Computational Modeling Lab
Department of Computer Science
Faculty of Sciences
Vrije Universiteit Brussel

Decentralised Reinforcement Learning in Markov Games

Peter Vrancx

Dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Sciences

supervisors:
Prof. Dr. Ann Nowé
Dr. Katja Verbeeck

Print: Silhouet, Maldegem

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 5487 715 8
NUR 984
Legal Deposit D/2010/11.161/036

To Petra and Ruben.

Committee Members:

Prof. dr. Ann Nowe Vrije Universiteit Brussel (supervisor)
Prof. dr. Dirk Vermeir Vrije Universiteit Brussel
Prof. dr. Bernard Manderick Vrije Universiteit Brussel
Prof. dr. Luc Steels Vrije Universiteit Brussel
dr. Katja Verbeeck KaHo Sint-Lieven (KU Leuven) (supervisor)
Prof. dr. Gerhard Weiss Maastricht University
Prof. dr. Martin Riedmiller Albert-Ludwigs-University Freiburg
Prof. dr. Marco Saerens Université Catholique de Louvain

Abstract

This dissertation introduces a new approach to multi-agent reinforcement learning. We develop the Interconnected Learning Automata for Markov Games (MG-ILA) algorithm, in which agents are composed of a network of independent learning units, called learning automata (LA). These automata are relatively simple learners, that can be composed into advanced collectives and provide very general convergence results.

An advantage of our approach is that it has very limited information requirements, since the automata coordinate using only their own reward signals. This allows us to view a multi-state learning problem as a single repeated normal form game from classical game theory. We use this observation to develop a new analysis for multi-agent reinforcement learning. Using this method we show the convergence of MG-ILA towards pure equilibrium points between agent policies.

We then proceed by investigating the properties of our algorithm and proposing a number of extensions. Using results from evolutionary game theory, we analyse the learning dynamics of our system and develop a novel visualisation method for multi-state learning dynamics. We also show how an updated learning rule is able to overcome local optima and achieve global optimality in common interest problems. In conflicting interest cases, we show how this technique can be combined with a simple coordination mechanism to ensure a fair distribution of payoffs amongst all agents.

We conclude the dissertation by examining some possible applications of our system. We start by applying MG-ILA to multi-robot navigation and coordination simulations. We show that, even when only partial state information is available, the algorithm still finds an equilibrium between robot policies. We also consider applications in the field of swarm intelligence. We demonstrate how our system can be used as a model for systems using stigmergetic communication. In these settings agents exchange information by sharing local pheromone signals. Our model allows us to apply our game theory based analysis to this class of algorithms, providing a new method to analyse the global results of local pheromone interactions.

Samenvatting

Deze thesis introduceert een nieuw algoritme voor reinforcement leren met meerdere agenten. Dit algoritme, dat we Interconnected Learning Automata for Markov Games (MG-ILA) noemen, stelt agenten voor door een netwerk van onafhankelijke, lerende automaten. Deze leerautomaten kunnen worden samengesteld tot geavanceerde collectieven en bieden ook uitgebreide convergentieresultaten.

Een belangrijk voordeel van onze aanpak is dat leerautomaten maar een minimale hoeveelheid informatie nodig hebben om hun gedrag te coördineren. Elke automaat in het collectief baseert zich enkel op zijn individuele beloningen om een strategie te leren, zonder gegevens over de andere automaten te vereisen. Deze eigenschap laat ons toe om de multi-toestand leerproblemen die we in dit werk behandelen te modelleren als een spel uit de klassieke speltheorie. Dit leidt tot een nieuwe analysemethode voor reinforcement leren met meerdere agenten, die we gebruiken om de convergentie van MG-ILA naar pure equilibriumpunten tussen agentstrategieën aan te tonen.

Vervolgens, onderzoeken we de eigenschappen van ons algoritme en stellen we een aantal uitbreidingen voor. Door gebruik te maken van bestaande resultaten uit de evolutionaire speltheorie, kunnen we de dynamica van het algoritme onderzoeken en een originele visualisatiemethode voorstellen. Voor problemen waar agenten hetzelfde doel nastreven, introduceren we ook een alternatieve leerregel die het algoritme toelaat te ontsnappen uit lokale optima en het globaal optimum te bereiken. In problemen waar de agenten conflicterende doelen hebben, gebruiken we deze techniek in combinatie met een eenvoudig coördinatiemechanisme om een gelijkwaardige verdeling van beloningen voor de agenten te bekomen.

Tenslotte sluiten we de verhandeling af met een aantal mogelijke toepassingen voor ons raamwerk. We passen MG-ILA toe in simulaties van coördinatieproblemen met mobiele robots. We tonen aan dat, zelfs als de robots onvolledige informatie over het systeem hebben, het algoritme nog steeds een equilibrium vindt. Hierna, bekijken we toepassingen in het swarm intelligence domein. We demonstreren hoe ons systeem kan dienen als model voor algoritmen waar agenten informatie uitwisselen door het verspreiden van lokale feromoonsignalen. Dit laat ons toe om onze speltheorie-analyse ook op deze klasse van algoritmen toe te passen en resulteert in een nieuwe methode voor bepalen van de globale resultaten van lokale feromooninteracties.

Acknowledgements

This dissertation was realised thanks to the encouragement and support of many people. I am indebted to following people for their contributions, both scientific and otherwise:

The research contained in this dissertation was funded by a Ph.D grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT Vlaanderen).

I am also very grateful to my supervisors Katja Verbeeck and Ann Nowé, who provided me with the opportunity to do a PhD. They have continuously guided me during the years of research and introduced me to most of the techniques used in this work.

Further, I would like to thank the members of the examination committee, who took the time to read this dissertation and provided many helpful suggestions and constructive criticisms.

A special thanks also goes to Karl Tuyls, who was kind enough to invite me to visit the university of Maastricht. This stay led to many interesting discussions and a fruitful collaboration. His guidance was essential to the realisation of Chapter 5 of this dissertation.

During my PhD I have also had the privilege to collaborate with several excellent researchers, all of whom have significantly contributed to the work I present here: Maarten Peeters, Yann-Michaël De Hauwere, Steven de Jong, Ronald Westra and Ville Könönen.

I would also like to thank all other current and former colleagues at CoMo, both for creating a pleasant working environment, as well as for providing many fun distractions from work. Thank you, Bernard, Yann-Aël, Tom, Stijn, Kris, David, Yifei, Walter, Pasquale, Ruben, Bart, Nyree, Feng, Yailen, Mike, Allan, Abdel, Jonatan, Sven, Bert, Saba, Mohamed, Pieter, Anne, Sam, Johan, Steven, Yoshi and Bram.

Finally, I would like to thank my family for their never-ending support. I especially want to mention my parents, who have always encouraged me during my education, and of-course Petra and Ruben for their love and support.

Contents

List of Figures

List of Tables

Glossary

abbreviation	meaning
CALA	continuous action learning automaton
EGT	evolutionary game theory
ESS	evolutionary stable strategy
ESRL	exploring selfish reinforcement learning
ILA	interconnected learning automata
MAS	multi-agent system
MARL	multi-agent reinforcement learning
MG	Markov game
MDP	Markov decision process
MMDP	multi-agent Markov decision process
$L_{R-\epsilon P}$	linear reward-ϵ penalty
L_{R-I}	linear reward-inaction
L_{R-P}	linear reward-penalty
LA	learning automaton
PLA	parameterised learning automaton
RD	replicator dynamic
RL	reinforcement learning

symbol	meaning
general:	
λ	learning rate
γ	discount factor
$\vec{p}$	probability vector (learning automata)
S_r	(r-1)-dimensional unit simplex
$\mu(A)$	set of probability distributions over set A
$Pr\{A\}$	probability of A
$\mathcal{P}(A)$	power set of the set A
single agent:	
π	policy
$\pi(s,a)$	action probability for action in state s according to policy π.
$\pi(s)$	action selected in state s under deterministic policy π.
$A(s)$	action set in state s
$V^\pi(s)$	value of state s under policy π
$Q^\pi(s,a)$	Q-value of state action pair under π
$d^\pi(s)$	stationary probability of state s under policy π
J^π	expected average reward under π
$R(s,a,s')$	one step reward for transition from s to s' using action a
$R^\pi(s)$	expected reward in state s under π
$T(s,a,s')$	transition probability from s to s' using action a
$T^\pi(s,s')$	transition probability induced by π
multi-agent:	
n	number of agents playing (Markov) game
$A_i(s)$	agent i's action set in state s.
$\vec{a}$	joint action $(a_1,\ldots,a_n)$, specifying action for each agent.
$\vec{\pi}$	joint policy $(\pi_1,\ldots,\pi_n)$, specifying a policy for each agent
$\vec{\pi}(s,\vec{a})$	probability of playing joint action $\vec{a}$ in state s under $\vec{\pi}$
$\vec{\pi}(s)$	joint action played in state s under joint policy $\vec{\pi}$, consisting of deterministic policies for all agents
$R_i(s,\vec{a},s')$	immediate reward received by agent i for transition from s to s' under joint action $\vec{a}$

Chapter 1

Introduction

This dissertation studies reinforcement learning (RL) in multi-agent systems. Reinforcement learning is a technique that allows an agent to learn optimal behaviour through trial-and-error interactions with its environment. By repeatedly trying actions in different situations the agent can discover the consequences of its actions and identify the best action for each situation.

However, when multiple learners use this approach in a shared environment traditional RL approaches often fail. In the multi-agent setting common assumptions that are needed for convergence are often violated. Even in the simplest case where agents share a non-changing environment and need to learn an action for only one situation, many new complexities arise. When agent objectives are aligned and all agents try to maximise the same reward, coordination is still required to reach the global optimum. When agents have opposing goals, a clear optimal solution may no longer exist. In this case we typically look for an equilibrium between agent strategies. In such an equilibrium no agent can improve its payoff when the other agents keep their actions fixed.

When we assume a dynamic environment, in addition to multiple agents, the problem becomes even more complex. Now agents do not only have to coordinate, they also have to take into account the current state of their environment. This problem is further complicated by the fact that agents typically have only limited information about the system. In general, they cannot observe actions of other agents, even though these actions have a direct impact on their rewards and their environment. In the most challenging case, an agent may not even be aware of the presence of other agents, making the environment seem non-stationary. In order to develop a suc-

cessful multi-agent approach, all these issues need to be addressed.

Despite the added complexity, a real need for multi-agent systems exists. Often systems are inherently decentralised, and a single agent learning approach is not feasible. This situation may arise because data or control is physically distributed, because several different objectives are present or simply because a single centralised controller requires to much resources. Examples of such systems are multi-robot set-ups, decentralised network routing, distributed load-balancing, electronic auctions, traffic simulations and many others.

Different approaches toward independent multi-agent learning already exist. However, most have only limited applicability or lack theoretical results. Many approaches only consider stateless environments and cannot deal with an environment that is affected by the agents' actions. Others are restricted to specific payoff structures or require unrealistic amounts of information about the system to assure convergence. Some may lack convergence guarantees altogether and are developed experimentally for very specific applications.

In this dissertation we propose a new approach for learning in multi-agent environments. Rather than creating monolithic agents, we use agents that are composed of a network of independent learning units, called learning automata (LA). Using these automata as building blocks for our agents has several advantages. Both the long term convergence properties and the learning dynamics of these automata are well studied [TS04, Tuy04]. Furthermore, while they are relatively simple learners, LA offer a lot of flexibility in terms of interconnections between the automata, making it possible to design agents consisting of a large interconnected collective of automata. These advantages are further complemented by the fact that automata algorithms have low information requirements and have even been shown to work in partially observable environments [VVN08b, PM97], making them ideally suited for multi-agent systems.

In the following sections we give more information on the learning setting we use and assumptions we made. Next, we describe our approach in short. Finally, a detailed outline for the remainder of the dissertation is added.

1.1 Intelligent Agents

Before introducing our problem setting, we will take a closer look at the concept of an *agent*. Below we give a short discussion of basic agent con-

cepts, based on [Wei99].

While most researchers in artificial intelligence have some notion of what exactly constitutes an agent, no generally agreed upon definition exists. Different fields of AI, may associate different properties with an agent. Despite this discrepancy there are some general properties most people agree an agent should have. These are summarised in the following definition from [JSW98]:

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

This definition deliberately neglects to define the *environment* in which an agent is situated. This is because agents can be used in a wide range environments with very different properties. In the next section we will discuss the environment models considered in this dissertation. The *autonomy* requirement for an agent, is used here to indicate that agents can act without the intervention of a human controller. Thus, the agent has control over its own internal state and its behaviour.

The definition above, is ofcourse very broad and allows us to view almost any control system as an agent. The definition covers simple systems such as a thermostat controlling a room's temperature or software daemons running background operating system processes. When considering agents used in AI research, we typically want to restrict our attention to learning agents that posses certain qualities that can be interpreted as intelligence. Wooldridge [Woo09] specifies some additional properties that are often associated with an *intelligent* agent:

- *reactivity:* the ability to respond to a timely fashion to changes in the environment.
- *pro-activeness:* to exhibit goal-directed behaviour and take the initiative to satisfy design requirements.
- *social ability:* being capable of interaction with other agents (or humans).

Designing a system with these properties can be more challenging than it appears at first. Consider for example the reactivity and pro-activeness properties. Simple procedures in any program could be construed as examples of goal-directed behaviour, being executed when their pre-conditions

hold in order to achieve some post-condition. However, this view relies on an environment which is static and fully known. In a (multi-)agent system, we typically have to deal with with changes in the environment to which an agent must react, i.e. the agent also needs to be reactive. The simple thermostat described above, can be interpreted as such a reactive system, continually reacting to changes in the room's temperature. This system however, is purely reactive. It does not systematically try to achieve its goal, it only reacts to external changes. We want our agents to be able to focus on a goal and take initiative to reach it. Thus the challenge in agent systems is to find a good balance between reacting to changes and trying to perform goal-directed behaviours. Below we will list some architectures that implement these properties. Our interest is in learning systems, where the agents try to identify the consequences of their actions, so that they can perform those actions that lead to the desired outcome.

Many different architectures can be considered in order to implement an agent's internal functions. Logic based agents rely on symbolic representations and logic deduction for their decision making, purely reactive agents implement a direct mapping from situations to actions, while belief-desire-intention (BDI) agents keep internal data-structures representing representing the agents beliefs, desires and intentions. In this dissertation, however, we will focus on reinforcement learning agents, as described in [BS98].

Reinforcement learning agents learn a mapping from environment situations to actions (called a *policy*) by repeated interaction with their environment. Depending on the algorithm used, an RL agent keeps an internal state that can be, for example: a value function representing expected future rewards of actions, some (possibly parameterised) representation of a policy, or even a complete environment model. Most RL agents rely on techniques from stochastic approximation or classical control theory to learn this internal state and use it to select actions. Chapter 3 includes an overview of common RL techniques. In the next section we take a closer look at the problems our agents will deal with.

1.2 Learning in Markov games

The problems considered in this dissertation can be classified as shown in Figure 1.1. On the one hand we can distinguish between single state (also called stateless) systems and multi-state systems. On the other hand we can also classify problems as either single agent or multi-agent. In a single state

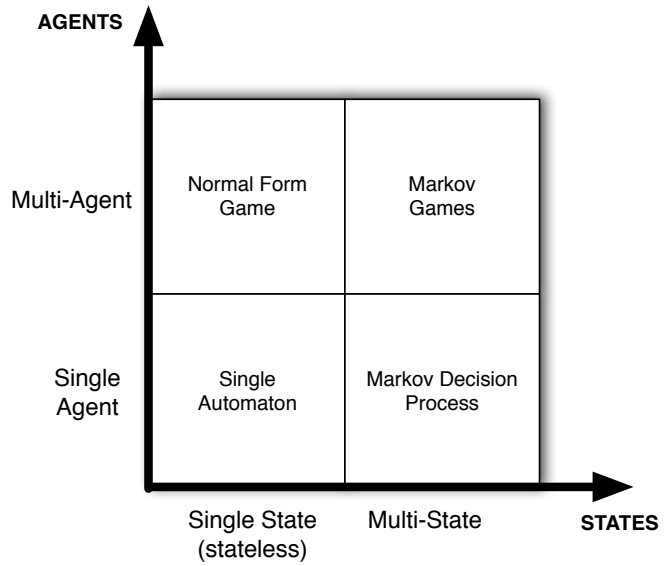


Figure 1.1: Overview of the learning settings considered in this dissertation.

system the agent has to learn a single strategy in order to optimise its reward. The environment can be stochastic, but expected rewards for actions are stationary and do not change over time. In a multi-state system, however, actions result in both a reward and a state transition. The expected reward of actions now depends on the state in which the action was taken. Agents can even have different action sets, depending on the current state. The agent's task is now to map each system state to an appropriate strategy. In doing so, the agent has to take into account not only the expected reward for an action, but also the future states the action may lead to. The single agent - multi-agent dimension distinguishes between systems with a single learner and those in which multiple learners are present in the same environment (either stateless or multi-state). Below we give a short description of these settings.

Reinforcement Learning was originally developed for Markov Decision Processes (MDPs) [BS98]. It allows a single agent to learn a policy that maximises a possibly delayed reward signal in a stochastic stationary environment. RL guarantees convergence to the optimal strategy as long as the

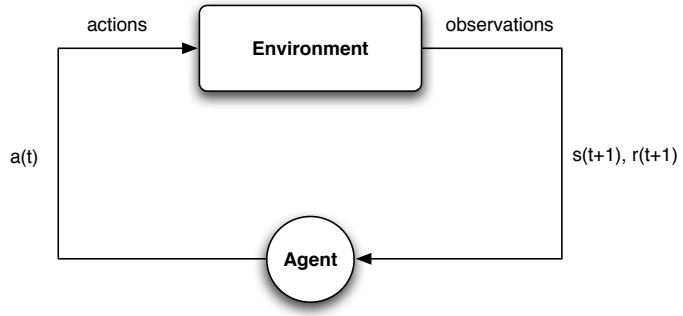


Figure 1.2: Schematic representation of single agent reinforcement learning.

agent can sufficiently experiment and the environment in which it operates has the Markov property [1].

A single agent RL setting consist of a set of system states, a set of agent actions and a function specifying the rewards the agent receives for performing an action in a certain system state. Interactions between the agent and its environment take place at discrete *time steps* or *stages* $t = 1, 2, \ldots$. At each stage t the agent selects an action $a(t)$, which serves as input to the environment. The response an agent receives consists of 2 pieces of information: an immediate reward $r(t+1)$resulting from the action and a description $s(t+1)$ of the new system state after the action is performed. The agent's goal is to learn a *policy*, mapping each possible state of the system to an action in order to maximise the long term expected reward.

When we allow multiple agents to act in the same environment, the preconditions for convergence of RL algorithms are generally no longer satisfied. In this setting both agent rewards and the global system state depend on the actions taken by all agents. As such, the learning agents not only have to deal with the stochasticity of the environment, but also have to take into account other agents with whom they might need to coordinate or even compete. Moreover, agents have individual reward functions, and in general, it is impossible to maximise the rewards for all agents simultaneously. In order to deal with the additional complexity caused by multiple agents, the standard RL framework of Markov decision processes is extended to the formal multi-agent setting of Markov games.

[1] A system has the Markov property if the present state predicts future states as well as the whole history of past and present states do, meaning that the system is memoryless.

In a multi-agent RL system, n agents are present in the same environment. In the simplest case we consider a non-changing (i.e. stateless) environment, and arrive at the repeated normal form game model from game theory. A game models strategic interactions between a set of participants called *players*. Each player has a set of actions he can perform and attempts to optimise a payoff function. The payoff he receives, however, is determined not only by his own action but also by those of the other players. Interactions between the players take place at discrete time steps $t = 1, 2, \ldots$ The players independently select an action from their private action set, using a set of associated action probabilities, also called a *strategy*. The *joint action* resulting from these independent action selections, triggers an individual payoff for each player. Based on this payoff, players can then update their strategies before the next round of the game begins. The main solution concept for these games is the *Nash equilibrium*. In a Nash equilibrium no player can increase his expected payoff by changing his current strategy while other players keep their strategies constant.

In the general multi-agent setting of Markov games, we combine multiple agents with a changing environment consisting of a set of possible states. At every time step each agent individually selects an action. The input to the environment at time t is now the vector $\vec{a}(t) = (a_1, \ldots, a_n)$ containing the joint action of all agents. The result is again a transition to a new system state $s(t+1)$ and an immediate reward $r_k(t+1)$, individual to each agent k. This setting can be seen as a set of games (one for each system state), with the agents transitioning between the games based on their joint actions. The main difference with single agent reinforcement learning is that the both the system state and the individual rewards now depend on all agents. Agents in this setting still attempt to learn a policy mapping states to actions, but the ultimate outcome depends on the joint policy, i.e. the combination of the policies of all agents in the system. Since the problems can be conflicting interest, we now consider Nash equilibria between the agent policies as the solution concept. A Nash equilibrium between policies can be defined as the situation where no agent can obtain a higher reward by switching policies (i.e. possibly changing strategies in multiple states simultaneously), when all other agents keep their policies fixed.

Different approaches can be taken with regard to learning in the multi-agent set-up, depending on the information available to the agents. One can assume that agents have full access to all system information and as such know the system state as well as the actions and rewards of all other agents. This allows agents to model their opponents and to estimate rewards for the combinations of agent actions. Many recent algorithms take

this approach, letting their agents learn over the joint actions instead of individual agent actions. However, assuming that agents have all this information at their disposal is often unrealistic. In a fully distributed system an agent might not be in a position to observe the environment interactions of all other agents. In this case, providing the agent with this information may result in a costly communication overhead. Therefore, other approaches deal with situations in which only limited information is available to the agents. In such settings the agents receive only partial information on the state, action or reward variables. In Chapter 4, we give an overview of current multi-agent RL, classified according to their information requirements.

In this dissertation we focus on the most restrictive case, in which the agents observe only the current system state, their own action and their individual reward. From the point of view of a single agent it may even seem as if the agent is alone in the environment. The influences of other agents can only be observed through their effects on rewards and transitions. In Chapter 7 we go even further, and allow agents only partial information on the system state. Despite these restrictions, we will demonstrate that our LA based approach is able to achieve coordination between the agents in a wide range of possible settings.

1.3 Assumptions

Before describing our approach to learning in Markov games, we describe some common assumptions and important concepts we consider in our learning setting.

- *discrete time:* As mentioned above, we consider systems in which the agents are required to select at discrete time steps. We do not consider continuous time systems or issues that arise from discretising such systems.

- *stochasticity:* The environment in which agents operate can be stochastic. This means that both state transitions and rewards do not have fixed values, but are drawn from a (stationary) distribution. As such the same (joint) action may generate different results.

- *state dependent environments:* All approaches considered in this work are applicable to general multi-state problems. By this we mean that the environment can assume several different states and both the reward and transition probabilities for a certain action depend on the

current state. While these probabilities can vary from state to state, we do assume that given the current state, these probabilities are stationary and do no vary over time. As was already mentioned above, the special case in which the system consists of only a single state is referred to as *stateless*.

- *finite problems:* For all problems we consider, we assume that both the state space and the agents' action sets are finite.

- *unknown environment:* As is common in RL, we assume that the environment in which agents operate is initially unknown. The agents do not have any model or any other form of background knowledge for their surroundings. Moreover, the agents cannot observe the transition or reward probabilities for environment states, only the transition that actually occurs and the reward generated for this transition are observed. As such the only way to gather this information is through repeated experimentation.

- *exact state observation:* While the environment is unknown, we do assume that agents have perfect observation of the current system state i.e. we do not consider possibly noisy observations of the system state. In Chapter 7 we do consider partial state observation, but again these observations are noise free.

- *common or conflicting interest:* The problems in this dissertation can be broadly classified as either common or conflicting interest. We consider a problem to be common interest when agents goals are aligned and all agents prefer the same joint actions. Problems where agents have different preferences for certain joint actions are referred to as conflicting interest. In general, agents can still have different reward functions in common interest problems (as long as they agree on the ordering of joint actions), but the only common interest problems we consider are identical payoff games, i.e. problems where all agents have the same reward function.

- *fairness:* In Chapter 6 we consider fair solutions. By this we mean that we attempt to equalise the agents' average rewards over time, while still allowing each agent to periodically obtain its preferred learning outcome.

- *distributed system:* All methods are designed to operate in a distributed manner. As such agents can run on separate systems and do not re-

quire a centralised implementation. The only way in which the agents are connected, is by the influences they exert on their shared environment and each other's rewards.

- *homogeneous agents:* The both the empirical and theoretical results demonstrated in this dissertation always assume homogeneous agents. By this we mean that all agents use identical learning algorithms. For the theoretical results this also implies that convergence results only hold when the algorithm is used in self-play.

- *independent agents:* All agents are considered to be fully independent. By this we do not only mean that they do not share a centralised control, but also that they act without information on the rewards and actions of other agents. Again with exception of the communicating agents in Chapter 6, agents in our approach always act without any knowledge of the existence of other agents.

1.4 Our Approach

All algorithms which are developed in this work share the common element that they rely on learning automata as basic building blocks. A single learning automaton consists of a probability vector which governs action selections and a learning rule which is used to update these probabilities when feedback is received. These comparatively simple learning units can be composed into larger networks in order to represent more complex agents. We employ such networks to develop an algorithm for general sum Markov games. This algorithm is then extended to provide additional functionality in order to achieve optimality in common interest problems, learn fair reward divisions or deal with partial state information.

The starting point for our Markov game algorithm is a learning automata based system for solving MDPs, described in Chapter 3. This algorithm which we call MDP-ILA (interconnected learning automata for MDPs), associates a single learning automaton with each possible system state. The agent is then represented by this network of LA, in which control passes from one automaton to the next when state transitions are made. Coordination between the automata is achieved based solely on the feedback signal, which consists of an estimate of the average reward received between 2 subsequent activations of an automaton.

We extend the MDP algorithm to an algorithm for Markov games, which we denote MG-ILA. The extended algorithm represents each agent by a

separate network of LA. This results in one automaton being associated to each system state, for each agent. When a state transition occurs, control now passes from one set of automata to the next. Automata again coordinate using only the reward signal, which is still the same for automata representing the same agent, but can differ for automata representing different agents.

One of the most interesting aspects of the LA based approach is that the interactions between automata can be viewed as a repeated normal form game. This approximation is a powerful tool for analysing the convergence properties of LA based algorithms. For the single agent MDP-ILA case, this results in a game in which the LA in each state are the players. As such a play in this game selects an action for each state and represents a deterministic policy for the MDP. Using the properties of MDPs, one can show that the equilibria in this game correspond exactly to the optimal policies of the MDP [WJN86]. Together with existing equilibrium convergence results for LA, this establishes the optimality of MDP-ILA.

When we move to the Markov game framework and the corresponding MG-ILA algorithm, we can still apply the game based analysis. Building on the approach described above for MDPs, we develop a corresponding analysis for Markov games. We start out by considering the approximating automata game, but also show that a Markov game can be analysed as a game at different levels, yielding additional insights to the problem under study.

When we approximate the interactions between LA, the result is a game where we have a player for each possible agent-state combination. Plays in this game assign an action to each state, for each player and thus correspond to a (deterministic) joint policy in the Markov game. We show that an equilibrium point of this approximating automata game now correspond to an equilibrium between agent policies. We combine this result again with the equilibrium convergence of LA to establish equilibrium convergence results for MG-ILA.

In addition to the automata game view which is the straightforward extension of the MDP approximation, we show that Markov games can be analysed at different levels using similar normal form game approximations. We demonstrate how common interest Markov games can be treated from a centralised control perspective in the superagent view. This view associates a player with each system state and simulates the situation where agents have a shared, central control. While this solution method is usually unfeasible, the superagent view demonstrates the results that could be obtained by applying centralised control to the problem. This view also

illustrates the problems that can arise from miscoordination when moving to a distributed approach.

Another view we consider is the so called agent view. Here players in the approximating game correspond to the agents in the Markov game. The entire Markov game is then seen as a repeated game in which players' actions correspond to their deterministic policies in the Markov game. This view was considered before in [VNP04], but only for tree-structured Markov games. It provides a convenient tool to analyse interactions between agent policies and allows us to reason about the Markov game at a higher level, by considering policies instead of individual state strategies. The usefulness of this approach is clearly demonstrated in Chapter 6, where we consider a class of non-stationary policies. These policies can be seen as mixed strategies in the agent view of the Markov game, and cannot be expressed by state-action probabilities only. We show how these policies can be implemented using learning automata and demonstrate their usefulness for obtaining a fair reward division among agents.

