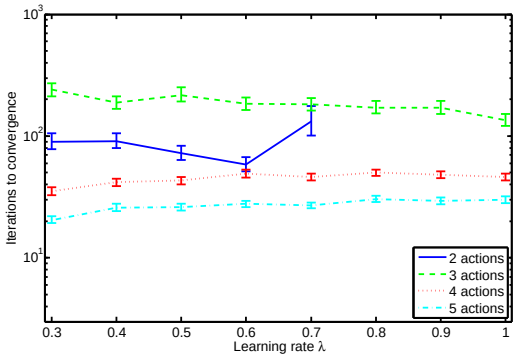
Similarly to WSLpS, the convergence time of QL for 2 actions in the grid topology is faster than that in a game with 3 actions (e.g. see Figure 4.4d). Moreover, a game with 2 actions has only 2 possible solutions, which underlies the erratic pattern of the corresponding graphs (all dark blue lines). We are able to determine from the reported results, that the values that perform best in both topologies and for all actions are $\tau = 0.1$ and the corresponding $\lambda = 0.8$. The QL algorithm can be extended by considering a variable learning rate for each agent. For example, Bowling & Veloso [2002] propose the WoLF principle (Win or Learn Fast), where the learning rate is adjusted based on the performance of the agent.

4.5.3 Results

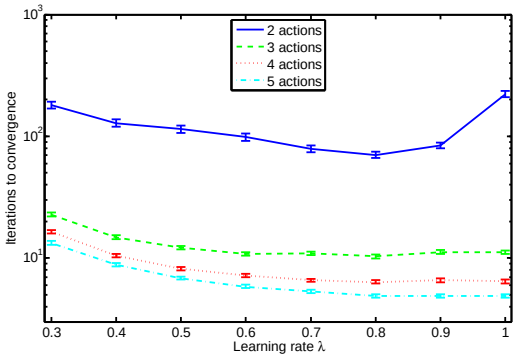
We see in Figures 4.5a and 4.5b that the Q-Learning algorithm and our Win-Stay Lose-probabilistic-Shift have comparable performance in terms of convergence time,



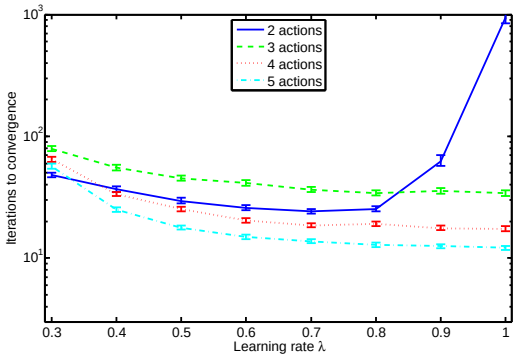
(a) Ring topology, $\tau = 0.05$.



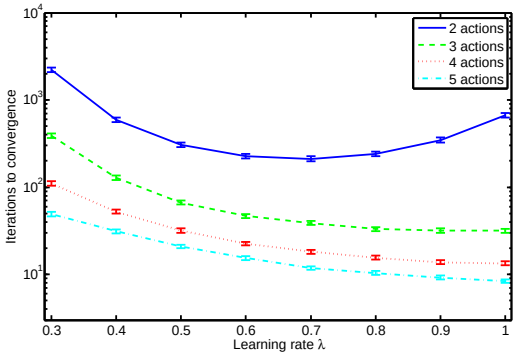
(b) Grid topology, $\tau = 0.05$.



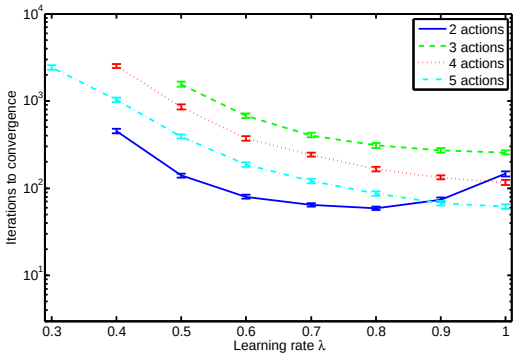
(c) Ring topology, $\tau = 0.1$.



(d) Grid topology, $\tau = 0.1$.



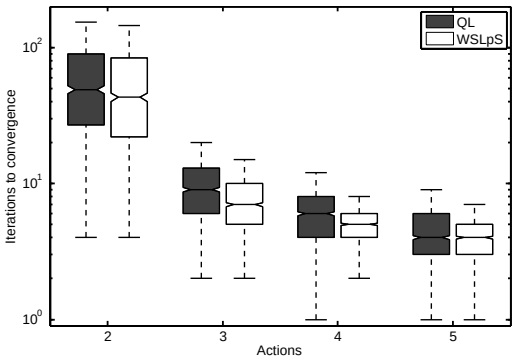
(e) Ring topology, $\tau = 0.15$.



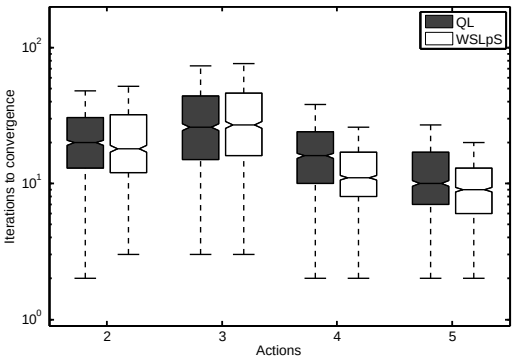
(f) Grid topology, $\tau = 0.15$.

Figure 4.4: Convergence time of QL in ring and grid topologies for different values of the learning rate λ and the temperature τ. Error bars show the 95% confidence interval of the mean.

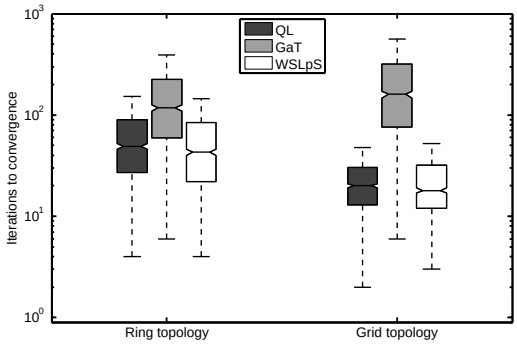
although the 95% confidence interval of our algorithm is almost always slightly lower than that of QL. The samples (but not results!) obtained from each of the two algorithms are significantly different (with a p-value in the order of 10^{-10}) according to a Mann-Whitney U-test with $\alpha = 0.05$, which is not surprising, since the distributions are generated by different algorithms. For two available actions, both algorithms outperform the Give-and-Take algorithm, as shown in Figure 4.5c. In the ring topology we notice that convergence time for QL and WSLpS decreases for higher number of available actions. Since each agent has only two neighbors, the chance of agents anti-coordinating increases with the number of actions. Agents



(a) Ring topology with $N = 20$ agents.



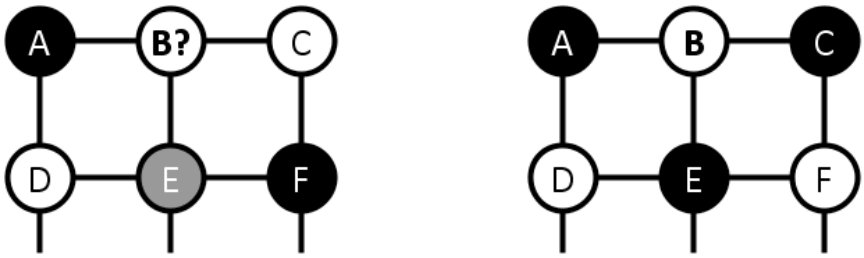
(b) Grid topology with $N = 25$ agents.



(c) The three algorithms in both topologies with 2 available actions.

Figure 4.5: Comparison between the convergence times of QL with $\tau = 0.1$ and $\lambda = 0.8$, WSLpS with $\beta = 0.1$, and GaT with $\theta = 0.3$ in ring and grid topologies.[3]

in the grid topology (Figure 4.5b) have more neighbors to anti-coordinate with and therefore the convergence time is on average higher than the convergence time in the ring.



(a) Grid topology with $k = 3$ available actions.

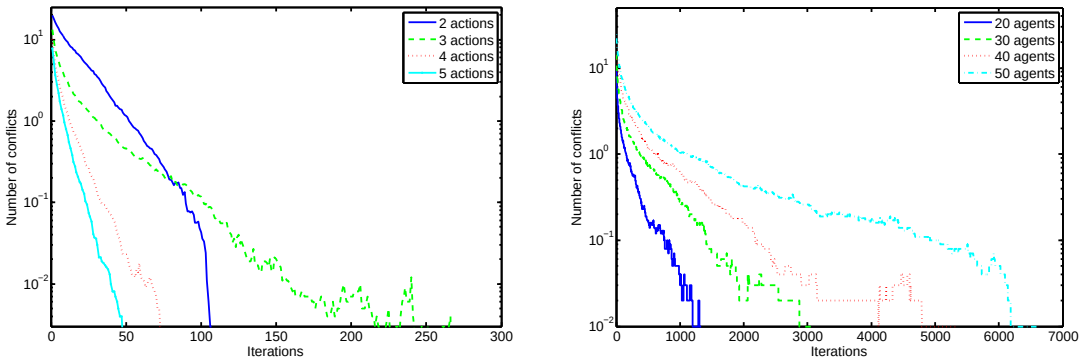
(b) Grid topology with $k = 2$ available actions.

Figure 4.6: A snapshot of an anti-coordination problem between agents in a grid topology. Each circle displays the name of the agent, while the color shows its selected action.

Interestingly, both for QL and WSLpS, anti-coordinating with 2, 4 and 5 actions in the grid topology is on average faster than with 3 actions. We attempt to explain this phenomenon in Figure 4.6. Although all neighbors in Figure 4.6a, except **B** can choose different actions, there is no feasible solution for agent **B**. The actions of **A**, **C** and **E** receive a high payoff, since they are different from those of their neighbors **D** and **F**. Only one of those three agents will be in conflict with **B** (agent **C** in this case). Thus the multi-agent system can take more time to escape from the outcome shown in Figure 4.6a, since all agents will have a high probability to select the same actions. With two available actions, on the other hand, such situation cannot occur, as illustrated in Figure 4.6b. If the actions of **A**, **C** and **E** agree with those of **D** and **F**, agent **B** can also select a conflict-free action. Inversely, if all three agents are in conflict with **D** and **F**, they will also be in conflict with **B** and therefore have a higher probability of shifting their actions and escaping this outcome. We show in Figure 4.7a the average number of conflicts between agents for different number of actions. A conflict is when two neighboring agents select the same action. Each conflict between two agents is counted once. We see that the conflicts in 3-action

[3] On each box, the central mark is the median, the edges of the box are the q_1 and q_3, i.e. the 25^{th} and 75^{th} percentiles. The notches show the 95% confidence interval of the median. The lower and upper whiskers extend to the most extreme data points not considered outliers, i.e. to the data points adjacent to $q_1 - (q_3 - q_1)$ and $q_3 + (q_3 - q_1)$, respectively. Outliers are not shown.

games are initially less that those in 2-action games. However, as time progresses, there are often some agents who find it difficult to anti-coordinate with 3 actions due to the above problem, resulting in a longer tail of the curve. This behavior explains why convergence with 3 actions is slower than with 2 in the grid topology in Figures 4.2 and 4.4.



(a) Grid topology with $N = 25$ agents. WSLpS with $\beta = 0.1$.

(b) Fully connected topology with $k = N$ available actions. WSLpS with $\beta = 0$.

Figure 4.7: Average number of conflicts in the pure anti-coordination game.

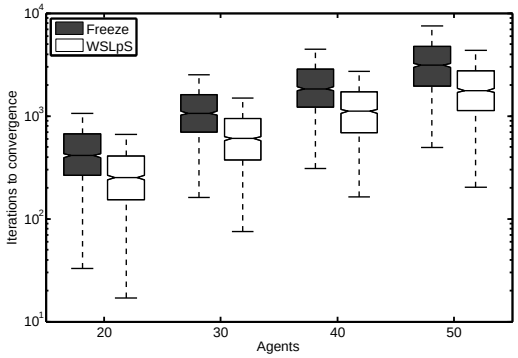


Figure 4.8: Comparison between the convergence times of Freeze and WSLpS with $\beta = 0$ in the fully connected topology with actions equal to the number of agents.

Lastly, we compare the Freeze algorithm to WSLpS in the fully connected topology in Figure 4.8. We see that the convergence duration of both algorithms increases with the number of agents. This effect can also be observed in Figure 4.7b, which shows the average number of conflicts. However, WSLpS is on average faster than

Freeze and this difference becomes more pronounced in larger networks. Again, a Mann-Whitney U-test with $\alpha = 0.05$ confirms (with a p-value in the order of 10^{-10}) that the obtained samples belong to two different distributions, as they are generated by two different algorithms.

Despite the comparable performance of QL and WSLpS in ring and grid, we point out that the former relies on two parameters and it is sensitive to the initial q-values. WSLpS, in contrast, has only one parameter to tune and is quite robust. In addition, WSLpS performs well in all topologies we tested for both different number of agents and actions, while QL, Freeze and GaT cannot be applied in all settings (cf. Table 4.1).

4.6 A game of coordination and anti-coordination

In Chapter 3 we studied the pure coordination game where all agents need to learn to select the same action. So far in Chapter 4 we explored the pure anti-coordination game, where each agent has to select an action unlike those of all its neighbors. In this section we move one step closer to the full problem of (anti-)coordination in wireless sensor networks.

In the beginning of this chapter we stated that the main difference between these games is the way the payoff signal is computed. We investigate here the performance of the same WSLpS approach we used so far, but in a game where agents need to both coordinate with some neighbors and at the same time anti-coordinate with others. We examine again the grid topology, but this time agents distinguish between their vertical and horizontal neighbors. This assumption is common in WSN, since nodes are usually aware of their hop distance to the base station and therefore can distinguish between nodes on the same hop (horizontal neighbors) and nodes on a higher or a lower hop (vertical neighbors). If we place the base station at the bottom of the grid and choose for a shortest path routing protocol, nodes need to forward their data vertically towards the sink. Although nodes are in range with their horizontal neighbors as well, horizontal message forwarding is not allowed. Nodes in such a WSN need to synchronize their communication with their vertical neighbors, in order to forward messages, and at the same time desynchronize with horizontal neighbors, in order to avoid interferences. However, nodes do not explicitly use this information of vertical and horizontal neighbors when synchronizing and desynchronizing, but are guided by the feedback they receive from the interactions. Note that we are still studying abstract (anti-)coordination games, but we refer to the WSN domain in this section to motivate our design choices.

4.6.1 The (anti-)coordination game

We design the (anti-)coordination game in this section to resemble the above WSN scenario, but we no longer speak about sensor nodes. Each agent receives a positive payoff for the number of its vertical neighbors with the same action and horizontal neighbors with different actions. Formally, the payoff p_i to agent i is computed in the following way:

$$p_i = \frac{1}{2}\left(\frac{n_i^v|_{a_j=a_i}}{n_i^v} + \frac{n_i^h\Big|_{a_j\neq a_i}}{n_i^h}\right)$$

where n_i^v and n_i^h are respectively the number of vertical and horizontal neighbors of i, $n_i^v|_{a_j=a_i}$ is the number of vertical neighbors with the same action as i and $n_i^h\Big|_{a_j\neq a_i}$ is the number of horizontal neighbors with different actions. Agents will keep their action in the next iteration according to Equation 4.1.

For a grid topology with 25 agents and 2 available actions, the two possible global solutions are shown in Figure 4.9; for 3 actions in the same topology there are 48 solutions and so on. In general, the number of possible solutions in a grid topology with N agents and k actions is $k(k-1)^{\sqrt{N}-1}$. We study here the performance of WSLpS in the 5-by-5 grid topology for up to 5 actions. Although the number of possible solutions increases exponentially, we will show in the next subsection that the convergence time of agents becomes slower.

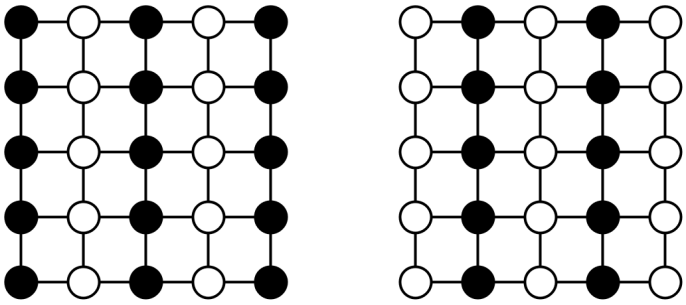
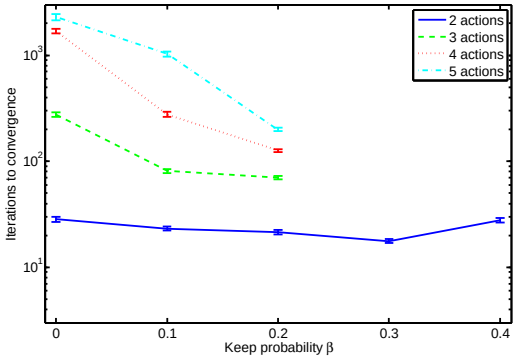


Figure 4.9: The two solutions of the (anti-)coordination game in grid topology for $N = 25$ agents and $k = 2$ actions (black and white).

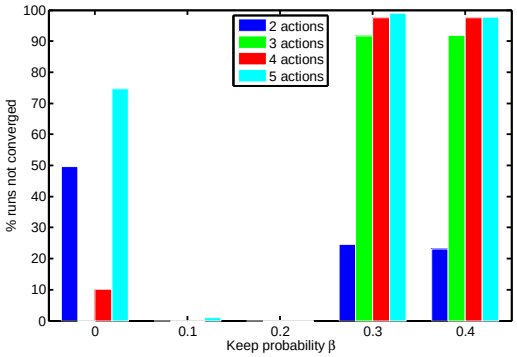
4.6.2 Parameter study

As with the pure anti-coordination game (cf. Section 4.5.2), we will perform here a parameter study for the keep probability β of our WSLpS approach. In Figure 4.10a we show how β affects the convergence time of our algorithm for 25 agents with different number of actions. We observe here the same effect as in Section 4.5.2. For

$\beta \geq 0.25$ the (anti-)coordination game cannot always converge, as confirmed by Figure 4.10b. In contrast to the pure anti-coordination game, however, the convergence time in our (anti-)coordination game increases for more available actions.



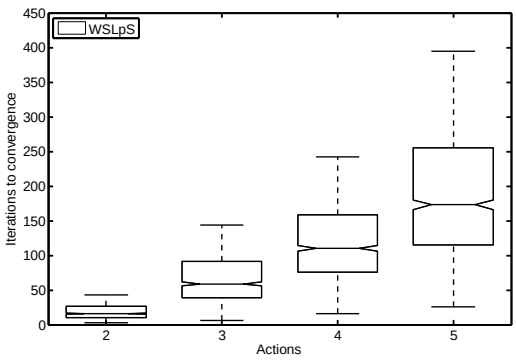
(a) Convergence time of WSLpS. Error bars show the 95% confidence interval of the mean.



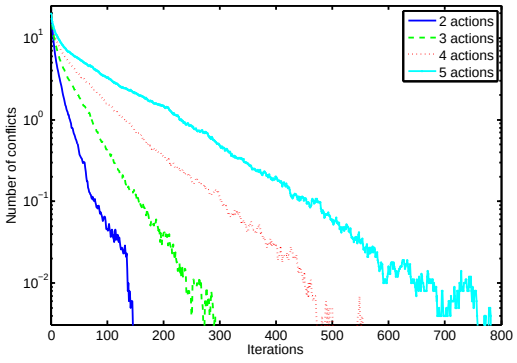
(b) Percentage of runs that did not converge within T_{max} iterations.

Figure 4.10: Results from the (anti-)coordination game with $N = 25$ agents in the grid topology for different values of the keep probability β.

4.6.3 Results and discussion



(a) Convergence times.



(b) Average number of conflicts.

Figure 4.11: Convergence time and conflicts of $N = 25$ agents in the grid topology for different number of actions in an (anti-)coordination game using WSLpS with $\beta = 0.2$.