Another application of this view is shown in Chapter 7, where we consider Markov games in which not all state information is visible to the agents. Here the agents cannot uniquely identify states and considering equilibria in the normal set of policies becomes meaningless, as agents might not be able to express the necessary policies. However, by considering the agent view game consisting only of expressible policies we can still describe the equilibrium points that can be learned by the agents, and demonstrate convergence to these points.

In the following section we conclude this introduction with a more detailed overview of the contents of this dissertation.

1.5 Outline of the Dissertation

The simplest learning setting we consider is the single agent, stateless environment setting. This setting is closely related to the *n-armed bandit* problem from machine learning literature. We consider a single learner (in our case a learning automaton), who at every time step must choose an action to perform. Its goal is to identify the optimal action through repeated experimentation. This problem is considered in Chapter 2. We will focus on how learning automata deal with this setting and show how different automata learning schemes can be evaluated. When we extend the problem to include multiple-agents, we arrive at the repeated (normal form) game setting from classical *game theory*. In a repeated game, rewards for the agents

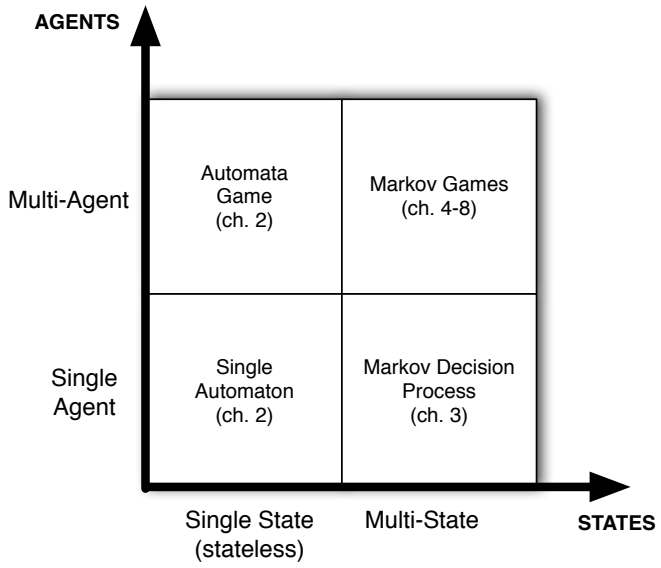


Figure 1.3: Outline of the dissertation.

depend on the joint action selected by all agents. As such the agents have to discover rewards as well as coordinate with other agents.

When moving to the single agent, multi-state setting, we arrive at Markov decision processes, the traditional single agent reinforcement learning setting which is also described above. In Chapter 3 we describe several traditional RL algorithms, that can be used to solve these MDPs. We will also demonstrate how a network of learning automata can be used to learn an optimal policy in MDPs. Moreover, we explain how the automata approach leads to a non-traditional view of MDPs, by transforming the multi-state, single learner problem into a stateless, multi-learner situation, i.e. a repeated game. This observation will be the basis for the contributions we introduce in subsequent chapters.

Next, in Chapter 4 we move to the full Markov game case. This multi-agent, multi-state framework is the most general setting we consider in this dissertation, and contains the previous 3 settings as special cases. In Chapter 4 we introduce the interconnected learning automata for Markov games (MG-ILA) algorithm, which is our learning automata based approach for Markov games. We also extend the limiting game analysis of Chapter 3

to the Markov game case, and show the convergence of MG-ILA to pure equilibrium points. We previously described the MG-ILA algorithm and its properties in [VVN08a, VNVP08, VVN08b].

In the following chapters we consider a number of extensions to MG-ILA. We start out by studying the dynamics of MG-ILA in Chapter 5. Based on existing work using *evolutionary game theory* to study learning dynamics in repeated games, we develop a novel framework for studying learning dynamics in Markov games. Using the limiting game view of Chapter 4 we also show how to visualise the dynamics of MG-ILA in simple 2-state problems. This chapter is based on our work in [VTWN08].

In Chapter 6 we show how we can extend the basic MG-ILA algorithm, in order to create more advanced solution methods. We start out by showing how we can use a different learning scheme in order to obtain global optimal convergence in common interest problems. We then move to general Markov games and introduce a solution concept which focuses on assuring a fair reward distribution among the agents. By allowing limited communication, agents can correlate their policies in order to alternate between outcomes preferred by different agents. We show that this approach can equalise the average payoff of agents, while avoiding the lower payoff some agents would receive in an equilibrium outcome. This chapter is based on results we published in [VVN07d]

Finally, in Chapters 7 and 8 we consider some possible applications of MG-ILA. In Chapter 7 we focus on coordination between mobile robots. An additional difficulty in this setting is that we no longer assume that the agents have full knowledge of the system state. Instead we consider a more realistic setting in which agents learn using a local state, i.e. their own location in the environment. Despite this restriction, we can still show that agents find an equilibrium between their policies. Chapter 7 summarises our results from [VVN08b, VVN07a, VVN07c]

In Chapter 8 we use MG-ILA as the basis for a model for stigmergetic algorithms. In these algorithms multiple agents coordinate their behaviour and share information using indirect, environment mediated communication. This means that rather than direct message passing, agents communicate by locally altering their environment. In the case we consider, they do this by leaving pheromone trails. We model this pheromone system by embedding learning automata in the environment, and allowing multiple agents to use and update the same automata. This model allows us to view stigmergy as a special case of MG-ILA and to apply the same convergence analysis as in earlier chapters. We also described the results in this chapter in [VVN07b, VVNar]

We conclude the dissertation in Chapter 9, where we give an overview of the contributions that were made and list some possible future work. This is followed by a short appendix which provides some background material on dynamical systems theory.

Chapter 2

Automata Learning in Stateless Environments

This chapter introduces the basic learning automaton that we use throughout this dissertation. We explain how these automata learn in stateless environments, both in single as well as in multi-automata settings. In the single automaton case the automaton's goal is to maximise its expected reward over a set of available actions. We show how an automaton's behaviour can be evaluated and give results for the basic automata update schemes.

After examining the single automaton case, we introduce some basic notions of game theory and show how normal form games can be used to implement multi-automata systems. In these systems rewards depend on the actions of all automata. If all automata receive the same reward, they need to coordinate to optimise their rewards. When automata have individual rewards an equilibrium is sought. In an equilibrium, no automaton can improve its rewards without another automaton also changing its action. We conclude with an overview of convergence results for automata learners in games. References for this chapter are: [NT89, TS04, OR99]

2.1 Single Automata systems

We start out by considering a single learner (in our case an automaton) interacting with its environment. This interaction takes the form of a feedback configuration in which the learner tries actions and receives a scalar feedback signal in return. In this chapter we only consider relatively simple stateless environments. In this kind of environment each action has a fixed reward probability which is kept stationary during the entire learn-

ing period. The learner's objective is to learn which action has the highest expected reward.

In the following sections we formalise the notions of a learning automaton and an environment. We also introduce a number of automata learning schemes and show how their behaviour can be evaluated in the single automaton setting.

2.1.1 Learning Automata

Learning automata have their roots in mathematical psychology [BM55]. A learning automaton formalises a simple learning unit, that attempts to learn an optimal action through repeated interaction with its environment. While early automata designs relied on internal state transitions to learn responses [Tse61], we will only consider so called variable structure automata. These automata keep a set of action probabilities that are updated at every stage using a reinforcement scheme. Formally, a learning automaton is defined as follows:

Definition 2 *A variable structure learning automaton is described by a quadruple $\{A, B, \vec{p}, U\}$ for which:*

- *A is the action or output set $\{a_1, a_2, \ldots a_r\}$ of the automaton*
- *B is the set of possible responses*
- *$\vec{p} = (p_1, \ldots, p_r)$ is a vector of the automaton's action probabilities and*
- *U denotes the learning scheme the automaton uses to update $\vec{p}$.*

An automaton is connected to its environment in a feedback loop. This environment defines the learning problem the automaton faces. At each time step t the automaton selects an action $a(t) \in A$, based on its internal action probability vector. This action serves as input to the environment which produces a response $b(t) \in B$ that is given to the automaton. The automaton can then alter its action probabilities based on the action it selected, and the feedback it received.

Depending on the set B, different models of environments can be distinguished. P-model environments produce a binary feedback using $B = \{0, 1\}$. In a Q-model environment the feedback can take on a finite number of values $B = \{b_1, \ldots, b_m\}$ with $0 \leq b_1, \ldots, b_m \leq 1$. Finally, in an S-model environment B is the interval $[0, 1]$ and the feedback is a continuous random variable. In the following we will focus on P-model environments,

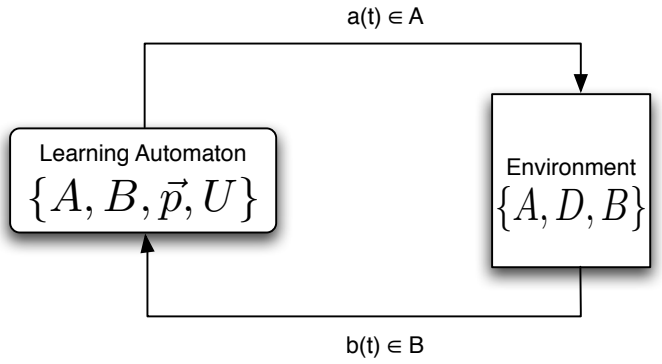


Figure 2.1: Learning automaton and environment feedback loop.

i.e. we assume that automata receive a binary feedback $b(t) \in \{0, 1\}$. Thus 0 represents a negative feedback, while 1 represents a positive feedback. Note that while there are only 2 possible feedback signals, this does not mean that the expected feedback for an action needs to be 0 or 1. Each action has an associated (unknown) reward probability, that specifies the probability of receiving a feedback of 1 when the automaton selects that action. The automaton's goal therefore, is to identify the action which has the highest reward probability. This leads to the following formal definition of an environment:

Definition 3 *An automata environment is defined by the triple $\{A, c, B\}$ with:*

- $A = \{a_1, \ldots, a_r\}$ *the set of possible inputs the environment can accept.*
- $D = \{d_1, \ldots, d_r\}$ *the set of reward probabilities corresponding to the inputs with* $d_i = Pr\{b(t) = 1 \mid a(t) = a_i\}, i : 1 \ldots r$
- B *is the set of outputs the environment can produce.*

As was mentioned above, the sets A and B correspond to those used in the definition of a learning automaton.

Several learning schemes for updating automata action probabilities have been proposed. Among these, the linear Reward Penalty family is probably the most studied. The basic idea behind these updates is to increase the probabilities of successful actions, while lowering the probabilities of actions that result in a low reward. The general linear reward penalty update is given by:

$$p_i(t+1) = p_i(t) + \lambda_1 b(t)(1 - p_i(t)) - \lambda_2(1 - b(t))p_i(t) \quad \text{if } a(t) = a_i \tag{2.1}$$

$$p_j(t+1) = p_j(t) - \lambda_1 b(t)p_j(t) + \lambda_2(1 - b(t))(\frac{1}{r-1} - p_j(t)) \quad \text{if } a_j \neq a_i \tag{2.2}$$

with r the number of actions in the set A. λ_1 and λ_2 are constants, called the reward and penalty parameter respectively. Depending on the values of these parameters 3 distinct variations of the algorithm can be considered. When $\lambda_1 = \lambda_2$ the algorithm is referred to as Linear Reward-Penalty (L_{R-P}) while it is called Linear Reward-ϵPenalty ($L_{R-\epsilon P}$) when $\lambda_1 >> \lambda_2$. If $\lambda_2 = 0$ the algorithm is called Linear Reward-Inaction (L_{R-I}). In this case λ_1 is also sometimes called the learning rate:

$$p_i(t+1) = p_i(t) + \lambda_1 b(t)(1 - p_i(t)) \quad \text{if } a(t) = a_i \tag{2.3}$$

$$p_j(t+1) = p_j(t) - \lambda_1 b(t)p_j(t) \quad \text{if } a_j \neq a_i \tag{2.4}$$

Despite the fact that all these updates derive from the same general scheme, they exhibit very different learning behaviours. In the next sections, we will discuss how these behaviours can be analysed and evaluated.

2.1.2 Norms of behaviour

When acting in a stationary environment, where the c_i parameters are constants, the goal of a learning automaton is to find the optimal action a_m for which $d_m = \max_i d_i$. The performance of the automaton is typically evaluated by examining the expected reward given the current probability vector $\vec{p}(t)$. This quantity is referred to as $W(t)$. For a P-model environment this quantity is given by:

$$W(t) = E[b(t) \mid \vec{p}(t)] \tag{2.5}$$

$$= \sum_{i=1}^{r} Pr\{b(t) = 1 \mid a(t) = a_i\} Pr\{a(t) = a_i\}$$

$$= \sum_{i=1}^{r} d_i p_i(t) \tag{2.6}$$

One possibility to evaluate the automatons performance is to compare it to a pure-chance automaton, i.e. an automaton that selects all actions with uniform probability without updating its action probabilities. The expected reward for such an automaton is a constant, denoted W_0, which can be calculated using Formula 2.6:

$$W_0 = \frac{1}{r}\sum_{i=1}^{r} d_i \tag{2.7}$$

This leads to the following definition:

Definition 4 *A learning automaton is called expedient if*

$$\lim_{t\to\infty} E[W(t)] > W_0 \tag{2.8}$$

Thus an automaton is expedient if it outperforms the pure chance automaton. Of course this is not a very demanding goal and ideally we would like our automata to be optimal, meaning that it learns to maximise the expected reward:

Definition 5 *A learning automaton is called optimal if*

$$\lim_{t\to\infty} E[W(t)] = d_m \tag{2.9}$$

where

$$d_m = \max_i d_i$$

Unfortunately, achieving optimality is often not feasible in practice. Therefore we introduce the concept of ϵ-optimality:

Definition 6 *A learning automaton is called ϵ-optimal if for arbitrary $\epsilon > 0$:*

$$\lim_{t\to\infty} E[W(t)] > d_m - \epsilon \tag{2.10}$$

can be achieved by a proper choice of automaton parameters.

In arbitrary environments the ϵ-optimality of an automaton may depend on the specific reward probabilities of the environment or the initial conditions of the automaton. An alternative requirement for desired behaviour can be defined by absolute expediency:

Definition 7 *A learning automaton is said to be absolutely expedient if:*

$$E[W(t+1) \mid \vec{p}(t)] > W(t) \tag{2.11}$$

This imposes an inequality on the conditional expectations of $W(t)$ at every time step. Taking expectations of both sides it can be seen that $E[W(t)]$ must be strictly monotonically increasing. In the stationary random environments considered here, it can be shown that absolute expediency implies ϵ-optimality [NT89].

2.1.3 Analysis of behaviour

When the reward probabilities are fixed $\vec{p}(t+1)$ is completely determined by $\vec{p}(t)$ and $\{\vec{p}(t)\}_{t\geq 0}$ is a discrete-time homogeneous Markov process. The state space of this process is the unit simplex:

$$S_r = \{\vec{p} \mid \vec{p} = (p_1, \ldots, p_r), 0 \leq p_i \leq 1, \sum_{i=1}^{r} p_1 = 1\} \tag{2.12}$$

A state $\vec{p}^*$ in this space is called *absorbing* when $\vec{p}(t) = \vec{p}^*$ implies $\vec{p}(k) = \vec{p}^*$ for all $k \geq t$.

The learning algorithms defined in Section 2.1.1 represent a mapping $U : S_r \rightarrow S_r$. When the corresponding Markov process has absorbing states, the algorithms are referred to as absorbing algorithms. This is the case for instance for the L_{R-I} algorithm for which all unit vectors $\vec{e}_i, i = 1 \ldots r$ are absorbing states. It can be shown [NT89] that the L_{R-I} scheme converges to one of these states with probability 1. Furthermore, by making the learning rate λ_1 sufficiently small, one can make the probability of the optimal action arbitrarily close to one, meaning that L_{R-I} is ϵ-optimal.

In contrast the L_{R-P} and $L_{R-\epsilon P}$ schemes are non-absorbing. Both these schemes are *ergodic*, meaning that the action probabilities converge in distribution to a random variable $\vec{p}^*$,independent of the initial conditions. In the case of the $L_{R-\epsilon P}$ scheme the mean of $\vec{p}*$ can be made arbitrarily close to the optimal unit vector by choosing sufficiently small learning parameters, giving an ϵ-optimal scheme.

2.1.4 Generalisations of Learning Automata

Several variations of the basic learning automaton have been developed. We conclude this section with a brief overview of relevant developments of the original LA scheme of Section 2.1.1.

The *pursuit automaton* modifies the update mechanism used by the standard automaton model. Instead of using the feedback $b(t)$ to directly update the last action performed, pursuit LA keep estimates $\vec{d} = (d_1, \ldots, d_r)$ which estimate the average reward received for each action. To calculate these estimates the pursuit algorithm stores two vectors $\vec{z} = (z_1, \ldots, z_r)$ and $\vec{\eta} = (\eta_1, \ldots, \eta_r)$, which respectively store the total reinforcement received for each action and the number of times each action has been performed. When the automaton receives response $b(t)$ for performing action $a(t) = a_i$ these vectors are updated as follows:

$$\begin{aligned} z_i(t+1) &= z_i(t) + b(t) \\ z_j(t+1) &= z_j(t),\ i \neq j \\ \eta_i(t+1) &= \eta_i(t) + 1 \\ \eta_j(t+1) &= \eta_i(t),\ i \neq j \\ d_i(t+1) &= \frac{z_i(t+1)}{\eta(t+1)},\ i = 1, \ldots, r \end{aligned} \tag{2.13}$$

The action probabilities of a pursuit automaton are updated based on these estimates. More specifically, rather than updating the action that was performed last, the pursuit algorithm updates the action which has the highest estimated average reward. In vector notation this update is described by:

$$\vec{p}(t+1) = \vec{p}(t) + \lambda(\vec{e}_{m(t)} - \vec{p}(t)) \tag{2.14}$$

Here λ is the learning rate, $m(t)$ is the index of the action with the highest estimated reward at time t and $\vec{e}_{m(t)}$ is the unit vector with a 1 at index $m(t)$. From this update it is clear that the pursuit algorithm performs an L_{R-I} update of action $a_{m(t)}$ with a feedback of 1.

Theoretically the pursuit update can be shown to exhibit the same behaviour as the reward-inaction update of Section 2.1.1. This means that it is ϵ-optimal and converges to the set of unit vectors. The algorithm does have some advantages over L_{R-I}. Since the feedback $b(t)$ is not directly used in the update of the probabilities, it does not need to be restricted to the interval $[0, 1]$. Furthermore, empirical evidence suggests that the pursuit algorithm converges much faster than L_{R-I}.

The *parameterised Learning Automata* (PLA) was introduced in [TP95]. The main innovation of PLA is that they no longer directly update the action probabilities. Instead, they keep a vector of parameters $\vec{u} \in \mathbb{R}^r$, with

one parameter u_i corresponding to each action a_i. From this parameter vector the action probabilities are calculated, using some probability generating function $g : A \times \mathbb{R}^r \to [0, 1]$. Here $g(a_i, \vec{u})$ gives the probability of selecting action a_i. The advantage of this parameterisation is that it is easier to implement more complex update schemes for the parameter vector, since it is not constrained to a probability vector. A common function used to generate the action probabilities is the *Boltzmann* function:

$$g(a_i, \vec{u}) = \frac{e^{u_i}}{\sum_j e^{u_j}} \tag{2.15}$$

The basic PLA update scheme uses an adapted version of the REINFORCE algorithm [Wil92]. This scheme defines a gradient descent procedure, which has the classic L_{R-I} learning rule as a special case. After selecting action a_i at time t and receiving feedback $b(t)$, a PLA will update its parameters using following update:

$$u_i(t+1) = u_i(t) + \lambda b(t) \frac{\delta \, ln \, g}{\delta \, u_i}(a_i, \vec{u}(t)) + \lambda h'(u_i(t)) + \sqrt{\lambda} s_i(t) \tag{2.16}$$

with:

$$h(x) = \begin{cases} -K(x-L)^{2n} & x \geq L \\ 0 & |x| \leq L \\ -K(x+L)^{2n} & x \leq -L \end{cases} \tag{2.17}$$

Here $h'(x)$ is the derivative of $h(x)$, $\{s_i(t) : t \geq 0\}$ is a set of i.i.d. variables with zero mean and variance σ^2, λ is the learning parameter, σ and K are positive constants and n is a positive integer.

In this update rule, the first two terms implement the basic REINFORCE gradient following rule. The third term was introduced to address issues with the REINFORCE algorithm giving rise to unbounded solutions. It keeps parameters bounded with $|u_i| \leq L$. The final term is a random noise term that allows the algorithm to escape local optima. PLA are mainly used in settings containing multiple automata, where they offer stronger convergence results than the basic L_{R-I} scheme. The precise convergence properties of this learning scheme will be discussed in more detail in Section 2.2.3.

2.2 Multi-Automata Systems

In the previous sections we considered the setting of a single automaton interacting with its environment. While interesting, this model has only

limited practical applicability as it is relatively simplistic. In this section we move to multi-automata systems, where several automata interact with the same environment. We introduce the basic method for interconnecting automata, called the automata game. In subsequent chapters we will use this model to develop and analyse algorithms for more complex *multi-state environments*.

Automata games view the LA as players in a strategic game from classical game theory. Their interactions can then be analysed using tools and solution concepts from game theory. In the next section we give a brief overview of these tools, before continuing with the analysis of automata update schemes in multi-automata settings.

2.2.1 Game Theory Basics

The central idea of game theory is to model strategic interactions as a game between a set of players. In this section we review basic game theoretic terminology and define two common solution concepts in games: Nash equilibria and Pareto optimality. A detailed overview of normal form games and their solutions can be found in [Gin00, OR99].

Definition 8 *A normal form game is a tuple* $(n, A_{1,\ldots,n}, R_{1,\ldots,n})$*, where*

- $1, \ldots, n$ *is a collection of agents, also called players*
- A_k *is the individual (finite) set of actions available to agent* k*.*
- $R_k : A_1 \times \ldots \times A_n \rightarrow \mathbb{R}$ *is the individual reward function of agent* k

The agents repeatedly play a game in which each agent k independently selects an individual action a from its private action set A_k.The combination of actions of all agents at any time-step, constitute a *joint action* or *action profile* $\vec{a}$ from the joint action set $\mathbb{A} = A_1 \times \ldots \times A_n$. For each joint action $\vec{a} \in \mathbb{A}$, $R_k(\vec{a})$ denotes agent k's expected payoff .

In the two player case, normal form games are often represented by their *payoff matrix*. An example of this can be seen in Example 1. In this case the action selected by player 1 selects a row in the matrix, while that of player 2 determines the column. The corresponding entry in the matrix then gives the payoffs player 1 and player 2 receive for the play. Players 1 and 2 are also referred to as the row and the column player, respectively.

Depending on the reward functions of the players different classifications of games can be made. When all players share the same reward function the game is called a *identical payoff* or *common interest* game. If the total

of all players rewards adds up to 0 the game is called a *zero-sum game*. In the latter games wins for certain players translate to losses for other players with opposing goals. Therefore these games are also referred to as pure competitive games. When considering games without special restrictions we speak of a *general sum game*.

Example 1 (Normal Form Game)

	C	D
C	$(5,5)$	$(0,10)$
D	$(10,0)$	$(1,1)$

The Prisoner's Dilemma Game: 2 prisoners committed a crime together. They can either cooperate (C) with each other and deny their crime (i.e. play the first action) or defect (D) and betray their partner (i.e. play the second action). When only one prisoner defects, he gets the maximum reward of 10*, while the other one takes all the blame for the crime and receives no reward. When they both cooperate, a reward of* 5 *is received, otherwise they only get reward* 1*.*

A *strategy* $\sigma_k : A_k \to [0,1]$ is an element of $\mu(A_k)$, the set of probability distributions over the action set A_k of player k . A strategy is called pure if $\sigma_i(a) = 1$ for some action $a \in A_i$ and 0 for all other actions, otherwise it is called a *mixed strategy*. A *strategy profile* $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ is a vector of strategies, containing one strategy for each agent. If all strategies in $\vec{\sigma}$ are pure it corresponds to a joint action $\vec{a} \in \mathbb{A}$. The expected reward for agent k for a strategy profile $\vec{\sigma}$ is given by:

$$R_k(\vec{\sigma}) = \sum_{\vec{a} \in \mathbb{A}} \prod_{j=1}^{n} \sigma_j(a_j) R_k(\vec{a})$$

We can now introduce the concept of a Nash equilibrium:

Definition 9 *Let* $\vec{\sigma}_{-k}$ *denote the profile* $\vec{\sigma} = (\sigma_1, \ldots, \sigma_k, \ldots, \sigma_n)$ *minus the strategy* σ_k *used by agent* k*. A strategy profile* $\vec{\sigma}$ *is then called a Nash equilibrium when we have for all* k*:*

$$R_k(\vec{\sigma}) \geq R_k(\vec{\sigma}_{-k} \cup \{\sigma'_k\}) \;\; \forall\, \sigma'_k \in \mu(A_k)$$

where we use $\vec{\sigma}_{-k} \cup \{\sigma'_k\}$ *to denote the strategy profile equal to* $\vec{\sigma}$*, but with agent* k *using* σ'_k *instead of* σ_k*.*

A Nash equilibrium $\vec{\sigma}$ is said to be pure when all strategies in $\vec{\sigma}$ are pure. From the definition it is clear that in a Nash equilibrium each agent is playing a best response to the current strategies of the other players. This means that, no player has an incentive to unilaterally deviate from this strategy profile. Nash proved [Nas50] the existence of a (possibly mixed) Nash equilibrium for any finite normal form game:

Theorem 1 (Nash,1950) *Every finite normal form game has at least one Nash equilibrium.*

A Nash equilibrium represents a local optimum for self-interested agents, since no agent can improve its payoff without the help of others. It does not, however, consider group rationality and joint actions that result in higher payoffs for all agents may exist. This leads to an alternative solution concept known as *Pareto optimality*:

Definition 10 *A strategy profile $\vec{\sigma}_1$ is said to be Pareto optimal if there is no other strategy profile $\vec{\sigma}_2$ in which all players simultaneously do better and at least one player is doing strictly better. The set of Pareto optimal strategies is called the Pareto front.*

So in a Pareto optimal solution it is not possible to improve an individual's payoff, without making someone else worse off. Note that a Nash equilibrium is not necessarily Pareto optimal and vice versa, a Pareto optimal solution is not necessarily Nash. This can be seen in the Prisoner's Dilemma Game of Example 1. The Nash equilibrium in this game is for both players to defect (i.e. betray each other) and receive a reward of 1. This equilibrium is not Pareto optimal, since both players could simultaneously do better if they both denied the crime. On the other hand, the Pareto optimal solution where both agents play 'cooperate' is not individually rational, since individually an agent can do better by switching to 'defect'.

2.2.2 Automata Games

The basic method for interconnecting multiple automata is the *automata game*. This model is based on the concept of normal form games described above, with each automaton representing a player in the game.

In an automata game setting, multiple automata are connected to the same, shared environment. The input to this environment is now a play of all automata, rather than a single action. A play $\vec{a}(t) = (a_1(t), \ldots, a_n(t))$ of n automata is a vector of pure strategies chosen by the automata at stage

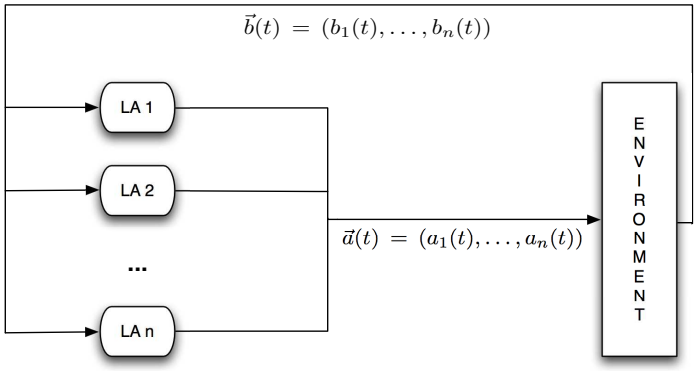


Figure 2.2: Graphic representation of an n-player automata game

t, such that $a_k(t)$ is an element of the action set A_k of the kth automaton. Correspondingly the outcome is now also a vector $\vec{b}(t) = (b_1(t), \ldots, b_n(t))$. Each outcome $b_k(t)$ can depend on the entire play $\vec{a}(t)$, not just on the action selected by the corresponding automaton k. At every time-step all automata update their probability distributions based on the responses of the environment. Figure 2.2 gives a visual representation of an automata game system.

This setting can be seen as a group of automata repeatedly playing a normal form game. In contrast to the standard game theoretic setting, however, players in the game are not aware of their opponents. All participants are considered to be ignorant, i.e. they update their action probabilities without information concerning the number of other participants, their strategies, actions or payoffs. In fact since they have no information regarding other players, participants in the game need not be aware that they are playing a game. From the point of view of an individual automaton, the LA are facing a non-stationary environment rather than playing a game, since they cannot observe their opponents. This setting is sometimes also referred to as the *unknown game model* [Ban68]. Despite this lack of information, we will in the next section see that LA still offer broad convergence guarantees in games.

2.2.3 Convergence in Automata Games

In the case of common interest games we can extend the notion of ϵ-optimality to teams of automatons. Since all players in these games receive the same payoff, joint actions which maximise the expected payoff for all agents are guaranteed to exist. This means that the team of automata is said to be ϵ-optimal, if their payoff is within distance ϵ of the maximum expected reward, for arbitrary ϵ. Since it is not possible to improve on the payoff of the optimal joint actions, they are also pure strategy Nash equilibria of the game. For the reward-inaction update following result has been shown:

Theorem 2 (Wheeler & Narendra, 1986) *Let Γ be an identical payoff game among n automata, all using identical L_{R-I} schemes. If Γ has a unique pure strategy Nash Equilibrium, then the collective of all learning automata is ϵ-optimal.*

However, in general, other possibly sub-optimal Nash equilibria can exist. In this case the reward-inaction scheme can only guarantee convergence to a pure Nash equilibrium and is only locally optimal. The selected equilibrium in this case will depend on the initial probabilities of the automata.

Unfortunately, in general automata games the norms of behaviour defined in Section 2.1.2 are ill-suited for the analysis of behaviour. This is due to the fact that automata get individual rewards and therefore requirements such as $(\epsilon-)$optimality may not be achievable for all automata at the same time. Instead we typically try to achieve convergence of the automata team to a Nash equilibrium[1]. This means that no automaton alone can improve its payoff by adjusting its strategy while other automata keep theirs constant. Convergence results of this type are available for a wide range of game settings. In two-person zero-sum games the L_{R-I} scheme converges to the Nash equilibrium when it exists in pure strategies, while the $L_{R-\epsilon P}$ scheme is able to approximate mixed equilibria. As already mentioned above, in n-person common interest games reward-inaction also converges to a pure Nash equilibrium. In [SPT94], the dynamics of reward-inaction in general sum games are studied. The authors proceed by approximating the update in the automata game by a system of ordinary differential equations.[2] Following properties are found to hold for the L_{R-I} dynamics:

[1] Although recent debate in the multi-agent community [SPG07] calls into question the focus on Nash equilibria, it is still the most commonly used solution concept in MARL. In Chapter 6 we shall discuss some methods of improving on equilibrium convergence.

[2] An overview of basic dynamic systems theory can be found in Appendix A.

- All Nash equilibria are stationary points.
- All strict Nash equilibria are asymptotically stable.
- All stationary points that are not Nash equilibria are unstable.

Furthermore, in [Ver04] it is shown that an automata team using the reward-inaction scheme will convergence to a pure joint strategy with probability 1. Together these results imply local convergence towards Nash equilibria [Ver04]. The above results for the L_{R-I} algorithm are summarised in the following theorem:

Theorem 3 (Sastry,1995 - Verbeeck, 2004) *In an n-player automata game with each player using the L_{R-I} update with sufficiently small learning rate, local convergence towards pure Nash equilibria is established.*

The parameterised learning automata described in Section 2.1.4 were introduced to improve on the local optimality obtained by reward-inaction in common interest problems. In [TP95] it was shown that the learning scheme used by PLA converges weakly to the Langevin equation, implying that the algorithm globally maximises the reward function. However, in order to keep solutions bounded, the parameters $\vec{u}$ used by the automata are restricted to a compact set. As such PLA may not be able to express the optimal policy, but instead learn to maximise the reward function over the compact set of parameters. This can be seen, for instance, when the algorithm is used with Boltzmann action selection. Because the optimal strategy in a common interest game is pure, the optimal reward is reached as action parameters $\vec{u}$ go to infinity. As the parameterised LA update of Equation 2.16 keeps parameters bounded with a predefined maximum absolute value of L, this strategy cannot be expressed by the PLA. By setting the boundaries L sufficiently large, however, the optimal strategy can be approximated as as closely as is necessary. These results are summarised in the following theorem:

Theorem 4 (Thatchahar & Phansalkar, 1995) *Let $\Gamma = (n, A_{1,\ldots,n}, R)$ be an identical payoff game among n automata, all using the PLA update scheme from Equation 2.16. Then the automata will globally maximise $E[R \mid \vec{u}]$, subject to $|u_i| \leq L$.*

2.3 Summary

In this chapter we discussed automata learning in stateless environments. We started by defining the basic concepts of a learning automaton and an environment. This was followed by an explanation of how a single automaton interacts with its environment and a discussion on the evaluation of an automaton's performance.

After providing some known results for the basic automata update schemes, we move to multi-automata settings. We first gave a brief overview of some basic game theoretic notions and then demonstrated how normal form games can be used as a framework for multi-automata learning. We finished the chapter with convergence results for these automata games.

Chapter 3

Learning in Finite MDPs

In this chapter we move from the stateless environments considered in the previous chapter, to problems where the environment can assume a finite set of states. In these situations the learner is required not just to learn a single action, but rather to sequentially select actions depending on the current environment state. At each time step the action selected by the agent produces a feedback and triggers a transition to the next state. The ultimate goal is to maximise not just the immediate, but rather the long term rewards.

In a realistic learning setting a learning agent will often face a dynamic environment. The actions that an agent performs have an effect on its surroundings and cause changes to which the agent must react. Decisions an agent makes now may have a large impact on its future reward prospects and the future state of the system. This also means that actions which the agent preferred in the past, may no longer be applicable in the current state. Therefore, in these settings, an agent's goal is no longer to learn a single optimal action, but rather to optimise a sequence of actions, which take into account the current state of the agent and its environment.

The sequential decision making problem described above can be formalised by the *Markov Decision Process* (MDP) model, which will be described in detail in the next section. The MDP model has its roots in operations research of the 1950s. It has since become the standard model to describe a wide range of control and optimisation applications. Often these MDPs are tackled by way of *dynamic programming*. Dynamic programming algorithms use knowledge of the MDP's state transition probabilities and expected rewards to calculate an optimal solution. Most of these approaches can be traced back to Bellman [Bel57], who formulated

the optimality condition (the *Bellman equation*) for dynamic programming in MDPs.

The dynamic programming approach can be contrasted with the learning approaches which are the focus of A.I. research on MDPs. Here an agent or learner no longer has explicit access to the underlying model of the MDP. Algorithms which solve MDPs in this manner are collectively referred to as reinforcement learning (RL). In a reinforcement learning setting the agent has access only to a scalar feedback signal, which indicates the reward for the last action taken. As such the learner has to use trial and error to discover the effects of its actions and estimate their expected future payoffs. In contrast with dynamic programming which is typically used to compute an optimal solution off-line, reinforcement learning is generally used on-line since it relies on direct experimentation to explore the environment[1]. RL is closely related to dynamic programming however, and is sometimes described as model-free dynamic programming, referring to the lack of knowledge about transition probabilities and expected rewards.

The remainder of this chapter is structured as follows. We start by introducing the formal model that we shall use to describe these sequential decision problems, the Markov Decision Process (MDP). Subsequent sections will list some standard results for this model from reinforcement learning theory. Finally, we show how learning automata can be used to find an optimal policy in MDPs and introduce the *limiting game*. This method allows us to analyse multi-state LA algorithms by approximating their interactions with a repeated automata game. This game can then be analysed using the tools from game theory introduced in the previous chapter. The limiting game view is not new [Wit77, WJN86], but has received little attention in RL literature. We promote this view as it provides a useful tool and can also provide a unified analysis for both single and multi-agent reinforcement learning. This approach will play an important role in the remainder of this dissertation. In the following chapters we will show how it can be extended to multi-agent multi-state problems. We will demonstrate that this formalism provides a useful tool for the design and analysis of multi-agent algorithms. References for this chapter are: [Put94, BS98, WJN86]

[1] Although batch RL approaches which use stored data do exist. See for instance [EGW06]

3.1 Markov Decision Processes

We now introduce the formal model for the sequential decision problems studied in this chapter. At every time-step $t = 1, 2, \ldots$ the system is described by the current state $s(t) = s$. The agent must select an action $a(t)$ from the set of possible actions for this state. The result of this action is that the environment transitions to a new state $s(t+1)$ and the agent receives a scalar reward $r(t+1)$ for this transition. Additionally, it is assumed that the probability for the transition and the expected reward associated with it depend only on the current state and action. That is, the history of the process before the current time step does not provide any supplementary information if the current state is known.

Formally, a Markov Decision Process can be described as follows:

Definition 11 *A (finite) Markov decision process is a tuple (S,A,R,T), where:*

- $S = \{s^1, \ldots, s^N\}$ *is a (finite) set of states.*
- $A : \cup_{s \in S} A(s)$*, where A(s) is the (finite) set of available actions in state* $s \in S$*.*
- $T : S \times A \times S \to [0, 1]$ *is the transition function,* $T(s, a, s')$ *specifying the probability of going to each state* s' *when action* a *is played in state* s*.*
- $R : S \times A \times S \to \mathbb{R}$ *is the reward function,* $R(s, a, s')$ *giving the expected reward for the transition from state* s *to state* s' *under action* a*.*

Additionally the process is assumed to obey the *Markov property*:

Definition 12 (Markov Property) *A system is said to posses the Markov property if the future behaviour of the process is independent of previous states, given the current state:*

$$T(s(t+1) \mid s(t), a(t), \ldots, s(0), a(0)) = T(s(t+1) \mid s(t), a(t))$$

A *policy* in the MDP is sequence $\pi = \{\mu(0), \mu(1), \ldots\}$, where each $\mu(t)$ denotes a function which maps each state s to a probability distribution over the action set $A(s)$. Thus, each $\mu(t)$ denotes the decision rule used a time t to select an action $a(t)$. A policy π is called *stationary* or time-independent when $\mu(t)$ is constant over all t and the same decision rule

is used for all time steps. With the exception of Chapter 6 only stationary policies are treated in this dissertation.

For a stationary policy π, we use $\pi(s, a)$ to denote the probability of taking action a in state s. When in all states s, $\pi(s, a) = 1$ for some action a and 0 for all other actions, the policy is said to be deterministic , otherwise it is called *stochastic*. For deterministic (stationary) policies we also write $\pi(s)$ to indicate the action selected in state s under policy π.

The goal of the agent is to learn a policy which maximises the expected future reward the agent receives. Several criteria can be used to define the concept of future reward. In discounted MDPs rewards received at future time steps are multiplied by a discount factor, in order to represent the current value of future rewards. Starting from state s, the expected discounted value for the process under a policy π is given by:

$$V^{\pi}(s) = E^{\pi}\left\{\sum_{t=0}^{\infty} \gamma^t r(t+1) \mid s(0) = s\right\} \tag{3.1}$$

Here $0 \leq \gamma < 1$ is called the *discount factor* and the operator E^{π} denotes expectation with regard to the policy π.
Alternatively, in an average reward MDP, the agent attempts to optimise its expected average reward over time:

$$J^{\pi}(s) = \lim_{T\to\infty} \frac{1}{T} E^{\pi}\left\{\sum_{t=0}^{T} r(t+1) \mid s(0) = s\right\} \tag{3.2}$$

The optimal policy π^* is defined as the policy which maximises the used criterion for all states s. The corresponding optimal reward criteria are denoted as V^* or J^* respectively. It should be noted that, in general these different criteria need not agree. Depending on which criterion is used, different policies can be optimal. This is the case since the discounted reward criterion focuses more on short term rewards, while the average reward only takes into account the long run behaviour of the process. However, in any finite MDP a discount factor for which both criteria agree can be found [Sch93].

Example 2 (MDP) *As a simple example of an MDP we consider the Recycling Robot Problem, based on the example by Barto and Sutton [BS98]. A mobile robot is tasked with recycling cans in an office building. The robot has the necessary systems for navigating the building, finding cans and recycling them. It has to learn however, which high level strategy is most efficient. It can either actively search for cans to recycle, or simply remain stationary and wait for people to bring*

it cans. The robot runs on a rechargeable battery. When this battery runs low, the robot can decide to return to its charging station in order to restore the battery to full power. If the robot does not recharge in time, its battery dies and it needs to be rescued. Below we describe this setting as an MDP.

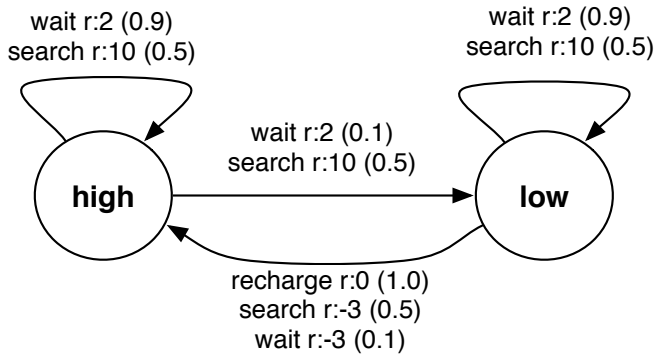


Figure 3.1: The recycling robot problem.

The system state is determined by the robot's battery charge and can either be $s(t) = high$ *or* $s(t) = low$*. The corresponding action sets are* $A(high) = \{search, wait\}$ *and* $A(low) = \{search, wait, recharge\}$*. Transition probabilities and expected rewards are summarised in Table 3.1. Figure 3.1 gives a transition diagram for the states of this problem.*

In this problem 6 *possible deterministic policies exist. Table 3.2 lists the expected discounted* ($\gamma = 0.9$) *and average rewards for both states. The optimal rewards for both criteria are listed in bold. As is clear from the table, in this case both criteria agree on the optimal policy '(search,search)'. So the best strategy for the robot is to always search for cans, even when it means that sometimes a penalty for depleting its battery is incurred.*

3.1.1 Properties of MDPs

Before proceeding with solution methods for MDPs, we give a short overview of some relevant properties. These features are of importance mainly when using the average reward criterion, as will be the case in the remainder of this dissertation. A more detailed discussion of these facts can be found in [Put94].

$s(t) = s$	$s(t+1) = s'$	$a(t) = a$	$T(s, a, s')$	$R(s, a, s')$
high	high	search	0.5	10
high	low	search	0.5	10
high	high	wait	0.9	2
high	low	wait	0.1	2
low	high	search	0.5	-3
low	low	search	0.5	10
low	high	wait	0.1	-3
low	low	wait	0.9	2
low	high	recharge	1.0	0
low	low	recharge	0.0	0

Table 3.1: Transition probabilities and expected rewards for the recycling robot problem.

π	$V^\pi(high)$	$V^\pi(low)$	$J^\pi(high)$	$J^\pi(low)$
(search,search)	**70.8**	**64.3**	**6.75**	**6.75**
(search,wait)	40.2	27.0	2.29	2.29
(search,recharge)	69.0	62.1	6 .67	6.67
(wait,search)	22.1	24.5	2.25	2.25
(wait,wait)	15.5	10.5	1.75	1.75
(wait,recharge)	18.3	16.5	1.82	1.82

Table 3.2: Expected discounted reward and limit average rewards for all deterministic policies in the recycling robot problem.

Markov Chains

The classification of MDPs is based on the underlying concept of *Markov chains*. We will now give a short description of basic Markov chain theory. For more information, we refer the reader to [Nor98]. A Markov chain is a sequence of random variables $\{X(t)\}_{t\geq 0}$ taking values in a discrete state space S. The transition probabilities $T(X(t+1)|X(t))$ of the chain are required to satisfy the Markov property as given in Definition 12, i.e. they are independent of past states given the current state. We will only consider *stationary chains*, in which transition probabilities are also independent of time. In this case we can write $p_{ij} = T(X(t+1) = s^j \mid X(t) = s^i)$ for the probability of going to state s^j when we are in state s^i. The Markov chain

can then be described by its *transition matrix* $P = (p_{ij}), i = 1, \ldots, |S|; j = 1, \ldots, |S|$. Additionally, we can define the n-step transition probability as $p_{ij}^{(n)} = Pr\{X(t+n) = s^j \mid X(t) = s^i\}$, i.e. the probability of going from s^i to s^j in n steps.

A state s^i of a stationary Markov chain is characterised based on the probability of returning to that state after a visit, $p_{ii}^{(n)}$. It is called *transient* if there is a non-zero probability that the chain will never return to that state. States that are not transient are called *recurrent states*. Equivalently, a state is recurrent, if and only if:

$$\sum_{n=0}^{\infty} p_{ii}^{(n)} = \infty$$

Additionally, if $gcd(\{n \mid p_{ii}^{(n)} > 0\}) = d$ (with d¿1) then state i is said to be *periodic*[2] with period d. This means that when starting in state i at time t, the chain has a positive probability of returning to this state only after an exact multiple of d time steps have past, i.e at times $t + kd, k \in \mathbb{N}$. When $d = 1$, state s^i is called *aperiodic*.

A set of states C is called *closed* when no state outside the set can be reached from a state inside the set, i.e. $p_{ik} = 0$ for every $i \in C$ and $k \notin C$. When a closed set does not have any closed proper subsets it is called irreducible. We now classify Markov chains based on their *chain structure.* The structure of a Markov chain is determined by the partitioning if its recurrent states into disjoint closed irreducible sets $C_i, i = 1, \ldots, m$. For any (finite) Markov chain with state space S, we can write S as:

$$S = C_1 \cup C_2 \cup \ldots \cup C_m \cup T$$

Where the C_i are the closed irreducible sets and T is the set of transient states. Based on this partitioning Markov chains are classified as follows. If S consists of a single closed set (i.e. $m = 1$ and $T = \emptyset$), the chain is *irreducible*. When S is finite and consists of a single closed irreducible set together with a set of transient states, the chain is *unichain* . Otherwise the chain is called *multichain* . In the following we will mainly deal with chains that have the following property:

Definition 13 *A Markov chain is ergodic if* $\lim_{n\to\infty} p_{ij}^{(n)} = p_j^*$ *exists for all* j*, independently of* i *with* $\sum_{j\in S} p_j^* = 1$ *and* $p_j^* > 0, \forall j$*. In this case* $(p_1^*, \ldots, p_{|S|}^*)$ *is called the limiting or stationary distribution of the Markov chain.*

[2] gcd denotes the greatest common divisor.

The necessary conditions for ergodicity are that the chain is irreducible and the states are aperiodic[3]. The stationary distribution of an ergodic Markov chain can be calculated by finding the probability vector $\vec{p}$ which satisfies $\vec{p} = \vec{p}P$. This stationary distribution is unique and independent of the starting state of the chain.

Example 3 (Markov Chains) *We now give a simple example of a Markov chain, using a very simple weather model. Let $X(t)$, represent the weather conditions on day t. This variable can take values in the state space $S = \{sunny(s), cloudy(c), raining(r)\}$. Transition probabilities indicate the probability of observing a certain weather type tomorrow, given the current conditions. We consider stationary transition probabilities specified by the following matrix:*

$$P = \begin{array}{c|ccc} & s & c & r \\ \hline s & 0.7 & 0.2 & 0.1 \\ c & 0.4 & 0.2 & 0.4 \\ r & 0.1 & 0.1 & 0.8 \end{array}$$

Entries in the matrix represent the probabilities for the next day given the current weather. We can now analyse our model by looking at the Markov chain $X(t)$ giving the state of the weather on day t. Let $\vec{p}(t)$ denote the probability vector giving the expected probabilities over the states at time t, i.e. $\vec{p}(t)$ contains the probabilities of $X(t)$ being sunny, cloudy or raining. Suppose we start on a sunny day, which we represent as probability $\vec{p}(0)$ vector having a probability 1 for sunny:

$$\vec{p}(0) = (1, 0, 0)$$

The probabilities for the weather on the next day $X(1)$ are then given by:

$$\vec{p}(1) = \vec{p}(0)P = (0.7, 0.2, 0.1)$$

So we have probability 0.7 of another sunny day and probabilities 0.2 and 0.1 for clouds or rain, respectively For day 2 we get:

$$\vec{p}(2) = \vec{p}(1)P = \vec{p}(0)P^2 = (0.58, 0.19, 0.23)$$

This process continues for time steps $t = 3, 4, \ldots$. In general, we have:

$$\vec{p}(t) = \vec{p}(t-1)P = \vec{p}(0)P^t$$

[3] Although when studying periodic chains, it is possible to proceed by replacing the limit in the definition with a Cesaro-limit.

Using the matrix P defined above, all states are recurrent (as we always have a positive probability of going to each state) and aperiodic (since we can return to each state in a single step), meaning that the chain is ergodic. We can now calculate stationary probabilities for the states by solving: $\vec{p}P = \vec{p}$. This results in (rounded) probabilities $(0.3636, 0.1515, 0.4848)$, i.e. the long term probability of having a sunny day is about 0.36, independent of the starting state.

This result depends completely on the matrix P. For example if we were to use following matrix:

$$P = \begin{array}{c|ccc} & s & c & r \\ \hline s & 1.0 & 0.0 & 0.0 \\ c & 0.5 & 0.0 & 0.5 \\ r & 0.0 & 0.0 & 1.0 \end{array}$$

we obtain a multichain example. The state space can now be decomposed into closed sets $\{sunny\}$ and $\{raining\}$ and the transient set $\{cloudy\}$. In this case we arrive in either the state $sunny$ or $raining$ and never leave this state, since the probability to stay is 1. The state $cloudy$ is transient as we can never return to this state after leaving it.

Classification of MDPs

When a fixed policy is played in an MDP, the sequence of states is a Markov chain. For this reason MDPs are sometimes called *controlled Markov chains*. Based on the properties of the Markov chains generated by the stationary deterministic policies, MDPs can be divided into following classes:

- *Recurrent*: an MDP is *recurrent* if all states belong to a single recurrent set under *all* stationary deterministic policies.
- *Unichain*: an MDP is if the Markov chain under *all* deterministic policies has a single recurrent set plus a (possibly empty) set of transient states.
- *multi-chain*: an MDP is called multichain if *at least one* stationary policy generates multiple irreducible recurrent sets.

In this dissertation we will focus on the ergodic case, where the Markov chains under all deterministic stationary policies are ergodic, also see Assumption 1 below. This case is a subset of the recurrent MDP case[4] above

[4]In fact since we can also deal with the periodic case(see previous footnote), many authors do not distinguish between the ergodic and the recurrent case. See for example [Put94, FVV97].

and has some nice analytical properties. Firstly, since all states are recurrent, we are sure that every state is visited infinitely often, under any policy. This is often a requirement for the convergence of learning algorithms. Secondly, as noted above we can determine a stationary distribution for the Markov chain under any stationary policy π. This distribution, over the states $s \in S$, denoted by $d^\pi(s)$, can be used to express the average expected reward for the policy (defined in Equation 3.2) as follows:

$$J^\pi = \sum_{s \in S} d^\pi(s) \sum_{a \in A(s)} \pi(s,a) \sum_{s' \in S} T(s,a,s') R(s,a,s') \tag{3.3}$$

Where $d^\pi$is the stationary distribution over the states under policy π. This expected average reward for a policy, is also constant over all states. Furthermore in the ergodic case an optimal deterministic policy, which maximises this reward always exists.

3.2 Reinforcement Learning

Reinforcement learning (RL) algorithms attempt to learn an optimal policy for an MDP, using only the reward signals obtained through repeated interactions with the environment. RL approaches can be divided intro 2 classes. *Value iteration* algorithms learn the optimal value function. From this learned value function an optimal policy can then be derived. Below we introduce *Q-learning*, one of the most popular reinforcement learning algorithms, which is an example of value iteration.

Alternatively, *policy iteration* algorithms directly build an optimal policy. Generally they consist of two interleaved phases: policy evaluation and policy improvement. During the policy evaluation phase the value of the current policy is estimated. Based on this estimate the policy is the locally improved during the policy improvement phase. This process continues until no further improvement is possible and an optimal policy is reached. The learning automata introduced in the previous chapter can be seen as an example of simple policy iterators. In Section 3.3 we will show how multiple LA can be used to develop a policy iteration RL algorithm for MDPs.

Both policy iteration and value iteration reinforcement learning are derived from their dynamic programming counterparts. However, as RL does not assume a given model for the MDP, several new problems are encountered. One of the key issues in RL is that learners need to explore the environment in order to discover the effects of their actions. This implies

that the agent cannot simply play the actions that have the current highest estimated rewards, but also needs to try new actions in an attempt to discover better strategies. This problem is usually known as the exploitation-exploration trade-off. Two basic ways exist to address this question.

On-policy methods estimate values of the policy that is currently being used and attempt to improve on this policy. Since on-policy algorithms also use the policy that is being learned for exploration, they can only implement policies that include sufficient exploration while still learning. The actor-critic methods discussed in Section 3.2.3 can be seen as an example of this approach. In contrast, in *off-policy* methods the agent uses a behaviour function or control policy, which differs from the goal policy that is being learned. In this case the process is often divided in a learning phase during which the optimal policy is learned, and a control phase during which it is used for control. The Q-learning algorithm in the next section is an example of this type of algorithm.

3.2.1 Q-learning

On of the most popular model-free approaches is *Q-learning* [WD92]. The goal of the Q-learning algorithm is to learn optimal state-action values or *Q-values*

$$Q^\pi(s,a) = E[r(s,a) + \gamma V^\pi(s_{t+1}) \mid s(t) = s, a(t) = a] \tag{3.4}$$

The Q-value $Q^\pi(s,a)$ denotes the expected (discounted) value of return for starting in state s, taking action a and following policy π thereafter. A policy π^* is optimal when

$$Q^*(s,a) = \max_\pi Q^\pi(s,a), \;\; \forall s \in S, \; a \in A(s) \tag{3.5}$$

These Q-values can be seen to correspond to the value V by noting that:

$$V^*(s) = \max_{a \in A(s)} Q^*(s,a) \tag{3.6}$$

Given the optimal values Q^*, an optimal policy can easily be formulated by selecting the action with the highest Q-value in each state. The Q-learning algorithm uses stochastic approximation techniques to make successive estimates of the optimal Q^*. At each time step the system is in a certain state s and the Q-learner performs an action a, selected by its control policy. After performing action a, the Q-learner observes its immediate reward r and the new system state s'. This information is then

used to update the Q-value estimate for the state-action pair (s, a), using the Q-learning update:

$$Q(s,a) \leftarrow (1-\lambda)Q(s,a) + \lambda[r + \gamma \max_{a'} Q(s',a')] \tag{3.7}$$

Where $\lambda \in (0,1)$ is the learning rate, and $\gamma \in [0,1]$ is the discount factor. Under quite general conditions it can be shown that the estimated values of the Q-learning algorithm converge to Q^* [Tsi94]. The complete Q-learning algorithm is listed in Algorithm 1.

Algorithm 1 Q-Learning

initialise $Q(s,a)$ to zero, $\forall s, a$.
$s \leftarrow s(0)$
loop
- Select an action a and execute it
- Observe immediate reward r and new state s'
- Update $Q(s,a)$:

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

- $s \leftarrow s'$

end loop

One of the key insights used in developing the Q-learning algorithm is that the control policy used to generate actions need not correspond to the greedy policy that is being learned by the algorithm. In fact provided that it keeps updating all state-action pairs any behaviour policy can be used [Tsi94]. Therefore Q-learning is an example of off-policy learning. To meet the theoretical convergence requirements an control policy is required to visit every state-action pair infinitely often. In practice, however, one generally tests if the Q-values have stabilised and reduces the amount of exploration accordingly as the algorithm proceeds.