We see in Figure 4.11a that the convergence time in the grid topology for the (anti-)coordination game is higher when agents have more available actions. The

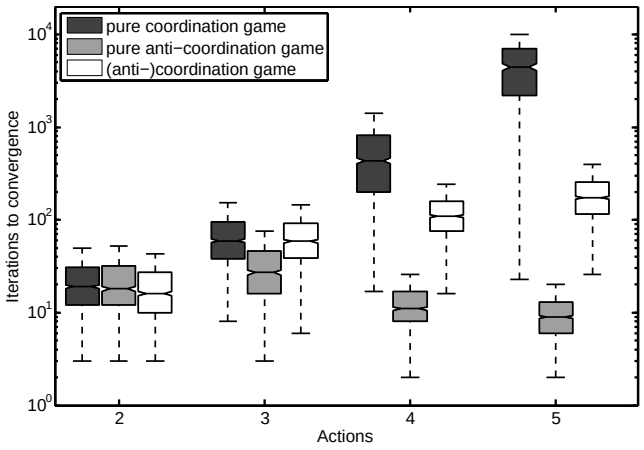


Figure 4.12: Convergence time of WSLpS with $\beta = 0.2$ in all three game types on the grid topology with $N = 25$ agents and $k = 2, \ldots, 5$ actions.

number of conflicts in each setting can be observed in Figure 4.11b. Although (anti-)coordination games with more actions have naturally more solutions, the coordination between vertical neighbors becomes more difficult with more available actions. The anti-coordination between horizontal neighbors, on the other hand, becomes easier for more actions, as we saw in Section 4.5.3. Lastly, in Figure 4.12 we notice that the convergence time in the (anti-)coordination problem on the grid topology is proportional to both coordination and anti-coordination games. The latter figure compares the three game types on the grid topology. Note that for two available actions each of the three games has exactly two solutions and therefore — comparable convergence times. However, the convergence time of the pure coordination problem increases exponentially[4] with the number of actions, while in the (anti-)coordination game, time increases only linearly (cf. Figure 4.11a). This is a positive result, since in wireless sensor networks nodes need to both coordinate and anti-coordinate at the same time. Moreover, the coordination problem is much smaller than the anti-coordination problem. For a successful message forwarding, for example, a node needs to coordinate with only one partner, but anti-coordinate with possibly many neighbors. Thus, network convergence of the combined game increases at most linearly with the number of actions.

[4] Note that the y-axis is logarithmic.

4.7 Conclusions

Guided by research question **Q2**, the main aim of this chapter is to show that a simple approach like WSLpS is able to make agents in different configurations quickly self-organize with no history of past plays and based only on local interactions with limited feedback. In addition, agents dispersion games reach a global favorable outcome without additional communication overhead, such as communicating current states or exchanging local information about the strategies of neighbors. We showed that the same approach we presented for pure coordination in Chapter 3, namely Win-Stay Lose-probabilistic-Shift, can be applied in pure anti-coordination games and in games that involve characteristics of both coordination and anti-coordination. Moreover, our WSLpS approach imposes minimal system requirements and can be used by agents in any topology and for any number of actions. Our empirical results indicate that WSLpS performs at least comparable to other (and sometimes more complex) algorithms presented in literature on anti-coordination.

We saw that solutions in pure coordination games always exist, while dispersion games on certain topologies have no conflict-free solutions. Note that grid topologies with two actions have exactly two global solutions for both pure coordination games and pure anti-coordination games and therefore agents perform equally well in both game types. When pure anti-coordination with $k > 2$ actions *is* possible, the game has more solutions and agents can typically find a favorable outcome much faster than agents playing a coordination game with the same number of actions. Lastly, we saw that (anti-)coordination games, which involve equal amount of coordination and anti-coordination, have convergence time proportional to its two aspects. However, the convergence time of these (anti-)coordination games is much closer to that of pure anti-coordination, than to pure coordination, which increases exponentially in the number of actions. Thus we can conclude that (at least in grid topologies) the element of anti-coordination has a much stronger influence on the convergence time of the (anti-)coordination problem than has the influence of coordination. To put it bluntly, pure anti-coordination speeds up the convergence time of (anti-)coordination games much faster than pure coordination slows it down. Nevertheless, deeper analysis needs to be performed to understand this relationship in any topology for arbitrary number of agents and actions.

Chapter 5

(Anti-)Coordination in time: wireless sensor networks

Until now we analyzed the pure coordination and pure anti-coordination problems separately, as well as the combined problem of coordination and anti-coordination in abstract single-stage repeated games. Here we explore the challenging domain of wireless sensor networks (WSNs), where sensor nodes are involved in a repeated multi-stage (anti-)coordination game in time. We show how the (anti-)coordination games studied so far map to the WSN coordination problem, by addressing the following question:

Q3: *How can highly constrained sensor nodes organize their communication schedules in a decentralized manner in a wireless sensor network?*

Our simple decentralized Win-Stay Lose-probabilistic-Shift (WSLpS) approach, presented in Chapters 3 and 4 allows agents in different topologies to successfully achieve (anti-)coordination through only local interactions and with no communication overhead. Most importantly, WSLpS imposes minimal system requirements, allowing highly constrained agents to (anti-)coordinate with only limited environmental feedback. These characteristics of our approach allows us to apply it in the real-world domain of wireless sensor networks. Due to the decentralized nature of the WSN scenario and the limited information available to nodes, individual agents are unable to measure the global system performance and hence optimize their long-term

behavior. Nevertheless, we demonstrate how optimization of immediate payoffs can still result in near-optimal outcomes.

In this chapter we study how WSLpS can be used by computationally bounded sensor nodes to organize their communication schedules in an energy-efficient decentralized manner. We propose two adaptive communication protocols based on WSLpS and demonstrate the importance of (anti-)coordination in WSNs, as opposed to pure coordination and pure anti-coordination. We show how our approach outperforms a state-of-the-art communication protocol in terms of two typical performance measures — lifetime and latency.

5.1 Introduction

A wireless sensor network is a collection of small autonomous devices (also nodes, or agents), which gather environmental data with the help of sensors. These battery-powered nodes use radio communication to transmit their sensor measurements to a terminal node, called the sink. The sink is the access point of the observer (or user), who can process the distributed measurements and obtain useful information about the monitored environment.

Though the sink is vital to the operation of the whole network, it does not constitute a central controller, since it has no global knowledge, no knowledge of the internal states of nodes, such as remaining battery power, and cannot directly communicate with all nodes. Collecting such global and local information and controlling individual nodes comes at a high communication cost for the entire network. Although the sink does constitute a single point of failure, a fault can easily be detected by the user and fixed, as opposed to a failure in one of the nodes.

Nodes have small transmission range and therefore data packets (or messages) cannot be sent directly to the sink, but need to be forwarded by other nodes within range, which we call neighboring nodes (or neighbors). Thus sensor data travels through the network in the form of data packages, transmitted at discrete time intervals (or time slots). At each time slot agents interact by attempting to forward messages towards the sink. We see each interaction in a given time slot, as a single-stage multi-player (anti-)coordination game between neighboring nodes. However, unlike the (anti-)coordination games studied in the previous chapters, here each game is influenced by the games at the previous time intervals, due to the forwarding of messages. For example, if at time slot t a node $\mathbf{A}$ transmits a message to its neighbor $\mathbf{B}$, in the next time slot $t+1$ node $\mathbf{B}$ would have to forward that message to another node. Depending on whether the transmission between $\mathbf{A}$ and $\mathbf{B}$ is

successful at time t, node **B** would have to take different actions at time $t + 1$. Due to the relation between games at different time slots, we see the collection of these single-stage games as one multi-stage game, played through time. That multi-stage game starts with each node having one sensor measurement to send and ends when all measurements are delivered to the sink. Since measurements are made periodically, we say that the WSN game is a repeated multi-stage (anti-) coordination game in time.

Successful message forwarding requires both synchronization with the intended receiver as well as desynchronization with all other nodes in range. Here the term synchronization refers to coordination of activities in time, while desynchronization stands for anti-coordination in time. Similarly, (de)synchronization stands for (anti-) coordination in time. In Chapters 3 and 4 we used multi-channel communication as a real-world example that illustrates abstract coordination and anti-coordination games respectively. As multi-channel communication poses numerous additional challenges in the domain of WSNs, for simplicity in this chapter we assume single channel communication. In Phung *et al.* [2012] we extend our work to multi-channel coordination. Note that single channel communication is, at the time of this writing, still an active area of research and that many state-of-the-art communication protocols are still single-channel. In this chapter we focus on another challenging problem that illustrates the need for (de)synchronization, namely wake-up scheduling. Due to the decentralized nature of most WSN applications, agents need to (de)synchronize their communication schedules without the help of a central entity. We illustrate the problem of (de)synchronization in Example 12.

Example 12 (WSN (de)synchronization)**.** *Consider a number of wireless sensor nodes, arranged in an arbitrary topology. For a successful transmission between two nodes, the sender needs to put its radio in transmit mode, the intended receiver needs to listen to the channel, while all other nodes in range need to turn off their radios. In the absence of central control, how can all nodes in the wireless sensor network learn over time to (de)synchronize their activities, such that they successfully forward data to the sink?*

Sensor nodes have the common goal of (de)synchronizing their activities, but have no individual preferences, since all nodes belong to the same user. The aim of nodes is to learn the best action at each time slot, resulting in an energy-efficient behavior that allows them to successfully forward their data in a timely fashion. The paradigm for the designer of such a decentralized system is to apply a learning algorithm that allows sensor nodes to (de)synchronize their activities through only

local interactions and using incomplete knowledge. Moreover, due to the limited resources available to sensor nodes, the learning algorithm should impose minimal system requirements and communication overhead.

The rest of this chapter is organized as follows. In the next section we describe in more detail the challenging domain of wireless sensor networks. We provide an overview of related work in Section 5.3 and then describe the underlying (anti-) coordination problem in WSNs in Section 5.4. We present our experimental results in Sections 5.5 and 5.6 before we outline the conclusions of our work on WSNs in Section 5.7.

5.2 Wireless sensor networks

Given the current technological trend, wireless sensor networks are envisioned to be mass produced at low cost in the next decade, for applications in a wide variety of domains. These include, to name a few, ecology, industry, transportation, or defense. Large scale sensor network applications can be classified in two main categories — environmental monitoring applications and applications for event detection. A typical WSN monitoring scenario consists of a set of sensor nodes, scattered in an environment, which conduct sensor measurements (e.g. temperature, humidity, light conditions) and periodically report their data to the base station. Sensor networks for event detection, in contrast, continuously sense their environment for specific phenomenon (e.g. smoke, intruders, vehicle movement) and report to the sink only when such events occur. In this thesis we are interested in optimizing periodic behavior, and therefore we will focus more on monitoring applications.

For example, WSNs in habitat monitoring become increasingly significant, due to the disturbance effects that human presence introduces to animal populations and plants. The traditional personnel-rich approach, used by researchers in field studies, is usually more expensive and potentially dangerous (e.g. to dormant plants, breeding animals, or even to scientists themselves), as compared to the more economical and less invasive method of wireless sensor monitoring [Mainwaring *et al.*, 2002]. This remote observation is done by deploying a set of sensor nodes over the environment of interest and thus minimizing the human impact on animal populations and plants, by remotely monitoring their habitation.

The resources of the untethered sensor nodes are often strongly constrained, particularly in terms of energy and communication range. The base station usually possesses much larger resources, comparable to those of a standard laptop or desktop computer. The limited resources of the sensor nodes make the design of a WSN

application challenging. Application requirements, in terms of lifetime, latency, or data throughput, often conflict with the network capacity and energy resources. We first outline the **network model** of WSNs and then we report on the (anti-) coordination **challenges** in this domain.

5.2.1 Network model

Communication in WSNs is achieved by means of networking protocols, and in particular by means of the Medium Access Control (MAC) and the routing protocols [Akyildiz *et al.*, 2002; Yick *et al.*, 2008]. The MAC protocol is the data communication protocol concerned with sharing the wireless transmission medium among the network nodes. This protocol controls the radio of nodes and is responsible for the efficient node-to-node message delivery. The routing protocol, on the other hand, handles the end-to-end packet delivery. It allows to determine via which paths sensor nodes have to transmit their data so that messages eventually reach the sink. When being forwarded, messages are stored in the finite buffer (or queue) of nodes.

5.2.1.1 Communication and routing

Initially, nodes in WSNs were used to directly transmit (pre-processed) sensor measurements to a base station, located within all nodes' transmission range, which then compiles and further processes the measured data [Martinez *et al.*, 2004]. However, monitoring large environments requires the deployment of high number of devices over (ever increasing) regions, making it difficult to choose a location for the base station that will be in range with all nodes. Increasing the transmission range of nodes, in order to reach the base station, results in a higher interference and energy consumption and therefore decreases the overall lifetime of the sensor field.

To reduce these problems Zhao & Guibas [2004] proposed a multi-hop routing protocol that allows data packets to be forwarded by neighboring nodes to the sink, rather than directly transmitting the data to the end point. This solution reduces the requirement for the size of the transmission range and hence, the energy consumption[1], but leads to the necessity of coordination between neighboring nodes to ensure a viable transmission route. This communication method is called multi-hop routing. It allows for bigger sensor fields, where nodes fall outside the transmission range of the base station. Therefore, a direct centralized control over the network is not possible, so nodes have to organize their schedules and communication in a

[1] The simplest energy consumption model suggests that the energy, required for transmission, is proportional to the squared distance for this transmission.

decentralized fashion.

In WSNs the wireless medium is a shared resource. Most state-of-the-art wireless motes are equipped with an omnidirectional antenna that transmits data in all directions. Although directional antennas overcome some challenges of omnidirectional ones, they are typically more expensive and come with their own limitations. In this chapter we assume that data is sent in all directions and therefore nodes need to coordinate on using the shared resource. The *MAC protocol* handles packet transmission and must ensure the proper and efficient usage of that resource in the envisaged application. Two major types of MAC protocols have been proposed: contention based and scheduling based. In contention based protocols like Carrier Sense Multiple Access (CSMA) nodes can forward their data at any time, without following any particular schedule. In order to reduce the probability of a collision, nodes compete for the wireless medium typically with the help of additional control messages prior to the transmission of the actual data. Scheduling based MAC protocols, on the other hand, rely on a specific schedule of channel access for each node and therefore do not require contention-introduced control messages. In the Time Division Multiple Access (TDMA) protocol the signal is divided into frames, while each frame is further divided into time slots. This scheme allows nodes to reserve time slots for data transmission/reception such that multiple nodes can use different parts of the bandwidth of the same radio channel. A drawback of TDMA-like protocols is that they usually require clock synchronization[2], such that (neighboring) nodes maintain a similar notion of time. Recent work, however, reports on an adaptive MAC protocol that achieves sender-receiver time coordination without the need for tight clock synchronization between nodes [Borms *et al.*, 2010]. In the WSN applications that we consider, we assume a TDMA protocol, due to its natural advantage of energy conservation when exploiting the periodic behavior of nodes. The only control message used by this protocol is the ACKnowledgment packet, which is transmitted after a DATA packet is received successfully. Although it introduces communication overhead, the ACK packet is necessary for the proper and reliable forwarding of messages. As typically done in WSNs, we assume here that the frame length equals the period of data collection, while each slot is long enough to allow a single IEEE 802.15.4 maximum length DATA packet (of 128 bytes) to be transmitted and acknowledged with an ACK packet, resulting in a slot duration of around 5 milliseconds. Each sensor measurement takes one slot to be transmitted and acknowledged between two nodes. Due to the periodic nature of sensor measurements, we assume a constant (or static) traffic flow. For simplicity we assume also single-

[2] Not to be confused with the term synchronization as coordination in time.

channel communication, but in recent work we described how an approach similar to WSLpS can be used to achieve distributed contention-free access in a multi-channel setting [Phung *et al.*, 2012].

When the WSN is deployed, the *routing protocol* requires that the nodes determine a routing path to the sink [Al-Karaki & Kamal, 2004; Ilyas & Mahgoub, 2005]. This is achieved by letting nodes broadcast packets immediately after deployment in order to discover their neighbors. Nodes in communication range of the sink propagate this information to the rest of the network. During the propagation process, each node chooses a parent, i.e. a node to which the data will be forwarded in order to reach the sink. The choice of a parent can be done using different metrics. A typical multi-hop routing protocol is to rely on a shortest path tree with respect to the hop distance, i.e. the minimum number of nodes that will have to forward their packets [Couto *et al.*, 2005; Woo *et al.*, 2003]. The nodes determine the neighbor node which is the closest (in terms of hops) to the sink, and use it as the parent (or relaying node) for the multi-hop routing. This is the type of routing protocol we assume in our WSN application, due to its simplicity and low-overhead implementation. Note that the focus of this dissertation is more on the node-to-node coordination, rather than the end-to-end message delivery. This routing scheme organizes the traffic flow in the network as a static tree, with the sink being the root. Nodes on one routing branch need to synchronize their wake-up schedules with each other in order to increase the throughput, and at the same time desynchronize with nodes from neighboring routing branches, so that interference is minimized.

The drawback of static routing protocols, however, are that they are unable to perform well in harsh and dynamic environments, where nodes may move, fail, or new nodes may be introduced. For this reason one must rely on dynamic routing approaches [Boyan & Littman, 1994; Nowé *et al.*, 1998]. A good overview of adaptive routing algorithms is presented by Förster [2007]. In the presence of multiple (mobile) sinks, the routing protocol needs to efficiently coordinate the flow of data towards the different base stations. Förster & Murphy [2007] introduce FROMS: an adaptive energy efficient routing protocol, based on Q-learning, that disseminates data to multiple mobile sinks. In this dissertation, however, for the reasons explained above, we assume a single end station.

5.2.1.2 Modes of operation

Since wireless sensor nodes operate in most cases on finite energy resource, low-power operation is one of the crucial design requirements in sensor networks. The challenge of energy-efficient operation must be tackled on all levels of the network

stack, from hardware devices to protocols and applications. Although sensing and data processing may incur significant energy consumption, it is commonly admitted that most of the energy consumption is caused by the radio communication. A large amount of research has therefore been devoted in the recent years to the design of energy-efficient communication protocols [Akyildiz *et al.*, 2002; Ilyas & Mahgoub, 2005; Ye *et al.*, 2004; Yick *et al.*, 2008].

In our WSN model each sensor node operates according to a schedule that defines three different modes:

- a node goes in **transmit** mode when it starts to send a message through the channel. Although the message is addressed only to the parent, the omnidirectional antenna of the node broadcasts the message to all nodes in range, which we call neighbors (or neighboring nodes).
- when in **listen** mode, the sensor node is actively listening for broadcasts in the medium. When a signal is detected, the message is decoded and stored in the node's memory buffer (or queue) for later forwarding. Nodes discard a broadcasted message, not addressed to them.
- when a node is in **sleep** mode, its radio transceiver is switched off and therefore no communication is possible. Nevertheless, the node continues its sensing and processing tasks.

These three operation modes pose potential problems to the communication, because two nodes have to be synchronized with each other, prior to exchanging data. Two nodes are synchronized (or coordinated in time), when the sender is in transmit mode, while the receiver is in listen mode. Only then a successful transmission can take place, provided no collisions occur at the receiver's antenna. A collision happens when more than one signal arrives at the same time (and on the same channel) at the node's antenna. Thus nodes need also to desynchronize with their neighbors (or anti-coordinate in time), such that collisions do not occur during communication.

Figure 5.1 reports the radio characteristics of several representative and often used radio platforms. An important observation is that for these typical radios, the transmit and listen power are comparable, and that the sleep power is at least two orders of magnitude lower. Taking into account the energy consumption of the different modes, we can identify four major sources of energy waste:

- **idle listening** happens when a node is listening to the channel when no neighbor is transmitting a message. Since no messages are being sent, the node is better off sleeping, as it is orders of magnitude cheaper than listening.

Mote Year	Mica2Dot 2002	T-mote Sky 2005	Imote2 2007	Waspmote 2009	Lotus 2011
Radio	CC1000	CC2420		Xbee-802.15.4	RF231-802.15.4
Outdoor range	150 m	50/100m		500 m	100+ m
Data rate	76 Kbps	250 Kbps		250 Kbps	250 kbps
Sleep power	100 µW	60 µW		<30 µW	30 µW
Listening power	36 mW	63 mW		150 mW	48 mW
Transmit power	75 mW	57 mW		135 mW	51 mW

Figure 5.1: Typical wireless sensor hardware developed in the recent years, together with their main radio characteristics.