3.2.2 R-learning

The Q-learning algorithm described above is able to learn a policy that maximises the expected discounted future rewards in a wide range of settings. It can also be used for undiscounted problems (i.e. $\gamma = 1$), provided that the Q-values are bounded. However, for undiscounted MDPs alternative approaches have been formulated, which are specifically tailored to

such problems. Schwartz [Sch93] developed an algorithm called *R-learning* for average reward MDPs. This algorithm, which follows the same template as Q-learning, attempts to learn policies which maximise the average reward, while preferring policies which give higher transient payoffs. The algorithm attempts to learn *relative values* for the state-action pairs, called R-values, instead of Q-values:

$$R^{\pi}(s,a) = \sum_{k=1}^{\infty} E^{\pi}\{r(t+k) - J^{\pi} \mid s(t) = s, a(t) = a\}$$

These values correspond to the transient difference in reward relative to the average reward under policy π, when starting from state s and taking action a. While in the ergodic MDPs to which R-learning can be applied the average reward J^{π} is the same for all states under a policy π, the transient rewards R^{π} may differ. R-learning attempts to learn those optimal polices which have the highest transient reward. To approximate the R-values it replaces the Q-learning update in Algorithm 1 with following rule:

$$R(s,a) \leftarrow (1-\lambda_1)Q(s,a) + \lambda_1[r - J + \max_{a'} R(s',a')]$$

Here λ_1 is again a learning rate and J is an approximation of the average reward for the greedy policy with respect to the R-values. This approximation is made by using following rule:

$$J \leftarrow (1-\lambda_2)J + \lambda_2[r + \max_{a'} R(s',a') - \max_{a} R(s,a)]$$

whenever the updated R-value $R(s,a)$ corresponds to $\max_a R(s,a)$, with λ_2 another a learning rate. While the algorithm has been thoroughly evaluated empirically [Mah96], currently no convergence guarantees are known for R-learning.

3.2.3 Actor-Critic Methods

Policy iteration methods learn optimal policies by iteratively improving the current policy used by the learner. At each time-step the learner has a current policy π, which is used for action selection. This policy is then locally improved by first evaluating the policy, and then finding states in which the policy can be improved. In the original dynamic programming version of policy iteration, policy evaluation was achieved by calculating the value function V^{π} for the current policy. In RL methods, however, we can no

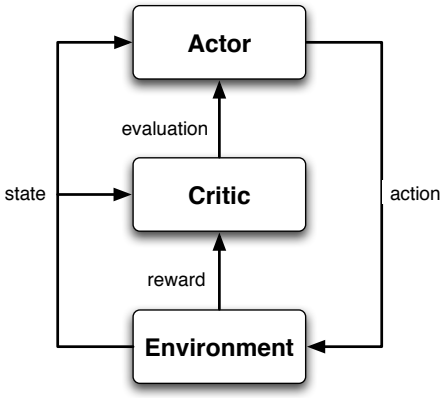


Figure 3.2: Schematic overview of the actor critic framework.

longer directly compute this value, because we no longer have a model of the environment. This limitation gave rise to the *Actor-Critic* framework.

Actor-critic methods are composed of 2 separate components. In addition to a representation of the current policy (the actor), actor-critic algorithms also learn an evaluation of this policy (the critic). The actor part is responsible for selecting actions based of the current policy. Rewards received for select actions are given to the critic, however. The critic uses these rewards to estimate the value function for the current policy. Based on this value function, it provides a feedback to the actor, which can then improve the policy. A schematic overview of this system is given in Figure 3.2.

In the next section we give an example of a simple implementation of this framework, using learning automata. In such a system, the current policy is represented by the set of all action probability vectors of all automata. A feedback for this policy is calculated by estimating the current state-action values in each state.

3.3 Automata Learning in MDPs

We now show how a network of learning automata can be used to solve an MDP in a completely decentralised manner. We focus on the algorithm for average reward MDPs proposed by Wheeler and Narendra [WJN86], but a similar algorithm exist for the discounted case [Wit77].

The basic idea is that each automaton is responsible for learning the appropriate action in a single state. Control of the system passes from one automaton to the next as the system changes states. When the automaton in state s is activated, it selects an action $a \in A(s)$ according to its action probability vector. The automata are not informed of the immediate reward that is received after their action is performed. Instead, when the system returns to a previously visited state s, the automaton receives 2 pieces of information: the current time step and the total reward gathered up to the current time step. These quantities allow the automaton to calculate Δr, the reward gathered since the last visit to state s, and Δt, the time that has passed since this visit. Both are added to the respective cumulative total reward $\rho(s, a)$ and time $\eta(s, a)$, corresponding to the action a taken on the last visit to state s. This action is subsequently updated using following feedback:

$$\beta(t) = \frac{\rho(s, a)}{\eta(s, a)} \tag{3.8}$$

This feedback signal, together with the reward-inaction updated described in Equation 2.3 are denoted by the authors in [WJN86] as learning scheme T1. The complete algorithm, which we refer to as the *Interconnected Learning Automata* (ILA) algorithm is listed in Table 2.

The average reward received for a policy depends on the structure of the Markov chain generated by that policy. To apply the ILA algorithm following assumption is made:

Assumption 1 (ergodicity) *The Markov chain of system states generated by each policy π is ergodic.*

This assumption assures us that there are no transient states and that for each policy a stationary distribution over all states exists. In [WJN86], the authors show that, under Assumption 1 following result holds:

Theorem 5 (Wheeler & Narendra,1986) *Associate an automaton LA_i, using update $T1$, with each state of an N state MDP. Provided that the Markov Chain corresponding to each policy π is ergodic, the learning algorithm is ϵ-optimal.*

The proof of this theorem relies on the following properties:

1. The asymptotic behaviour of the MDP-ILA algorithm can be approximated by an identical payoff automata game.

Algorithm 2 MDP-ILA

initialise $r_{prev}(s)$, $t_{prev}(s)$, $a_{prev}(s)$,t, r_{tot},$\rho(s,a)$, $\eta(s,a)$ to zero, $\forall s, a$.
$s \leftarrow s(0)$
loop
if s was visited before **then**
• Calculate received reward and time passed since last visit to state s:

$$\Delta r = r_{tot} - r_{prev}(s)$$

$$\Delta t = t - t_{prev}(s)$$

• Update estimates for action $a_{prev}(s)$ taken on last visit to s:

$$\rho(s, a_{prev}(s)) = \rho(s, a_{prev}(s)) + \Delta r$$

$$\eta(s, a_{prev}(s)) = \eta(s, a_{prev}(s)) + \Delta t$$

• Calculate feedback:

$$\beta(t) = \frac{\rho(s, a_{prev}(s))}{\eta(s, a_{prev}(s))}$$

• Update automaton LA_s using L_{R-I} update with $a(t) = a_{prev}(s)$ and $\beta(t)$ as above.
end if
• Let LA_s select an action a.
• Store data for current state visit:

$$t_{prev}(s) \leftarrow t$$

$$r_{prev}(s) \leftarrow r_{tot}$$

$$a_{prev}(s) \leftarrow a$$

• Execute a, observe immediate reward r and new state s'
• $s \leftarrow s'$
• $r_{tot} \leftarrow r_{tot} + r$
• $t \leftarrow t + 1$
end loop

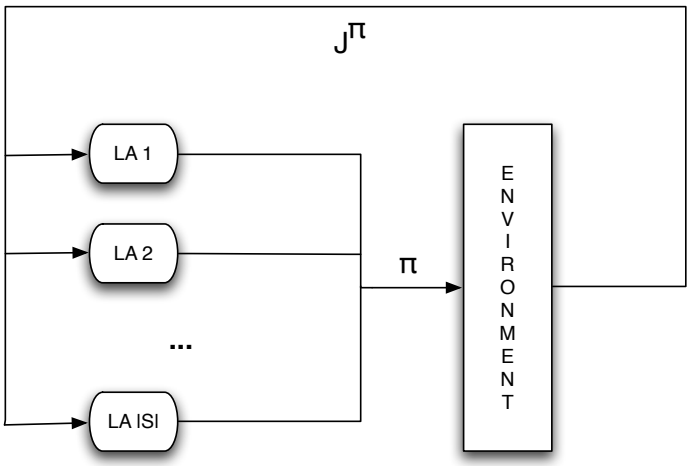


Figure 3.3: MDP automata algorithm as an automata game

2. The automata game corresponding to an ergodic MDP has a unique equilibrium, corresponding to the optimal policy. This property is explained in detail in the next Section.

3. According to Theorem 2 in Chapter 2, a team of automata using L_{R-I} in an identical payoff game with a unique equilibrium, converges to this equilibrium.

In the following section we explain in detail how this automata game approximation is applied.

3.4 Limiting Games

Limiting games were introduced in [Wit77, WJN86] as a tool to analyse the convergence behaviour of the MDP-ILA algorithm described above. The purpose of a limiting game is to approximate the asynchronous updating in a multi-state automata environment, by the synchronous interactions of an automata game.

For a given MDP, its limiting game can be constructed as follows. The automata used to select actions in each state of the MDP are the players in the game. Each automaton has the action set corresponding to the available actions in its state. This means that a play in the game selects an action for

every state of the MDP and as such corresponds to a deterministic policy for the MDP. At each time step, the joint strategy used by all automata in the game maps action probabilities to each state and as such describes the stationary policy currently used in the MDP.[5] The payoff received by players in the limiting game is exactly the expected average reward for the deterministic policy corresponding to the last play. Under the Assumption 1 noted in the previous section, the expected reward is the same starting from all states, and can be calculated using Equation 3.3.

Definition 14 (Limiting Game) *Given an ergodic MDP (S, A, R, T), the corresponding limiting game is an automata game $(|S|, A_{1,...,|S|}, R)$, where:*

- $1, ..., |S|$ *are the automata playing the game, one for each state in* S
- $A_i = A(s^i)$ *is the action set of automaton* i
- $R(\vec{a}) = J^\pi$ *is the common reward function for all agents. Here for any play* $\vec{a} = (a_1, \ldots, a_{|S|})$, π *is the corresponding deterministic policy which has* $\pi(s^i) = a_i$, *with* a_i *the action taken by automaton* i *in* $\vec{a}$.

Theorem 6 (Wheeler & Narendra,1986) *For any ergodic MDP, the corresponding limiting game has a unique, optimal equilibrium.*

We now give an outline of the proof for this theorem, provided in [WJN86]. The proof follows the proof of convergence for policy iteration given by Howard [How60]. Assume we have an ergodic, N-state MDP and let $\pi = (a_1, \ldots, a_N)$ be a non-optimal equilibrium point of its limiting game. Following Howard we can then calculate the relative state values $v^\pi(s^i)$ for states s^i under this policy by solving:

$$J^\pi = q^\pi(s^i) + \sum_{j=1}^{N} T(s^i, a_i, s^j) v^\pi(s^i) - v^\pi(s^i) \tag{3.9}$$

Where $q^\pi(s^i) = \sum_{j=1}^{N} T(s^i, \pi(s^i), s^j) R(s^i, \pi(s^i), s^j)$ and $v^\pi(s^N)$ is set to zero to guarantee a unique solution. Since π is non-optimal, a better policy can be found using policy iteration. Assume that s^i is one of the states where this better policy differs from π. We can then build a policy π', which differs from π only in state s^i, by maximising following quantity over all actions $k \in A(s^i)$:

[5] In view of this relation between policies and strategies in the game, we will sometimes use game theoretic terminology to describe policies as pure or mixed, instead of deterministic or stochastic as is more common in RL literature.

$$\tau^{\pi}(s^i, k) = q^k(s^i) + \sum_{j=1}^{N} T(s^i, k, s^j) v^{\pi}(s^j) - v^{\pi}(s^i) \tag{3.10}$$

Here the values $v^{\pi}(s^i)$ are those calculated above for the original π. One property of the quantities τ^{π} is that $J^{\pi'} = \sum_{j=1}^{N} d^{\pi'}(s^j)\tau^{\pi}(s^j, \pi')$, where $d^{\pi'}$ are the stationary probabilities under policy π' and $\tau^{\pi}(s^j, \pi')$ is the quantity obtained by following policy π' in state s^j, but using values v^{π} associated with π. Since we have maximised over all actions in state s^i, we have:

$$\begin{aligned} \tau^{\pi}(s^i, \pi') &> \tau^{\pi}(s^i, \pi) \\ \tau^{\pi}(s^j, \pi') &= \tau^{\pi}(s^j, \pi),\ j \neq i \end{aligned}$$

From this together with the observation that $\tau^{\pi}(s^j, \pi) = J^{\pi}$ it follows that:

$$J^{\pi'} = \sum_{j=1}^{N} d^{\pi'}(s^j)\tau^{\pi}(s^j, \pi') > \sum_{j=1}^{N} d^{\pi'}(s^j)\tau^{\pi}(s^j \pi) = J^{\pi} \tag{3.11}$$

This means that the play π' gets a strictly higher payoff than π, while differing from π only in the action taken by player i. This contradicts the assumption that π is an equilibrium point of the limiting game, since player i can change its action to improve its payoff. This completes the proof that only the optimal policy can be an equilibrium in the limiting game. □

The above results, together with Theorem 2, indicate that the linear reward-inaction update scheme will converge to the unique equilibrium in any limiting game corresponding to an ergodic MDP. To see that this behaviour is also valid for the asynchronous update scheme $T1$ described in the previous section, we must compare the rewards in the limiting game, with the actual rewards received using $T1$: $s(\pi, t) = \{\beta(t) \mid \pi_t = \pi\}$. Since these returns depend on t, the $T1$ update is not the same as learning in the synchronous game setting. However, in an ergodic MDP following property holds: $\lim_{t\to\infty} s(\pi, t) = J^{\pi}$ [WJN86]. So for sufficiently large t, the ordering among $s(\pi, t)$ and J^{π} entries in the game will be the same and the limiting game analysis will provide a good approximation of the MDP-ILA algorithm's behaviour. This implies that in order to get accurate convergence in an MDP the learning rate used by the automata should be sufficiently small, so that the returns for the current policy can be accurately estimated.

Additionally, Wheeler and Narendra [WJN86] note that updates in the scheme $T1$ used above occur asynchronously, as opposed to the standard repeated game setting, where all players update synchronously. This does not cause issues,however, as the ergodic assumption ensures that all automata continue to be updated. These observations allow us to conclude that the limiting game analysis will provide an accurate prediction for the behaviour of the $MDP - ILA$ algorithm. Additionally, since Theorem 6 holds for all limiting games, we can conclude that the algorithm is optimal in ergodic MDPs, without having to analyse the limiting games for specific MDPs.

Example 4 (Limiting Game for a MDP) *Consider again the recycling robot problem described in the previous example. We will now show how this MDP can be cast as a limiting automata game. Since this problem has only* 2 *states, the corresponding game will have* 2 *automata players. Player* 1 *corresponding to state* $high$ *has* 2 *actions:* $search$ *and* $wait$*. Player* 2 *corresponding to state* low *has* 3 *actions:* $search$*,* $wait$ *and* $recharge$*. Each play in the game corresponds to one of the deterministic policies in Table 3.2. Since the MDP is ergodic, the expected reward starting from both states is the same. Table 3.3 shows the game in matrix form (with rewards normalised to be in* $[0, 1]$ *in order to apply* L_{R-I}*). The unique pure equilibrium is indicated in bold. It is immediately clear that this equilibrium corresponds to the optimal deterministic policy.*

		Player 2		
		search	wait	recharge
Player 1	search	**0.75**	0.45	0.74
	wait	0.40	0.36	0.37

Table 3.3: Limiting game for the Recycling Robot problem from Example 2

3.5 Summary

In this chapter we treated single agent learning in multi-state environments. In this setting an agent needs to learn a mapping or policy which determines the best action to take dependent on the current state of its surroundings. We defined the Markov decision process model which describes this setting, and briefly discussed the reinforcement learning approach to solv-

ing these problems. We then showed how a network of the learning automata, that were introduced in the previous chapter, can be used to find optimal policies. Finally we showed that the behaviour of such an automata network can be predicted by analysing a corresponding automata game.

Chapter 4

Decentralised Learning in Markov Games

After considering multi-agent, stateless problems and single agent, multi-state problems in the previous chapters, we now move to the full multi-agent, multi-state model. Introducing multiple agents to the MDP model significantly complicates the problem agents face. Both rewards and changes in the environment now depend on the actions of all agents present in the system. Agents are therefore required to learn in a joint action space. Moreover, since agents can have different goals an optimal solution which maximises rewards for all agents simultaneously may fail to exist.

To accommodate the increased complexity of this problem we use the representation of Markov games [Sha53]. While they were originally introduced in game theory, Markov games were more recently proposed as a standard framework for multi-agent reinforcement learning [Lit94]. As the name implies, Markov games still assume that state transitions are Markovian, however, both transition probabilities and expected rewards now depend on a joint action of all agents. Markov games can be seen as an extension of MDPs to the multi-agent case, and of repeated games to multiple states. If we assume only 1 agent, or that other agents play a fixed policy, the Markov game reduces to an MDP. When the Markov game has only 1 state, it reduces to a repeated normal form game.

In recent years many different approaches have been proposed to tackle Markov games. Most of these approaches focus on finding an equilibrium between agent policies, i.e. a situation in which no agent can single-handedly improve its payoff. However, almost all of the systems currently developed have either a very limited applicability, or lack convergence

guarantees in all but the most restrictive cases. In Section 4.5 we give an overview of current approaches, together with their applicability.

In this chapter we introduce our own Markov game algorithm. Our goal is to develop a broadly applicable system, which nonetheless can be assured to converge in self-play. The algorithm we propose is an extension of the MDP-ILA algorithm of the previous chapter. Like MDP-ILA it learns using only the basic reinforcement model. That is, agents only know the actions they selected and their individual feedback signal. They do not require any information on the actions or rewards of other agents present in the system.

In addition to the new algorithm, we also extend the limiting game analysis to multi-agent problems. We show how limiting games can be calculated for Markov games, and how these games can be valuable tools for the analysis and design of algorithms. Applying this analysis we will show the convergence of the proposed Markov game algorithm. We will see that for common interest Markov games, like for the MDPs in the previous chapter, we can approximate the problem by identical payoff automata game. However, due to the additional need for coordination between agents, we can only guarantee local optimality using the standard reward-inaction update. In general Markov games we can still apply the limiting game approximation, but the resulting games are now general sum games. The convergence results of Chapter 2 assure us that the collective automata can find pure equilibria in these games. Additionally, we will show that the pure equilibria of the limiting automata game are exactly the pure Nash equilibria of the Markov game.

The remainder of the chapter is structured as follows. We start out by formally defining the Markov game framework and the common interest special case of Multi-agent MDPs. Subsequently, we show how the MDP-ILA algorithm can be extended to accommodate multiple agents. This is followed by an extension of the limiting game analysis, which is first applied first to the common interest and then to the general Markov game case. Finally, we use this analysis to demonstrate the convergence of the extended ILA algorithm. We conclude the chapter with a discussion on related work. References for this chapter are: [Lit94, BBDS08, VVN08b, VVN08a]

4.1 Markov Games

An extension of the single agent Markov decision process (MDP) to the multi-agent case can be defined by Markov Games [Sha53]. In a Markov Game, actions are the joint result of multiple agents choosing an action independently.

Definition 15 *A Markov game is a tuple* $(n, S, A_{1,\ldots,n}, R_{1,\ldots,n}, T)$*, with:*

- n *the number of agents in the system.*
- $S = \{s^1, \ldots, s^N\}$ *a finite set of system states*
- A_k *the action set of agent* k*.*
- $R_k : S \times A_1 \times \ldots \times A_n \times S \to \mathbb{R}$*, the reward function of agent* k
- $T : S \times A_1 \times \ldots \times A_n \to \mu(S)$ *the transition function.*

Note that $A_k(s^i)$ is now the action set available in state s^i for agent k, with $k : 1 \ldots n$ and $i : 1, \ldots, N$. Transition probabilities $T(s^i, \vec{a}^i, s^j)$ and rewards $R^k(s^i, \vec{a}^i, s^j)$ now depend on a starting state s^i, ending state s^j and a joint action from state s^i, i.e. $\vec{a}^i = (a_1^i, \ldots a_n^i)$ with $a_k^i \in A_k(s^i)$. The reward function $R_k(s^i, \vec{a}^i, s^j,)$ is now individual to each agent k. Different agents can receive different rewards for the same state transition. Transitions in the game are again assumed to obey the Markov property of Definition 12.

As was the case in MDPs, agents try to optimise either their future discounted or their average reward over time. The main difference is that now these criteria also depend on the policies of other agents. This gives following definition for the expected discounted reward for agent k under a joint policy $\vec{\pi}$:

$$V_k^{\vec{\pi}}(s) = E^{\vec{\pi}} \left\{ \sum_{t=0}^{\infty} \gamma^t r_k(t+1) \mid s(0) = s \right\} \tag{4.1}$$

while the average reward for agent k is defined as:

$$J_k^{\vec{\pi}}(s) = \lim_{T \to \infty} \frac{1}{T} E^{\vec{\pi}} \left\{ \sum_{t=0}^{T} r_k(t+1) \mid s(0) = s \right\} \tag{4.2}$$

Expectations are now taken with respect to a *joint policy* $\vec{\pi} = (\pi_1, \ldots, \pi_n)$, which includes a policy π_k for every agent k. We will focus on the average

reward criterion. We will again assume that the Markov Game is ergodic in the sense that there are no transient states present and a limiting distribution $d^{\vec{\pi}}(s)$ on the states exists for all joint policies $\vec{\pi}$. We will only treat non-randomised, stationary policies:

Assumption 2 (Markov game ergodicity) *The Markov chain of system states generated by each joint policy $\vec{\pi}$ is ergodic.*

Let $\vec{\pi} = (\pi_1, \ldots, \pi_n)$ denote a joint policy in which agent k uses the (deterministic) policy π_k, and $\vec{\pi}(s^i)$ is the joint action played in state s^i, i.e $\vec{\pi}(s^i) = (\pi_1(s^i), \ldots, \pi_n(s^i))$. In this case the expected average reward for agent k under $\vec{\pi}$ is the same for all states and can be written as follows:

$$J_k^{\vec{\pi}} = \sum_{i=1}^{N} d^{\vec{\pi}}(s^i) \sum_{j=1}^{N} T(s^i, \vec{\pi}(s^i), s^j) R_k(s^i, \vec{\pi}(s^i), s^j) \tag{4.3}$$

Due to the existence of different reward functions, it is in general impossible to find an optimal policy for all agents. Instead, equilibrium points are sought. In an equilibrium, no agent can improve its reward by changing its policy if all other agents keep their policy fixed.

Definition 16 *Let $\vec{\pi} = (\pi_1, \ldots, \pi_n)$ denote a joint policy in a Markov game Γ in which agent k uses policy π_k and let $\vec{\pi}_{-k}$ denote the same joint policy without the policy of agent k. $\vec{\pi}$ is an average reward Nash equilibrium of Γ if for all $s \in S$ and $k : 1, \ldots, n$:*

$$J_k^{(\vec{\pi}_{-k}, \pi_k)}(s) \geq J_k^{(\vec{\pi}_{-k}, \pi'_k)}(s), \quad \forall \pi'_k \in \Pi_k$$

where J_k is defined in Equation 4.2 and Π_k is the set of policies available to agent k.

In general, a Markov game using the average reward criterion in Equation 4.2 does not necessarily have an equilibrium point in stationary (i.e. non-history dependent) policies. Examples of Markov games which only have equilibria in history dependent strategies can be found [Gil57]. It can be shown however that Markov games, satisfying Assumption 2 have at least one equilibrium in stationary policies [Sob71]. This equilibrium need not consist of pure policies, however.

In a special case of the general Markov game framework, the so-called team Markov games or multi-agent MDPs (MMDPs) [Bou96] optimal policies still exist. In this case, the Markov game is purely cooperative and all

agents share the same reward function. This specialisation allows us to define the optimal policy as the joint agent policy, which maximises the payoff of all agents.

Definition 17 (Multi-agent MDP) *A Multi-agent Markov Decision Process is a cooperative Markov game* $(n, S, A_{1,\ldots,n}, R, T)$ *in which all agents share the same reward function* R*.*

Because the agents share the same transition and reward function, one can think of the collection of agents being a single super agent with joint actions at its disposal and whose goal is to learn the optimal policy for the joint MDP.

Since the agents' individual action choices may be jointly suboptimal, the added problem in MMDPs is for the agents to learn to coordinate their actions so that joint optimality is achieved. The value of a joint policy $\vec{\pi} = (\pi_1, \ldots, \pi_n)$ with π_k the policy of agent k can still be defined by Equation 4.3, but is now the same for all agents.

Under Assumption 2, it is sufficient to only consider joint policies in which the agents choose pure strategies. This can easily be seen, as the problem can be reduced to an ergodic MDP by considering the super agent view described above. As noted in the previous chapter an optimal, deterministic policy always exist for these MDPs. Moreover, since no agent can improve on the payoff of this optimal joint policy, it will also be a pure Nash equilibrium for the MMDP. Unfortunately, the MDP approach used in the Chapter 3 can only be applied if we allow centralised control of all agents. As we are specifically considering multi-agent systems, this approach is generally not applicable. In this chapter we shall examine the difficulties that we encounter when moving from centralised control to a decentralised system with multiple independent agents.

Example 5 (Markov game) *Consider again the recycling robot problem from Chapter 3. We now add a second robot to the problem. The first robot has exactly the same actions as in the original problem: search (s), wait (w) and recharge (r). The second robot can choose either to aid the first robot (a) or wait for cans to recycle (w). Robots recycle the most when one goes out to search and one waits for the other robot to bring it cans. In this situation, the robot who goes to search gets a slightly lower reward since in expends more energy. If both robots go out to search they hinder each other and receive lower rewards. Similarly if both simply wait around, they collect less cans. Both robots get a reward depending on the amount recycled.*

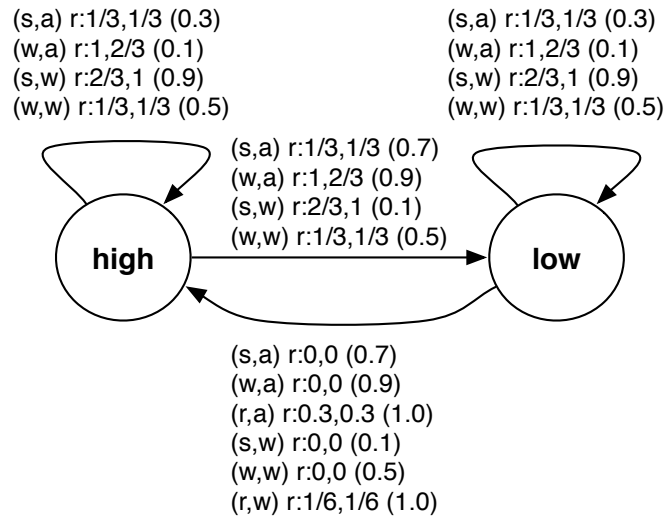


Figure 4.1: State diagram for the example Markov game.

To simplify matters we do not consider the battery level of the second robot (for example because it is solar powered). Thus the resulting still has 2 *states corresponding to the battery level of the first robot (i.e. high or low). When in the 'low' state robot* 1 *has the option to choose action recharge to immediately return to state 'high'. If it does not recharge but depletes its batteries, it is rescued and returned to state 'high' but both robots are punished with a reward of* 0*.*

The resulting problem is depicted in Figure 4.1. Here arrows indicate state transitions. The labels on the arrows give the transition probability and expected rewards for both robots (r) when this transition is made under each joint action.

4.2 Learning in finite Markov Games

In this section we extend the ILA-model of the previous chapter to the framework of Markov games. Instead of putting a single learning automaton in each state of the system, we propose to put an automaton LA (k, i) in each state s^i with $i : 1, \ldots, N$, for each agent k, $k : 1, \ldots, n$. At each time step only the automata of a single state are active; a joint action triggers the LA from that state to become active. Each automaton then individually selects an action. The resulting joint action triggers the next state transition and immediate rewards.

As before, LA (k, i) active for agent k in state s^i is not informed of the one-step reward $R(s^i, \vec{a}^i, s^j)$ resulting from choosing joint action $\vec{a}^i = (a^i_1, \ldots, a^i_n)$ with $a^i_k \in A_k(s^i)$ in s^i and leading to state s^j. When state s^i is visited again, all automata $LA(k, i)$, $k : 1, \ldots, n$ present in this state, receive two pieces of data: the cumulative reward r_k gathered by agent k up to the current time step and the current global time t. From these, all $LA(k, i)$ compute the incremental reward Δr_k generated for agent k since this last visit to the state and the corresponding elapsed global time Δt. These values are then added to the cumulative totals for the action a^i_k:

$$\begin{aligned} \rho_k(a^i_k) &\leftarrow \rho_k(a^i_k) + \Delta r_k \\ \eta_k(a^i_k) &\leftarrow \eta_k(a^i_k) + \Delta t \end{aligned}$$

Here a^i_k is the action that agent k chose as part of joint action $\vec{a}^i$ that was selected when the state s^i was visited the last time. The environment response $LA(k, i)$ receives for selecting a^i_k is exactly the same as in the MDP version of the algorithm, but now calculated for each agent separately:

$$\beta_k(t) = \frac{\rho_k(a^i_k)}{\eta_k(a^i_k)} \tag{4.4}$$

The complete algorithm is listed in Algorithm 3.

4.3 Limiting game analysis

In this section we extend the limiting game analysis to Markov games. This extension allows us to depart from the traditional state-action value perspective on Markov games, and instead to view them as repeated normal form games. We also show how we can examine the problem at different levels, to better understand the issues at hand.

First, we will examine the special case of MMDPs were all agents share the same payoff function. Three different viewpoints emerge; the single superagent view, the agent view and the LA-game view. Each of these views considers the problem at a different level, introducing additional complexities that arise at that level. The superagent view is concerned only with selecting the optimal joint action at each state and (unrealistically) assumes that centralised control of the agents is feasible. In the agent view we study the interactions between individual agent policies and the coordination required between agents to find the optimal joint policy. The lowest level LA

Algorithm 3 MG-ILA

initialise $r_{prev}(s,k)$, $t_{prev}(s)$, $a_{prev}(s,k)$,t, $r_{tot}(k)$,$\rho_k(s,a)$, $\eta_k(s,a)$ to zero, $\forall s,k,a$.
$s \leftarrow s(0)$
loop
for all Agents k **do**
if s was visited before **then**
• Calculate received reward and time passed since last visit to state s:

$$\Delta r_k = r_{tot}(k) - r_{prev}(s,k)$$

$$\Delta t = t - t_{prev}(s)$$

• Update estimates for action $a_{prev}(s,k)$ taken on last visit to s:

$$\rho_k(s, a_{prev}(s,k)) = \rho_k(s, a_{prev}(s,k)) + \Delta r_k$$

$$\eta_k(s, a_{prev}(s,k)) = \eta_k(s, a_{prev}(s,k)) + \Delta t$$

• Calculate feedback:

$$\beta_k(t) = \frac{\rho_k(s, a_{prev}(s,k))}{\eta_k(s, a_{prev}(s,k))}$$

• Update automaton $LA(s,k)$ using L_{R-I} update with $a(t) = a_{prev}(s,k)$ and $\beta_k(t)$ as above.
end if
• Let $LA(s,k)$ select an action a_k.
end for
• Store data for current state visit:

$$t_{prev}(s) \leftarrow t$$

$$r_{prev}(s,k) \leftarrow r_{tot}(k)$$

$$a_{prev}(s,k) \leftarrow a_k$$

• Execute joint action $\vec{a} = (a_1, \ldots, a_n)$, observe immediate rewards r_k and new state s'
• $s \leftarrow s'$
• $r_{tot}(k) \leftarrow r_{tot}(k) + r_k$
• $t \leftarrow t + 1$
end loop

View	Players	Action set	Applicability
Super agent	1 player per state	joint action set in each state: $\times_k A_k(s)$	Only MMDPs
Agent	1 player per agent	agent's pure policies: $\times_s A_k(s)$	MMDPs and Markov Games
Automata	1 player per state-agent combination	agent's action set in each state $A_k(s)$	MMDPs and Markov Games

Table 4.1: Overview of the different limiting game views. Each view analyses the problem at a different level. The table lists for each view the players and corresponding action sets in the limiting games as well as the problems to which this view is applicable.

view, not only deals with the coordination between agents, but also examines the interplay between state level action selections of the same agent.

As will become clear later, when applying the same analysis to general Markov games we lose the superagent perspective, but we are still able to study the agent and LA views. Table 4.1 lists the different views together with their applicability and describes the normal form game they use to study the problem.

We also show that under the Assumption 2, the agent and LA game view share the same pure equilibrium points. Using this result we can apply the automata game convergence proofs from Chapter 2 to show the convergence of our extended MG-ILA algorithm to a pure equilibrium.

4.3.1 MMDPs

Since the utility of a state in an MMDP is the same for all agents, the problem can be stated as an MDP in which the actions in each state are the joint actions of the agents. This means that a pure optimal policy in which all agents maximise their pay-off can still be found, using the MDP-ILA algorithm of the previous chapter. The use of this model is possible only if we assume that the agents are of one mind and select a joint action together. However this assumption is far from realistic. In general, agents are independent and select their actions individually. This complicates learning considerably since individual agent actions may be jointly suboptimal.

Therefore an important issue in learning MMDPs, is that of coordination.

Example 6 (MMDP) *As an example of an MMDP we consider the 2-robot recycling problem from Example 5, but with both agents having the reward function listed in Table 4.2. The problem states, transitions and actions are kept identical to the previous example. The main difference is that now both robots receive the same reward for each transition. Despite this fact, we will show that local optima can still occur and both robots need to coordinate to achieve the optimal payoff.*

s_t	s_{t+1}	$\vec{a}$	$R(s_t, \vec{a}, s_{t+1})$
high	low	(search,aid)	0.25
high	low	(wait,aid)	0.75
high	low	(search,wait)	1.0
high	low	(wait,wait)	0.25
high	high	(search,aid)	0.25
high	high	(wait,aid)	0.75
high	high	(search,wait)	1.0
high	high	(wait,wait)	0.75
low	low	(search,aid)	0.25
low	low	(wait,aid)	0.75
low	low	(search,wait)	1.0
low	low	(wait,wait)	0.25
low	high	(search,aid)	0
low	high	(wait,aid)	0
low	high	(recharge,aid)	0.3
low	high	(search,wait)	0
low	high	(wait,wait)	0
low	high	(recharge,wait)	0.1

Table 4.2: Common rewards for state transitions in the recycling robot MMDP.

Single Super Agent View

We start by examining a centralised view on the Markov game. While this may not correspond to a realistic multi-agent learning setting, it is nonetheless interesting to see what can be achieved when coordinated selection of join-actions in each state is possible. When we (unrealistically) assume that

the agents are of one mind, we can think of a single super agent being active. The joint actions of the MMDP are the actions of the super agent. In this case, the problem reduces to an MDP and can be solved by the model of Chapter 3, i.e. put one learning automaton in each action state. We can now look at the limiting game for this MDP, which we will call the super-agent view of the MMDP. The resulting game will have a player for every state. The action set for player i is the joint action set $A_1(s^i) \times \ldots \times A_n(s^i)$ of the corresponding state s^i. A play in this game selects a joint action $\vec{a}^i = (a_1^i, \ldots, a_n^i)$ for every state s^i. Such a play $\vec{a} = (\vec{a}^1, \ldots, \vec{a}^N)$ can be seen to correspond to a deterministic joint policy $\vec{\pi} = (\pi_1, \ldots, \pi_n)$ where $\vec{\pi}(s^i) = \vec{a}^i$.

Definition 18 *The super-agent view of an MMDP $(n, S, A_{1,\ldots,n}, R, T)$ is the limiting game for the MDP (S, A, R, T) with:*

$$A(s^i) = A_1(s^i) \times \ldots \times A_n(s^i) \quad \forall s^i \in S$$

The MMDP in Example 6 is then approximated by a 2 player identical pay-off game, i.e. 2 player because there are 2 states. Player 1 corresponds to state $high$ and has action set $\{(search, aid), (wait, aid), (search, wait), (wait, wait)\}$ consisting of all possible joint actions in this state. Player 2 corresponds to state low and can perform the same actions, in addition to actions $(recharge, wait)$ and $(recharge, aid)$. The rewards in the game are given by the long-term expected reward per time step. In total 24 pure joint policies are possible. This game is given in Table 4.3. One can see that only optimal equilibria are present.

As proved in Theorem 5, the interconnected learning model of Section 3.3 will find the optimal joint policy of the problem, i.e the super agent chooses joint action $(search, wait)$ in both states low and $high$.

The multi-agent View

In the multi-agent view, we no longer assume that agents jointly select their action in each state. In this view we examine the outcomes of joint policies from the perspective of the agents. The entire Markov game is represented as a single normal form game in which each agent has its deterministic policies in the Markov game as actions in the repeated game. Note that despite the fact that we have different players and action sets, a play in this game still corresponds to a joint (deterministic) policy for the Markov game. The reward each player receives for such a play is again the reward for the joint policy as given by Equation 4.3.

		$state : low$					
		(search,aid)	*(wait,aid)*	*(recharge,aid)*	*(search,wait)*	*(wait,wait)*	*(recharge,wait)*
state: high	$(search, aid)$	0.1625	0.1734	0.2706	0.8187	0.1771	0.1882
	$(wait, aid)$	0.3703	0.4125	0.5368	0.8850	0.3482	0.4421
	$(search, wait)$	0.9364	0.9075	0.9364	**0.9500**	0.8542	0.9182
	$(wait, wait)$	0.1771	0.1875	0.2667	0.8850	0.1875	0.2000

Table 4.3: An identical payoff game with 2 players that approximates the single agent view of the MMDP of Example 6. The unique equilibrium is indicated in bold.

We will see that by decentralising decision making over the agents, we lose the global optimality of the super-agent view. In this view of the Markov game suboptimal Nash equilibria can occur, due to miscoordinations of the agents. The pure Nash equilibria given in this view are exactly the pure equilibria of the Markov game as given by Definition 16.

Definition 19 *The multi-agent view of an ergodic MMDP game (n,S,A,R,T) is a normal form game* $(n, \Pi_{1,\ldots,n}, J)$*, where:*

- *each agent* $k : 1, \ldots, n$ *in the Markov game corresponds to a player.*
- *each player* k *has its set of possible deterministic policies* Π_k *in the Markov game as action set.*
- *the reward received by all players for a play* $\vec{\pi} = (\pi_1, \ldots, \pi_n)$ *is* $J^{\vec{\pi}}$ *as defined by Equation 4.3*

This means that the underlying game played now depends on the agents' individual policies. Consider the agent game for Example 6. Again this game is a 2-player identical payoff game. But here we have a 2-player game, because we have 2 agents. Player one corresponds to the first agent and can take actions $[search, search]$, $[search, wait]$, $[wait, search]$, $[wait, wait]$, $[search, recharge]$ and $[wait, recharge]$. Player 2 corresponds to the second agent and has access to actions $[aid, aid]$, $[aid, wait]$, $[wait, aid]$ and

$[wait, wait]$. Note that here $[a_1, a_2]$ denotes a policy instead of a joint action, i.e. the agent takes action a_1 in state $high$ and action a_2 in state low.

In Table 4.4 we show the game matrix for the MMDP in the example. Surprisingly, in this game, 4 equilibria are present of which 1 is optimal and 3 are sub-optimal. While, as is shown in Theorem 6, in the super agent view only optimal equilibria can appear, this is clearly not the case in the multi-agent view.

It should be noted that in the agent view we are looking at the possible outcomes for *deterministic* policies of the agents. This analysis is no longer valid when one considers stochastic policies and mixed strategies of the game. This is due to the fact that mixed strategies in the multi-agent view need not correspond to stationary strategies. Consider for example the game given in Table 4.4. A mixed strategy in this game where the agent 1 plays each of the policies (wait,wait) and (search,search) 50% of the time, i.e. either select 'wait' in both states or select 'search' in both states. This strategy cannot be implemented in the Markov game by using stationary policies. Any policy which randomises the action selection based only on the system state will result in sometimes combining action 'wait' with action 'search'. Thus while the pure strategies of the agent view correspond to the (stationary) deterministic policies of the Markov game, this is not always the case for mixed strategies. In subsequent chapters we will consider a method for implementing non-stationary policies corresponding to mixed strategies of the agent game.

		$agent2$			
	$policies$	$[aid, aid]$	$[wait, aid]$	$[aid, wait]$	$[wait, waits]$
agent 1	$[search, search]$	0.1625	0.8844	0.8187	**0.9500**
	$[wait, search]$	0.3703	0.1771	**0.8850**	0.7917
	$[search, wait]$	0.1734	0.9075	0.1771	0.8542
	$[wait, wait]$	0.4125	0.1875	0.3482	0.1875
	$[search, recharge]$	0.2706	**0.9364**	0.1882	0.9182
	$[wait, recharge]$	**0.5368**	0.2667	0.4421	0.2000

Table 4.4: An identical payoff game with 4 actions that approximates the multi agent view of the MMDP of Example 6. Equilibria are indicated in bold.

The Learning Automata View

The third and last view we have on the control problem is the automata view. In this view we consider the game between all the learning automata that are present in the different action states, using the MG-ILA algorithm. This limiting automata is the direct extension of the limiting game for the MDPs of Chapter 3.

Each player in the game is again an automaton. The difference with the MDP case is that instead of a single player per state, we now have n players for each state, one representing each agent. A player (k, i) representing agent k in state s^i, has the corresponding action set $A_k(s^i)$ available. This means that a play in this game results in 1 action for each agent, for each state and so is again a joint policy (as was also the case in the previous views). Rewards for a play are again given by calculating the expected average reward as given by Equation 4.3. In the MMDP case this means that all automata receive the same payoff for a joint policy. Unfortunately, as was the case in the agent view, the unique equilibrium property of MDPs no longer holds and sub-optimal equilibria can occur. In the next section we will show that this view shares the same pure equilibria with the multi-agent view and that the ILA model is able to find an equilibrium policy of the Markov game.

Definition 20 *The automata view for an MMDP* $(n, S, A_{1,\ldots,n}, R, T)$ *is the limiting automata game* $(n \times |S|, A^{1,\ldots,|S|}_{1,\ldots,n}, J)$ *with:*

- *One player* (k, i) *corresponding to each possible state-agent pair* (k, s^i).
- $A^i_k = A_k(s^i)$ *the action set of player* (k, i)
- $J^{\vec{\pi}}$ *the reward all players receive for a play* $\vec{\pi}$.

For the MMDP of Example 6 this results in a 4 automata game. The complete game is shown in Table 4.5. For reasons of clarity we abbreviate each action to its first letter. Players are indicated by the agent and state combination they represent, i.e. $(2, l)$ is the automaton used by agent 2 in state low.

4.3.2 Markov Games

In this section we extend our approach to general Markov games. In these games each agent k has its own reward function R_k. The expected individual reward J_k of agent k can now be calculated by using Equation 4.3.

((1,l),(1,h),(2,l),(2,h))	$J^{\vec{\pi}}$	((1,l),(1,h),(2,l),(2,h))	$J^{\vec{\pi}}$
(S,S,A,A)	0.1625	(S,S,A,W)	0.8187
(W,S,A,A)	0.3703	(W,S,A,W)	**0.8850**
(S,W,A,A)	0.1734	(S,W,A,W)	0.1771
(W,W,A,A)	0.4125	(W,W,A,W)	0.3482
(S,R,A,A)	0.2706	(S,R,A,W)	0.1882
(W,R,A,A)	**0.5368**	(W,R,A,W)	0.4421
(S,S,W,A)	0.8844	(S,S,W,W)	**0.9500**
(W,S,W,A)	0.1771	(W,S,W,W)	0.7917
(S,W,W,A)	0.9075	(S,W,W,W)	0.8542
(W,W,W,A)	0.1875	(W,W,W,W)	0.1875
(S,R,W,A)	**0.9364**	(S,R,W,W)	0.9182
(W,R,W,A)	0.2667	(W,R,W,W)	0.2000

Table 4.5: An identical payoff game between 4 players that approximates the LA view of the MMDP of Example 6. Equilibria are indicated in bold.

Since there is no longer a single reward function we cannot treat the problem as a single agent MDP anymore, and the single agent view of the previous section is no longer applicable.

It is still possible, however, to approximate the Markov Game using the agent and learning automata views. The definitions of both limiting games can easily be extended to include individual rewards. The corresponding definitions are given below in Definition 21 (agent view) and Definition 22 (LA view). The main difference with the MMDP section is that the resulting approximating games are no longer common interest, but rather conflicting interest. In the agent game, each agent can now get a different reward for a joint policy. In the automata game, automata belonging to the same agent k all get the same payoff J_k, while those belonging to different agents can get different rewards. In the next section we give convergence results for MG-ILA for these more general games, which include the MMDPs of the previous section as a special case.

Definition 21 *The multi-agent view of an ergodic Markov game (n,S,*$A_{1,\ldots,n}$*,* $R_{1,\ldots,n}$*,T) is a normal form game* $(n, \Pi_{1,\ldots,n}, J_{1,\ldots,n})$*, where:*

- *each agent* $k : 1, \ldots, n$ *in the Markov game corresponds to a player.*
- *each player* k *has its set of possible deterministic policies* Π_k *in the Markov game as action set.*

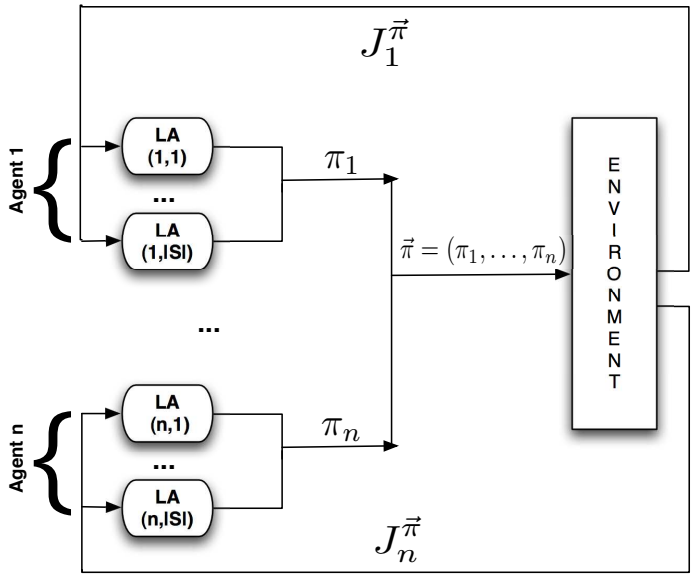


Figure 4.2: Game representation of the automata Markov game algorithm.

- *the reward received by player k for a play $\vec{\pi} = (\pi_1, \ldots, \pi_n)$ is $J_k^{\vec{\pi}}$ as defined by Equation 4.3*

Definition 22 *The automata view for a Markov game $(n, S, A_{1,\ldots,n}, R_{1,\ldots,n}, T)$ is the limiting automata game $(n \times |S|, A_{1,\ldots,n}^{1,\ldots,|S|}, J)$ with:*

- *One player (k, i) corresponding to each possible state-agent pair (k, s^i).*
- *$A_k^i = A_k(s^i)$ the action set of player (k, i)*
- *$J_k^{\vec{\pi}}$ the reward all players $(k, i), i : 1, \ldots, N$ receive for a play $\vec{\pi}$.*

The resulting limiting game for the multi-agent view is shown in Figure 4.6

4.3.3 Theoretical Results

In this section we theoretically examine the convergence properties of the MG-ILA algorithm. We do this by showing a correspondence between the

		$agent2$			
	policies	$[aid, aid]$	$[wait, aid]$	$[aid, wait]$	$[wait, wait]$
agent 1	$[search, search]$	0.2167	0.5958	0.5667	**0.6333**
	$[wait, search]$	0.4937	0.2361	**0.6400**	0.5556
	$[search, wait]$	0.2313	0.6100	0.2361	0.5833
	$[wait, wait]$	0.5500	0.2500	0.4643	0.2500
	$[search, recharge]$	0.3196	**0.6333**	0.2647	0.6212
	$[wait, recharge]$	**0.6684**	0.3222	0.6053	0.2778

		$agent2$			
	policies	$[aid, aid]$	$[wait, aid]$	$[aid, wait]$	$[wait, wait]$
agent 1	$[search, search]$	0.2167	0.8875	0.8292	**0.9500**
	$[wait, search]$	0.3479	0.2361	**0.8767**	0.8056
	$[search, wait]$	0.2167	0.9067	0.2361	0.8611
	$[wait, wait]$	0.3667	0.2381	0.3452	0.2500
	$[search, recharge]$	0.3196	**0.9364**	0.2647	0.9242
	$[wait, recharge]$	**0.4930**	0.3222	0.4298	0.2778

Table 4.6: A conflicting interest game with 4 actions that approximates the multi-agent view of the Markov game of Example 5. The first matrix gives payoffs for agent 1, the second for agent 2. Equilibrium payoffs are indicated in bold.

pure equilibria of the agent view and LA view limiting games. Because of this correspondence, we can use MG-ILA to search equilibria at the automata level, and still be assured that the equilibrium that is found, is also an equilibrium between agent policies. We only treat the general Markov game case, since results for this case also hold for MMDPs as a special case.

Note first that a play for the LA-view is automatically also a play for the agent view. The same notation can be used, only the interpretation is different. A play in the multi-agent view gives for each agent a set of individual actions, one for each action state: $\vec{\pi}_{multi-agent} = (\pi_1, \ldots, \pi_n)$ with $\pi_k = [a_k^1, \ldots, a_k^N]$. In the LA-view a play is composed of an individual action for each LA in the system. As such $\vec{\pi}_{LA} = (a_1^1, \ldots, a_1^N, a_2^1, \ldots, a_2^N, \ldots, a_n^1, \ldots, a_n^N)$ which is exactly the expansion of $\vec{\pi}_{multi-agent}$. As note above, both these plays correspond to a deterministic joint policy for the Markov game.

We can now state the following:

Theorem 7 *Given a Markov Game* $\Gamma = (n, S, A_{1,\ldots,n}, R_{1,\ldots,n}, T)$*. Assume that the Markov Chain, corresponding to each joint policy* $\vec{\pi}$ *is ergodic. If* $\vec{\pi}$ *is a pure equilibrium policy in the multi-agent view,* $\vec{\pi}$ *is also an pure equilibrium policy for the LA-view and vice versa.*

Proof:

First we note that an equilibrium in the agent game must always be an equilibrium in the LA game. This is easy to see, because a learning automaton switching its action corresponds to a single agent changing its policy in a single state. So in any situation where no agent alone can improve its expected reward by changing policies, it must also be impossible for a single learning automaton to improve its payoff.

Now suppose that $\vec{\pi} = (\pi_1, \ldots, \pi_n)$ is an equilibrium for the LA game view but is not an equilibrium point for the agent game view. This means no automaton alone can improve its payoff by switching to another action. To get out of the equilibrium at least 2 LA should change their action choice. However since we assumed that $\vec{\pi}$ is not an equilibrium point in the agent view, it is possible to find a single agent k that can improve its payoff simply by changing its strategy $\pi_k = [a_k^1, \ldots, a_k^N]$. Because of the above, it has to do this by changing the actions of 2 or more different LA. For simplicity assume that agent k changes its strategy in state s^i and state s^j with $i \neq j$[1] and call this new strategy $\pi'_k = [b_k^1, \ldots, b_k^N]$.

Denote $\vec{\pi}_{-k}$ to be the joint policy $\vec{\pi}$ without the policy of agent k and $(\vec{\pi}_{-k}(s^i), \pi_k^*)$ with $\pi_k^* \in A_k(s^1) \times \ldots \times A_k(s^N)$ the joint policy in which all agents play follow $\vec{\pi}$, except agent k who follows policy π_k^*. Furthermore define $(\vec{\pi}_{-k}(s^i), a_k^*))$ as the joint action in state s^i where all agents follow $\vec{\pi}$ but agent k plays action $a*_k \in A_k(s^i)$. Then by construction we have that for $\pi_k = [a_k^1, \ldots, a_k^N]$ and $\pi'_k = [b_k^1, \ldots, b_k^N]$, the following holds: $a_k^i \neq b_k^i$, $a_k^j \neq b_k^j, \forall l : 1 \ldots N, l \neq\ i, j : a_k^l = b_k^l$ and $J_k^{(\vec{\pi}_{-k}, \pi'_k)} > J_k^{(\vec{\pi}_{-k}, \pi_k)}$.

We will now show that $\vec{\pi}$ cannot be an equilibrium of the automata game, by showing that agent k can change its policy in exactly 1 state (i.e. let 1 LA change its action) and receive a strictly higher payoff. Remember that $\Gamma = (n, S, A_{1,\ldots,n}, R_{1,\ldots,n}, T)$ is the Markov game under consideration. We now construct a corresponding MDP $\Gamma_k = (S, A_k, R', T')$ where:

- Γ and Γ_k have the same set of states S
- A_k is the action set of agent k in the Markov game.

[1] The case where 3 or more automata should switch their action is analogous.

- $R'(s^i, a_k^i, s^j) = R_k(s^{'}(\vec{\pi}_{-k}(s^i), a_k^i), s^j)$, where R_k is the reward function of agent k in the Markov game.
- $T'(s^i, a_k^i, s^j) = T(s^i, (\vec{\pi}_{-k}(s^i), a_k^i), s^j)$ with T the transition function of Γ, gives the transition probabilities in the MDP.

So the MDP Γ_k is exactly the Markov game but with all agents $m \neq k$ playing according to the fixed joint policy $\vec{\pi}$. Since we have assumed that the Markov game is ergodic, the MDP will also be ergodic. This means that according to Theorem 5 of Chapter 3 the limiting game for this MDP only has optimal equilibria. As was shown in the proof of the theorem, for each sub-optimal policy, a better policy which differs only in a single state can be constructed. Because of the definition of T' and R', the expected average reward of a policy π in the MDP is exactly:

$$J^{\pi} = J_k^{(\vec{\pi}_{-k}, \pi)}$$

Now consider the policies π_k and π'_k for agent k in Γ. These are also policies in the MDP and we have:

$$J^{\pi'_k} = J_k^{(\vec{\pi}_{-k}, \pi'_k)} > J_k^{(\vec{\pi}_{-k}, \pi_k)} = J^{\pi_k}$$

So π_k cannot be an optimal policy for the MDP Γ_k, and we can find a better policy $\pi*$ which differs from π_k in only 1 state. This policy $\pi*$ is also a valid policy for agent k in the original Markov game Γ. So we now have a policy $\pi*$ in the original agent game which differs from the original policy π_k only in the action of a single automaton $LA(k, *)$, and receives a strictly higher payoff $J^{\pi*} = J_k^{(\vec{\pi}_{-k}, \pi*)}$. This means that a single automaton of agent k can change its action to receive a higher payoff and thus $\vec{\pi}$ cannot be an equilibrium in the full automata game. This contradicts our original assumption that $\vec{\pi}$ is an equilibrium for the automata game, but is not an equilibrium for the agent game view. Therefore we can conclude that both views share the same pure equilibria.

□

Corollary 1 *The MG-ILA model proposed in 4.2 converges locally to an equilibrium in pure strategies in a Markov game that satisfies Assumption 2.*

Proof:

According to Theorem 3, of Chapter 2 the automata will converge a pure equilibrium point of the corresponding automata game. But because

of Theorem 7 this point will also be an equilibrium point of the limiting agent game.

□

4.4 Experiments

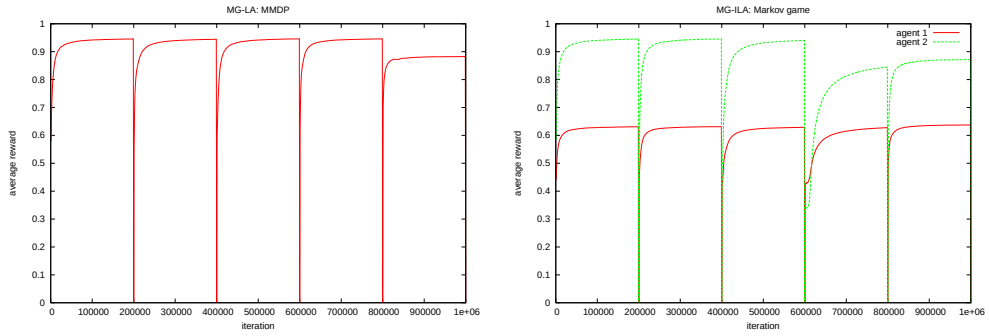
We now demonstrate the behaviour of the LA learning model on the sample problems described in the examples above. Figures 4.3(a) and (b) show the results of the algorithm on the sample MMDP and Markov game of Examples 6 and 5, respectively. Since we are interested in the long term convergence, we show a typical run, rather than an average over multiple runs. To demonstrate convergence to the different equilibria, we use a single very long run (1 million time steps) and restart the automata every 200000 steps. After every restart the automata are initialised with random probabilities to allow them to converge to different equilibria.

In Figure4.3(a) we see the common average reward for both agents in the MMDP of Example 6. The agents converge once to the suboptimal and three times to the optimal equilibrium point. After 200000 the time steps the average reward approaches the predicted values of 0.9500 and 0.8850 very closely. In Figure 4.3(b) we show the average reward for both agents on the Example 5, since each has its own reward function. Out of the 5 initialisations, the algorithm converges 3 times to the equilibrium play where agent 1 uses policy $[search, search]$ and 2 uses $[wait, wait]$. The last 2 the agents converge 2 to the equilibrium in which they play $[wait, search]$ and $[aid, wait]$, respectively. Again the average rewards can be seen to closely approximate the predicted values, now with different payoffs for each agent.