- **overhearing** occurs when a node receives a packed that is addressed to another node and not to itself. This event can happen due to the broadcasting nature of the antennas.
- **collision** is another event that happens at the receiver's radio, i.e., the node detects more than one signal at the same time and is unable to distinguish between them. In this case energy is wasted both at the sender's and at the receiver's side, because the message was not received properly and needs to be retransmitted.
- **control packet overhead** represents the energy loss due to the exchange of control packets prior to, during or after the transmission of the actual message. The frequency and size of the control packets should be kept low to minimize the effect of this problem.

Here energy waste refers to the energy spent on an action (e.g. transmit, or listen) that does not result in successful message delivery. In order to maximize energy efficiency of the network, the communication protocol should minimize the above sources of energy waste, while maximizing sleep mode and considering the latency requirements of the observer.

5.2.1.3 Wake-up scheduling

It is clear that in order to save energy, a node should turn off its radio (or go to sleep). However, when sleeping, the node is not able to send or receive any messages, therefore it increases the latency of the network, i.e., the time it takes for messages to reach the sink. High latency is undesirable in any real-time applications. On the other hand, a node does not need to listen to the channel when no messages are being sent, since it loses energy in vain. Therefore, the only way to significantly

reduce power consumption is to have the radio switched off most of the time, and to turn it on only if messages must be received or sent. This problem is referred to as *wake-up scheduling*. Analogously, a node's wake-up schedule contains the time slots for each of the node's three modes of operation, i.e. transmitting, listening and sleeping. Since measurements are taken periodically with the frame length being the period, nodes should repeat their wake-up schedule in each frame. We assume here that all nodes take environmental measurements at the beginning of each frame, such that at the first slot each node has one new message to forward to the sink.

Wake-up scheduling in wireless sensor networks is an active research domain, and a good survey on wake-up strategies in WSNs is presented by Schurgers [2007]. Three types of wake-up solutions can be identified, namely, on-demand paging, synchronous and asynchronous wake-up.

In on-demand paging, the wake-up functionality is managed by a separate radio device, which consumes much less power in the idle state than the main radio. The main radio therefore remains in a sleeping state, until the secondary radio device signals that a message is to be received on the radio channel. This idea was first proposed with the PicoRadio and PicoNode projects [Guo *et al.*, 2001] for extremely low power systems, and extended in Shih *et al.* [2002]; Agarwal *et al.* [2005] with hand-held devices. On-demand paging is the most flexible and energy-efficient solution, but adds non-negligible costs in the hardware design.

In synchronous wake-up approaches, nodes duty-cycle their radio (or alternate active and inactive modes) in a coordinated fashion. The duty cycle is the ratio of active time to sleep time within a frame. Several MAC protocols have been proposed, allowing nodes to wake up at predetermined periods in time at which communication between nodes becomes possible. A standard paper detailing this idea is that of S-MAC (Sensor-MAC) [Ye *et al.*, 2004]. The basic scheme is that nodes rely on a fixed duty cycle, specified by the user, where nodes periodically and simultaneously switch between the active and sleep states. S-MAC suffers from energy loss due to overhearing, since all nodes are awake at the same time. Several extensions to S-MAC have been proposed. In particular, van Dam & Langendoen [2003] proposed T-MAC, which aims at improving the energy efficiency by making the active period adaptive. This is achieved by making the active period very small, e.g., only the time necessary to receive a packet, and by increasing it at runtime if more packets have to be received. Another extension is D-MAC [Lu *et al.*, 2004], which staggers the wake-up cycles along the routing tree, so that nodes send data when their parent's radio is in the receive mode. The main concern with protocols based on synchronous wake-up is the overhead which can be caused by

maintaining the nodes synchronized. However, when the periodic message reporting is relatively frequent, the synchronization costs become negligible compared to the communication costs [Borms *et al.*, 2010].

Finally, in asynchronous wake-up solutions, the nodes are not aware of each other's schedules, and communication comes at an increased cost for either the sender or the receiver. In sender-based asynchronous wake-up, such as X-MAC [Buettner *et al.*, 2006], the sender continuously sends beacons until the receiver is awake. Once the receiver gets the beacon, it sends an acknowledgment to notify the sender that it is ready to receive a packet. This scheme is the basis for the low-power listening [Hill & Culler, 2002] and preamble sampling [El-Hoiydi, 2002] protocols. The receiver-based wake-up solution is the mirror image of sender-based, and was exposed in the Etiquette protocol [Goel, 2005]. Sender-based and receiver-based asynchronous protocols can achieve very low power consumption. Asynchronous wake-up solutions however require an overhead due to the signaling of wake-up events, which makes them inefficient when wake-up events are relatively frequent [Schurgers, 2007]. For this reason we focus more on synchronous TDMA approaches applied in monitoring applications with a relatively high message rate.

5.2.2 Design challenges

From the above described network model we see that the wireless network consists of highly constrained sensor nodes that need to coordinate their behavior in a decentralized manner in order to fulfill the requirements of the WSN application. Here we summarize some of the main challenges in the WSN domain, together with the design requirements for an efficient communication protocol:

- A message transmission by one node may cause **communication interference** in another, resulting in message loss. Therefore, the sender needs to coordinate its transmissions not only with the receiver but also with other nodes within range.
- There is **no central control**, as the sensor nodes are typically scattered over a vast area. There is no single unit that can monitor and coordinate the behavior of all nodes. As a result, nodes need to coordinate their transmissions in a decentralized manner.
- **Communication is expensive** in terms of battery consumption, since the radio transmitter consumes the most energy. For this reason agents cannot coordinate explicitly using (energy-expensive) control messages, such as a node

saying to all nodes in range *"I will transmit a message in* 5 *seconds, so everyone please stay silent"*.

- Due to the small transmission and sensing range, nodes have **only local information** and lack any global knowledge (e.g. of the network topology). Again, communicating such local information comes at a certain cost. Thus, nodes should be able to adapt their behavior based on local interactions alone.
- Nodes possess **limited memory and processing** capabilities and therefore cannot store large amounts of data, or reliably execute complex algorithms. The coordination behavior needs to be simple and have low memory requirements.
- Sensor nodes **cannot observe** directly the actions of others, but only the effect of their own actions. When a sensor node selects transmit and the message is not acknowledged by the recipient, the sender does not know if the receiver was itself transmitting, sleeping, or it was listening but encountered interference. Only after successful communication can the node infer the action of its communication partner.

In the next sections we will address the (de)synchronization problem of wireless nodes, as posed in **Q3**, taking into account the above mentioned challenges.

5.3 Related work

Coordination and cooperative behavior has recently been studied for digital organisms by Knoester & McKinley [2009], where it is demonstrated how populations of such organisms are able to evolve coordination algorithms based on biologically inspired models for synchronization while using minimal information about their environment. Synchronization in WSNs, based on the Reachback Firefly Algorithm, is more specifically applied to WSNs by Werner-Allen *et al.* [2005]. The purpose of the study is to investigate the realistic radio effects of synchronization in WSNs. Two complementary publications to the aforementioned work present the concept of desynchronization in WSNs as the logical opposite of synchronization [Degesys *et al.*, 2007; Patel *et al.*, 2007], where nodes perform their periodic tasks as far away in time as possible from all other nodes. Agents achieve that in a decentralized way by observing the firing messages of their neighbors and adjusting their phase accordingly, so that all firing messages are uniformly distributed in time.

The latter three works are based on the firefly-inspired mathematical model of pulse-coupled oscillators, introduced by Mirollo & Strogatz [1990]. In this seminal

paper the authors proved that, using a simple oscillator adjustment function, any number of pulse-coupled oscillators would always converge to produce global synchronization irrespective of the initial state. More recently, Lucarelli & Wang [2004] applied this concept in the field of WSNs by demonstrating that it also holds for multi-hop topologies.

The underlying assumption in all of the above work on coordination in time is that agents can observe each other's actions (e.g. firing frequencies) and thus adapt their own policy (e.g. own firing phase), such that the system is driven to either pure synchronization or pure desynchronization, respectively. However, as we mention in Section 5.2.2 in WSNs agents cannot observe the actions of others. For example a sensor node could be in sleep mode while its neighbor wakes up, thus the sleeping node is unable to detect this event and adjust its wake-up schedule accordingly. Moreover, achieving either global synchronization (e.g. all nodes wake up at the same time) *or* global desynchronization (e.g. one one node awake at a time) alone in most WSNs can be impractical or even detrimental to the system. In Section 5.4 we will present how we tackle these challenges using different decentralized reinforcement learning approaches.

Paruchuri *et al.* [2004] propose a randomized algorithm for asynchronous wake-up scheduling that relies on densely deployed sensor nodes with means of localization. It requires additional data to be piggybacked to messages in order to allow for making local decisions, based on other nodes. This bookkeeping of neighbors' schedules, however, introduces larger memory requirements and imposes significant communication overhead. A different asynchronous protocol for generating wake-up schedules [Zheng *et al.*, 2003] is formulated as a block design problem with derived theoretical bounds. The authors derive theoretical bounds under different communication models and propose a neighbor discovery and schedule bookkeeping protocol operating on the optimal wake-up schedule derived. However, both protocols rely on localization means and incur communication overhead by embedding algorithm-specific data into packets. Adding such data to small packets will decrease both the throughput and the lifetime of the network. A related approach that applies reinforcement learning in WSNs is presented by Liu & Elhanany [2006]. As in the former two protocols, this approach requires nodes to include additional data in the packet header in order to measure the incoming traffic load.

A related methodology is the collective intelligence framework of Wolpert & Tumer [2008]. It studies how to design large multi-agent systems, where selfish agents learn to optimize a private utility function, so that the performance of a global utility is increased. In previous work [Mihaylov *et al.*, 2008] we investigated how this

framework can be applied in WSNs to overcome the challenge of decentralized (anti-) coordination. This framework, however, requires agents to store and propagate additional information, such as neighborhood's efficiency, in order to compute the world utility, to which they compare their own performance. The approach therefore causes a communication overhead, which is detrimental to the network lifetime.

5.4 (Anti-)coordination in wireless sensor networks

The design objectives of individual nodes are to forward their sensor measurements towards the sink in a timely fashion. As stated earlier, successful communication between two nodes requires good coordination with all nodes in range. When a node needs to *transmit* a message at a given time, the intended receiver must *listen* for messages. We refer to this type of coordination in time between a sender and a receiver as **synchronization**. The two nodes perform the same "meta" action at the same time, i.e. forward a message towards the sink. Throughout this chapter we use the term *coalition* to refer to a pair of agents that are synchronized at a given time slot, i.e. one agent selects transmit while the other listens. In addition to sender-receiver synchronization, no neighbors can forward a message at the same time, because their message will interfere with the transmission between the two communicating nodes. Therefore, the neighbors should *sleep* instead. This type of anti-coordination in time between the communicators and neighbors we call **desynchronization**, since the two groups cannot perform the same action at the same time, i.e. they cannot forward a message when another message is being forwarded. They need to desynchronize their activities in time, so that transmissions do not occur simultaneously in close proximity. Thus, the wake-up schedules of nodes require **(de)synchronization**, or (anti-)coordination in time, so that nodes follow their design objectives in an energy-efficient manner.

Coordinating the actions of agents can successfully be done using the reinforcement learning (RL) framework by rewarding successful interactions (e.g., transmission of a message) and penalizing the ones with a negative outcome (e.g., overhearing or packet collisions) [Mihaylov *et al.*, 2012a; 2011a; 2011b]. This behavior drives the nodes to repeat actions that result in positive feedback more often and to decrease the probability of unsuccessful interactions. In literature, RL techniques are typically applied to optimize the long-term performance of agents, as opposed to the immediate short-term reward [Sutton & Barto, 1998]. A far-sighted agent selects actions so as to maximize the sum of the possibly discounted future rewards, where an initial sequence of actions may result in low rewards, but obtain a very high reward

later on. However, in a message forwarding task in WSNs, agents cannot explicitly condition their actions on what other agents further in the network will do, since state information of others is not available. Moreover, sensor nodes should forward their messages as soon as possible, so as to maximize throughput (or minimize latency). Myopic agents are, therefore, well suited in a WSN scenario. Even though maximizing immediate rewards may in some cases result in sub-optimal routing, optimality is rarely sought in industrial applications where near-optimal solutions are well accepted. Far-sightedness in WSNs cannot be implemented on a global scale, since individual agents are unable to measure the performance of the whole system in order to optimize their long-term behavior. To achieve the latter, state information needs to be shared between agents, resulting not only in increased computational complexity, but also in higher communication overhead. Related work has studied the optimization of the long-term system performance in Markov games where agents exchange information in order to propagate the reward signals [Vrancx, 2010]. Another author has explored the long-term learning behavior of agents in a grid-world coordination task, where only agents in close proximity share state information, due to the implied costs of communication [De Hauwere, 2011]. In our WSN scenario, however, we attempt to address the decentralized coordination challenge of highly-constrained sensor nodes imposing minimal system requirements and communicational overhead. We therefore require that agents achieve successful (anti-) coordination without sharing any state information. Still, agents learn near-optimal wake-up schedules by maximizing immediate rewards, based only on the immediate outcome of their own actions. In this way the scheduling of the sensor nodes' behavior emerges from simple and local interactions without the need of central mediator or any form of explicit coordination.

As we outlined in Section 4.6, in WSNs nodes are usually aware of their hop distance to the base station and therefore can distinguish between nodes with the same hop distance (horizontal neighbors) and nodes on a higher or a lower hop (vertical neighbors). Depending on the routing protocol, coalitions (i.e. synchronized pairs of nodes) logically emerge across the different hops. Note that no explicit notion of coalition is necessary. Rather, these coalitions emerge from the global objective of the system, and agents learn by themselves with whom they have to (de)synchronize (e.g. to maximize throughput). As defined by the routing protocol, messages are not sent between nodes from the same hop, hence these nodes should desynchronize (or belong to separate coalitions) to avoid communication interference. If the routing would allow for message forwarding between neighbors on the same hop, coalitions could form "horizontally" as well.

Since sensor networks typically cover vast areas, precise node positioning is a tedious task. Sensor nodes are usually scattered randomly over the monitored field, and although we assume the network topology remains static after deployment, the precise configuration cannot be known in advance. Moreover, the transmission power of nodes, as well as the channel quality, influence the network connectivity. For these reasons the network designer cannot anticipate the network topology in advance in order to completely determine the wake-up schedules of nodes using off-line learning methods. Once the sensor network is deployed, on-line learning methods can help nodes adapt their schedules to the resulting topology. On-line adaptation, on the other hand, typically relies on trial-and-error methods, which are costly in the WSN domain. Therefore the network designer can use a combination of these techniques to improve the efficiency of the system. He can apply off-line learning methods to pre-configure the the nodes' schedules and then use planning techniques, such as Dyna-Q [Sutton & Barto, 1998], in order to speed up the on-line learning process. In addition, transfer learning techniques [Taylor, 2009] can be applied when a new node is added to the network. Neighboring nodes can transfer their their learned schedules, such that the new node can learn more quickly an efficient schedule. Similarly, transfer learning can help the user to change the purpose of its network and let nodes quickly adapt to the new task, for example from environmental monitoring application to intrusion detection.

We mentioned in Section 5.1 that agent interaction in WSNs can be seen as a sequence of repeated singe-stage (anti-)coordination games that are related in time and therefore comprise one multi-stage game. To illustrate this concept, we investigate the (de)synchronization problem in WSNs from two perspectives: per-slot learning, which studies the outcome of learning in individual slots (or stages) independently, and real-time learning, in which agents sequentially learn in each slot of the multi-stage game. We propose several techniques to coordinate the communication of nodes in a decentralized and self-organizing way. Nodes attempt to (anti-) coordinate their transmissions and learn a wake-up schedule based on their position in the network. For example, leaf nodes should learn to only transmit and sleep, since no messages are being sent to them, while nodes close to the sink need to forward more messages and therefore have to transmit and listen more. In our studies we use synchronous action updates where agents simultaneously update their actions at the end of each slot.

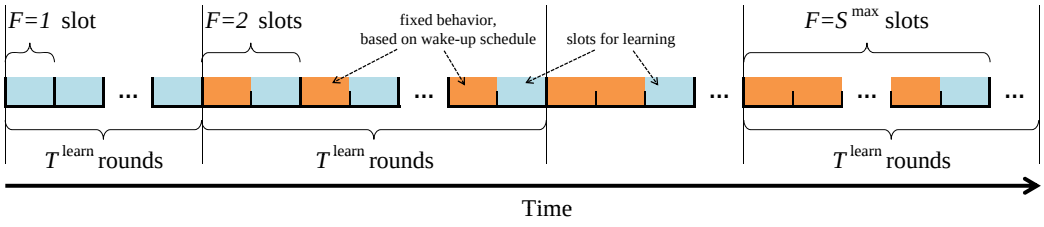
Recall that the frame captures the periodic behavior of agents, where at the beginning of each frame nodes generate a sensor measurement that has to be forwarded towards the sink. In **per-slot learning** nodes attempt to learn an energy-efficient

behavior one slot at a time. The game in each slot resembles a single-stage repeated (anti-)coordination game, similar to the games studied in the previous chapters. The frame length F is initially set to contain only one slot ($F = 1$) and repeated for T^{learn} rounds, determined by the user of the system. During that time the agent should learn which action to select, based on the actions of its neighbors, by using its learning approach (described below). The number of rounds should be high enough to allow agents to successfully (de)synchronize with others in that slot. After the T^{learn} rounds, agents store their learned action in their wake-up schedule and the frame length is then increased by one slot. During the next T^{learn} rounds, in the first $F-1$ slots of the frame each agent selects its actions according to its learned schedule, while in the last slot it again attempts to learn an energy-efficient action. This process is repeated until the frame contains $S^{\max}$ slots, which is the final length of the data collection round (or period) of the system. Thus, in the first $T^{\text{learn}} \cdot \sum_{n=1}^{S^{\max}} S_n$ slots after the deployment of the WSN, nodes attempt to learn an energy-efficient wake-up schedule at each slot separately, while forwarding messages. Thereafter, the learned schedules of nodes remain unchanged due to the periodic traffic flow. In this way, in every last slot of every frame agents are playing a single-stage (anti-) coordination game with their neighbors, repeated for T^{learn} rounds. The actions learned in that game determine the (anti-)coordination game in the next added slot. One can notice that these (anti-)coordination games have certain characteristics of Graphical Games [Vickrey & Koller, 2002], where agents have to independently decide on an action and receive payoff based on the actions of their neighbors in the network. Thus during learning our agents are engaged in sequential repeated graphical games, where each game is related to the preceding one, as a result of the traffic flow through the network. In other words, the game in each new slot is influenced by the actions of agents in the previous slots. The aim is to study how (anti-)coordination can emerge in a decentralized manner through local interactions and limited feedback. Moreover, we see how independently learning in a sequence of (anti-)coordination games can result in an overall efficient schedule even though agents are only aware of a single game at a time and do not take into account the relation between the different games (or slots). In addition, it is not obvious what information from previous games to use and how to integrate it when learning in the new slot.

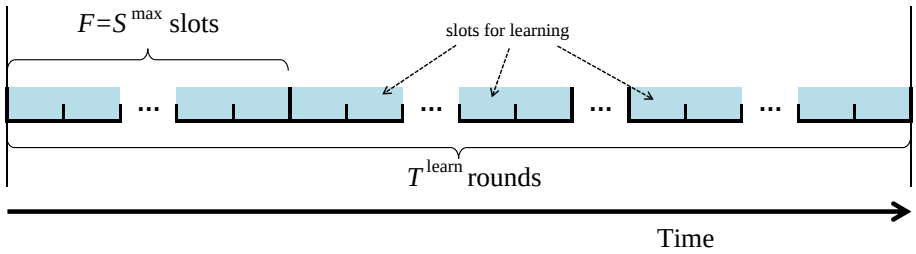
In **real-time learning**, in contrast, we study the real-world problem of WSN coordination as one multi-stage (anti-)coordination game. From deployment on, frames contain $F = S^{\max}$ slots, where measurements are generated at the beginning of each frame, and agents continuously learn for T^{learn} rounds by forwarding messages

towards the sink. In that perspective each node adapts its wake-up schedule on-line to the periodic traffic flow in the network, influenced by the behavior of others. The aim of nodes is to learn within T^{learn} rounds a good action at each slot within the frame.

In both perspectives the learning is done on-line in the sense that nodes adapt their behavior as messages are being forwarded towards the sink. In the per-slot perspective, agents sequentially learn an efficient action at each slot, while in real-time learning, agents adapt their behavior for each slot in parallel. Figure 5.2 illustrates the differences between the two learning perspectives we consider. Nodes take care of the on-line learning performance by constantly forwarding messages in the direction of the sink and thus the observer can obtain useful measurements during the learning phase. In Section 5.5.1.3 we measure the on-line performance.



(a) Per-slot learning.



(b) Real-time learning.

Figure 5.2: The two studied learning perspectives.

5.4.1 Per-slot learning perspective

We use this perspective to study how the (anti-)coordination problem we explored in the previous chapter maps to the WSN domain. Indeed, our WSN setting bares resemblance to the (anti-)coordination game in Section 4.6.1. Agents are arranged in a grid topology with the base station at the bottom. The shortest-hop routing protocol requires messages to be forwarded only vertically towards the sink.

At the beginning of each frame, only nodes with an empty queue generate a sensor measurement. Thus, nodes closer to the sink would take fewer measurements

during learning. Each game is defined by the number of messages in the queue of each agent. For example, in the first T^{learn} rounds after deployment, (i.e. the first repeated game, where $F = 1$) each node has at least one message. Agents attempt to (de)synchronize in that slot, such that when a node transmits a message, its lower hop neighbor listens, while other nodes in range stay silent. After T^{learn} rounds, the frame is extended to 2 slots. In the first slot agents perform their learned action from last game, while in the second slot they apply their learning approach in order to (anti-)coordinate. This frame, containing 2 slots, is repeated for T^{learn} rounds, followed by T^{learn} frames of 3 slots and so on, until the frames contain $S^{\max}$ slots and agents have learned an action at each slot (cf. Figure 5.2a).

To study this repeated (anti-)coordination problem, we apply once again our Win-Stay Lose-probabilistic-Shift approach, described in Section 4.4.1. Each node has three modes of operation, as outlined in Section 5.2.1.2 — *transmit*, *listen* and *sleep*. According to the WSLpS approach, nodes keep successful actions and shift with a certain probability if the action is not successful. Due to the constraints of wireless communication, the payoff for action a is binary — success ($p_i(a) = 1$) or failure ($p_i(a) = 0$), and is determined by the actions of other agents in the system, as outlined in Table 5.1. Maximizing the throughput requires both proper transmission as well as proper reception. Therefore, we treat the two positive rewards equally. Furthermore, most radio chips require nearly the same energy for sending, receiving (or overhearing) and (idle) listening [Langendoen, 2008], making the last three rewards equal. We consider these five events to be the most energy expensive or latency crucial in wireless communication. Although a payoff of 0 means the action performed resulted in failure, the payoff alone does not provide enough information on what the best action might be. Conversely, a payoff of 1 indicates success for the node (and its partner), but no information is given on the impact of the action on other neighbors.

Note that the *transmit* action is only possible if the node has a message to send. Moreover, while *sleep*ing, the agent cannot receive any feedback from the environment, since its radio is switched off. The agent is not aware whether its action is successful or not, unless additional control messages from neighbors are sent, which in turn introduces communication overhead. Therefore, during learning in every last slot, the agent will never select the *sleep* action, but will only *transmit* (if possible) and *listen*.

Each agent i will select action *transmit* with probability $\pi_i(\textit{transmit})$, depending on the previously selected action at that slot, its payoff p_i and the number of

action	**outcome**	**payoff**
transmit	ACK received	1
	no ACK received	0
listen	DATA received	1
	communication overheard	0
	nothing received	0
	several messages collided	0

Table 5.1: Payoffs depending on the outcome of the selected action.

messages m_i in the queue of agent i:

$$\pi_i(\textit{transmit}) \leftarrow \begin{cases} 1, & \text{if } \textit{transmit} \text{ AND } p_i = 1 \text{ AND } m_i > 0 \\ \alpha, & \text{if } p_i = 0 \text{ AND } m_i > 0 \\ 0, & \text{if } \textit{listen} \text{ AND } p_i = 1 \text{ OR } m_i = 0 \end{cases} \tag{5.1}$$

where $\alpha \in (0, 1)$ is the probabilistic component of WSLpS. A large transmit probability can ensure faster transmission, while a small α will decrease the chance of collisions. Note that α behaves both as shift probability (when listening was not successful) as well as keep probability (when transmission was not successful). The reason for this "duality" is because nodes experience different games, according to their role in the forwarding of messages. This concept will be further elaborated in Section 5.5. Lastly, agent i will select *listen* with probability $\pi_i(\textit{listen}) = 1 - \pi_i(\textit{transmit})$.

After T^{learn} rounds, agents store their best (i.e. current) action for the last slot of the frame in their wake-up schedule, while agents without a "winning" action (i.e. if $p_i = 0$) will use the *sleep* action for that slot instead. One game is played for each slot of the frame, so that agents learn which actions to apply as part of their periodic wake-up schedule. Gradually the frame becomes long enough to ensure that all generated sensor measurements at the beginning of the frame have enough time to be forwarded to the base station, before the new measurements are taken at the beginning of the next frame.

In real-world WSNs, even if two nodes are synchronized for communication it can happen that messages are occasionally dropped due to poor channel quality. In order for WSLpS to operate in such settings, where the channel quality varies through time, agents need memory in order to distinguish between occasional packet drops and click drifts or depleted neighbors. We study WSLpS both in perfect channel conditions, as well as in noisy environments, and report the results in Section 5.5.

5.4.2 Real-time learning perspective

In the previous perspective we explore how efficient behavior can emerge when agents play individual single-stage repeated games, similar to those studied in the literature on coordination and anti-coordination. In the real-time perspective, in contrast, we study how agents behave in the WSN scenario where they are involved in one continuous multi-stage (anti-)coordination game. While in per-slot learning agents learn in each slot independently of the next, in real-time learning agents adapt their actions sequentially for each slot of the frame. The frame is set to $F = S^{\max}$ slots from the beginning and agents learn in each slot for T^{learn} rounds (or frames) (cf. Figure 5.2b). Recall that frames capture the periodic behavior of nodes. As messages are being periodically forwarded towards the sink, nodes use a learning approach to adapt their actions to the traffic flow in the network, such that with time, each node learns an energy-efficient wake-up schedule — one action for each slot. To achieve the latter we propose DESYDE — DEcentralized SYnchornization and DEsynchronization communication protocol (or learning algorithm).

DESYDE is a real-time version of our WSLpS approach from Section 5.4.1. During a short learning phase, fixed by the user, agents always stay awake in order to learn the quality of their actions. Thus, at each slot during the learning phase each agent selects one of the two available actions — *transmit* and *listen*. Upon executing action a, each agent i receives a binary payoff $p_i(s, a)$ from the environment based on the outcome that occurred, as shown in Table 5.1. Note that each agent can select only one action during a slot and that agents select their action synchronously.

We use $Q_i(s, a)$ to indicate the expected reward (or "quality") of agent i taking action a at slot s. At first, this value is initialized to 0 for $Q_i(s, \textit{transmit})$ and 1 for $Q_i(s, \textit{listen})$. Upon executing action a at slot s, agent i updates its action quality, based on the payoff it receives: $Q_i(s, a) \leftarrow p_i(s, a)$. In this way $Q_i(s, a)$ represents the latest payoff obtained at slot s for action a. This update scheme allows agents to quickly find a good wake-up schedule, without necessarily looking for the optimal solution, since learning in WSNs is costly. A sub-optimal solution, on the other hand, might also be costly in terms of latency, but as we will see in Section 5.6 the loss in latency is negligible.

The probabilistic component of DESYDE accounts for action exploration, and is expressed by the channel contention window. Since collisions constitute the biggest obstacle in the pursuit of low latency, typically MAC protocols employ a backoff timer T that instructs the node when to transmit a packet. During learning, when a node receives a message or obtains a measurement, it will generate a uniformly random number t where $1 \leq t \leq T_{max}$ represents the number of slots after which

the node will attempt to transmit the packet. The node sets $T = t$ and decrements it every slot, such that when $T = 0$, the node will send its message. Thus the parameter of DESYDE is the value $T_{max} \in [2, S^{\max}]$, which is the maximum number of slots for the backoff timer. Note that a window of at least 2 slots is necessary to resolve collisions. A too low value of T_{max} will result in more frequent collisions, while a too high value will increase the latency of the system. During the learning phase each agent i will select action a at slot s in the following way:

$$a \leftarrow \begin{cases} \textit{transmit}, & \text{if } Q_i(s, \textit{transmit}) = 1 \text{ OR } T = 0 \\ \textit{listen}, & \text{if } Q_i(s, \textit{transmit}) = 0 \end{cases} \tag{5.2}$$

This behavior resembles a win-stay lose-shift strategy (cf. Section 2.3.3), where agents repeat successful actions and avoid unsuccessful ones. In particular, at slot s agent i will repeat action a only if it had a positive outcome at slot s in the previous frame. Recall that frames capture the periodic behavior of nodes. However, using the probabilistic retransmission model, the strategy resembles more our WSLpS approach. Thus, in every frame the agent repeats those actions that had positive outcome in the previous frame, or probabilistically attempts to transmit. For example, if $Q_i(s, \textit{transmit}) = 1$ for slot s, agent i will choose to transmit a packet during slot s in the next frame (provided that it has a packet in its queue). As a result, a payoff $p_i(s, \textit{transmit})$ will be generated and stored in $Q_i(s, \textit{transmit})$. If the transmission at slot s in the following frame was acknowledged, $Q_i(s, \textit{transmit})$ will remain 1 and the agent will repeat the same action next frame at slot s. Otherwise, it will choose *listen*. In the same way the agent will select an action in every slot within each frame.

Note that $Q(s, \textit{listen})$ does not influence the choice of action during the learning phase. This is because during learning agents always stay awake and thus select *listen* even if both actions are unsuccessful. Only after the learning phase, the agent selects *sleep* for all slots where neither listening nor transmitting is successful. Thus after learning, the action a that agent i will select in slot s of the frame is:

$$a \leftarrow \begin{cases} \textit{transmit}, & \text{if } Q(s, \textit{transmit}) = 1 \\ \textit{listen}, & \text{if } Q(s, \textit{listen}) = 1 \\ \textit{sleep}, & \text{if } Q(s, \textit{transmit}) = Q(s, \textit{listen}) = 0 \end{cases} \tag{5.3}$$

Every agent learns a periodical wake-up schedule based on the outcome of its actions. We therefore say that no explicit form of agent coordination is necessary to achieve equilibrium. Instead, coordination "emerges" as a result of packet forwarding and reasoning based on local interactions. We note that DESYDE makes use of the ACKnowledgment control packet to determine the payoff of *transmit*. However,

as we mentioned in Section 5.2.1.1, this packet is necessary for the proper and reliable forwarding of messages and it is not introduced by the learning algorithm. Moreover, DESYDE does not embed any additional information in packets, as do other protocols, such as RL-MAC [Liu & Elhanany, 2006].

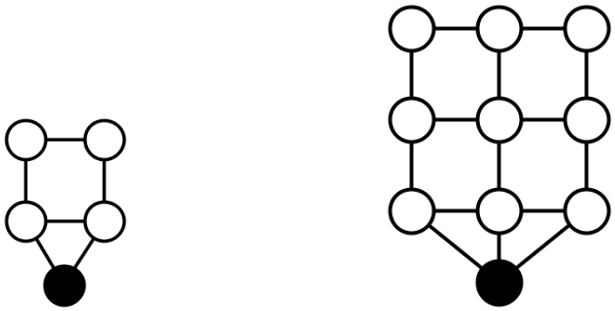
One drawback of DESYDE is that it requires nodes to stay awake during the learning phase, which is set by the user. Due to the decentralized nature of the (anti-)coordination problem, agents cannot determine when they have successfully reached (de)synchronization. In the WSN domain, staying awake in order to learn an efficient wake-up schedule is costly in terms of battery consumption. Although the sufficient learning time depends on the network size, we determined empirically that this learning duration is negligible compared to the lifetime of the network (cf Section 5.6).

5.5 Results from per-slot learning

The aim of the per-slot learning is to explore the WSN setting from pure game-theoretic perspective where agents are involved in sequential graphical games. Each graphical game is modeled as a separate repeated (anti-)coordination game where sensor nodes attempt to (de)synchronize their wake-up schedules at the last slot of each frame, without modeling the underlying relation between slots (i.e. games). We show that even without an explicit modeling of such a relation, near-optimal wake-up schedules can emerge from decentralized interactions.

The WSN sequential graphical games are best studied in grid topologies. They allow the routing algorithm to organize the network in a tree structure where nodes in one routing branch of the tree need to coordinate in time, while at the same time anti-coordinate from neighboring routing branches. Moreover, grid structures allow for a more straightforward analysis of the solutions, compared to random or small-world topologies. We apply WSLpS on nodes in 2-by-2 and 3-by-3 grid topologies where the sink is placed at the bottom of the network, as shown in Figures 5.3a and 5.3b.

Agents learn in each last slot of the frame for T^{learn} rounds, specified by the user. This learning is repeated as the frame is gradually expanded up to $S^{\max}$ slots. The overall performance indication of the learning outcome at the end of the learning phase (i.e. when $F = S^{\max}$) is the number of slots necessary for all messages to be forwarded to the base station according to the wake-up schedule of each node. This measure is known as the latency of the system and it also defines the minimum period for the data collection round. Our measure of latency is the slot in which



(a) 2-by-2 grid topology. **(b)** 3-by-3 grid topology.

Figure 5.3: Studied topologies.

the last message in the network is transmitted to the sink, according to the learned wake-up schedule of each node.

Another important performance measure is the lifetime of the network, which is defined as the duration between the deployment of the network and the first time any node runs out of battery. Due to the properties of our learning algorithms, after the learning phase, nodes will only become active if they can successfully forward a message and sleep otherwise. As a result, each node learns the minimal duty cycle that will allow the node to forward all (received) messages. For this reason we do not explore the lifetime performance indicator. Note that for simplicity we ignore here the realistic radio effects, such as the energy used for changing between modes.

5.5.1 Evaluation

As mentioned in Section 5.4.1 the parameter α of the algorithm defines the probability with which the node will select *transmit* if its last selected action was unsuccessful and the node has a message to send. The system parameter is T^{learn} and defines the number of rounds during which nodes will attempt to (anti-)coordinate in the last slot of the frame. The action selected at the last round in each slot is the one that will be stored in the respective slot of the node's wake-up schedule.

We apply WSLpS in two grid topologies — one contains 4 nodes, arranged in a 2-by-2 grid, and the other is a 3-by-3 grid with 9 nodes. We study the transmit probability α of our approach and report the resulting latency of the network in the number of slots necessary to forward all generated messages within the frame. Each reported value is averaged over 1000 runs in MATLAB, which were sufficient to obtain statistically significant results, as determined using a Mann-Whitney U-test with $\alpha = 0.05$. We set here the number of learning rounds per slot to $T^{\text{learn}} = 200$ and the final data collection period to $S^{\max} = 20$. These values result in a learning

duration of $T^{\text{learn}} \cdot \sum_{n=1}^{S^{\max}} S_n = 200 \cdot 210 = 42000$ slots. With a typical slot length of 5ms (according to the IEEE 802.15.4 standard), the length of the learning duration is thus around 3.5 minutes, which is negligible compared to the lifetime of the system.

Next, we measure the **latency** of the system and show the learned **schedules** of nodes. We demonstrate the **on-line learning performance** investigate how WSLpS handles real-world phenomena, such as **clock drift**, **rerouting** and **packet loss**.

5.5.1.1 Latency

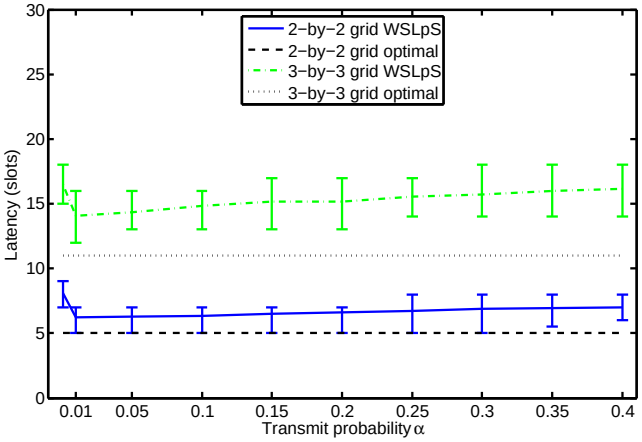
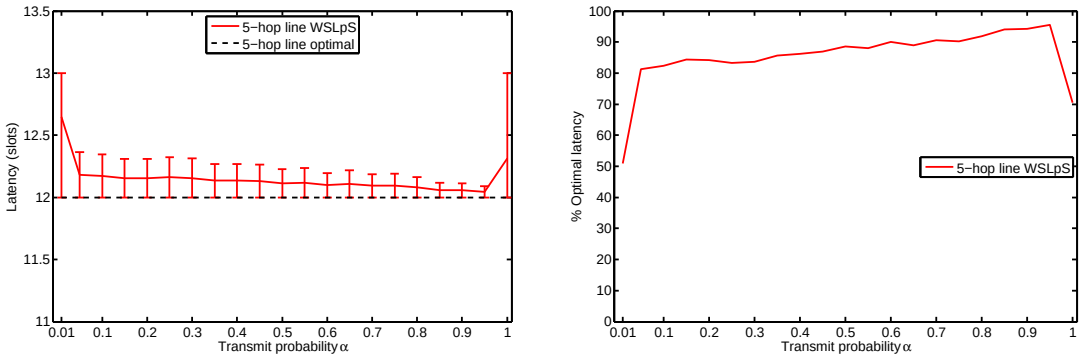


Figure 5.4: Latency of the WSLpS approach in grid topologies for different values of the transmit probability α. Lower and upper error bars show the 25^{th} and 75^{th} percentile, respectively.

Figure 5.4 shows the latency of the 2-by-2 and 3-by-3 grids, together with the optimal latency computed for the corresponding grid. We see that the resulting latency is close to optimal and that both networks achieve the lowest latency with a small transmit probability. The reason for obtaining good results with small α is twofold. According to table 5.1 listening nodes receive a payoff of 0 when they overhear a packet, or detect several packets in the channel, while the payoff is 1 only if exactly one higher hop neighbor within range is transmitting. Thus, more frequent transmissions increase the chance that listening nodes overhear or detect a collision and therefore not reply with an ACK packet to any sending node. Secondly, higher transmit probability drives more nodes to send measurements, which results in fewer listening nodes and therefore more failed transmissions. A low transmit probability, on the other hand, may result in idle listening (and thus a payoff of

0) for listening nodes, but since the chance of shifting to transmit is also low, any sending nodes have a higher probability of having a listening lower hop neighbor.



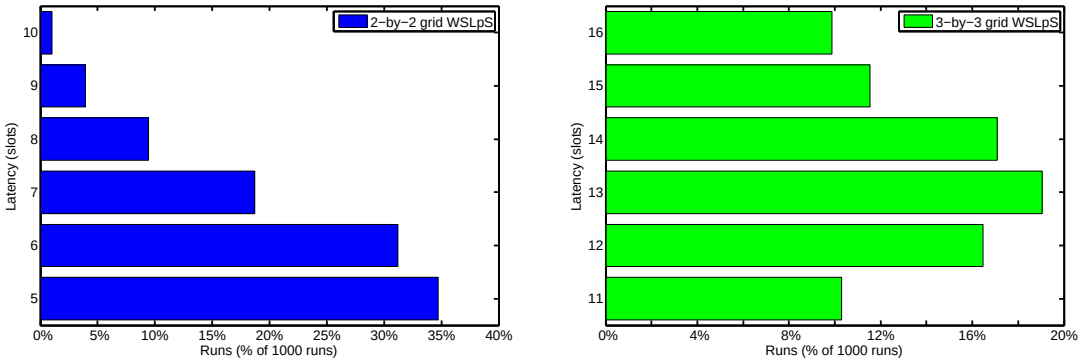
(a) Latency for different values of α. Lower and upper error bars show the 25^{th} and 75^{th} percentile, respectively.