4.5 Related Work

Stochastic or Markov games were introduced by Shapley [Sha53] and extended to the infinite horizon in [HK66]. Early Markov game research focused mainly on existence criteria and computation methods for equilibria [FVV97]. Littman first suggested the use of Markov games as a framework for multi-agent reinforcement learning [Lit94]. A recent overview of multi-agent reinforcement learning (MARL) approaches for Markov games can be found in [BBDS08].

Many of the MARL approaches for Markov games are based on the



(a) (b)

Figure 4.3: Typical run of the LA learning model of Section 4.2 (a)Results on the MMDP of Example 6. (b)Results for the Markov game of Figure 4.1. Both experiments used automata with the L_{R-I} update scheme and a learning rate of $\lambda_1 = 0.05$.

Q-learning algorithm detailed in Section 3.2.1. The most straightforward strategy is to directly apply Q-learning (or another reinforcement learning algorithm), by using independent agents who effectively ignore each other. This setting is examined in [SSH94, CB98], for instance. However, not taking into account the other agents means that learners face a non-stationary environment and convergence guarantees are lost. This issue has lead to the development a large number of Q-learning extensions, with varying assumptions and convergence results.

In MMDPs the *Team Q-learning* algorithm [Lit01b] lets agents independently learn the optimal the common Q-values. The approach assumes that the optimal joint actions are unique, however, and agents may fail to coordinate if this is not the case. Alternatively, *Distributed Q-learning* [LR00] allows agents to find the common, optimal joint policy, but the algorithm is only applicable in deterministic environments. The *Sparse Tabular Q-learning* (STQ) technique uses the *Coordination graphs* introduced in [GLP02] to prevent miscoordination. These graphs explicitly specify the states in which agents need to coordinate their actions. Agents can then rely on single agent Q-learning by default, but learn over joint actions when coordination is required. While STQ requires the coordination graphs to be specified manually beforehand, in [KtHBV05] the authors an extended method, called *Utile Coordination*, to learn the graphs.

In 2-player zero-sum Markov games *minimax-Q* can be used. This algorithm alters the Q-learning update rule to use the *minimax* value to es-

timate returns for state-action pairs. This method is extended in [Lit01a] to the *Friend or Foe Q-learning* (FFQ) algorithm for general sum Markov games. This algorithm classifies agents as either friend or foes depending on whether or not they cooperate with the learner to maximise its expected payoff. While FFQ can be shown to converge, convergence to a Nash equilibrium is only achieved in common interest and zero-sum games.

Nash-Q [HW98, HW03] views the learning problem as a sequence of *stage games*, which are determined by the Q-values of the current state. The Q-values are then updated by assuming that the agents will play according to a computed Nash equilibrium in all subsequent states. Under some very restrictive conditions on the structure of the stage games, Nash-Q can be shown to converge to a Nash Equilibrium.

In [LRD07, Li03] Nash-R, an average reward version of Nash-Q, is proposed. This method, together with the MG-ILA approach proposed in this dissertation, are the only MARL approaches we are aware of that do not use the discounted reward criterion. While both Nash-R and MG-ILA are defined for ergodic Markov games, Nash-R agents need additional information on their opponents' actions and rewards in order to compute the stage games. Furthermore, convergence to a Nash equilibrium requires the same restrictive conditions as Nash-Q.

Another issue with both Nash-Q and Nash-R is that currently no efficient method for computing Nash equilibria is known. This makes the Nash update rule costly, since it requires the calculation of Nash equilibria for the stage games. This can be remedied by using other equilibrium concepts to update Q-values. *Correlated Q-learning* (CE-Q) [GHS03] and *Asymmetric Q-learning* [Kon03a] rely on the *correlated equilibrium* [Aum74] and the *Stackelberg equilibrium* [vS34] concepts, respectively. Both algorithms follow the Nash-Q principle of updating Q-values using equilibrium values, but their respective equilibrium concepts can be computed efficiently.

As an alternative to equilibrium based updates, Q-values may also be estimated by using models of the other agents. *Nonstationary Converging Policies* (NSCP) agents, for example, learn models of opponents and uses these models to compute a best response. This best response policy is then used to estimate state values in the Q-learning update. Finally, the *Hyper-Q* algorithm [Tes04] estimates the strategies of other agents and uses this information as part of its state representation. The idea is that this will allow the agent to adapt better adapt to non-stationary opponents. However, since the strategies of agents are continuous variables, it cannot use the standard Q-learning formulation and needs to rely on function approximation techniques. This makes obtaining general convergence results more

difficult.

Regrettably, recent research [ZGL06] indicates that learning Q-values may not be sufficient in order to always guarantee convergence to an equilibrium in all general Markov game settings. Naturally, other learning methods, that are not based on Q-learning do exist. The *Policy Search* algorithm [PKKM00] is able to find local optima over factored policies in MMDPs, but cannot guarantee equilibrium convergence. Similarly *Policy gradient* [Kon03b] method and can be used with function approximation techniques. The strongest optimal convergence guarantees for common interest problems are provided by *Optimal Adaptive Learning* (OAL) [WS03]. This algorithm learns the optimal joint policy by constructing virtual games on top of every stage game of the MMDP. In Chapter 6 we will define another ILA algorithm, *MMDP-ILA*, which is also able to find optimal policies, but does not need additional information on other agents.

Bowling [BV01a] proposes the "Win or Learn Fast" (WoLF) approach. This heuristic method increases or decreases the learning rate based on the performance of the agent. This approach can be combined with different learning schemes, such as: *Infinitesimal Gradient ascent* (IGA) [SKM00], Generalised IGA (GIGA) [Zin03]and Policy Hill Climbing (PHC) [BV01b] This has lead to a whole family of WoLF algorithms: WoLF-IGA [BV01a], GIGA-WoLF [Bow05], WoLF-PHC [BV01b] and PDWoLF [BP03]. While WoLF-IGA and GIGA-WoLF are evaluated only for repeated normal form games, WoLF-PHC and PDWoLF have been shown to achieve good results in general Markov games.

As is clear from the descriptions above a large body of work on multi-agent reinforcement learning exists. Unfortunately, none of these methods can offer general equilibrium convergence. Additionally, most methods do not only require a scalar feedback signal, but also need information on other agents' actions and rewards which can limit their applicability. Table 4.7 gives an overview of the approaches described above, their information requirements and their target tasks.

4.6 Summary

In this chapter we study the problem of learning Markov Games with independent agents that only have knowledge of their own payoff, reward and the current state. We propose a model based on interconnected learning automata algorithm for MDPs. The extended algorithm puts an additional learning automaton for each agent in each state and updates the automata

		Applicabillity	
		MMDP	General Markov Games
Information Requirement	Independent Agents	Policy Search [PKKM00] Policy Gradient [Kon03b] MMDP-ILA [VVN07d]	MG-ILA [VVN08a] (WoLF-)PHC [BV01b] PDWoLF [BP03] Independent learners [CB98]
	Joint Action Learners	OAL [WS03] Team-Q [Lit01b] Distributed Q [LR00] STQ [KV04] Utile Coordination [KtHBV05]	Nash-Q [HW98] Nash-R [Li03] FFQ [Lit01a] Hyper-Q [Tes04] Asymmetric Q [Kon03a] NSCP [WR04]

Table 4.7: Overview of current MARL approaches. Algorithms are classified by their applicability (common interest or general Markov games) and their information requirement (scalar feedback or joint-action information).

for each agent as was done in the single agent case.

Additionally, we extend the limiting game method introduced in the previous chapter, and show that limiting games can be used to study Markov games from different perspectives. These limiting games view the Markov game as a single large normal form game, rather than taking the common state-action view employed by most RL techniques. For the common interest MMDP case 3 viewpoints emerge. In the centralised superagent perspective, we maintain the optimality results obtained for MDPs in Chapter 3. When looking at the problem from a multi-agent point of view, we lose the optimality guarantee, due to the possibility of agent miscoordination. Finally,we can also examine the problem from the state-based automata view. This view corresponds to treating the problem as an automata game, as was done for MDPs. When moving from the common interest case to general Markov games, the centralised approach becomes impossible, but the other perspectives remain.

An important result is also obtained, linking the low level automata view to the multi-agent view, and showing that both contexts share the same pure equilibria. Together with existing automata convergence results, this property ensures the equilibrium convergence of the extended ILA algorithm. This result was also experimentally demonstrated on some small sample problems. We finish the chapter by situating our new algorithm in

the existing body of related work.

Chapter 5

Dynamics of Learning

A powerful tool to analyse learning in repeated games is the relation between RL and replicator dynamics (RD). More precisely, in [TtHV06, BS97, PTL08] the authors derived a formal link between the replicator equations of Evolutionary Game Theory (EGT) and reinforcement learning techniques such as Q-learning and Learning Automata. In particular this link showed that in the limit these learning algorithms converge to a certain form of the RD. This allows us to use the RD to establish what states a given learning system will settle into over time and what intermediate states it will go through. Recently it was demonstrated that there are a number of benefits to exploiting this link: the model predicts desired parameters to achieve Nash equilibria with high utility, the intuitions behind a specific learning algorithm can be theoretically analysed and supported by using the basins of attraction, and it was shown that the framework could easily be adapted and used to analyse new MAL algorithms, such as for instance lenient Q-learning [PTL08].

A limitation of using the RD as a model of MAL, however, is that it has only been applied in stateless repeated games. In this chapter we propose an extension of this method to general Markov games. By applying the limiting game approach introduced in the previous chapters, we show that the EGT connection can still be used to study multi-state problems. We demonstrate the EGT analysis in multiple state problems for multiple agents using the MG-ILA algorithm as RL technique. We do this by introducing a combination of switching dynamics and RD, which we call Piecewise Replicator Dynamics, to describe the learning processes over the multiple states. We calculate a limiting average reward game for each state, which takes into account the rewards that are obtained in the other states. This game can

then be studied using the replicator dynamics. The resulting dynamics apply under the assumption that agents are only learning in a single state. In reality, however, agents update their action probabilities in all states, and these probabilities will all change in parallel. Changes in action probabilities in one state will cause the average reward games to change in other states. The dynamics observed for an average reward state game are only a snapshot of the true learning dynamics.

To account for these changes we model the system as a piecewise dynamical system. Based on the qualitative changes in the dynamic, we partition the state space in a number of cells which correspond to the different possible attractors in the state games. We assume that each cell has its own fixed replicator dynamic. The entire system can then be modeled by updating the current action probabilities in each state, according to the replicator dynamics of the current cell. When the action probabilities leave this cell the equilibria of the state game change, and we get a new replicator dynamic that drives the system. In this way we follow the trajectory in all states through multiple cells and dynamics until an equilibrium is reached.

The remainder of this chapter is structured as follows. In Section 5.1.1 we elaborate on the necessary background. Section 5.2 introduces piecewise replicator dynamics for modeling multi-learning in multi-state problems. Section 5.3 demonstrates our approach with some experiments. Finally, we conclude with a discussion in Section 5.4. References for this chapter are: [Gin00, TtHV06, VTWN08]

5.1 Evolutionary Game Theory

Evolutionary Game Theory (EGT) studies the propagation of strategies through a population of agents. Rather than the rationality based approach of traditional game theory, EGT uses biological principles like fitness and natural selection to determine the outcome of a repeatedly played game. The underlying idea here is that strategies which have a higher payoff will be adopted by more agents and will spread faster.

In EGT, games are played by a population of agents. Members of the population are selected randomly to play the game pairwise. Each individual uses a fixed strategy. The fitness of an individual is the expected payoff of that individual's strategy, considering the current distribution of strategies in the population. After playing the game, the population is replaced by a new generation. Each agent produces offspring using the same strategy, to take its place in the next generation. However, the number of

offspring an agent produces is determined by its performance in the game. This ensures that strategies that have a high fitness will reproduce faster.

In the following sections we provide a short overview of the key concepts of EGT. This is followed by an explanation of the relation between EGT and reinforcement learning. A more detailed account of EGT can be found for instance in [Wei97, Gin00, Sam98].

5.1.1 Evolutionary Stability

One of the fundamental questions of EGT is whether a strategy is *evolutionary stable*. Evolutionary Stable strategies (ESS) are those strategies that are resistant to invasion by another strategy. This means that when the entire population plays the strategy, a small number of mutants that switch to a different strategy cannot take over the population. The mutants will always have a lower expected payoff and thus a lower rate of reproduction. This ultimately leads to the extinction of the invading strategy. Let $R(\sigma, p)$ denote the expected payoff strategy σ receives when playing against population p, i.e. $R(\sigma, p)$ is the expected payoff in the game when an opponent is drawn randomly from population p. The population p here is described in terms of the fractions of members that play each available strategy. We can now formally define the ESS concept as follows:

Definition 23 *A strategy σ is an evolutionary stable strategy if for all $\sigma' \neq \sigma$, there exists a $\delta \in]0, 1[$ such that for all $0 < \epsilon < \delta$,*

$$R(\sigma, (1-\epsilon)\sigma + \epsilon\sigma') > R(\sigma', (1-\epsilon)\sigma + \epsilon\sigma')$$

where R is the payoff function for the game being studied and $((1-\epsilon)\sigma + \epsilon\sigma')$ describes a population with fraction $1-\epsilon$ of individuals playing strategy σ and fraction ϵ playing strategy σ'.

This definition describes the requirement for a strategy to be resistant to takeover by a mutant strategy. Suppose that a population originally consisting entirely of players using strategy σ, incurs a small fraction ϵ of mutations, causing the mutated players to switch to strategy σ'. For this mutant strategy to take over the entire population they need to achieve a higher payoff than strategy σ when we draw an opponent from the mutated population, i.e. have a probability $1-\epsilon$ of drawing an opponent playing σ and probability ϵ of drawing an opponent playing σ'. When no such mutant strategy can be found, σ is an ESS.

Maynard-Smith [Smi82] demonstrated that evolutionary stability is actually a refinement of the Nash equilibrium concept:

Theorem 8 (Maynard-Smith,1982) *The set of evolutionary stable strategies for a particular repeated game is a subset of the set of Nash equilibria for that game.*

Thus an evolutionary stable strategy implies a Nash equilibrium, but not necessarily the other way around, as non evolutionary stable Nash equilibria may exist. This means that ESS can be considered a refinement of the Nash equilibrium concept. In the next section we relate the ESS concept with the dynamics underlying strategy reproduction.

5.1.2 Replicator Dynamics

A second approach in EGT is to look at the dynamics of strategy reproduction in a population. Here the population is considered to consist of agents playing pure strategies. Now we consider a population of $N(t)$ agents at time t, with $N_i(t)$ agents playing pure strategy $\sigma_i, i = 1, \ldots, r$. The proportion of agents playing σ_i at time t is then $x_i(t) = N_i(t)/N(t)$ and the state of the population can be described by the vector $\vec{x}(t) = (x_1(t), \ldots, x_r(t))$. We now look at the changes in this population state as time proceeds.

Denote by $\bar{R}(\vec{x})$ the expected payoff over the entire population when it is in state $\vec{x}$:

$$\bar{R}(\vec{x}) = \sum_i x_i R(\sigma_i, \vec{x})$$

If we assume that the amount of new individuals playing strategy σ_i is proportional to $R(\sigma_i, \vec{x}(t))$, then the expected proportion of individuals playing strategy σ_i in the next generation is given by:

$$\begin{aligned} x_i(t+1) &= \frac{N_i(t) R(\sigma_i, \vec{x}(t))}{\sum_{j=0}^{r} N_j(t) R(\sigma_j, \vec{x}(t))} \\ &= x_i(t) \frac{R(\sigma_i, \vec{x}(t))}{\bar{R}(\vec{x})} \end{aligned} \tag{5.1}$$

Which in turn means the evolution of σ_i in the population can be written as:

$$x_i(t+1) - x_i(t) = x_i(t) \frac{R(\sigma_i, \vec{x}(t)) - \bar{R}(\vec{x})}{\bar{R}(\vec{x})} \tag{5.2}$$

The description above assumes that generations are not overlapping. Each generation lives for one period before being replaced by its offspring. We now consider the case where agents are being continuously replaced,

i.e. we consider a period of time τ in which a fraction τ of the population reproduces. With each selected individual playing σ_i giving birth to $R(\sigma_i, \vec{x}(t))$ offspring, the expected number of agents playing σ_i at time $t+\tau$ is:

$$N_i(t+\tau) = N_i(t) + \tau N_i(t) R(\sigma_i, \vec{x}(t))$$

This results in a population evolution described by:

$$x_i(t+\tau) - x_i(t) = x_i(t) \frac{\tau R(\sigma_i, \vec{x}(t)) - \tau \bar{R}(\vec{x}(t))}{1 + \tau \bar{R}(\vec{x}(t))}$$

By taking the limit $\tau \to 0$ and a proper rescaling of time, we arrive at the continuous time replicator dynamic:

$$\frac{dx_i}{dt} = x_i(R(\sigma_i, \vec{x}(t)) - \bar{R}(\vec{x}(t))) \tag{5.3}$$

When we take into account that the rewards are obtained by playing a normal form game, we can further rewrite these equations. Assume that the game rewards are given by the payoff matrix A. The average payoff $\bar{R}$ for the entire population can then be written as the product $\vec{x}A\vec{x}$. The expected payoff for strategy σ_i is the ith component of the vector $A\vec{x}$, which we denote as $(A\vec{x})_i$. This results in the following equation:

$$\frac{dx_i}{dt} = x_i((A\vec{x})_i - \vec{x}A\vec{x})) \tag{5.4}$$

The discussion above assumes a single population playing the game. One can also consider a multi-population version of this system. In this case opponents in the game are drawn from different populations. This is necessary, for instance, in asymmetric games where the different players of the game have different action sets. For simplicity, we restrict the discussion to two player games. As a result, we now need two systems of differential equations: one for population representing the row agent (X) and one for the column agent (Y). This translates into the following replicator equations for the two populations:

$$\frac{dx_i}{dt} = [(A\vec{y})_i - \vec{x} \cdot A\vec{y}]x_i \tag{5.5}$$

$$\frac{dy_i}{dt} = [(B\vec{x})_i - \vec{y} \cdot B\vec{x}]y_i \tag{5.6}$$

As can be seen in Equations 5.5 and 5.6, the growth rate of the types in each population is additionally determined by the composition of the other population.

The replicator equations can be linked to evolutionary stable strategies by following theorem [HSS79]:

Theorem 9 (Hofbauer et al.,1979) *If a strategy $\vec{\sigma} \in S_r$ is an ESS[1], then the population state $\vec{x} = \vec{\sigma}$ is asymptotically stable under the replicator dynamic.*

This means that any path starting sufficiently close to state $\vec{x}$, will converge to this state under the replicator dynamic. Thus the ESS of the previous section are attractor points for the RD.

5.1.3 Connection with Reinforcement Learning

A connection between evolutionary game theory and reinforcement learning was first demonstrated by Börgers and Sarin [BS97]. In their paper they study the continuous time limit of an RL update called Cross' Learning [Cro73]. It is shown that this limit converges to the continuous time replicator dynamic described above. This result provides a formal link, which allows us to use the RD as tool to study the behaviour of RL algorithms. This method was used for instance in [TVL03] to visualise basins of attraction.

Cross' learning is a simple reinforcement scheme for learning in repeated games. It was first proposed by Bush and Mosteller [BM55]. Cross [Cro73] studied a special case in which reinforcements are limited to be positive. Players using Cross' learning keep a probability vector $\vec{p}$, which is updated after every play in the game. For a 2-player normal form game, with payoff matrix $A = (a_{uv})$, the update equations are given by:

$$p_i(t+1) = a_{ik} + (1 - a_{ik})p_i(t) \quad (5.7)$$

$$p_j(t+1) = (1 - a_{ik})p_j(t) \quad (5.8)$$

where i is the index of the action selected by the agent and a_{ik} is the matrix entry corresponding to the joint action chosen in the game at time t. As is noted in [Tuy04], these updates are a special case of the Linear Reward-Inaction scheme described in Section 2.1.1. This can easily be seen by taking the learning rate $\lambda_1 = 1$ and setting the feedback $b(t)$ to the reward a_{ik} achieved in the game.

[1] Recall that S_r is the $(r-1)$-dimensional unit simplex, defined in Equation 2.12.

To see the relation with the RD we calculate the expected change in the probability vectors $\vec{p}$ and $\vec{q}$ for two agents using Cross' learning in a repeated normal form game with payoff matrices A and B. Define $\Delta p_j = p_j(t+1) - p_j(t)$ and $\Delta q_j = q_j(t+1) - q_j(t)$. Then the expected updates given the current probabilities are:

$$E[\Delta p_j \mid \vec{p}(t), \vec{q}(t)] = p_j(t)[(A\vec{q}(t))_j - \vec{p}(t)A\vec{q}(t)] \quad (5.9)$$
$$E[\Delta q_k \mid \vec{p}(t), \vec{q}(t)] = q_k(t)[(B\vec{p}(t))_k - \vec{q}(t)B\vec{p}(t)] \quad (5.10)$$

Taking the probability vectors $\vec{p}$ and $\vec{q}$ as population states and defining fitness for strategies j and k as $(A\vec{q}(t))_j$ and $(B\vec{p}(t))_k$, respectively, the equations above are the discrete replicator dynamics for 2 populations. Letting the time between updates in the Cross' learning scheme go to 0, we arrive at the continuous RD of Equations 5.5, 5.6. Thus the expected changes in probability updates of Cross' Learning (and L_{R-I}) are given by the RD. In [BS97], Börgers and Sarin go on to proof that the actual movements of the learning process match the expected updates.

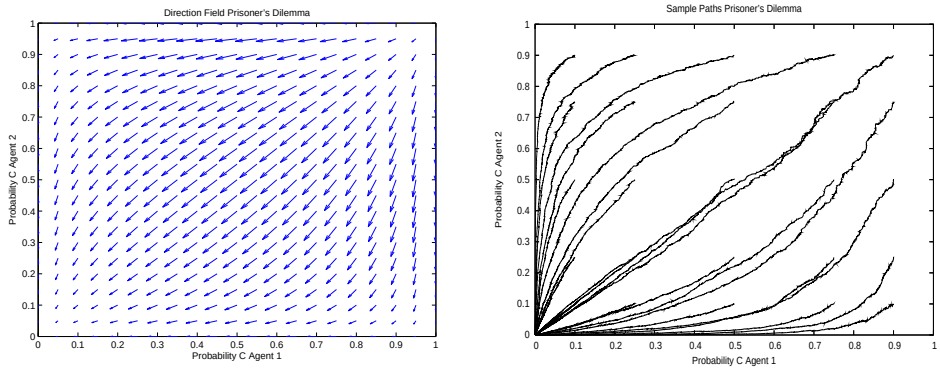
Example 7 *As an example for the application of RD in stateless games, we illustrate the dynamics of the well known Prisoner's Dilemma (PD) game. Recall from Example 1 that the (normalised) reward matrices for this game are:*

$$A = \begin{bmatrix} 0.6 & 0.0 \\ 1.0 & 0.2 \end{bmatrix} \quad B = \begin{bmatrix} 0.6 & 1.0 \\ 0.0 & 0.2 \end{bmatrix}$$

When we put these rewards into Equations 5.5, 5.6, we get following dynamic system:

$$\frac{dp_1}{dt} = p_1[0.6q_1 - (0.6p_1q_1 + p_2q_1 + 0.2p_2q_2)]$$
$$\frac{dq_1}{dt} = q_1[0.6p_1 - (0.6p_1q_1 + p_1q_2 + 0.2p_2q_2)]$$

Where p_1 and q_1 are the probabilities to play action 1 (cooperate) for agent 1 and 2, respectively. Figure 5.1(a) shows the direction field for this dynamic. Figure 5.1(b) plots the action probability trajectories generated by LA playing the repeated game. These trajectories are generated by initialising 2 automata with random action probabilities and repeatedly letting them play the game, using a reward-inaction update. As is clear from the figures, the behaviour of the automata, closely follows the paths predicted by the direction field.



(a) (b)

Figure 5.1: Dynamics for the stateless Prisoner's Dilemma game (a) Direction field for the replicator equations on this game. (b) Sample paths showing the evolution of the action probabilities in a repeated automata game. Both automata use the L_{R-I} update with learning rate 0.001.

5.2 Piecewise Replicator Dynamics

Analyzing the learning dynamics becomes significantly more complex when we move from stateless games to multi-state problems. As the agents have independent action probabilities for each state, the result is a very high dimensional problem. In order to deal with this high dimensionality we present an approach to analyse the dynamics per state.

5.2.1 Average Reward State games

In order to deal with the high dimensionality of multi-state dynamics, we map the dynamics on the states. This allows us to deal with the different states separately. The main idea is to define the limiting games of previous chapters for individual states. For each state of the Markov game, we define an average reward state game. This game gives the expected reward for all joint actions in the state, under the assumption that the agents play a fixed strategy in all other states. When we assume that the action probabilities in the other states remain fixed, we can use Equation 4.3, which we repeat below, to calculate the expected average rewards for each joint policy that results from playing a joint action in this state, while following the fixed policy elsewhere. The game obtained by these rewards can then be studied using the replicator dynamics, exactly as was described in the previous

section.

$$J_k^{\vec{\pi}} = \sum_{i=1}^{N} d^{\vec{\pi}}(s^i) \sum_{j=1}^{N} T(s^i, \vec{\pi}(s^i), s^j) R_k(s^i, \vec{\pi}(s^i), s^j)$$

This method allows us to calculate limiting games, as was introduced in the previous chapters. We now follow the same approach but map the limiting games onto the states of the Markov game. The *average reward state game* for a state s is obtained by calculating the expected average rewards for possible plays in s, while assuming that a fixed policy is used in all other states:

Definition 24 *Let be (n, S, A, R, T) be an ergodic Markov game. Denote by $\vec{\pi}^{-s}$ the joint policy $\vec{\pi}$ without the components corresponding to state s. Then the average reward state game for state s is defined as the normal form game $(n, A_{1,..,n}(s), J)$, in which each agent i receives reward $J_i^{(\vec{a},\vec{\pi}^{-s})}$ for the play $\vec{a}$, i.e. the average expected reward agent i receives for the joint policy which plays $\vec{a}$ in state s, but is equal to $\vec{\pi}$ in all other states.*

Consider for example the Markov game in Table 5.1. In this 2 agent game we have 2 states, both with immediate rewards following the structure of the Prisoner's dilemma game. When the agents both play the same action (i.e joint action (D,D) or (C,C)) the system has a 0.9 probability of staying in the same state and a 0.1 probability of moving to the other state. When the agents play different actions (i.e. joint actions (C,D) or (D,C)) these probabilities are reversed.

As the rewards in each state have the same structure as the PD repeated game of the previous section, one might assume that the agents will converge to the equilibrium point (D,D) in both states. The only pure equilibria in the multi-state example, however, are the points where one agent plays *defect*(D) in state 1 and *cooperate*(C) in state 2, and the other agent does exactly the opposite. This means that instead of mutual defection, the agents converge to a situation, where an agent is exploited in one state, but exploits the other agent in the other state. This is an important difference from the stateless game, where the players converge to mutual defection.

We now demonstrate how average reward state games can be used to analyse the dynamics. Assume both agents play a fixed strategy in state 2. The possible rewards they can then obtain are determined by their play in state 1. We now calculate the expected average reward for each of the 4 plays (C, C), (C, D), (D, C), and (D, D), assuming that play in state 2 remains fixed.

2 State PD		
	State 1	State 2
Rewards	C D C 0.3, 0.3 0.0, 1.0 D 1.0, 0.0 0.2, 0.2	C D C 0.4, 0.4 0, 1 D 1, 0 0.1, 0.1
Transitions	(C,C)→(0.9,0.1) (C,D)→(0.1,0.9) (D,C)→(0.1,0.9) (D,D)→(0.9,0.1)	(C,C)→(0.1,0.9) (C,D)→(0.9,0.1) (D,C)→(0.9,0.1) (D,D)→(0.1,0.9)

Table 5.1: Example Markov games with 2 states and 2 agents with 2 actions in each state. Rewards for joint actions in each state are given in the first row as matrix games. The second row specifies the transition probabilities to both states under each joint action. Rewards in both states have the same structure as the Prisoner's Dilemma game.