(b) Percentage of runs resulting in the optimal latency for different values of α.

Figure 5.5: Results from WSLpS in a 5-node line topology.

Thus in grid topologies the transmission probability needs to be low, so that nodes avoid overhearing of horizontal neighbors. To illustrate this result we applied WSLpS in a 5-node topology, where agents are arranged in a line, such that nodes have no horizontal neighbors. We see in Figures 5.5a and 5.5b that the higher the transmission probability, the closer the average latency is to the optimal one. Nevertheless, a transmission probability of 1 increases the latency, since nodes will always try to transmit if their previous action failed and thus only nodes with an empty queue will listen.

Back to the grid topologies, we group in Figures 5.6a and 5.6b the runs according to their resulting latency in each topology. We remind the reader that agents optimize only their immediate payoffs and do not consider any long-term goals, due to the constraints in wireless communication and the lack of informative feedback. Still, we see that in the 4-node grid (Figure 5.6a), one third of the runs have the optimal latency, while the runs in the 9-node network typically finish only 2 slots later than the optimal. The latency in both topologies can be improved by increasing the number of rounds T^{learn} spent to learn in each slot. Of course, the larger the number of rounds, the more costly the learning phase. Depending on the envisioned application and requirements, the user can set T^{learn} in accordance with the desired network performance.



(a) 2-by-2 grid topology.

(b) 3-by-3 grid topology.

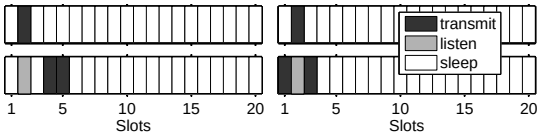
Figure 5.6: Percentage of runs resulting in the corresponding latency for WSLpS with $\alpha = 0.01$.

5.5.1.2 Schedules

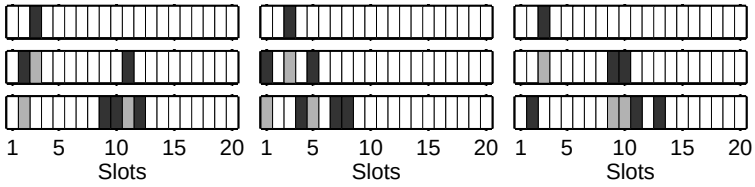
For each grid topology we present in Figure 5.7 an example of the learned wake-up schedules from a typical run of the WSLpS approach with transmit probability $\alpha = 0.01$. In Figure 5.7a we see the resulting schedules of nodes in the 2-by-2 grid topology (with resulting latency of 5 slots), while in Figure 5.7b the schedules of the 3-by-3 topology are displayed (with resulting latency of 13 slots). We can observe in both examples the following outcomes:

- Transmissions (black slots) are always synchronized vertically with listening (gray slots) or with the sink.
- Slots for listening are always synchronized with vertical transmission slots and desynchronized with horizontal ones.
- Nodes transmit as many times as they receive, plus an additional transmission for their own message.
- Each node is active exactly for the slots necessary to forward all messages.

We see that leaf nodes are active for 1 slot, since each node generates one message in a frame. Nodes on the next hop are active for one listening slot (to receive the message of the upper neighbor) and two transmission slots (to send their own and their neighbor's message), and so on. Here we assume that neighboring nodes are able to transmit at the same time, as long as they are outside the range of each other's communication partners. In reality, due to the radio effect known as "fading", the interference range of a transmission is larger than the actual communication range



(a) Learned schedules in 2-by-2 grid topology. The schedules result in latency of 5 slots, which is the optimal one for this topology.



(b) Learned schedules in 3-by-3 grid topology. The schedules result in latency of 13 slots, which is 2 slots more than the optimal one for this topology.

Figure 5.7: Examples of learned wake-up schedules from a typical run in each topology using WSLpS with $\alpha = 0.01$. Schedules are arranged according to node position in the corresponding network (see Figures 5.3a and 5.3b). The sink is at the bottom and is always listening.

and therefore neighboring nodes should not transmit at the same time to avoid interference. In theory, WSLpS should be able to handle such realistic effects, since neighboring transmitting nodes will not receive an ACK from their partners, due to the interference from fading, and thus learn to desynchronize transmissions between neighbors.

5.5.1.3 On-line performance

As mentioned earlier, nodes take care of the on-line learning performance by constantly forwarding messages in the direction of the sink. In Figure 5.8 we display the number of messages received by the sink per frame during learning, averaged over 1000 runs in the 3-by-3 grid topology using per-slot learning. Recall that after deployment the frame contains only one slot and every $T^{\text{learn}} = 200$ repetitions the frame is expanded by one slot (cf. Figure 5.2a). The sink can receive only one message per slot and at most 9 messages per frame, since there are 9 nodes in the network, each generating 1 message at the beginning of the frame. In the 3-by-3 grid topology the minimum number of slots necessary to deliver all messages is 11. Thus the upper bound in the figure shows the theoretical maximum number of messages that the sink can receive within a frame. Note that this upper bound is loose and can never be fully achieved, since in this topology messages from leaf nodes need at least 3 time slots to reach the sink. Therefore the sink can never receive a message

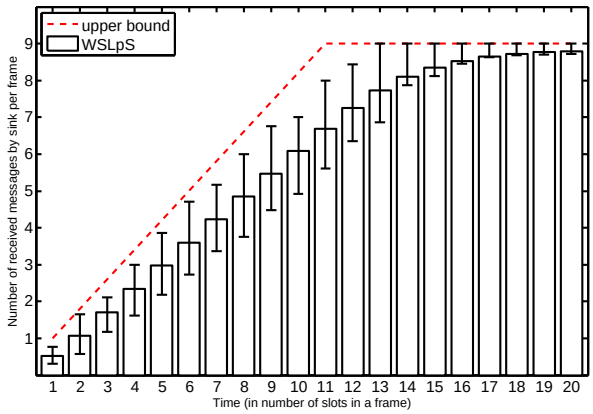


Figure 5.8: On-line performance during the learning phase, measured in the number of messages that the sink receives per frame in a 3-by-3 grid topology. Lower and upper error bars show the 25^{th} and 75^{th} percentile, respectively. The upper bound indicates the limit on the possible number of received messages, as the sink can receive at most one message per slot and at most nine messages in a frame — one from each of the nine nodes in the network.

at every slot in the frame for all frame sizes.

We do not analyze the on-line latency, as it cannot be correctly determined due to the transient effect at the beginning of the learning phase. Instead, we measure the ratio of delivered versus dropped messages during and after learning. For simplicity we consider the worst case scenario where nodes have queue length of 1, i.e. all nodes drop their undelivered messages at the end of each frame in order to generate new ones at the beginning of the next. Thus, at the beginning of each frame (regardless of size) there are 9 messages in the network. As in the beginning of the learning phase the frame consists of only 1 slot, at least 8 messages will have to be dropped; at least 7 messages will be dropped when the frame contains 2 slots, and so on. Figure 5.9 displays this cumulative delivery ratio for the first 70 minutes after deployment (or the first $8.2 \cdot 10^5$ slots), assuming a slot duration of 5 milliseconds. We see that the graph asymptotically reaches a ratio of 1 and that in the first one hour of runtime, already 90% of all generated messages since deployment have reached the sink. Thus, although many messages are dropped in the first 3.5 minutes of learning (up to the dashed line in the figure), this ratio is negligible compared to the quality of the final solution in the long run.

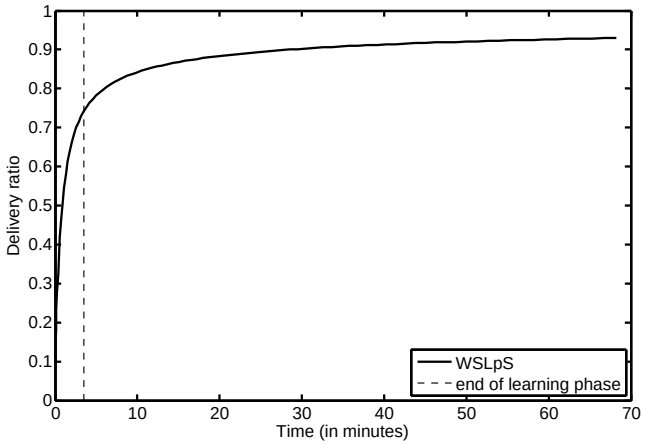
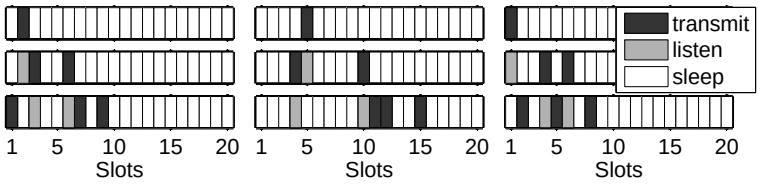


Figure 5.9: Cumulative ratio of delivered versus dropped messages in a 3-by-3 grid topology during and after the learning phase. The dashed line indicates the end of the 3.5 minute learning phase at which point all nodes start executing their learned schedules. The figure shows the worst case scenario, where nodes have queue size of 1. Each node generates a message at the beginning of each frame and drops all its undelivered messages at the end of each frame.

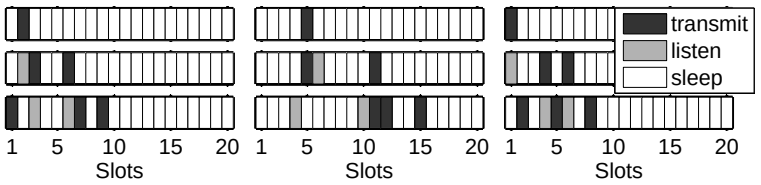
5.5.1.4 Clock drift

So far we assumed perfect clock synchronization where all nodes count slots equally. However, in reality, internal clocks may speed up or slow down relative to the clocks of other nodes, leading to different notions of time. Although the amount of clock drift is small compared to the size of a slot, this drift may accumulate and over time shift the learned schedule one slot sooner or later. As WSLpS relies on fixed behavior after learning, once a drift builds up in a node to cause disturbance on the schedules of neighboring nodes, the learning phases of all affected nodes need to be restarted, so that nodes adapt to the new time. Each node will restart its learning phase of 3.5 minutes once it detects that its schedule is no longer good, i.e. if the node experiences collisions, unsuccessful transmissions, etc.

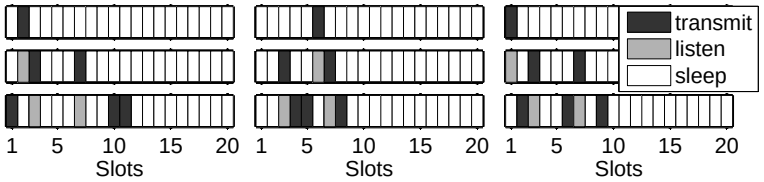
We show in Figure 5.10a the learned schedule of nodes in a 3-by-3 grid topology, which results in a latency of 15 slots. At a certain point in time we simulate that the clock drift of the middle node has accumulated, such that its learned schedule is delayed with one slot, as shown in Figure 5.10b, compared to the schedules of the other nodes in the network. The first node to detect this problem is its parent — the bottom middle node. At slot 4 the latter node expects to receive data, but listens idly to the channel. At slot 5 the middle node itself and its child — the top middle node,



(a) Learned schedules.



(b) The clock of the middle node is late, causing its learned schedule to drift one slot to the right with respect to the schedules of the other nodes, which remain the same.



(c) Re-learned schedules after each node has detected the problem in the previous schedule.

Figure 5.10: Demonstrating re-learning of schedules after clock drift in a typical run in the 3-by-3 grid topology using WSLpS with $\alpha = 0.01$. Schedules are arranged according to node position in the network (see Figure 5.3b). The sink is at the bottom and is always listening.

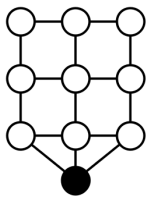
both experience unsuccessful transmissions. Analogously, these problems propagate to all other nodes in the network. After a number of consequent failures to transmit or receive, each node determines that the problems occur too often to be caused by occasional packet loss, and are therefore due to clock drift. Each node restarts its learning phase of 3.5 minutes whenever it supposes that clock drift has occurred. Although not all nodes start learning at the same time, the learning phase is long enough to allow all nodes to find a good schedule. The learning phase is similar to the initial one after deployment, except that q-values are initialized to the current schedule and not all to *listen*, as in the beginning. Moreover, the frame length is not restarted and remains $S^{\max}$ slots. Nodes still learn one slot at a time for T^{learn} rounds before learning in the next slot, while in the rest of the frame they repeat

the (previously) learned actions. Note that frame boundaries need not coincide for all nodes. Agents will learn a schedule relative to the actions of others, regardless when the frame starts or ends. We show in Figure 5.10c the newly learned schedules after the clock drift of the middle node has caused all nodes to restart their learning phases. Since learning is restarted from the last learned schedule (and not from scratch), actions that are not in conflict with others have a high chance to remain the same. We see that the schedules of the three leftmost nodes remain the same in the first 5 slots as before the new learning phase, while the schedules of the three rightmost nodes is the same only in the first two slots. Restarting from the last learned schedule results in a better on-line performance during each learning phase, as nodes are able to successfully deliver more messages to the sink. Moreover, the newly learned schedule in Figure 5.10c has a latency of 11, which is the optimal latency.

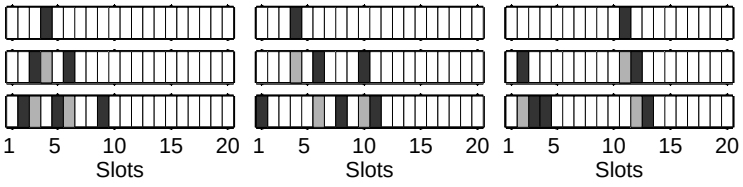
5.5.1.5 Rerouting

Another potential problem in WSN communication, besides clock drift, is when a node is damaged or runs out of battery. In that case the routing protocol needs to build a new routing tree in order to redirect the network traffic around the depleted node. Note that in this thesis we are not concerned with routing algorithms and simply assume that the rerouting takes care that all nodes have a path to the sink. As a result of the new routing tree, the schedules of nodes may change according to the (new) traffic rate.

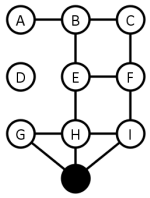
We show in Figure 5.11b an example of a learned schedule in the 3-by-3 grid topology when all nodes are functioning, as depicted in Figure 5.11a. We then simulate that the middle left node (or node **D**) runs out of battery and becomes disconnected from the network (see Figure 5.11c). After a number of consecutive failed transmissions and receptions, nodes **A** and **G** detect that their schedule is no longer good and restart their learning phase, as they did after a clock drift in Section 5.5.1.4. Note that the nodes are not aware that their neighbor, node **D** has run out of battery. The restart of the learning triggers other nodes to initiate their learning phase. During learning, the shortest-hop routing protocol re-builds the routing tree, which leaves node **A** at hop 4, since its only neighbor is node **B**, who is at hop 3. As a result, shown in Figure 5.11d, nodes **B**, **E** and **H** need to forward an additional packet every frame, that of agent **A**. Node **G**, on the other hand, becomes a leaf node and therefore needs to forward only its own message to the sink. Lastly, nodes **C**, **F** and **I** experience the same traffic rate as before, but learn a new schedule as a result of the interfering communications of their neighbors.



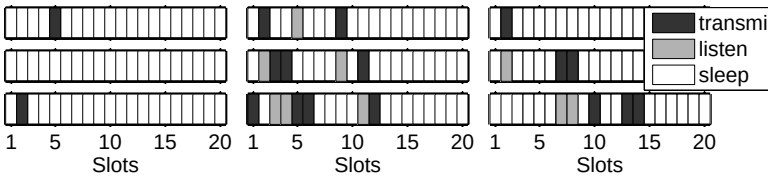
(a) Topology with all nodes functioning.



(b) Learned schedules.



(c) Topology after node **D** runs out of battery.



(d) Re-learned schedules as a result of rerouting.

Figure 5.11: Demonstrating re-learning of schedules after a node runs out of battery in a typical run in the 3-by-3 grid topology using WSLpS with $\alpha = 0.01$. The depleted node is disconnected from the network and the shortest-hop routing scheme selects **B** as the parent of **A**, who is now at hop 4. Schedules are arranged according to node position in the network. The sink is at the bottom and is always listening.

In summary, the depletion of node **D** triggered a re-learning phase that propagated through the network. As a result of the new routing tree, nodes learned a new schedule according to the traffic rate they experience.

5.5.1.6 Packet loss

Lastly, we investigate the effect of noisy communication on the performance of the system. So far in this chapter we assumed perfect communication and let nodes probabilistically shift their action upon every conflict. However, in a noisy WSN environment even if agents are properly synchronized for communication, packets may occasionally get dropped. Therefore agents should not always consider occasional packet losses as conflicts and re-learn their schedules. Unfortunately, with no memory, agents are unable to distinguish between a dropped message due to packet loss and failed communication as a result of conflicting schedules (e.g. due to clock drift).

In Sections 5.5.1.4 and 5.5.1.5 we required that agents wait for a number of consecutive observations of the same conflict, in order to rule out packet loss as the cause of the disturbance. Here we elaborate on this topic and discuss how WSLpS can be modified to function in noisy environments. In particular, we extend the memory of each agent i to contain a window of the last W_i payoffs p_i^t, obtained at time steps $\{t - W_i + 1, \ldots, t\}$, in order to recognize if only few of the last W_i actions were unsuccessful, or all of them. In this way agents ignore any occasional unsuccessful actions and treat them as if they were successful, as long as $\sum_{t=1}^{W_i} p_i^t > 0$. Messages from unsuccessful transmissions remain in the queue of agents. If all last W_i actions failed, i.e. $\sum_{t=1}^{W_i} p_i^t = 0$, only then will the agent probabilistically shift its unsuccessful action. Thus, we modify policy 5.1 to account for the last W_i payoffs, instead of only the last one. The length of this payoff window should be large enough to skip any occasional packet losses, and yet small enough to converge quickly to a good solution. Moreover, the window should not be the same for all agents, so that there is a chance that an agent, who restarts learning, quickly resolves a conflict before its partner restarts its learning phase. Although W_i affects the overall system performance, we will not study extensively this parameter, but use a good range that we determined empirically. Thus, we set the window length W_i of each agent i to a uniformly random number between 5 and 10 last payoffs.

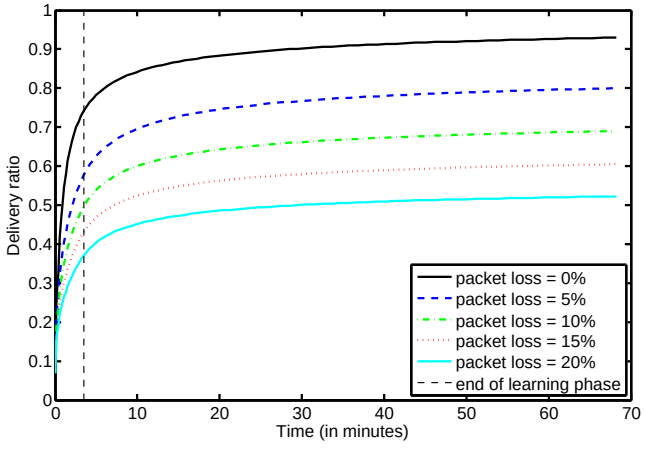


Figure 5.12: Cumulative ratio of delivered versus dropped messages in a 3-by-3 grid topology during and after the learning phase for different packet loss rates. The black dashed line indicates the end of the 3.5 minute learning phase at which point all nodes start executing their learned schedules. The figure shows the worst case scenario, where nodes have queue size of 1. Each node generates a message at the beginning of each frame and drops all its undelivered messages at the end of each frame.

Each agent i will select action *transmit* with probability $\pi_i(transmit)$, depending on the previously selected action at that slot, the last W_i payoffs p_i^t, and the number of messages m_i in the queue of agent i:

$$\pi_i(transmit) \leftarrow \begin{cases} 1, & \text{if } transmit \text{ AND } \sum_{t=1}^{W_i} p_i^t > 0 \text{ AND } m_i > 0 \\ \alpha, & \text{if } \sum_{t=1}^{W_i} p_i^t = 0 \text{ AND } m_i > 0 \\ 0, & \text{if } listen \text{ AND } \sum_{t=1}^{W_i} p_i^t > 0 \text{ OR } m_i = 0 \end{cases} \tag{5.4}$$

where $\alpha \in (0, 1)$ is the probabilistic component of WSLpS. We show in Figure 5.12 the delivery ratio of WSLpS up to 20% packet loss. For every 5% packet loss we see 10% drop in the delivery ratio, since in not all sample runs agents are able to find a good schedule for a frame length of 20 slots. We show in Figure 5.13 the learned schedule of nodes for a packet loss rate of 10%. As dropped messages need to be retransmitted at a later slot, the resulting schedule in the figure has a higher latency of 19 slots. Thus, in noisy environments, the frame length needs to be long enough to ensure that agents will deliver all their messages to the sink. Further analysis needs to be performed to determine the size of the frame and the length of the payoff window.

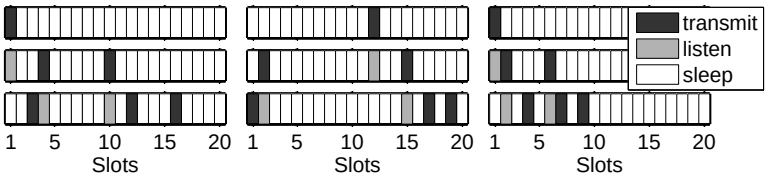


Figure 5.13: Learned schedules in 3-by-3 grid topology with 10% packet loss. The schedules result in latency of 19 slots. Schedules are arranged according to node position in the network. The sink is at the bottom and is always listening.

5.5.2 Discussion

An intriguing aspect of the (anti-)coordination problem in WSNs is that nodes experience different games, according to their role in the forwarding of messages. The game that transmitters experience has characteristics of pairwise pure coordination games from Chapter 3 (Section 3.4). A node **A** deciding to forward a message selects only one other node **B** as the intended receiver and attempts to form an implicit coalition with **B** by trying to send a message. The transmitter obtains a payoff of 1 only if **B** has chosen to belong to the same coalition, i.e. if **B** decides to help **A** in forwarding its message. In contrast, the game that listening nodes experience

bares resemblance to the multi-player (anti-)coordination games of Chapter 4 (Section 4.6). A listening node has to both coordinate with a transmitter and at the same time anti-coordinate with all other nodes in range. For these reasons the transmit probability α behaves both as keep probability (after unsuccessful transmission) and as shift probability (after unsuccessful listening). Thus the (de)synchronization problem in WSNs is certainly challenging and combines aspects of all games we studied so far.

Here we would like to shortly depart from the WSN domain and discuss how WSLpS can be applied in other domains. In particular, we are interested in traffic light control problems since they fit well in the topic of this dissertation. In fact, traffic light control problems to a certain extend resemble coordination problems in peer to peer communication networks, such as load balancing and throughput optimization. A traffic light system in a city can be regarded as a multi-agent system where central control is costly both in terms of infrastructure maintenance (connecting all traffic lights to a central system) and in computational resources. The complexity and costs of such a system can be reduced by addressing the problem from a decentralized perspective where individual traffic lights are represented by decision-making agents with no global knowledge. The challenge then is to enable the decentralized coordination between agents, based on the traffic flow of vehicles. The aim is to optimize the global traffic throughput, minimize waiting times and certainly avoid traffic accidents (or collisions), caused by conflicting light signals. The latter requirement once more illustrates the importance of having the *whole* system converge to a good solution, rather than only 90% of the agents. Similarly to WSNs, the dependence between games at different light cycles (or time slots) is determined by the vehicle flow. However, contrary to WSNs, individual traffic lights interact indirectly via this traffic flow — a green signal at one intersection will send vehicles towards the traffic light at the next intersection. The payoff from each interaction can be measured by the local throughput and waiting (or queuing) times at each agent. The action space consists in selecting durations for both red and green signals, analogous to the sleep and awake times in WSNs. However, here red and green light durations are contiguous periods, while in WSNs the sleep and awake times may be scattered in the frame. Each agent needs to synchronize its activities with some lights, in order to propagate more vehicles, and at the same time desynchronize with other lights in order to avoid accidents or traffic jams. Moreover, since traffic flow patterns change at different times of the day, agents need to learn over larger time windows.

5.6 Results from real-time learning

Recall that the main purpose of our research is to study decentralized approaches that make agents (anti-)coordinate and achieve good collective performance imposing minimal system requirements and overhead. The real-time protocol that we propose helps highly constrained wireless nodes achieve (de)synchronization in a decentralized manner with limited feedback. We stress that it is not our aim here to propose out-of-the-box MAC protocols that are robust against all WSN settings and traffic conditions. In other words we focus on addressing the (anti-)coordination problem in WSNs, rather than on designing a MAC protocol for ad-hoc WSNs. With the experiments below we illustrate the importance of (de)synchronization, as opposed to implementing MAC protocols that achieve pure synchronization. We present here a revised version of the results from DESYDE, published in Mihaylov *et al.* [2011a].

We evaluate the performance of our learning approach in networks of different size and topology. Each network is run for 3600 seconds in the OMNeT++ simulator[3] and results are averaged over 30 runs. This network runtime was sufficiently long to eliminate any initial transient effects. To illustrate the performance of the system at high data rates, we set the data sampling period of nodes to one message every 10 seconds, which is long enough for all messages to reach the sink between two data samplings. Frames have the same length as the sampling period and were divided in $F = 2000$ slots of 5 milliseconds each. The duration of the slot was chosen such that only one DATA packet can be sent and acknowledged within that time. All hardware-specific parameters, such as transmission power, bit rate, etc., were set according to the data sheet of our radio chip — CC2420. In addition, we chose the protocol-specific parameters, such as packet header length and number of retransmission retries as specified in the IEEE 802.15.4 standard [Gutierrez *et al.*, 2002].

To address the latency issues of synchronized wake-up protocols, such as S-MAC [Ye *et al.*, 2004], Lu *et al.* [2004] propose D-MAC, which employs a staggered wake-up schedule to enable continuous data forwarding on the multi-hop path. Instead of having all nodes synchronized at the beginning of the frame, D-MAC schedules the radio activity of sensor nodes in such a way that only pairs of vertical neighbors in the routing tree synchronize their radio transmission/reception slots. While this strategy appears at first sight to offer significant benefits over traditional S-MAC, it only performs well if nodes are arranged in a line topology. Indeed, whenever the routing tree contains several branches, horizontal neighbors wake at the same

[3] A C++ simulation library and framework, primarily for building network simulators.

time and may interfere, causing packet losses, and possibly important delays (once a transmission fails in D-MAC, the packet is queued until the next frame). In addition to staggering the duty cycles, D-MAC adjusts the length of the active period according to the traffic load in the network.

5.6.1 Evaluation

We compare the performance of our DESYDE protocol to D-MAC in order to determine whether synchronous staggering the adaptive duty cycles across hops is an efficient strategy to improve latency, as opposed to (de)synchronization. In addition, we present the case where all nodes remain active in all frames and never switch off their radio transmitter (called ALL-ON for short). The latter behavior serves only as a benchmark in terms of end-to-end latency, because the energy consumption of this protocol renders it impractical for real-world scenarios. Since nodes have a duty cycle of 100%, packets will not experience any sleep latency and will be quickly forwarded to the sink.

Experiments are carried out in three networks of different size and topology — a 4-hop line, a 16-node (4-by-4) grid topology and one with 50 nodes scattered randomly with an average of 5 neighbors per node. The first topology requires nodes to synchronize in order to successfully forward messages to the sink. The second topology illustrates the importance of combining synchronization and desynchronization, as neither one of the two behaviors alone is an efficient strategy. The random topology shows the scalability of our approach to larger networks where the topology is not known a priori. As in all other experiments we presented, in our simulations we use a shortest path routing scheme that creates a static routing tree.

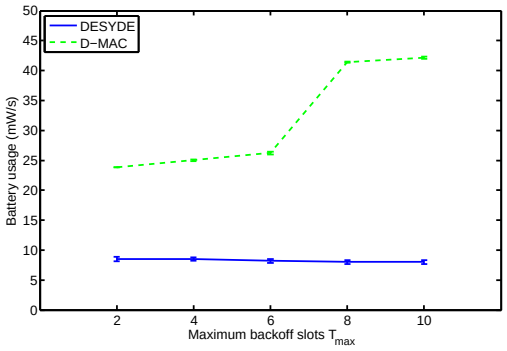
D-MAC copes with latency issues by "staggering" the wake-up cycles of nodes according to their hop distance to the sink. In other words, all nodes that lie at the same distance from the sink are synchronized to wake up at the same time and send a packet to their parents, who wake up at the slot just after their children. The length of the active period (or duty cycle) of each node under D-MAC is dependent on its traffic load (i.e. its position in the data gathering tree), as it is the case with our learning approach DESYDE.

As explained in Section 5.4.2, we attempt to reduce collisions by letting each node contend for the channel for uniform random number of slots within a fixed contention window of size T_{max}. D-MAC uses the same principle of contention to resolve conflicts. We therefore present the performance of D-MAC and DESYDE for different contention window sizes. According to the specifications of D-MAC, we define the size of its contention window in terms of the duration of a DATA packet.

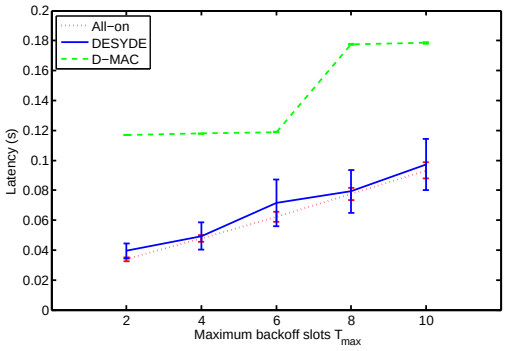
The design of DESYDE, however, requires us to set the contention window as a factor of the slot length instead (which is a DATA packet + an acknowledgment). We use the latter setting in ALL-ON as well. Since the difference between the two contention windows is negligible, we use the same horizontal axis in Figure 5.14 to plot the performance of both protocols.

In Figure 5.14 we see that in all three topologies DESYDE outperforms D-MAC in terms of energy consumption, for all contention window sizes. Due to the "win-stay lose-probabilistic-shift" behavior of DESYDE, our learning approach is not significantly influenced by the contention window size, since contention is used only during the learning phase. Each node, thereafter, learns to transmit in a different time slot within the frame and thus contention for the channel is not necessary under constant traffic pattern. Nodes under D-MAC, however, always wake up for one listen and one transmit slot, regardless of the node position in the network. A disadvantage is that leaf nodes still listen for one slot, when they need not, while all other nodes need to hold an additional listen + transmit slot for every packet they generate. The energy consumption of D-MAC is therefore higher than the one of DESYDE for each topology. Moreover, according to specifications, the active period of D-MAC includes the time for channel contention. Therefore its battery consumption increases with the size of the contention window. Due to the adaptiveness of the active period of D-MAC to the traffic conditions, we noticed that a contention window larger than 6 DATA packets requires nodes to hold an additional active slot, resulting in nearly $^2/_3$ times more energy.

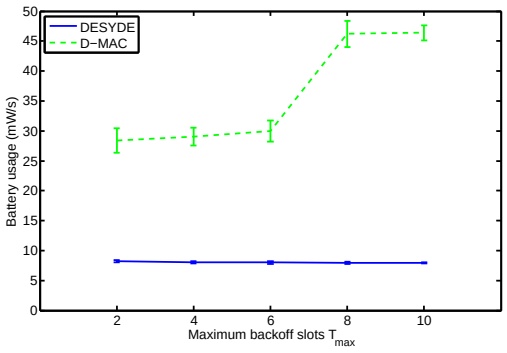
Lastly, we discuss the difference between ALL-ON, DESYDE and D-MAC in terms of the end-to-end latency averaged over 30 random topologies, each consisting of 50 nodes. Figure 5.14f compares the three protocols for different contention windows. One can notice that DESYDE once again outperforms D-MAC. DESYDE enables nodes to both synchronize with their parents and desynchronize with their horizontal neighbors. In the shortest path routing scheme that we employ, all nodes that lie on the same hop belong to different branches of the routing tree. In D-MAC, however, all those nodes wake up at the same time and therefore cause radio interferences, followed by packet retransmissions. Intuitively, latency under D-MAC decreases for larger contention windows, but nodes still require more than one active period to deliver all their packets. DESYDE, on the other hand, has comparable latency to ALL-ON, where nodes never switch off their antenna and therefore packets incur no sleep delay. While ALL-ON requires 100% duty cycle, DESYDE is able to achieve comparable latency with only 0.8% average active time within a frame.



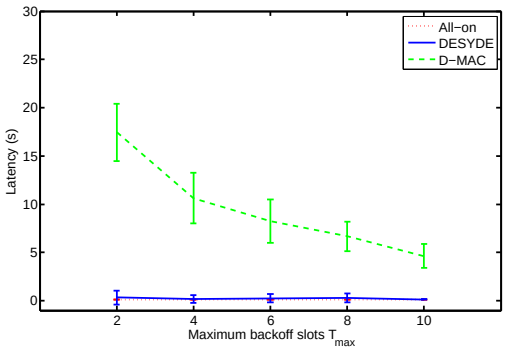
(a) Energy consumption in the line.



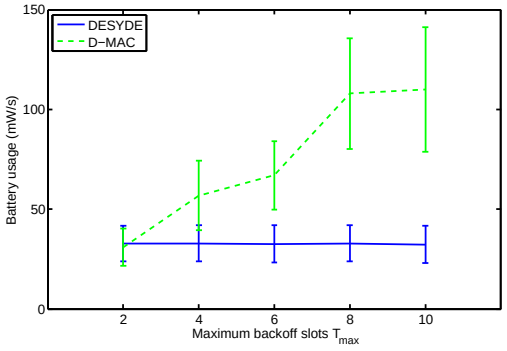
(b) End-to-end latency in the line.



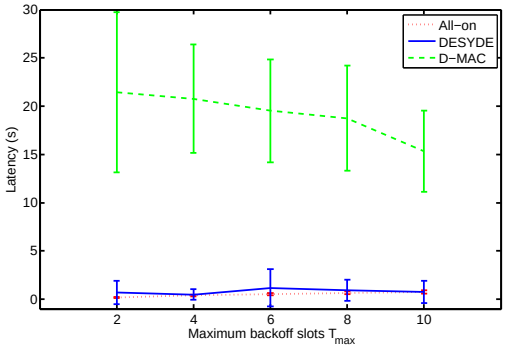
(c) Energy consumption in the grid.



(d) End-to-end latency in the grid.



(e) Energy consumption in the random.



(f) End-to-end latency in the random.

Figure 5.14: Performance of protocols in all three topologies for different values of the maximum backoff T_{max}. Error bars show standard deviation.

5.6.2 Discussion

The above experimental results illustrate that DESYDE is able to significantly improve the performance of a data collection task in wireless sensor networks by (de)synchronizing schedules in a decentralized manner. The two main metrics considered are the latency and the energy consumption. For both metrics, large gains could be observed, over a wide range of networking parameters. These results were particularly remarkable for large and random topologies. The main reason is that DESYDE relies on a learning strategy which can adapt to complex topologies and dense traffic patterns.

The win-stay lose-probabilistic-shift strategy which underlies DESYDE is a key aspect of the proposed approach. Several research directions can be pursued in order to further improve its performance. First, an advantage of the WSLpS is that it provides a way to reduce the exploration space and to accelerate the convergence of the learning stage. A direct drawback of this "aggressive" exploration is that more efficient solutions to the coordination of sensor nodes may be too quickly discarded. One of the research axes we plan to focus on consists in relying on "smoother" updating rules for the quality values of the actions. This could be done by using a learning factor which keeps tracks of past rewards during the learning process. In addition, coordinated exploration techniques can be applied to find more efficient schedules [Verbeeck, 2004].

A second important parameter is the convergence time of the learning process. We observed in all our experiments that this time is in practice very short, in the order of a few data collection rounds (or frames). Markov chain analysis can be performed on DESYDE, similar to the one carried out in Section 3.6.2, in order to study the convergence properties of learning. Unfortunately, network convergence does not seem to be detectable by individual agents without all nodes exchanging information about their state, which would be energy costly.

We assumed, as do most protocols which fall in the synchronous wake-up category, that the traffic patterns and the network topology are stationary for every run. As suggested earlier, DESYDE is not robust to topology changes, or to variations in the data collection rate. The common solutions to these issues is to rely on periodic checks concerning the amount of dropped packets, or queue sizes on the sensor nodes, and to restart the coordination of the nodes if necessary. Due to the short learning phase, DESYDE is able to quickly re-adjust the wake-up schedules of nodes after such changes in the network occur.

We also assumed a static routing protocol and focused on the wake-up scheduling problem, where each node needs to decide whether it should transmit, listen or

sleep at each time slot. DESYDE can naturally be extended to learn not only the scheduling, but also the routing tree of the network, based on the traffic flow. Each node can keep and update a pair of Q-values (one for transmit and one for listen) but now for each neighbor. Thus, upon successful communication with a given neighbor, the node will keep its action (e.g. *transmit to neighbor* 2) and otherwise with probability α will select a different neighbor and action at random. This behavior will ultimately extend the learning phase due to the larger action set of nodes. Nevertheless, it will allow the routing scheme to distribute the traffic flow more evenly across the network, as neighbors with high traffic rates will likely reject packets from new parents. In addition to routing and scheduling, DESYDE can also be extended to function in a multi-channel setting. In Phung *et al.* [2012] we use an approach similar to WSLpS and DESYDE and combine wake-up scheduling with route selection in order to increase the number of parallel transmissions in a multi-channel WSN. We demonstrate how our approach outperforms McMAC, a state-of-the-art parallel rendez-vous protocol, in terms of throughput and latency.

5.7 Conclusions

Synchronous wake-up protocols allow users to greatly reduce the duty cycle of wireless sensor nodes in a periodical monitoring task. We highlighted in this chapter, however, that they suffer from potential high latency and energy waste due to radio interferences and packet collisions. These deficiencies stem from the fact that neighboring sensor nodes need to synchronize their activities within their own routing branch, and at the same time desynchronize with nodes on other branches. Moreover, the decentralized nature of WSN communication and the constraints arising from the limited resources of sensor nodes make the (de)synchronization problem challenging.

To answer our research question **Q3**, we explored the WSN (de)synchronization problem from two perspectives. We studied agent coordination as a sequence of repeated singe-stage (anti-)coordination games in per-slot learning, as well as from the perspective of the resulting multi-stage game in real-time learning. Our proposed protocols address the WSN (anti-)coordination challenge, also shown in Example 12, in a decentralized manner, while imposing minimal system requirements and overhead. As a result, our approach makes it possible that (anti-)coordination *emerges* in time rather than is *agreed* upon. We applied our win-stay lose-probabilistic-shift approach in the per-slot perspective to study how individual agents can (anti-) coordinate without explicitly modeling the relation between slots. Due to the com-

parable quality of the final schedules in the two perspectives, we conclude that there is only a weak relation between the games at the different time slots. Nevertheless, this relation ultimately influences the overall system latency.

The core of DESYDE is based on the win-stay lose-probabilistic-shift approach, but is applied in the real-time perspective. It lets nodes individually desyde [*sic*] on their actions and thus quickly converge to an efficient wake-up scheme with no additional communication overhead. In this way, by introducing DESYDE, we are able to answer the question presented in Example 12.

Although optimization of long-term behavior will certainly improve network performance, nodes have insufficient information to consider the global long-term goals of the system. Our approaches are well-suited for myopic agents, who only optimize the immediate payoffs of their actions, but nevertheless achieve near-optimal performance. Still, one could explore the trade-off between the cost of information sharing and the quality of the final solution when considering long-term behavior. In addition, other routing protocols can be investigated, along with contention-based schemes and multiple (mobile) sinks.

Chapter 6

Conclusions and outlook

In this thesis we investigated the following problem, which served as the main motivation for our research: **How can the designer of a decentralized system, imposing minimal system requirements and overhead, enable the efficient coordination of highly constrained agents, based only on local interactions and incomplete knowledge?** We focused on decentralized systems with complex design objectives, beyond the capabilities of individual agents and designed an approach that allows these agents to efficiently coordinate in a decentralized manner. In this chapter we summarize how we addressed this problem and the underlying research questions.