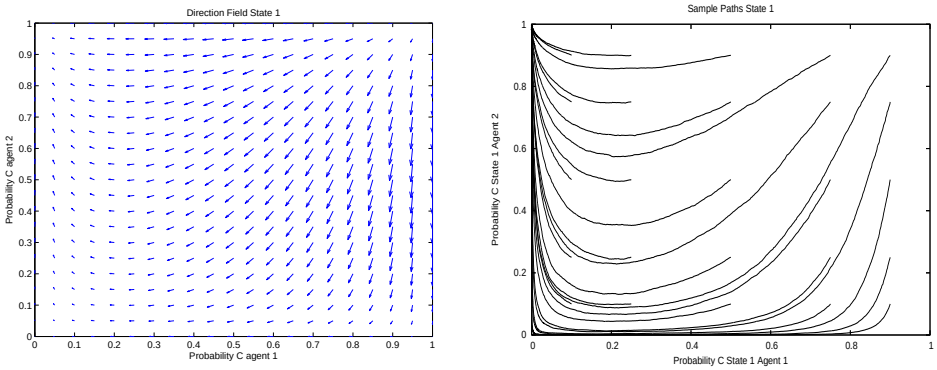
	C	D
C	0.28, 0.35	0.08, 0.78
D	0.48, 0.39	0.19, 0.26

Table 5.2: Average reward game for state 1 of the 2 state PD, when the agents 1 and 2 play action cooperate in state 2 with probabilities 0.7 and 0.2, respectively.

Table 5.2 gives an example average reward game obtained for state 1 of the 2 state PD game, when agent 1 and agent 2 have a fixed probability of 0.7 and 0.2 respectively, to play action 'Cooperate' (C) in state 2. The corresponding direction field is shown in 5.2(a). This figure was generated by applying the RD to the game in Table 5.2. Figure 5.2(b) are sample paths showing the evolution of action probabilities in state 1 when applying the MG-ILA algorithm, but keeping action probabilities in state 2 fixed at the values given above.

5.2.2 Piecewise Modelling

The main problem with analysing the dynamic on an average reward state game, is that it assumes that agents are only learning in a single state, and are keeping the action probabilities in other states fixed. In reality agents update their action probabilities in an interleaved way in all states, and



(a) (b)

Figure 5.2: Sample paths generated by the LA algorithm in state 1 when agents use fixed strategy of Table 5.2 in state 2. (learning rate: 0.0001)

these probabilities will all change in parallel. The dynamics observed for an average reward state game are only a snapshot of the true learning dynamics. As the probabilities in other states change, the state game and corresponding dynamic will also change.

To account for these changes we model the system as piecewise dynamical system. This means that we partition the state space of all action probabilities in all states into a number of discrete cells. Each cell corresponds to a different set of attractors in the average reward state games and has its own associated dynamics. This approach is based on the piecewise linear models described in [GK73, dJGH$^+$04]. Here, however, we don't consider linear models, but rather the non-linear RD.

More precisely, for each state we examine the boundaries where the replicator dynamics of the state game change qualitatively. We do this by looking for points where equilibria disappear or new equilibria appear. It is important to note that we focus on qualitative changes of the dynamic system. Within each region quantitative changes of the dynamic can still occur as the payoffs in the game change, but the same attractor points remain present. When the action probabilities cross a cell boundary, however, they will cause a radical change in the dynamic in the corresponding state. Inside each cell we assume that the probabilities are governed by a fixed replicator dynamic.

Consider for example, the 2 state, 2 action case. If we want to determine the equilibria in the state game for state 1 we need to look at the action probabilities in state 2. Let $(p, 1-p)$ and $(q, 1-q)$ be the strategies used in

	b1	b2
a1	$(J_1^{\vec{\pi}_1}, J_2^{\vec{\pi}_1})$	$(J_1^{\vec{\pi}_3}, J_2^{\vec{\pi}_3})$
a2	$(J_1^{\vec{\pi}_2}, J_2^{\vec{\pi}_2})$	$(J_1^{\vec{\pi}_4}, J_2^{\vec{\pi}_4})$

Table 5.3: Abstract average reward game for a state s with 2 actions and 2 agents.

state 2 by agent 1 and 2 respectively. By this we mean that agent 1 selects its first action with probability p and its second action with probability $1-p$. To determine the structure of the state game we need to look at 4 policies:

$$\begin{aligned}
\vec{\pi}_1 &= [(1,p),(1,q)] \\
\vec{\pi}_2 &= [(0,p),(1,q)] \\
\vec{\pi}_3 &= [(1,p),(0,q)] \\
\vec{\pi}_4 &= [(0,p),(0,q)]
\end{aligned}$$

Where we describe each joint policy by the action probabilities for action 1 in both states, for both agents. That is, $[(1,p),(1,q)]$ denotes the joint policy in which agent 1 plays his first action with probability 1 in state 1 and with probability p in state 2. The second agent plays action 1 with probabilities 1 and q in state 1 and 2, respectively. The abstract average reward state game, for this case is given in Table 5.3. We can now calculate when a play in the state 1 game is an equilibrium as p and q change. Consider for instance the play $(a1,b1)$, i.e. both agents playing their first action. In order for this play to be an equilibrium, the following inequalities need to be satisfied:

$$J_1^{\vec{\pi}_1} \geq J_1^{\vec{\pi}_2} \tag{5.11}$$

$$J_2^{\vec{\pi}_1} \geq J_2^{\vec{\pi}_3} \tag{5.12}$$

In these constraints J is calculated using Equation 4.3 for the joint policies listed above. These average rewards depend on both the stationary state distributions under the corresponding policies, as well as the expected rewards. We can expand the equation to write down the expected game rewards in the terms of p and q:

$$\begin{aligned}
J_1^{\vec{\pi}} = \; & d^{\vec{\pi_1}}(s_1)R_1(s_1,(a1,b1)) + d^{\vec{\pi_1}}(s_2)[pqR_1(s_2,(a1,b1)) \\
& +(1-p)qR_1(s_2,(a2,b1)) + p(1-q)R_1(s_2,(a1,b2)) \\
& +(1-p)(1-q)R_1(s_2,(a2,b2))]
\end{aligned} \tag{5.13}$$

Putting the results above back into the equilibrium constraints 5.11 and 5.12, we can solve for p and q and obtain the action probabilities where equilibria (dis)appear. In the example below we apply this method to the $2-state$ Prisoner's dilemma Markov game.

Example 8 *Consider the example 2-state Prisoner's dilemma in Table 5.1. We will now calculate when mutual defection, i.e. join-action (D, D), is an equilibrium in the average reward game for state 1. For this we need to calculate when the constraints $J_1^{\vec{\pi}_4} \geq J_1^{\vec{\pi}_3}$ and $J_2^{\vec{\pi}_4} \geq J_2^{\vec{\pi}_2}$ hold. Filling in the transition probabilities for joint policy $\vec{\pi}_4 = ((0,p),(0,q))$ we get following transition matrix:*

$$T^{\pi_4} = \begin{bmatrix} 0.9 & 0.1 \\ 4/5p + 4/5q - 8/5pq + 1/10 & 8/5pq + 9/10 - 4/5q - 4/5p \end{bmatrix}$$

In Section 3.1.1 we explained that we can find the stationary distribution for this matrix by solving $\vec{p} = \vec{p}\, T^{\pi_4}$. This gives following formula's for the stationary state probabilities:

$$d^{\vec{\pi}_4}(s^1) = \tfrac{(16pq-1-8q-8p)}{(16pq-2-8q-8p)} \quad , \quad d^{\vec{\pi}_4}(s^2) = \tfrac{-1}{(16pq-2-8q-8p)}$$

Inputting these values in Equation 5.13 we get:

$$\begin{aligned} J_1^{\vec{\pi}_4} &= \tfrac{(16pq-1-8q-8p)}{(16pq-2-8q-8p)} 0.2 + \tfrac{-1}{(16pq-2-8q-8p)} [0.4pq \\ &\quad +(1-p)q + 0.1(1-p)(1-q)] \end{aligned}$$

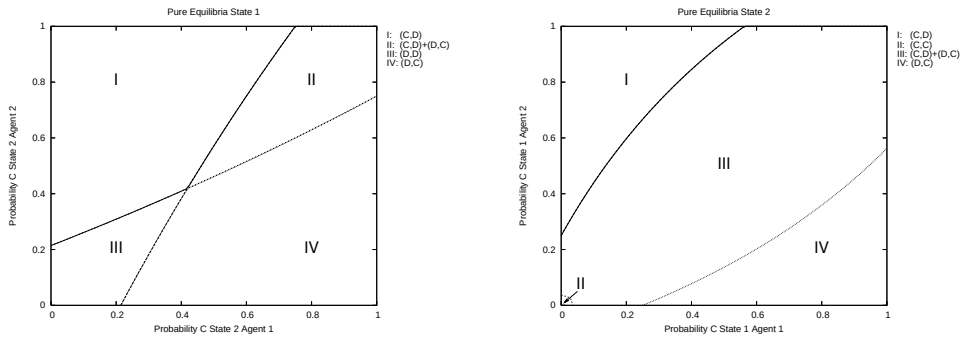
Repeating this procedure for policy $\vec{\pi}_3 = ((0,p),(1,q))$ we get:

$$J_1^{\vec{\pi}_3} = \frac{-9}{(16pq - 10 - 8q - 8p)} [0.4pq + (1-p)q + 0.1(1-p)(1-q)]$$

This means we can solve constraint $J_1^{\vec{\pi}_4} \geq J_1^{\vec{\pi}_3}$ to obtain:

$$p \leq \frac{14q - 3}{2q - 2}$$

This calculation can be repeated for the all other policies, equilibria and states. The end result is shown in Figure 5.3. Here we show for each state the boundaries which cause equilibria to (dis)appear in the other state. Each region in the state space is labelled with the pure equilibria present in the state game for the other state when the state's action probabilities are in that region.



(a) (b)

Figure 5.3: Boundaries which cause a change of equilibria in the other state's average reward game in the 2-state PD.

5.2.3 Sampling Cell Dynamics

In Figure 5.3 we show how the average reward games for both states change as a function of the current action probabilities. Since we have only 2 states, the average reward game in state 1 is completely determined by the strategies in state 2, and vice versa. Figures 5.3 (a) and (b) show the regions corresponding to different equilibria for state 1 and 2, respectively. For both states we get 4 possible regions that correspond to different dynamics. The regions here indicate the equilibria present in the average reward game of a state, when action probabilities *in the other state* are in that region.

To analyse the full dynamics we now need to associate dynamics with the different regions identified in the previous section. We proceed as follows. When we initialise the learning algorithm with action probabilities, these probabilities define an average reward state game and corresponding replicator dynamic for each state. We then assume that the system follows this dynamic, until the action probabilities cross one of the cell boundaries. When this happens the attractors in the corresponding state change and we get new equilibria in the state game with a new replicator dynamic. This dynamic then drives the dynamics in that state until another boundary is crossed. Here we obtain the representative dynamics for each of the different regions by sampling the average reward game at a single point of the interior of regions. Other approaches are possible, for example, one could sample multiple points inside a region and average the games before calculating the dynamic. In the experiments section we will see however, that

Common Interest Game		
	State 1	State 2
Rewards	b1 b2 a1 0.5 0.6 a2 0.6 0.7	b1 b2 a1 0 1 a2 0.5 0
Transitions	(a1,b1)→(0.1,0.9) (a1,b2)→(0.1,0.9) (a2,b1)→(0.1,0.9) (a2,b2)→(0.9,0.1)	(a1,b1)→(0.1,0.9) (a1,b2)→(0.1,0.9) (a2,b1)→(0.1,0.9) (a2,b2)→(0.9,0.1)

Table 5.4: Common Interest Markov game with 2 states and 2 agents with 2 actions in each state.

even with a single point sample, the predicted dynamics give a very good indication of the evolution of the action probabilities.

Figures 5.4 and 5.5 give direction fields for both states and all regions. So once we know the action probabilities in each state, we can determine the regions for both states and identify the corresponding dynamics. For example suppose we initialise LA in both state so that agent 1 plays its first action with probability 0.2 and agent 2 plays the first action with probability 0.8. This means the dynamics start in cell (II,III), i.e. state 1 starts with the dynamics corresponding to region I in Figure 5.4, while state 2 starts in region III of Figure 5.5. This dynamic will then drive the action probabilities until one of the boundaries in encountered, causing a switch of equilibria in the other state. In this way we can follow the trajectory in all states through multiple cells and dynamics until an equilibrium is reached. In the Experiments section we show results for this approach on 2 example Markov games.

5.3 Experiments

We now demonstrate our approach on 2 sample Markov games. We start by demonstrating the dynamics for the 2-state PD shown in Figures 5.4 and 5.5. We will do this by following a single sample path through multiple cells, until it reaches an equilibrium. Agents 1 and 2 are initialised to play their first action with probabilities 0.2 for agent 1 and 0.7 for agent 2 in state 1 and probabilities 0.3, 0.8 in state 2, respectively. Both agents use the MG-ILA algorithm described in Chapter 4, where the automata are updated

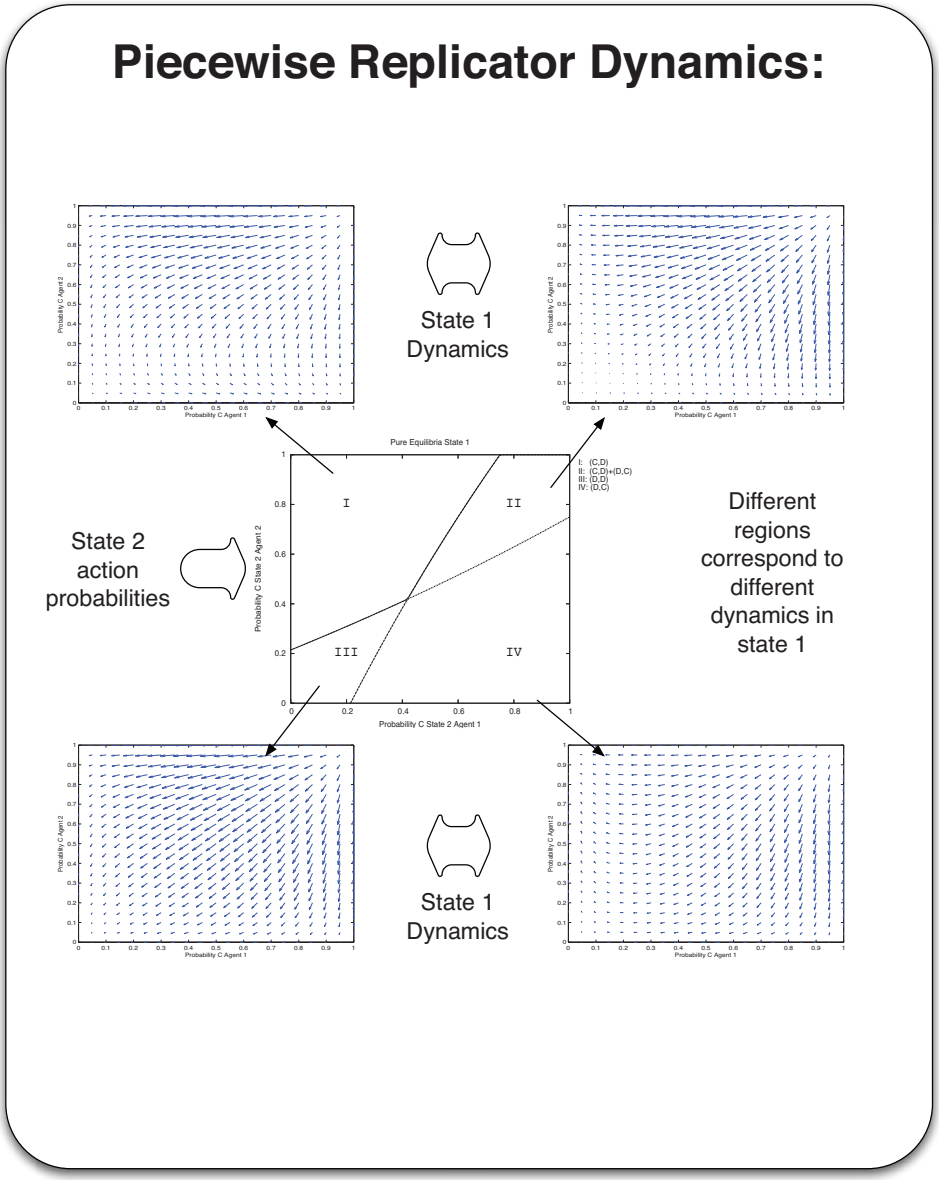


Figure 5.4: Piecewise Replicator dynamics for state 1 of the 2-state Prisoner's dilemma example.

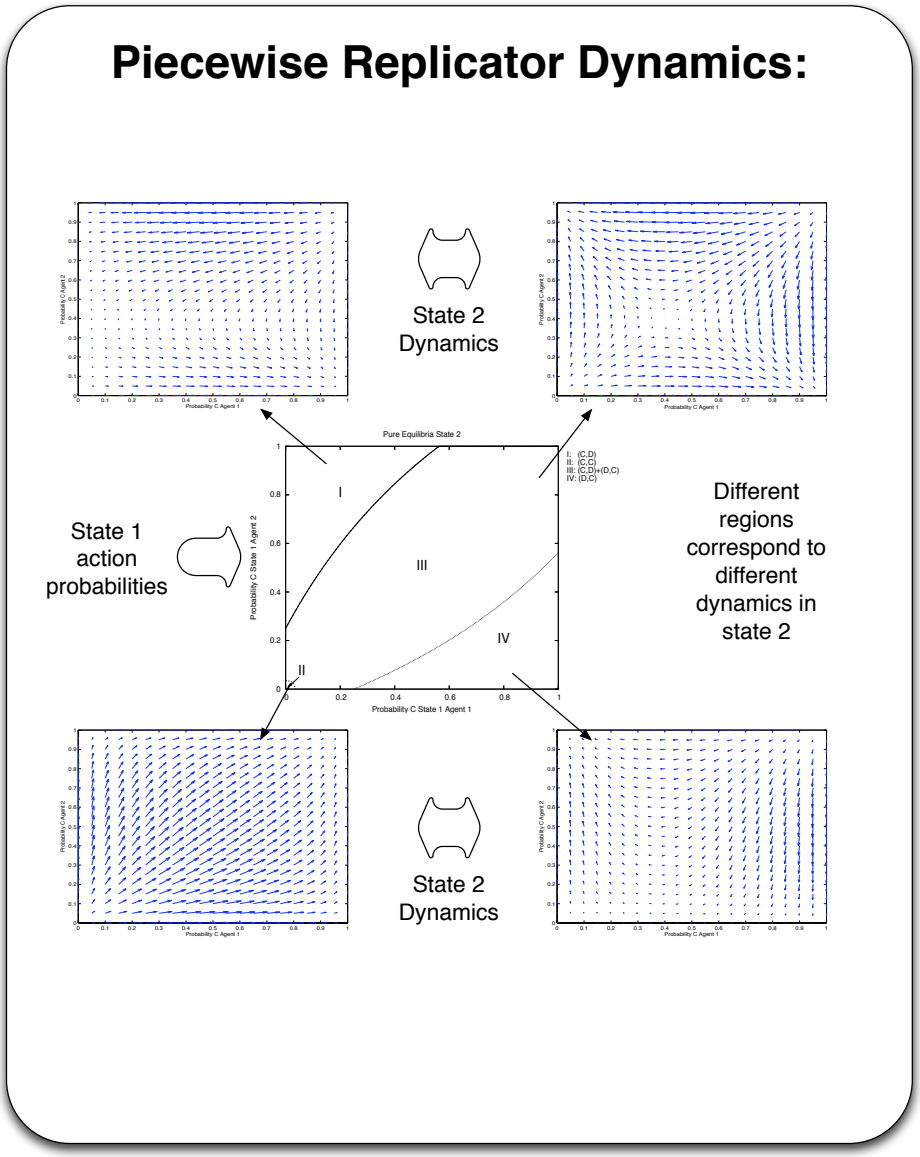
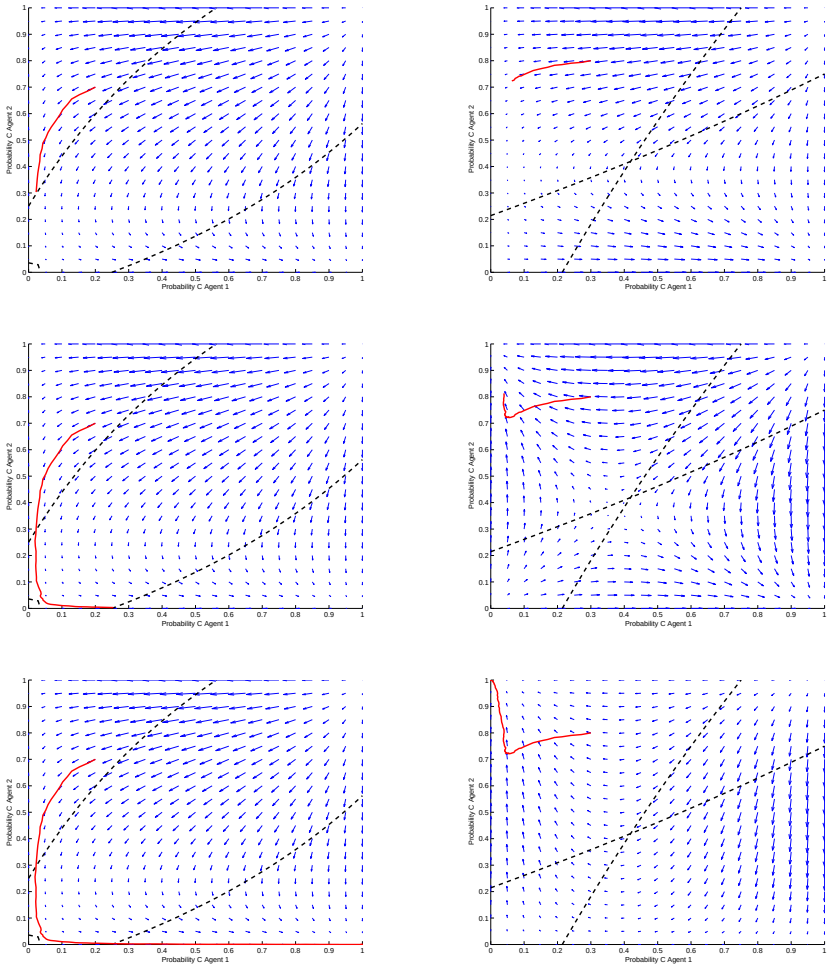
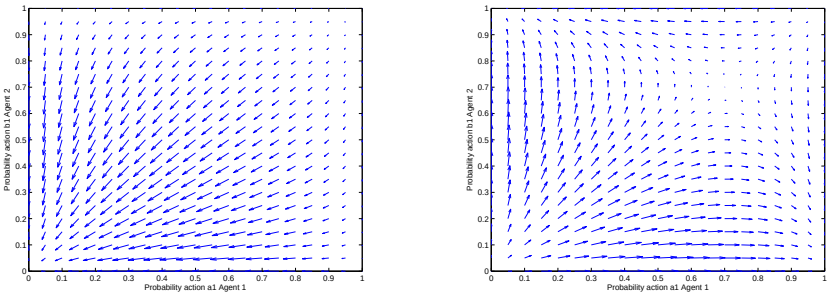


Figure 5.5: Piecewise Replicator dynamics for state 2 of the 2-state Prisoner's dilemma example.



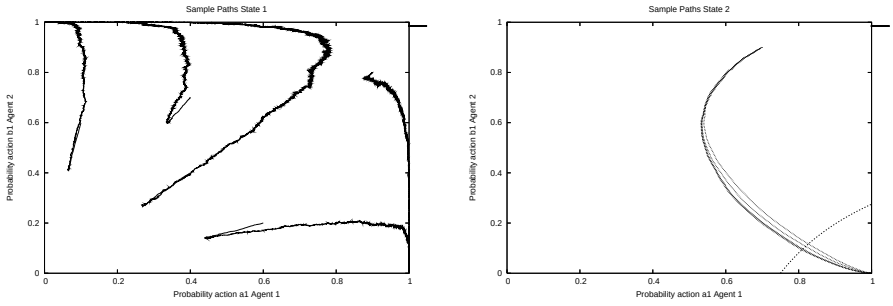
(a) (b)

Figure 5.6: Sample paths for both states of the 2-state PD, generated by automata using the L_{R-I} scheme with learning rate 0.0001. (a) State 1. (b) State 2. Each row shows the evolution of action probabilities (red) in both states until one of the region boundaries (black) is crossed. Blue arrows give the vectorfield for the replicator dynamics.



(a) (b)

Figure 5.7: Two direction fields for state 1 of the common interest Markov game.



(a) (b)

Figure 5.8: Sample paths for both states of the common interest Markov game, generated by automata using the MG-ILA with learning rate 0.0001. (a) State 1. (b) State 2.

using the average reward during the last episode.

Figure 5.6 shows the sample paths generated by the MG-ILA algorithm. The blue arrows show the predicted dynamics, black dotted lines give the region boundaries for each state, and red lines are the sample path generated by LA using the MG-ILA algorithm. The first row gives the predicted dynamics for the initial probabilities, and shows that both states will evolve towards the (C, D) joint action. It can be seen that sample paths in both states closely follow the predicted dynamics until the action probabilities of state 1 reach a region boundary. This event causes a switch in the dynamics of state 2, which now move towards joint action (D, C). In the second row of Figure 5.6 it can be seen that the sample paths in state 2 indeed change directions. The final row shows the convergence to the equilibrium which plays (C, D) in state 1 and (D, C) in state 2. The state 1 probabilities cross another boundary which causes the (C, D) equilibrium in state 2 to disappear completely, but does not affect the sample paths' convergence towards (D, C). The action probabilities in state 2 stay in the same region the whole experiment, so no switch is encountered in state 1.

In a final experiment we show results for another Markov game, shown in Table 5.4. This game is a common interest game, with both agents receiving identical payoffs. In both states, both agents have a choice between 2 actions: a_1 and a_2 for agent 1, b_1 and b_2 for agent 2. Two pure equilibria exist in this game. In the first the agents play joint action (a2,b1) in state 1 and (a1,b2) in state 2, while in the second they play (a1,b2) in state 1 and again (a1,b2) in state 2. For the experiment we completed multiple runs with different initial probabilities in state 1, and using the same starting point in state 2. From this point the dynamics in state 2 always drive the probabilities to the joint action (a1,b2). As the trajectories come closer to this point, they cross the boundary indicated in Figure 5.8(b). When this happens the dynamics in state 1 switch. In Figure 5.8(b) we see that the trajectories in state 1 change direction as this boundary is crossed. From following the dynamic shown in Figure 5.7(a) towards (a2,b2), the trajectories switch to following the dynamic in Figure 5.7(b) which takes the probabilities in very different directions towards (a1,b2) or (a2,b1).

5.4 Discussion and Related Work

As was mentioned in Section 5.1.3, the first use of EGT to study reinforcement learning was proposed in [BS97]. Here a link was shown between the replicator dynamics and Cross' learning. This connection later extended to

Boltzmann Q-learning [TtHV06] and learning automata [Tuy04]. More recently, several extensions have been developed based on these early applications. In [RGK09] the dynamics op ϵ-greedy Q-learning were considered, while [TW09] considered an extension to continuous strategy spaces. EGT has also been used to study the dynamics of learning in application such as poker [PTRD08] and auctions [KTTP08]. Other non-EGT approaches to study dynamics also exist. In [SPT94] the dynamics of LA were studied using ordinary differential equations and the theory of Markov processes. Another non-EGT based approach was used in [BV01a], where the authors study the dynamics of gradient learning methods.

All of the studies mentioned above were performed on repeated games. To the best of our knowledge the approach presented here, and originally published in [VTWN08], is the first attempt to study learning dynamics in multi-state problems. In this chapter we demonstrated our approach on 2-state, 2-agent, 2-action games. The analysis we demonstrated here, goes beyond the traditional EGT approaches. While an application to more complex problems is possible, several issues need to be considered. As is the case with repeated the repeated game analysis, increasing the number of agents or the number of actions will make it difficult to visualise dynamics as was done in this chapter. In such systems alternative visualisations will need to be developed. In repeated games this was done for example in [PTL08] to allow more actions and in [tHT04] to allow more agents.

Since our approach maps dynamics onto individual states, adding additional states does not make the representation used here impossible. However, an increase in the number of states, could lead to an increase of the number of cells. This can make the piecewise approach more difficult, since with a large amount of cells,the dynamics would continually switch between different replicator equations, negating the simplifying assumption of a constant dynamic in each cell. Additionally, considering a large number of state action probabilities will make explicit calculation of region boundaries difficult. In order to both deal with these issues, the authors in [HTR09] propose an extension of the approach described in this chapter. They no longer calculate cells, but rather recalculate the dynamics in each state. This avoids the problem of calculating boundaries or possible artefacts of a sudden switch in dynamics. However, as this approach needs to be applied to an individual initialisation of probabilities it does not allow a global overview of the dynamics as was done in Figures 5.4 and 5.5 for the 2-state PD.