Our work on decentralized coordination has been inspired mainly by the challenging domain of wireless sensor networks (WSNs). WSNs are an example of a decentralized system with complex objectives, but no central authority to compute a global solution. Agents (or sensor nodes) are fully cooperative, but also highly constrained with limited computational resources and restricted communication range. Moreover, agents interact with only small portion of the population and have no global knowledge. They receive a limited feedback signal and see only the outcome of their own actions. Although our focus is mainly on WSNs, other decentralized systems, such as fleets of robot vehicles or swarms of picosatellites, possess similar characteristics. All these systems require the efficient decentralized coordination between individual agents.

The WSN problem in particular requires agents to synchronize with some nodes, in order to improve message throughput, and at the same time desynchronize with

others, in order to reduce communication interference. We refer to this type of coordination as (de)synchronization, or (anti-)coordination in time. Throughout this thesis we analyzed the (anti-)coordination problem by studying its two building blocks separately — pure coordination (Chapter 3) and pure anti-coordination, as well as the combined problem of coordination and anti-coordination (Chapter 4). We then studied the full problem of (anti-)coordination in time, as seen in the WSN domain (Chapter 5). Here we summarize our findings and draw conclusions on the obtained results.

6.1 Summary and conclusions

Our research in Chapter 3 was guided by the following question:

Q1: *How can conventions emerge in a decentralized manner in pure coordination games?*

We studied the problem of convention emergence in pure coordination games and surveyed the related literature. We proposed a simple decentralized approach for fast on-line convention emergence, called Win-Stay Lose-probabilistic-Shift (WSLpS). It is based on the reinforcement learning (RL) framework and allows for a whole spectrum of strategies, two of which are the well-known strategies from game theory — Win-Stay Lose-Shift (WSLS) and Win-Stay Lose-Randomize (WSLR). We showed that WSLpS outperforms WSLS and WSLR. Within only a short number of time steps, agents involved in a repeated pure coordination game are able to reach a mutually beneficial outcome on-line based on only local interactions and limited feedback, and without a central mediator. We analytically studied the properties of our approach using the theory of Markov chains and proved its convergence in pure coordination games. Moreover, we laid out our analysis in such a way that it can be extended to other game types in a relatively straightforward manner.

We also investigated empirically the behavior of players in different topological configurations and concluded that densely connected agents reach a convention on average faster than agents in sparser networks. Another finding is that conventions emerge faster when agents have a large probability to change their action upon conflict. An interesting result is that information on the actions of others does not always lead to significant improvements and that observation of neighbors' actions, though informative, is only useful in dense networks.

In Chapter 4 we posed the following question:

Q2: *How can agents achieve pure anti-coordination in a decentralized manner in*

dispersion games?

We studied the problem of pure anti-coordination and the combined problem of coordination and anti-coordination in single-stage dispersion games. We showed that a simple approach like WSLpS is able to make agents in different configurations quickly self-organize with no history of past plays and based only on local interactions with limited feedback. We surveyed the literature on anti-coordination games and described the details of several algorithms that bare resemblance to WSLpS. Our empirical results indicate that WSLpS performs at least comparable to similar algorithms, but it can be applied in a wide range of scenarios, in which other, sometimes more complex algorithms are not suitable.

We also illustrated the relationship between the convergence time of agents in pure coordination and pure anti-coordination games, as well as in the combined (anti-)coordination game. We argued that the former two game types are inherently related and that the goal of agents in both games is the same — learning to select the appropriate actions, in order to avoid conflicts. Thus from the point of view of individual agents, the two games differ only in the way the payoff signal is defined. Nevertheless, there is an important difference for the system designer, concerning the learning duration. We saw that solutions always exist in convention games and that their number increases linearly in the number of actions. However, convergence time of agents increases exponentially in the number of actions. Provided solutions exist, dispersion games, on the other hand, converge much faster, but their feasibility depends on the topology and the number of actions. Lastly, we saw that the convergence time of (anti-)coordination games, which require equal amount of coordination and anti-coordination, is much closer to that of pure anti-coordination (i.e. faster), than to pure coordination.

Finally, in Chapter 5 we studied the problem of (anti-)coordination in time, guided by the following question:

Q3: *How can highly constrained sensor nodes organize their communication schedules in a decentralized manner in a wireless sensor network?*

We explored how the (anti-)coordination problem maps to the WSN problem of (de)synchronization. We studied the latter problem from two perspectives: as one multi-stage (anti-)coordination game in time, and as a sequence of repeated single-stage graphical games at different time intervals. Each time interval is dependent on the previous one in the sequence as a result of the message forwarding task. We observed that although from the latter perspective agents do not explicitly model the relation between time slots, we obtained comparable end results with the multi-stage

learning. We therefore conclude that there is only weak dependence between the interactions at different time slots, which however influence the end-to-end latency of the system. Since agents have no global information, they cannot model this long-term effect on the system performance. Myopic agents, therefore, proved well-suited for this learning scenario, as they were able to achieve near-optimal results at negligible learning costs. Since optimization of long-term goals is non-trivial and costly in WSNs, we demonstrated that maximizing immediate payoffs still results in near-optimal behavior. Moreover, the short learning duration allows myopic agents to quickly adapt to changes in the environment.

We studied how WSLpS can be used by computationally bounded sensor nodes to organize their communication schedules in an energy-efficient decentralized manner. We investigated the performance of WSLpS both in perfect channel conditions, as well as in noisy environments. We proposed an adaptive communication protocol for real-time learning. The DEcentralized SYnchronization and DEsynchronization protocol (DESYDE) is based on our simple WSLpS approach and lets nodes quickly converge to an efficient wake-up scheme with no additional communication overhead. As a result of our simple protocol, (anti-)coordination emerges in time rather than is agreed upon. Due to the high communication costs in WSNs, using our protocol agents are able to quickly find good solutions, without necessarily looking for the optimal ones. We compared DESYDE against D-MAC, a representative synchronization protocol in literature and demonstrated the importance of (anti-) coordination in WSNs, as opposed to pure coordination and pure anti-coordination.

We believe in this dissertation we adequately addressed our problem statement and the three related research questions. We motivated the need for decentralized coordination in the challenging domain of wireless sensor networks. We then developed a simple decentralized approach, called win-stay lose-probabilistic-shift, that allows the highly constrained sensor nodes to efficiently coordinate their behavior and thus achieve their complex design objectives. WSLpS requires no history of past interactions and imposes minimal system requirements due to its low computational complexity. Agents are able to efficiently coordinate without additional communication overhead and with no sharing of local information. Moreover, WSLpS features a short learning phase, reducing the high learning costs in WSNs. Due to the lack of global knowledge, individual agents cannot determine that a final solution has been reached. Nevertheless, an advantage of WSLpS is that global solutions are absorbing states of the resulting Markov chain and therefore agents, once converged, never leave a favorable outcome. If changes occur in the environment of agents, e.g. a sensor node runs out of energy, agents can quickly converge to a new favorable

state. Using WSLpS our highly constrained agents are able to efficiently coordinate their behavior in a decentralized manner and achieve their design objectives.

6.2 Directions for future research

In closing, we list here some directions that can be taken to extend the research presented in this dissertation.

One research topic that can be addressed is the structure of the underlying game topology. We studied agent interactions and convergence times in static networks, which are suitable to model, for example, agents in the smart grid, or wireless nodes in a field. Agents do not change their position and thus their neighborhood remains fixed. We also demonstrated that our approach can still adapt if a node runs out of energy or causes disturbance on neighboring nodes. In other real-world scenarios, however, the network topology is dynamic, such as agents in mobile computing or fleets of robot vehicles. One needs to study the relationship between, for example, the convergence times of agents and the rewiring mechanism. In addition, we explored (anti-)coordination games that involve equal amount of coordination and anti-coordination. These settings can be further studied in other proportions (e.g. 90% anti-cordination and 10% coordination) in order to find a more general relationship between the latter two game types and their influence on (anti-)coordination games.

We studied networks of fully cooperative agents, since they are all part of the same system, owned by the user. This research can be extended to explore the impact of private information and self-interest on the decentralized coordination problem. In some scenarios agents may belong to different users and serve different goals, thus one can apply mechanism design techniques to achieve efficient coordination between self-interested agents. Another interesting research perspective is a study on the evolution and dynamics of coordination and anti-coordination in this context.

As identified earlier, our methodology has a close relationship with the distributed constraint optimization framework and the related problem of graph coloring. A future line of work is the study of this relationship and how well DCOP problems map to the problem of decentralized (anti-)coordination. Also, to what extend can WSLpS be applied in graph coloring problems and how algorithms for graph coloring can be used by individual agents to (anti-)coordinate.

Lastly, since the source of our inspiration is WSNs we will list here some the directions for future work in the context of WSNs:

- We implemented myopic agents due to the cost of sharing information with others. One could explore in more detail the trade-off between this cost and the quality of the final solution when considering long-term behavior.
- We assumed static routing protocol and a single sink, which are suitable in environmental monitoring applications. Different scenarios can be explored, involving multiple (mobile) sinks and more dynamic routing schemes.
- When new nodes are added to the system, in theory, our approach will let them, as well as surrounding nodes, learn a new schedule. One can study the use of transfer learning techniques where surrounding nodes transfer their schedules to the new node in order to speed up the learning process.
- Although we used a state-of-the-art WSN simulator, it cannot fully capture the effect of real-world phenomena. Therefore, an actual deployment needs to be performed on real testbeds to gain insights into the actual coordination challenge of nodes.

In all games we studied synchronous updates, where agents interact with their neighbor(s) and then all agents simultaneously update their action. This is indeed the case in scheduling based protocols in WSNs, where slot boundaries are aligned, such that agents have a similar notion of time. One can study the application of WSLpS in contention based protocols where agents update their actions asynchronously.

Wireless sensor networks are indeed a challenging domain. However, many more decentralized systems exist where autonomous agents need to coordinate in order to solve their complex objectives. We believe the research presented in this dissertation provides the tools and methodology with which to study decentralized coordination problems in other multi-agent systems.

Publications

Part of the work in this thesis has already been published. Here we show a list of selected publications.

Journals and post-proceedings

- Mihaylov, M., Le Borgne, Y.A., Tuyls, K. & Nowé, A. (2012b). Reinforcement Learning for Self-Organizing Wake-Up Scheduling in Wireless Sensor Networks. In J. Filipe & A. Fred, eds., *Postproceedings of the 3rd International Conference ICAART 2011. Revised Selected Papers.*, vol. 271 of *Communications in Computer and Information Science*, 382 – 397, Springer-Verlag, Agents and Artificial Intelligence edn

- Mihaylov, M., Le Borgne, Y.A., Tuyls, K. & Nowé, A. (2012a). Decentralised Reinforcement Learning for Energy-Efficient Scheduling in Wireless Sensor Networks. *International Journal of Communication Networks and Distributed Systems*, **9**, 207–224

- Mihaylov, M., Tuyls, K. & Nowé, A. (2010). Decentralized learning in wireless sensor networks. In M. Taylor & K. Tuyls, eds., *Postproceedings of the 2nd Workshop ALA 2009, Held as Part of the AAMAS 2009 Conference. Revised Selected Papers.*, vol. 5924 of *Lecture Notes in Computer Science*, 60–73, Springer Berlin/Heidelberg, Adaptive Learning Agents edn

Full papers at international conferences

- Mihaylov, M., Le Borgne, Y.A., Tuyls, K. & Nowé, A. (2011a). Distributed cooperation in wireless sensor networks. In Yolum, K. Tumer, P. Stone & Sonenberg, eds., *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Taipei, Taiwan
- Mihaylov, M., Le Borgne, Y.A., Tuyls, K. & Nowé, A. (2011b). Self-Organizing Synchronicity and Desynchronicity using Reinforcement Learning. In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence (ICAART)*, 94–103, Rome, Italy
- Van Moffaert, K., Van Vreckem, B., Mihaylov, M. & Nowé, A. (2011). A learning approach to the school bus routing problem. In *23rd Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, Ghent, Belgium
- Naessens, V., Mihaylov, M., De Jong, S., Verbeeck, K. & Nowé, A. (2010). Carebook: Assisting elderly people by social networking. In *Proceedings of the 1st International Conference on Interdisciplinary Research on Technology, Education and Communication (ITEC)*, Kortrijk, Belgium
- Mihaylov, M., Nowé, A. & Tuyls, K. (2008). Collective intelligent wireless sensor networks. In *Proceedings of the 20th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, 169–176, Enschede, The Netherlands

Full papers at workshops

- Phung, K.h., Lemmens, B., Mihaylov, M., Zenobio, D.D., Steenhaut, K. & Tran, L. (2012). Multi-agent Learning for Multi-channel Wireless Sensor Networks. In *Proceedings of the 3rd IEEE International Workshop on SmArt COmmunications in NEtwork Technologies (SaCoNet)*, Ottawa, Canada
- Mihaylov, M., Tuyls, K. & Nowé, A. (2009). Decentralized learning in wireless sensor networks. In *Proceedings of the Adaptive and Learning Agents Workshop (ALA)*, Budapest, Hungary

List of examples

List of algorithms

List of tables

Bibliography

[Agarwal *et al.*, 2005] Agarwal, Y., Gupta, R. & Schurgers, C. (2005). Dynamic power management using on demand paging for networked embedded systems. In *Proceedings of the Asia and South Pacific Design Automation Conference*, vol. 2, 755–759. Cited on page 126.

[Akyildiz *et al.*, 2002] Akyildiz, I., Su, W., Sankarasubramaniam, Y. & Cayirci, E. (2002). A survey on sensor networks. *Communications Magazine, IEEE*, **40**, 102–114. Cited on pages 121 and 124.

[Al-Karaki & Kamal, 2004] Al-Karaki, J. & Kamal, A. (2004). Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE*, **11**, 6–28. Cited on page 123.

[Aras *et al.*, 2004] Aras, R., Dutech, A. & Charpillet, F. (2004). Cooperation through communication in decentralized Markov games. In *International Conference on Advances in Intelligent Systems - Theory and Applications - AISTA'2004*, Luxembourg-Kirchberg/Luxembourg. Cited on page 27.

[Arthur, 1994] Arthur, W. (1994). Inductive reasoning and bounded rationality. *The American economic review*, **84**, 406–411. Cited on pages 23 and 24.

[Auer *et al.*, 2002] Auer, P., Cesa-Bianchi, N. & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, **47**, 235–256. Cited on page 95.

[Aumann, 1974] Aumann, R.J. (1974). Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, **1**, 67–96. Cited on page 19.

[Axelrod, 1984] Axelrod, R. (1984). *The evolution of cooperation*. Basic Books, New York. Cited on page 17.

[Axelrod, 1986] Axelrod, R. (1986). An evolutionary approach to norms. *The American Political Science Review*, **80**. Cited on pages 21 and 45.

[Barabasi *et al.*, 1999] Barabasi, A.L., Albert, R. & Jeong, H. (1999). Mean-field theory for scale-free random networks. *Physica A: Statistical Mechanics and its Applications*, **272**, 19. Cited on page 69.

[Barrett & Zollman, 2009] Barrett, J. & Zollman, K.J.S. (2009). The role of forgetting in the evolution and learning of language. *Journal of Experimental & Theoretical Artificial Intelligence*, **21**, 293–309. Cited on pages 38, 45, 46, 47, 48, 50, 57, and 60.

[Borms *et al.*, 2010] Borms, J., Steenhaut, K. & Lemmens, B. (2010). Low-overhead dynamic multi-channel mac for wireless sensor networks. In *Proceedings of the 7th European conference on Wireless Sensor Networks*, EWSN'10, 81–96, Springer-Verlag. Cited on pages 122 and 127.

[Boutilier, 1996] Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, 195–210, Morgan Kaufmann Publishers Inc. Cited on pages 27 and 30.

[Bowling & Veloso, 2002] Bowling, M. & Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, **136**, 215–250. Cited on pages 33 and 106.

[Boyan & Littman, 1994] Boyan, J.A. & Littman, M.L. (1994). Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In J.D. Cowan, G. Tesauro & J. Alspector, eds., *Advances in Neural Information Processing Systems*, vol. 6, 671–678, Morgan Kaufmann Publishers, Inc. Cited on page 123.

[Bramoullé, 2001] Bramoullé, Y. (2001). Complementarity and social networks. University of Maryland. Cited on pages 93 and 94.

[Bramoullé, 2007] Bramoullé, Y. (2007). Anti-coordination and social interactions. *Games and Economic Behavior*, **58**, 30–49. Cited on pages 76 and 93.

[Bramoullé *et al.*, 2004] Bramoullé, Y., López-Pintado, D., Goyal, S. & Vega-Redondo, F. (2004). Network formation and anti-coordination games. *International Journal of Games Theory*, **33**, 1–19. Cited on pages 23, 52, and 96.

[Buettner *et al.*, 2006] Buettner, M., Yee, G., Anderson, E. & Han, R. (2006). X-MAC: A Short Preamble MAC Protocol For Duty-CycledWireless Sensor Networks. Tech. Rep. CU-CS-1008-06, University of Colorado at Boulder. Cited on page 127.

[CC2420, 2012] CC2420 (2012). Data sheet. `http://www.ti.com/product/cc2420`, last accessed on May 1, 2012. Cited on page 153.

[Challet & Zhang, 1997] Challet, D. & Zhang, Y.C. (1997). Emergence of cooperation and organization in an evolutionary game. *Physica A: Statistical Mechanics and its Applications*, **246**, 407 – 418. Cited on page 24.

[Challet *et al.*, 2005] Challet, D., Marsili, M. & Zhang, Y.C. (2005). *Minority Games*. Oxford University Press. Cited on page 24.

[Cigler & Faltings, 2011] Cigler, L. & Faltings, B. (2011). Reaching correlated equilibria through multi-agent learning. In Yolum, K. Tumer, P. Stone & Sonenberg, eds., *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Taipei, Taiwan. Cited on page 20.

[Claus & Boutilier, 1998] Claus, C. & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the National Conference on Artificial Intelligence*, 746–752, John Wiley & Sons Ltd. Cited on pages 27, 31, and 49.

[Couto *et al.*, 2005] Couto, D., Aguayo, D., Bicket, J. & Morris, R. (2005). A high-throughput path metric for multi-hop wireless routing. *Wireless Networks*, **11**, 419–434. Cited on page 123.

[De Hauwere, 2011] De Hauwere, Y.M. (2011). *Sparse Interactions in Multi-Agent Reinforcement Learning*. Ph.d. thesis, Vrije Universiteit Brussel. Cited on pages 80 and 131.

[de Jong *et al.*, 2008] de Jong, S., Uyttendaele, S. & Tuyls, K. (2008). Learning to reach agreement in a continuous ultimatum game. *Journal of Artificial Intelligence Research (JAIR)*, **33**, 551–574. Cited on page 49.

[De Vylder, 2008] De Vylder, B. (2008). *The Evolution of Conventions in Multi-Agent Systems*. Ph.D. thesis, Vrije Universiteit Brussel. Cited on pages 45 and 56.

[Degesys *et al.*, 2007] Degesys, J., Rose, I., Patel, A. & Nagpal, R. (2007). DESYNC: self-organizing desynchronization and TDMA on wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks (IPSN)*, 11–20, ACM, New York, NY, USA. Cited on page 128.

[Delgado *et al.*, 2003] Delgado, J., Pujol, J. & Sanguesa, R. (2003). Emergence of Coordination in Scale-Free Networks. *Web Intelligence and Agent Systems*, **1**, 131–138. Cited on pages 45, 46, 47, and 48.

[Easley & Kleinberg, 2010] Easley, D. & Kleinberg, J. (2010). *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press. Cited on page 42.

[El-Hoiydi, 2002] El-Hoiydi, A. (2002). Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks. In *IEEE International Conference on Communications*, vol. 5, 3418–3423. Cited on page 127.

[Förster, 2007] Förster, A. (2007). Machine Learning Techniques Applied to Wireless Ad-Hoc Networks: Guide and Survey. In *Proceedings of the The Third International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 365–370, IEEE, Melbourne, Australia. Cited on page 123.

[Förster & Murphy, 2007] Förster, A. & Murphy, A. (2007). FROMS: Feedback Routing for Optimizing Multiple Sinks in WSN with Reinforcement Learning. In *Proceedings of the The Third International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, Melbourne, Australia. Cited on page 123.

[Galeotti *et al.*, 2010] Galeotti, A., Goyal, S., Jackson, M.O., Vega-Redondo, F. & Yariv, L. (2010). Network Games. *Review of Economic Studies*, **77**, 218–244. Cited on pages 25 and 29.

[Galstyan *et al.*, 2005] Galstyan, A., Czajkowski, K. & Lerman, K. (2005). Resource allocation in the Grid with learning agents. *Journal of Grid Computing*, **3**, 91–100. Cited on pages 24 and 37.

[Goel, 2005] Goel, S. (2005). *Etiquette protocol for ultra low power operation in energy constrained sensor networks*. Ph.D. thesis, Rutgers University, New Brunswick, USA. Cited on page 127.

[Grenager *et al.*, 2002] Grenager, T., Powers, R. & Shoham, Y. (2002). Dispersion games: general definitions and some specific learning results. In *Proceedings of the Eighteenth national conference on Artificial intelligence*, Alpern 2001, 398–403, AAAI Press. Cited on pages 10, 23, 24, 92, 94, 96, 97, 99, 100, and 101.

[Guo *et al.*, 2001] Guo, C., Zhong, L. & Rabaey, J. (2001). Low power distributed MAC for ad hoc sensor radio networks. *GLOBECOM*, **5**, 2944–2948. Cited on page 126.

[Gutierrez *et al.*, 2002] Gutierrez, J., Naeve, M., Callaway, E., Bourgeois, M., Mitter, V. & Heile, B. (2002). IEEE 802.15.4: a developing standard for low-power low-cost wireless personal area networks. *Network, IEEE*, **15**, 12–19. Cited on page 153.

[Harsanyi & Selten, 1988] Harsanyi, J.C. & Selten, R. (1988). *A General Theory of Equilibrium Selection in Games*, vol. 1. The MIT Press. Cited on page 30.

[Hilgard, 1948] Hilgard, E.R. (1948). *Theories of Learning*. Appleton-Century-Crofts, New York, 2nd edn. Cited on page 36.

[Hill & Culler, 2002] Hill, J. & Culler, D. (2002). Mica: A wireless platform for deeply embedded networks. *IEEE micro*, **22**, 12–24. Cited on page 127.

[Ilyas & Mahgoub, 2005] Ilyas, M. & Mahgoub, I. (2005). *Handbook of sensor networks: compact wireless and wired sensing systems*. CRC Press. Cited on pages 123 and 124.

[Jennings *et al.*, 1998] Jennings, N.R., Sycara, K. & Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, **1**, 7–38. Cited on pages 1, 2, and 3.

[Jensen & Toft, 1995] Jensen, T. & Toft, B. (1995). *Graph Coloring Problems*. Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley. Cited on page 94.

[Kearns, 2007] Kearns, M. (2007). Graphical Games. *Algorithmic game theory*, 159–180. Cited on page 28.

[Kearns *et al.*, 2001] Kearns, M., Littman, M.L. & Singh, S. (2001). Graphical Models for Game Theory. *Association for Uncertainty in Artificial Intelligence*, **1**, 253–260. Cited on page 25.

[Kelley *et al.*, 1962] Kelley, H., Thibaut, J., Radloff, R. & Mundy, D. (1962). The Development Of Cooperation In The Minimal Social Situation. *Psychological Monographs: General and Applied*, **76**, 1–19. Cited on pages 37 and 47.

[Kemeny & Snell, 1969] Kemeny, J. & Snell, J. (1969). *Finite Markov chains*. VanNostrand, New York. Cited on page 66.

[Kittock, 1993] Kittock, J. (1993). Emergent conventions and the structure of multi-agent systems. In *Proceedings of the 1993 Santa Fe Institute Complex Systems Summer School*, vol. 6, 1–14, Citeseer. Cited on pages 45, 47, 48, and 57.

[Knoester & McKinley, 2009] Knoester, D.B. & McKinley, P.K. (2009). Evolving virtual fireflies. In *Proceedings of the 10th European Conference on Artificial Life*, Budapest, Hungary. Cited on page 128.

[Koulouriotis & Xanthopoulos, 2008] Koulouriotis, D. & Xanthopoulos, A. (2008). Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems. *Applied Mathematics and Computation*, **196**, 913–922. Cited on pages 34 and 99.

[Kraines & Kraines, 1995] Kraines, D. & Kraines, V. (1995). Evolution of Learning among Pavlov Strategies in a Competitive Environment with Noise. *Journal of Conflict Resolution*, **39**, 439–466. Cited on page 37.

[Langendoen, 2008] Langendoen, K. (2008). Medium access control in wireless sensor networks. *Medium access control in wireless networks*, **2**, 535–560. Cited on page 135.

[Lewis, 1969] Lewis, D. (1969). *Convention: A Philosophical Study*. Harvard University Press. Cited on pages 21, 43, 45, and 57.

[Liu & Zhao, 2010] Liu, K. & Zhao, Q. (2010). Distributed learning in multi-armed bandit with multiple players. *Trans. Sig. Proc.*, **58**, 5667–5681. Cited on pages 95 and 100.

[Liu & Elhanany, 2006] Liu, Z. & Elhanany, I. (2006). RL-MAC: a reinforcement learning based MAC protocol for wireless sensor networks. *International Journal of Sensor Networks*, **1**, 117–124. Cited on pages 129 and 139.

[Lu *et al.*, 2004] Lu, G., Krishnamachari, B. & Raghavendra, C. (2004). An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks. In *Proceedings of the 18th International Symposium on Parallel and Distributed Processing*, 224. Cited on pages 126 and 153.

[Lucarelli & Wang, 2004] Lucarelli, D. & Wang, I.J. (2004). Decentralized synchronization protocols with nearest neighbor communication. In *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys)*, 62–68, ACM, New York, USA. Cited on page 129.

[Mainwaring *et al.*, 2002] Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D. & Anderson, J. (2002). Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International workshop on Wireless Sensor Networks and applications*, 88–97. Cited on page 120.

[Martinez *et al.*, 2004] Martinez, K., Hart, J. & Ong, R. (2004). Environmental sensor networks. *IEEE Computer*, **37**, 50–56. Cited on page 121.

[Mihaylov *et al.*, 2008] Mihaylov, M., Nowé, A. & Tuyls, K. (2008). Collective intelligent wireless sensor networks. In *Proceedings of the 20th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, 169–176, Enschede, The Netherlands. Cited on page 129.

[Mihaylov *et al.*, 2009] Mihaylov, M., Tuyls, K. & Nowé, A. (2009). Decentralized learning in wireless sensor networks. In *Proceedings of the Adaptive and Learning Agents Workshop (ALA)*, Budapest, Hungary.

[Mihaylov *et al.*, 2010] Mihaylov, M., Tuyls, K. & Nowé, A. (2010). Decentralized learning in wireless sensor networks. In M. Taylor & K. Tuyls, eds., *Postproceedings of the 2nd Workshop ALA 2009, Held as Part of the AAMAS 2009 Conference. Revised Selected Papers.*, vol. 5924 of *Lecture Notes in Computer Science*, 60–73, Springer Berlin/Heidelberg, Adaptive Learning Agents edn.

[Mihaylov *et al.*, 2011a] Mihaylov, M., Le Borgne, Y.A., Tuyls, K. & Nowé, A. (2011a). Distributed cooperation in wireless sensor networks. In Yolum, K. Tumer, P. Stone & Sonenberg, eds., *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Taipei, Taiwan. Cited on pages 130 and 153.

[Mihaylov *et al.*, 2011b] Mihaylov, M., Le Borgne, Y.A., Tuyls, K. & Nowé, A. (2011b). Self-Organizing Synchronicity and Desynchronicity using Reinforcement Learning. In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence (ICAART)*, 94–103, Rome, Italy. Cited on page 130.

[Mihaylov *et al.*, 2012a] Mihaylov, M., Le Borgne, Y.A., Tuyls, K. & Nowé, A. (2012a). Decentralised Reinforcement Learning for Energy-Efficient Scheduling in Wireless Sensor Networks. *International Journal of Communication Networks and Distributed Systems*, **9**, 207–224. Cited on page 130.

[Mihaylov *et al.*, 2012b] Mihaylov, M., Le Borgne, Y.A., Tuyls, K. & Nowé, A. (2012b). Reinforcement Learning for Self-Organizing Wake-Up Scheduling in Wireless Sensor Networks. In J. Filipe & A. Fred, eds., *Postproceedings of the 3rd International Conference ICAART 2011. Revised Selected Papers.*, vol. 271 of *Communications in Computer and Information Science*, 382 – 397, Springer-Verlag, Agents and Artificial Intelligence edn.

[Mirollo & Strogatz, 1990] Mirollo, R.E. & Strogatz, S.H. (1990). Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, **50**, 1645–1662. Cited on page 128.

[Naessens *et al.*, 2010] Naessens, V., Mihaylov, M., De Jong, S., Verbeeck, K. & Nowé, A. (2010). Carebook: Assisting elderly people by social networking. In *Proceedings of the 1st International Conference on Interdisciplinary Research on Technology, Education and Communication (ITEC)*, Kortrijk, Belgium.

[Namatame, 2006] Namatame, A. (2006). *Adaptation and evolution in collective systems*. World Scientific Pub Co Inc. Cited on pages 94, 97, 101, and 102.

[Nash, 1950] Nash, J.F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, **36**, 48–49. Cited on page 17.

[Nowak & Sigmund, 1993] Nowak, M. & Sigmund, K. (1993). A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game. *Nature*, **364**, 56–58. Cited on pages 37, 47, and 60.

[Nowé *et al.*, 1998] Nowé, A., Steenhaut, K., Fakir, M. & Verbeeck, K. (1998). Q-learning for adaptive load based routing. In *Proceedings of the IEEE International Conference on Systems Man and Cybernetics*, 3965–3970, IEEE. Cited on page 123.

[OMNeT++, 2012] OMNeT++ (2012). An extensible, modular, component-based c++ simulation library and framework, primarily for building network simulators. `http://www.omnetpp.org`, last accessed on May 1, 2012. Cited on page 153.

[Owen, 1995] Owen, G. (1995). *Game Theory*. Academic Press. Cited on page 26.

[Paruchuri *et al.*, 2004] Paruchuri, V., Basavaraju, S., Durresi, A., Kannan, R. & Iyengar, S.S. (2004). Random asynchronous wakeup protocol for sensor networks. In *Proceedings of the First International Conference on Broadband Networks (BROADNETS)*, 710–717, IEEE Computer Society, Washington, USA. Cited on page 129.

[Patel *et al.*, 2007] Patel, A., Degesys, J. & Nagpal, R. (2007). Desynchronization: The theory of self-organizing algorithms for round-robin scheduling. In *Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 87–96, IEEE Computer Society, Washington, USA. Cited on page 128.

[Peeters, 2008] Peeters, M. (2008). *Solving Multi-Agent Sequential Decision Problems Using Learning Automata*. Ph.D. thesis, Vrije Universiteit Brussel, Brussels, Belgium. Cited on page 37.

[Peters, 2008] Peters, H. (2008). Extensive form games. In *Game Theory*, 197–212, Springer Berlin Heidelberg. Cited on page 15.

[Phung *et al.*, 2012] Phung, K.h., Lemmens, B., Mihaylov, M., Zenobio, D.D., Steenhaut, K. & Tran, L. (2012). Multi-agent Learning for Multi-channel Wireless Sensor Networks. In *Proceedings of the 3rd IEEE International Workshop on SmArt COmmunications in NEtwork Technologies (SaCoNet)*, Ottawa, Canada. Cited on pages 93, 119, 123, and 158.

[Posch, 1999] Posch, M. (1999). Win-stay, lose-shift strategies for repeated games-memory length, aspiration levels and noise. *Journal of theoretical biology*, **198**, 183–95. Cited on page 38.

[Robbins, 1952] Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 527–535. Cited on pages 34 and 37.

[Rousseau, 1754] Rousseau, J.J. (1754). *Discourse on Inequality*. Marc-Michel Rey, Holland. Cited on page 16.

[Santharam *et al.*, 1994] Santharam, G., Sastry, P. & Thathachar, M. (1994). Continuous action set learning automata for stochastic optimization. *Journal of the Franklin Institute*, **331**, 607 – 628. Cited on page 37.

[Schelling, 1960] Schelling, T.C. (1960). *The strategy of conflict*. Cambridge: Harvard University Press. Cited on page 22.

[Schurgers, 2007] Schurgers, C. (2007). *Wireless Sensor Networks and Applications*, chap. Wakeup Strategies, 26. Springer. Cited on pages 126 and 127.

[Segbroeck *et al.*, 2009] Segbroeck, S.V., Santos, F.C., Lenaerts, T. & Pacheco, J.M. (2009). Emergence of cooperation in adaptive social networks with behavioral diversity. In *Proceedings of the 10th European Conference on Artificial Life (ECAL)*, 434–441. Cited on page 49.

[Sen *et al.*, 1994] Sen, S., Sekaran, M. & Hale, J. (1994). Learning to coordinate without sharing information. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 426–431. Cited on page 49.

[Shapley, 1953] Shapley, L. (1953). Stochastic games. *Proceedings of the National Academy of Sciences*, **39**, 1095. Cited on page 25.

[Shih *et al.*, 2002] Shih, E., Bahl, P. & Sinclair, M. (2002). Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, 160–171. Cited on page 126.

[Shoham & Tennenholtz, 1993] Shoham, Y. & Tennenholtz, M. (1993). Co-learning and the evolution of social activity. Tech. rep., Stanford University. Cited on pages 21 and 46.

[Shoham & Tennenholtz, 1995] Shoham, Y. & Tennenholtz, M. (1995). On Social Laws for Artificial Agent Societies: Off-Line Design. *Artificial Intelligence*, **73**, 231–252. Cited on page 43.

[Shoham & Tennenholtz, 1997] Shoham, Y. & Tennenholtz, M. (1997). On the emergence of social conventions : modeling, analysis, and simulations. *Artificial Intelligence*, **94**, 139–166. Cited on pages 45, 47, 48, and 56.

[Sutton & Barto, 1998] Sutton, R.S. & Barto, A.G. (1998). *Reinforcement Learning: An Introduction*. MIT Press. Cited on pages 30, 34, 130, and 132.

['t Hoen & Bohte, 2003] 't Hoen, P. & Bohte, S. (2003). COllective INtelligence with task assignment. Coordinating choices in Multi-Agent Systems. Tech. rep., CWI. Cited on page 94.

[Tan, 1993] Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, vol. 337, 330–337, Amherst, MA. Cited on page 48.

[Taylor, 2009] Taylor, M. (2009). *Transfer in Reinforcement Learning Domains*, vol. 216 of *Studies in Computational Intelligence*. Springer-Verlag. Cited on page 132.

[Taylor *et al.*, 2011] Taylor, M., Jain, M., Tandon, P., Yokoo, M. & Tambe, M. (2011). Distributed on-line multi-agent optimization under uncertainty: Balancing exploration and exploitation. *Advances in Complex Systems*. Cited on page 95.

[Tewfik, 2012] Tewfik, A.H. (2012). Load balancing in wireless local area networks. patent US8098637. Cited on page 23.

[Thorndike, 1911] Thorndike, E. (1911). *Animal intelligence: experimental studies*. Macmillan, New York. Cited on page 37.

[Tsitsiklis, 1994] Tsitsiklis, J.N. (1994). Asynchronous stochastic approximation and q-learning. *Journal of Machine Learning*, **16**, 185–202. Cited on page 33.

[van Dam & Langendoen, 2003] van Dam, T. & Langendoen, K. (2003). An adaptive energy-efficient MAC protocol for wireless sensor networks. In *Proceedings Of The First International Conference On Embedded Networked Sensor Systems*, 171 – 180, Los Angeles, California, USA. Cited on page 126.

[Van Moffaert *et al.*, 2011] Van Moffaert, K., Van Vreckem, B., Mihaylov, M. & Nowé, A. (2011). A learning approach to the school bus routing problem. In *23rd Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, Ghent, Belgium.

[Verbeeck, 2004] Verbeeck, K. (2004). *Coordinated Exploration in Multi-Agent Reinforcement Learning*. Ph.D. thesis, Vrije Universiteit Brussel, Brussels, Belgium. Cited on page 157.

[Vickrey & Koller, 2002] Vickrey, D. & Koller, D. (2002). Multi-agent algorithms for solving graphical games. In *Proceedings of the National Conference on Artificial Intelligence*, 345–351. Cited on page 133.

[Villatoro *et al.*, 2011a] Villatoro, D., Sabater-Mir, J. & Sen, S. (2011a). Social Instruments for Robust Convention Emergence. In *Twenty-Second International Joint Conference On Artificial Intelligence (IJCAI)*, 6, Barcelona, Spain. Cited on pages 51 and 56.

[Villatoro *et al.*, 2011b] Villatoro, D., Sen, S. & Sabater-Mir, J. (2011b). Exploring the Dimensions of Convention Emergence in Multiagent Systems. *Advances in Complex Systems*, **14**, 201–227. Cited on pages 46, 47, 48, 52, 53, 62, and 68.

[Vrancx, 2010] Vrancx, P. (2010). *Decentralised Reinforcement Learning in Markov Games*. Ph.D. thesis, Vrije Universiteit Brussel, Brussels, Belgium. Cited on pages 37 and 131.

[Warneke *et al.*, 2001] Warneke, B., Last, M., Liebowitz, B. & Pister, K. (2001). Smart Dust: communicating with a cubic-millimeter computer. *Computer*, **34**, 44–51. Cited on page 4.

[Watkins, 1989] Watkins, C. (1989). *Learning from delayed rewards*. Ph.D. thesis, University of Cambridge, England. Cited on page 31.

[Werner-Allen *et al.*, 2005] Werner-Allen, G., Tewari, G., Patel, A., Welsh, M. & Nagpal, R. (2005). Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys)*, 142–153, ACM, New York, USA. Cited on page 128.

[Wolpert & Tumer, 2002] Wolpert, D.H. & Tumer, K. (2002). Collective Intelligence, Data Routing and Braess's Paradox. *Journal of Artificial Intelligence Research*, **16**, 359–387. Cited on page 94.

[Wolpert & Tumer, 2008] Wolpert, D.H. & Tumer, K. (2008). An introduction to collective intelligence. Tech. Rep. NASA-ARC-IC-99-63, NASA Ames Research Center. Cited on page 129.

[Woo *et al.*, 2003] Woo, A., Tong, T. & Culler, D. (2003). Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, 14–27, ACM. Cited on page 123.

[Wooldridge & Jennings, 1995] Wooldridge, M. & Jennings, N.R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review*, **10**, 115–152. Cited on page 2.

[Ye *et al.*, 2004] Ye, W., Heidemann, J. & Estrin, D. (2004). Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*, **12**, 493–506. Cited on pages 124, 126, and 153.

[Yick *et al.*, 2008] Yick, J., Mukherjee, B. & Ghosal, D. (2008). Wireless sensor network survey. *Computer Networks*, **52**, 2292–2330. Cited on pages 121 and 124.

[Young, 1993] Young, H.P. (1993). The Evolution of Conventions. *Econometrica*, **61**. Cited on pages 45, 48, and 62.

[Zhao & Guibas, 2004] Zhao, F. & Guibas, L. (2004). *Wireless Sensor Networks: An Information Processing Approach*. The Morgan Kaufmann Series in Networking, Morgan Kaufmann. Cited on page 121.

[Zheng *et al.*, 2003] Zheng, R., Hou, J.C. & Sha, L. (2003). Asynchronous wakeup for ad hoc networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc)*, 35–45, ACM, New York, USA. Cited on page 129.

Index

Z

Decentralized Coordination in Multi-Agent Systems

Mihail Mihaylov

Many computer systems are comprised of multiple entities (or agents) with common objectives. Individual agents are often restricted in their capabilities and have only limited knowledge of their environment. However, the group as a whole is capable of executing more complex tasks than a single agent can perform. Individual agents, therefore, need to coordinate their activities in order to meet the design objectives of the entire system.

Our main contribution is to propose a simple decentralized reinforcement learning approach, called Win-Stay Lose-probabilistic-Shift, that allows highly constrained agents to efficiently coordinate their behavior imposing minimal system requirements and overhead. We demonstrate that global coordination can emerge from simple and local interactions without the need of central control or any form of explicit coordination.

WWW.VUBPRESS.BE

9 789057 181429