5.5 Conclusions

In this chapter we introduced a new method for analysing the dynamics of multi-agent learning in multi-state problems. By combining piecewise linear dynamic systems with the replicator equations from evolutionary game theory, we obtained a new modeling system which we call Piecewise Replicator Dynamics. In this system the dynamics for each system state are modelled as a set of independent replicator dynamics, between which the system switches based on the current strategies in other states. This method allows us to move from stateless to multi-state games, while retaining the powerful methods EGT offers. More precisely, we are still able to predict the learning system's trajectories and attractors by studying the piecewise replicator dynamics.

Even in rather abstract 2-state problems, our methodology goes beyond the state-of-the-art, since thus far, MAL could only be studied in stateless games using RD. Moreover, even these 2-state problems have shown to be sufficiently complex to emphasise hard challenges for multi-agent reinforcement learning algorithms.

Chapter 6

Beyond Equilibrium

In the previous chapters we described an automata algorithm capable of finding pure Nash equilibria in general Markov games. As was mentioned in Chapter 2, however, the Nash equilibrium solution concept can have several downsides. In common interest games an equilibrium guarantees only local optimality and the global optimum may not be found. In general sum games equilibrium play can result in large payoff differences between agents, or even low payoffs for all agents collectively. Therefore, in this chapter we look at methods to extend the automata algorithm of Chapter 4. We introduce an extended algorithm to learn optimal joint policies in MMDPs and show how this result can be used to achieve fair reward divisions among agents in general sum games.

A large part of the multi-agent learning literature focuses on finding a Nash equilibrium between agents policies [HW98, VVN08a]. However, while a Nash equilibrium represents a local optimum, it does not necessarily represent a desirable solution of the problem at hand. For instance, as we demonstrated in the Chapter 4, in MMDPs where a clear optimum always exist, suboptimal equilibria can still occur. This means that approaches which focus on equilibria, such as MG-ILA may fail to find the optimal solution and offer only local optimality.

In the case of general sum games the notion of an optimal solution is no longer clearly defined. While most learning methods again focus on the Nash equilibrium concept, this may not yield satisfactory outcomes. This is clearly demonstrated by the famous prisoner's dilemma game described in Example 1, for instance. Here the single Nash equilibrium does not represent a desirable outcome, since both agents can simultaneously do better. In some cases this issue can be tackled by considering Pareto optimality.

Pareto optimality assures that it is not possible for all agents to do better simultaneously, however it does not consider the distribution of payoffs among agents. For instance in a constant sum game [1] any outcome where one agent receives the entire reward and others get nothing is Pareto optimal, since the best agent will do worse in all other outcomes. Moreover, in general, multiple Pareto optimal solutions exist (the so called Pareto front) and a selection problem is still present.

Therefore, in this chapter we present a new multi-agent coordination approach for learning patterns of desirable joint agent policies. To do so, we depart from the idea of jointly learning an equilibrium in the full Markov game. Instead, our main idea is to tackle a Markov game by decomposing it into a set of multi-agent common interest problems; each reflecting the preferences of one agent in the system. ILA agents using Parameterised Learning Automata are able to solve this set of MMDPs in parallel. The set of solutions to these MMDPs can then be used as a basis for composing more advanced solution concepts. Here, we use the basic solutions to obtain a fair outcome for the Markov game. More precisely, we consider the problem of agents equalising the reward division, while still giving each agent a chance to realise its preferred outcome. The main idea of our approach is to let agents learn a periodic policy in general sum Markov Games. In a periodic policy the agents play different optima alternatively in periods. The agents alternate between each agent's preferred solution, so that on average the payoffs of all agents are equalised.

Periodic policies were introduced in [NPV01, VNPT07] for learning fair reward divisions in repeated games. The main idea used to let independent agents learn a periodical policy in a repeated game, is to first let agents individually converge to an equilibrium and then exchange information about their payoffs. Better performing agents can then exclude actions, in order to allow other agents to catch up. The motivation for this work came from a simple principle present in a *homo egualis* society from sociology [FS99, Gin00], i.e. agents combat inequity. They are displeased when they receive a lower payoff than the other group members, but they are also willing to share some of their own payoff when they are winning. A detailed overview of this principle and fairness in general can be found in [FS99, dJTV08].

This chapter is organised as follows: in the next section we consider the Exploring Selfish Reinforcement learning algorithm. The Learning Automata algorithm for optimal equilibria in Multi Agent MDPs is given in

[1] i.e. games where the sum of payoffs is a constant over all possible plays.

section 6.2. In Section 6.3 our approach for learning a periodic policy in Markov Games is described. We demonstrate this approach on a simple 2-state Markov game and a larger grid world problem in section 6.4. We end with a discussion in Section 6.5. References for this chapter are: [VNPT07, VVN07d]

6.1 Exploring Selfish Reinforcement Learning

In this section we describe the Exploring Selfish Reinforcement Learning (ESRL) algorithm. This technique was introduced in [Ver04] and can be used in repeated normal form games to find the Pareto optimal Nash equilibrium in common interest games or to learn fair solutions in general sum games. In the following sections we will look at possible extensions of this technique to MMDPs and Markov Games.

The main idea behind ESRL is to divide learning into 2 alternating phases: an exploration phase and a synchronisation phase. During the exploration phase agents learn individually to optimise their own reward. When agents have converged, a synchronisation phase is triggered in order to allow agents to coordinate their exploration of the joint action space and possibly to communicate their performance to the other agents.

In the common interest case agents coordinate their search for the optimal equilibrium. Each agent in the game is represented by a single learning automaton. In the exploration phase agents learn action probabilities using the reward-inaction update, as in any automata game. The convergence results detailed in Theorem 3 of Chapter 2 assure that the agents converge to a (possibly suboptimal) Nash equilibrium. During the synchronisation phase each agent then excludes the action to which it has converged in the exploration phase. The next exploration phase is then started with each agent using the reduced action set. In diagonal games, i.e. games where each agent action is used in at most one pure Nash equilibrium, this system guarantees convergence to the optimal Nash equilibrium. In non-diagonal games random restarts can be used to ensure that the agents explore the joint action space. Provided that the sufficient restarts are used, the agents are again certain to find the Pareto optimal Nash equilibrium.

In general sum games ESRL can be used to play joint strategies that ensure an *optimally fair* reward division among agents. By this we mean that the algorithm attempts to equalise the average reward over time for the participating agents, but prefers the fair solutions that give the highest payoffs for the agents. In this setting, the algorithm is based on the *homo*

egualis assumption. This means that agents are inequity averse and are willing to sacrifice some payoff in order to reduce inequity in the population of agents. In the ESRL approach agents do this by playing *periodical policies*.

Again the algorithm proceeds in an exploration phase and a coordination phase. During the exploration phase each agent learns individually and selfishly tries to optimize its own reward. Provided all agents use L_{R-I} automata to learn their actions, the agents will converge to a pure equilibrium of the game[2]. When all agents have converged, the coordination phase takes place. During this phase agents are allowed limited communication. Each agent exchanges its average reward during the last exploration phase as well as its total reward over all phases. Based on this information the agents can determine the best performing agent in the system. This agent excludes the action it has converged to. This allows convergence to another joint action during the next exploration phase and gives other agents a chance to improve their reward. If after the next coordination phase another agent becomes the best agent, the previously best agent restore its action set. Pseudo code for both phases is given in Tables 4 and 5.

It should be noted that agents using the periodical policies approach can no longer be classified as fully independent agents, since they require communication to coordinate their actions. However, they could be implemented without communication in a joint action learner scenario, which assumes full observability of all agents' reward signals. The periodical policies approach actually has a lower information requirement than most joint action learners, since it does not need information on the opponents' action selections and only periodic updates on their received rewards.

Example 9 *We demonstrate the typical behaviour of the ESRL algorithm in the common interest and conflicting interest case. The algorithm is applied to the 'climbing game' and the 'Bach-Stravinsky game'. The first is an identical payoff game where the optimal Nash equilibrium is surrounded by heavy penalties. The second game is a coordination game where agents have different preferences. Both agents need to coordinate on the same activity (i.e. 'go to a Bach concert' or 'go to a Stravinsky concert'), but the agents have different preferences. Agent 1 would prefer Bach, while Agent 2 prefers Stravinsky. However, if they fail to coordinate, neither agent gets a payoff. Both games are listed in Table 6.1.*

The results of the ESRL algorithm on both games are shown in Figure 6.1. In the common interest game the agents explore the different Nash equilibria in

[2]The case of games only having mixed equilibria is treated in [Ver04]. Here it is shown that the agents will converge to a pure joint strategy, and the algorithm can still be applied.

Algorithm 4 Exploration phase for ESRL agent j in conflicting interest games.

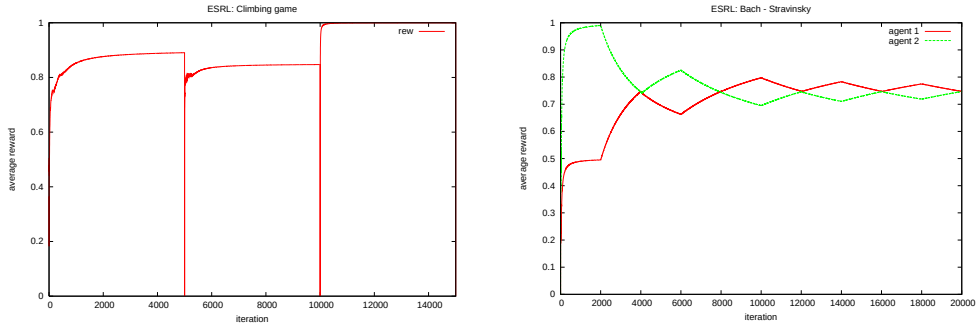
Initialize
time step $t \Leftarrow 0$,
for all actions i that are not excluded :
initialize action values $v_i(t)$ and action probabilities $p_i(t)$
(assume $l = |A_j|$ and $k \leq l$ is the number of not excluded actions)
repeat
$t := t + 1$
choose action a_i from A_j probabilistically using $p(t-1) = (p_1(t-1), \ldots, p_l(t-1))$
take action a_i, observe real immediate reward r in $[0, 1]$
with a given learning rate α, update the action value as follows:

$$\begin{aligned} v_i(t) &\Leftarrow (1-\alpha)v_i(t-1) + \alpha r \\ &\quad \text{for action } a_i \\ v_{i'}(t) &\Leftarrow v_{i'}(t-1) - \alpha v_{i'}(t-1) \\ &\quad \text{for all actions } a_{i'} \text{ in } A_j \text{ with : } a_{i'} \neq a_i \end{aligned}$$

normalize the action values $v_i(t)$ in probabilities $p_i(t)$
set the global average: $average \Leftarrow \frac{t-1}{t} average + \frac{r}{t}$
until $t = N$

Algorithm 5 Synchronization phase for ESRL agent j in conflicting interest games

$T \Leftarrow T + 1$
get action a_i in A_j for which the action probability $p_i((T-1)N) \approx 1$
broadcast action value $v^j = v_i((T-1)N)$ and global $average^j = average$ to all other agents
receive action value v^b and global average $average^b$ for all agents b
if $v^j > v^b$ and $average^j > average^b$ for all b **then**
temporarily exclude action a_i : $A_j \Leftarrow A_j \backslash \{a_i\}$
else
restore the original action set: $A_j \Leftarrow \{a_1, \ldots, a_l\}$ with $l = |A_j|$
end if



(a) (b)

Figure 6.1: Example runs of ESRL for repeated games (a) The Climbing game. (b) Bach-Stravinsky game.

	a1	a2	a3
a1	**1.0**	0.0	0.73
a2	0.0	**0.9**	0.88
a3	0.73	0.73	0.85

(a)

	B	S
B	**(0.5,1.0)**	(0,0)
S	(0,0)	**(1.0,0.5)**

(b)

Table 6.1: (a) The climbing game. (b) The Bach-Stravinsky game. Both games have stochastic rewards, normalised to be between 0 and 1. Nash Equilibria are indicated in bold.

subsequent exploration phases, this allows them to always play optimally when they finish exploring, without performing an exhaustive search of the joint strategy space. In the Bach-Stravinsky game, agents alternate between both pure equilibria. This allows both agents to achieve their preferred outcome some of the time, and assures both of them a payoff which is higher than the smallest Nash payoff.

6.2 Optimal Learning in MMDPs

We now consider the problem of learning the Pareto optimal Nash equilibrium in an MMDP. One method would be to apply the common interest ESRL method of the previous section in the automata view, discussed in the Chapter 4. The advantage of approximating the ILA algorithm's behaviour by an limiting game, is that we can immediately apply algorithms such

as ESRL to the Markov game case. Unfortunately, this approach does not scale well to larger problems. In MMDPs, the large joint action space may require a lot of exploration phases before the optimal point is found. In this case the time needed for the automata team to converge several times may be prohibitively large. Furthermore, as the underlying model of the MMDP is generally unknown, there is no way of knowing how many exploration phases will be needed, or even if the optimum has already been found. Therefore, we propose below an alternative method, which has the advantage that convergence to the optimum can be guaranteed.

6.2.1 Parameterised Learning Automata

As an alternative to the coordinated exploration technique above, we also give an alternative ILA learning update capable of converging to the global maximum. For this we rely on the parameterised learning automata (PLA) update.

Recall from Section 2.1.4 that PLA keep an internal parameter vector $\vec{u}$, of real numbers which is not necessarily a probability vector. After selecting an action a_i and receiving feedback $b(t)$, with learning rate λ, this state vector is updated as follows:

$$u_i(t+1) = u_i(t) + \lambda b(t) \frac{\delta \, ln \, g}{\delta \, u_i}(\vec{u}(t), a(t)) + \lambda h'(u_i(t)) + \sqrt{\lambda} s_i(t) \qquad (6.1)$$

h is a specially shaped function which keeps the vector $\vec{u}$ bounded, and $s(t)$ is a random noise term, inspired by the simulated annealing algorithm, which allows the PLA to escape from local optima.

The parameter vector $\vec{u}$ is used in combination with a function g, that normalises the parameters to action probabilities. For this purpose we use the Boltzmann action selection rule:

$$Pr\{a(t) = a_i \mid \vec{u}\} = g(a_i, \vec{u}) = \frac{e^{u_i}}{\sum_j e^{u_j}} \qquad (6.2)$$

To tackle the MMDP case, we propose to replace the reward-inaction automata in MG-ILA with PLA. To distinguish this adapted algorithm from the general Markov game version we refer to it as MMDP-ILA. Convergence results for this algorithm can be deduced from the results provided in previous chapters. When we combine the Theorem 4, which guarantees that a team of PLA will converge to the global optimum in a common interest game, with Theorem 7 which shows that the Nash equilibria for the

ILA automata game are exactly those of the MMDP, we immediately get following result:

Corollary 2 *The MMDP-ILA model, using the update scheme given in Equation 6.1 converges to an optimal equilibrium in pure strategies in an ergodic MMDP.*

To apply the theorem above in practice, however, some considerations need to be made. Firstly, we must take into account that PLA are guaranteed only to find the best strategy given their state vector $\vec{u}$. There is no guarantee that the global optimal strategy can be expressed by $\vec{u}$ and the function g. In fact, when using the Boltzmann function above, the PLA cannot express pure strategies, since these require values in $\vec{u}$ to go to infinity. This means that the limits $(-L, L)$ for the state vector $\vec{u}$ must be taken large enough to approximate a pure policy sufficiently close.

Additionally, while the algorithm continues updating, random noise $s(t)$ is added to $\vec{u}$ at every time step. This prevents the algorithm from fully converging, as action probabilities tend to keep moving in the neighbourhood of the maximum value. This issue can be addressed by switching to a greedy policy, once we are satisfied that the algorithm has found the maximum equilibrium.

In Section 6.4 we provide some experimental results for MMDP-ILA. In the next section we will show how the technique can still be applied in general sum Markov games in order to learn fair joint policies.

6.3 Periodic policies for Markov games

We now consider the issue of learning fair reward divisions in general sum Markov games. In this section we explain how agents can equalise their average rewards by switching between a set of deterministic stationary policies. Naturally, we do not wish to only equalise the rewards agents obtain, but also optimise these rewards. Therefore we develop a system in which agents take turns to play their preferred joint policies.

The idea is similar to the Exploring Selfish Reinforcement Learning and Periodical Policies concept described in Section 6.1. There however, the agents excluded actions in order to allow other agents to improve their payoff. Unfortunately this approach does not scale well to the large number of automata typically needed for solving Markov games. Several issues arise, such as a large number of exploration phases being needed to equalise rewards which, together with the slower convergence in large

Markov games, makes the algorithm very slow. Additionally, since all automata belonging to the same agent have the same expected rewards, more complex exclusion schemes are needed to prevent multiple automata of the same agent alternating to become the best player. These issues make a direct application of ESRL and periodical policies to Markov games difficult.

Therefore, instead of directly implementing ESRL for Markov games. we propose an adapted version for Markov Games. We proceed by using properties of both the original ESRL algorithm (alternating exploration and coordination) and MMDP-ILA (optimality in common interest problems). In the next section we first discuss some issues , which arise when allowing agents to switch between policies. We then introduce our periodic policies approach for Markov games.

6.3.1 Playing a mixture of policies

Following the ESRL approach, we will attempt to develop a system in which agents alternate between plays in order to assure fair payoffs for all agents. In Markov games this implies that instead of switching between different joint actions, we let agents alternate between different joint policies. For this to work we need a system to allow agents to play a mixture of stationary policies. As was noted in Chapter 4, mixing stationary policies is not the same as playing a mixed, stationary policy. The result of mixing stationary policies can be non-stationary, meaning that it cannot be implemented by simple state-based action probabilities.

The approach we use to implement these policy mixtures is similar to the *policy time-sharing* (PTS) system used in constrained MDP literature [AS93]. A similar idea has also been applied to multi-objective reinforcement learning in [MS02]. In a policy time sharing system the game play is divided into a series of episodes. An episode is defined as the time between two subsequent visits to a selected recurrent system state[3]. During an episode agents play a fixed stationary policy. Different stationary policies are calculated in order to satisfy several constraints on the average reward received in the MDP. When the agent returns to the selected recurrent state, a fixed rule is used to select the policy during the next episode of play. Policies in this system are specified by a vector of stationary policies $(\pi_1, \ldots, \pi_l)$ and a corresponding vector $\vec{\alpha} = (\alpha_1, \ldots, \alpha_l)$ such that $\sum_j \alpha_j = 1$. A PTS policy is then any policy which iterates over policies π_j, using a selection

[3] A recurrent state is a non-transient state. In the ergodic system under study here all states are recurrent.

rule such that for each π_j the limiting proportion of episodes in which it is used tends to α_j.

It should be noted that the expected average reward for a mixture is not simply the sum of their rewards weighted by proportions α_j, but also depends on the expected episode length under each policy. Altman and Schartz [AS93] show that the expected average reward of a PTS policy $(\vec{\alpha}, \vec{\pi})$ is given by:

$$J^{\vec{\alpha},\vec{\pi}} = \sum_{j=1}^{l} w_j J^{\pi_j} \tag{6.3}$$

Where the weight w_j of policy π_j is determined by:

$$w_j = \frac{\alpha_j \tau_j}{\sum_{i=1}^{l} \alpha_i \tau_i}$$

Here τ_j is the expected length of a episode under policy π_j, or equivalently the expected time between 2 subsequent visits to the episode switching state. In an ergodic MDP, with s being the recurrent state which determines episodes, we can calculate this quantity as:

$$\tau_j = \frac{1}{d^{\pi_j}(s)} \tag{6.4}$$

In Example 10 we show how this system can be applied to play (possibly non-stationary) mixed strategies in the agent view of a Markov game.

Example 10 *Consider again the* 2 *robot recycling problem from Chapter 4. The rewards obtained for deterministic stationary policies are listed in the agent game view of Table 4.6. We now demonstrate how we can let agents play mixed strategies in the agent game. We choose state* 1 *as the recurrent state which determines episodes. Let agent* 1 *randomise over deterministic policies* $[search, search]$ *and* $[wait, wait]$ *with probabilities* $1/3$ *and* $2/3$*, respectively. Agent* 2 *plays policies* $[aid, wait]$ *and* $[wait, aid]$ *with equal probability. This means the system switches between following joint policies:*

$$\begin{aligned}
\vec{\pi}_1 &= ([search, search], [aid, wait]) \\
\vec{\pi}_2 &= ([wait, wait], [aid, wait]) \\
\vec{\pi}_3 &= ([search, search), [wait, aid]) \\
\vec{\pi}_4 &= ([wait, wait], [wait, aid])
\end{aligned}$$

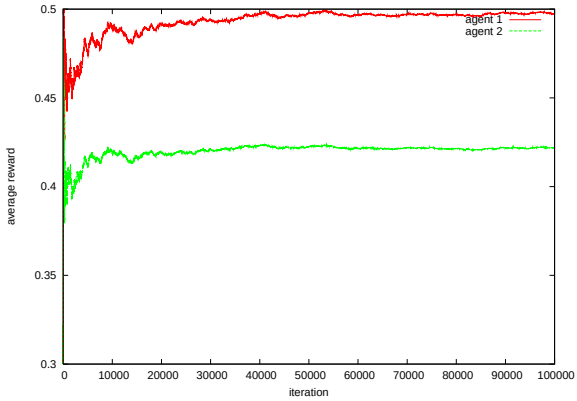


Figure 6.2: Rewards for 2 agents playing mixed agent game policies described in Example 10

Rewards for both agents under these joint policies can be looked up in Table 4.6 and are respectively: $(0.85, 0.72)$, $(0.15, 0.15)$, $(0.49, 0.54)$ *and* $(0.66, 0.46)$. *The proportions* $\vec{\alpha}$ *with which policies are played are* $(1/6, 2/6, 1/6, 2/6)$. *Calculating the stationary distribution we can use Equation 6.4 to obtain* $\vec{\tau} = (1.3, 2.8, 1.1, 6.0)$. *Finally, inputting above data in Equation 6.3 allows us to calculate expected rewards for the agents:* $(0.4925, 0.4185)$. *Figure 6.2 gives the rewards obtained by both agents in an experiment implementing PTS play with the above specifications.*

Below we introduce periodic policies for Markov games. This mechanism uses the policy mixing method described above, but does not limit agents to individually mixed policies. By using a coordination mechanism such as ESRL, agents not only mix their policies, but can also correlate their action choice. This means that agents are not limited to the product distributions, given by individually mixing policies. Instead, agents using the system below to coordinate their policy switches, can play only desired joint policies, rather than the entire cross product of their individual policy sets.

As an example consider the simple Bach-Stravinsky in Example 9. If both agents play mixed strategies in this game, they will always attach some probability to the plays (B, S) and (S, B), which result in 0 payoffs. This is unavoidable as agents action selections are not correlated and using mixed strategies, they will sometimes miscoordinate. When agents correlate their action switches using ESRL coordination, however, they are able

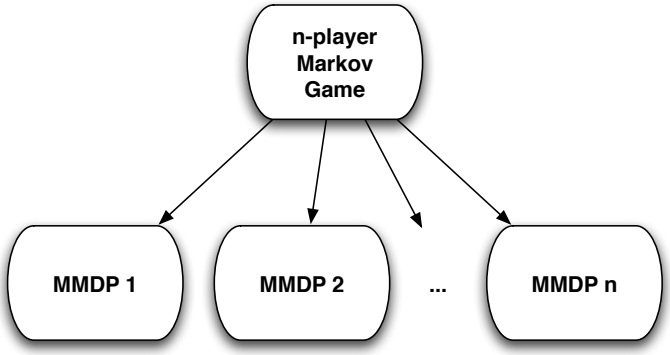


Figure 6.3: Idea behind the egualis learning algorithm.

to play only the Nash equilibrium joint actions and avoid the 0 payoffs. In the next section we implement a similar system in the Markov game setting.

6.3.2 Periodic Play in a Markov Game

The idea behind this algorithm is to split the Markov game into a number of common interest problems, one for each agent. These problems are then solved in parallel, allowing agents to switch between different joint policies, in order to satisfy different agents' preferences. The agents learn preferred outcome for each participant in the game. A fair reward division is then assured by continuously switching between playing each agent's preferred joint policy. This system improves over the ESRL approach in 2 ways. First, rather than randomly exploring the joint action space, we perform a directed search for each agent's desired outcome. Secondly, instead of running long exploration phases before coordinating, play is now divided in relatively short episodes and agents can switch between policies at the start of each episode. The more frequent coordination allows agents to equalise rewards from the start, rather than having to perform multiple exploration phases before equal average payoffs are achieved. The downside to this is that the algorithm requires more frequent communication than the original ESRL. It should be noted, however, that in a joint action setting where all rewards are available, it can still be implemented without the need for communication.

We apply this system as follows. At the start of each episode, agents

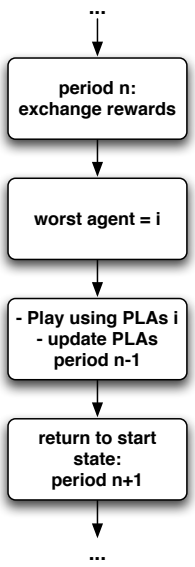


Figure 6.4: Outline of the egualis learning algorithm.

exchange their total reward up to the current time step. The agents then select the agent who is worst off, and choose the policy which maximises that agent's reward. Thus agents do not only learn their own policy, they also learn the preferences of the other agents. Agents do this by associating multiple LA with each state, one for each agent in the system. The first automaton is used to learn their own policy, while the others are used to learn the preferences of the other agents. When the agents agree to aid another agent, they use and update the automata corresponding to that agent in each state until the episode ends. Since the expected average reward under a given policy in an ergodic Markov game is the same for all states, the average rewards exchanged each episode contain sufficient information for each agent to estimate the average payoff received by the other agents during the last episode. This information can then be used to update the automata in visited states, in exactly the same manner as was described in the algorithm of Section 6.2.

This system effectively transforms the Markov game into a set of MMDPs. Each MMDP represents the problem of finding the joint policy that maximises the reward for a single agent in the system. By switching between different automata to learn different agents' preferences, the agents are ac-

	State 1			State 2		
R		b1	b2		b1	b2
	a1	0.2/0.1	0/0	a1	1.0/0.5	0/0
	a2	0/ 0	0.2/0.1	a2	0/0	0.6/0.9
T	(a1,b1)→(0.5,0.5)			(a1,b1)→(0.5,0.5)		
	(a1,b2)→(0.5,0.5)			(a1,b2)→(0.5,0.5)		
	(a2,b1)→(0.5,0.5)			(a2,b1)→(0.5,0.5)		
	(a2,b2)→(0.5,0.5)			(a2,b2)→(0.5,0.5)		

Table 6.2: Markov game with 2 states and 2 agents. Each agent has 2 actions in each state: actions $a1$ and $a2$ for agent 1 and $b1$ and $b2$ for agent 2. Rewards for joint actions in each state are given in the first row as matrix games. The second row specifies the transition probabilities to both states under each joint action.

tually solving each of the MMDPs in parallel using the algorithm described in the previous section.

Provided all LA in the system use the PLA update system the agents will find the optimal joint policy in each MMDP, which corresponds to the joint policy maximising the reward for the corresponding agent. Thus when the automata have converged, the agents continuously alternate between the stationary joint policies that are optimal for the different agents in the system. In the next section we will demonstrate how this system can be naturally applied in an example grid world setting.

6.4 Experiments

In this section we demonstrate the behaviour of our approach on 2 Markov games and show that it does achieve a fair payoff division between agents. As a first problem setting we use that Markov game of Table 6.2. From the limiting game in Table 6.3 we observe that the game has 4 pure equilibrium points. All of these equilibria have asymmetric payoffs with 2 equilibria favouring agent 1 and giving payoffs $(0.6, 0.3)$, and the other equilibria favouring agent 2 with payoffs $(0.4, 0.5)$.

Figure 6.5 gives a typical run of the algorithm, which shows that agents equalise their average reward, while still obtaining a payoff between both equilibrium payoffs. All PLA used a Boltzmann exploration function with update parameters: $\lambda = 0.05$, $\sigma = 0.001$, $L = 3.0$, $K = n = 1.0$. Over 20 runs of 100000 iterations the agents achieved an average payoff 0.42891

		Agent 2			
		[b1,b1]	[b1,b]	[b2,b1]	[b2,b2]
Agent 1	[a1,a1]	**0.6/0.3**	0.1/0.05	0.5/0.25	0/0
	[a1,a2]	0.1/0.05	**0.4/0.5**	0/0	0.3/0.45
	[a2,a1]	0.5/0.25	0/0	**0.6/0.3**	0.1/0.05
	[a2,a2]	0/0	0.3/0.45	0.1/0.05	**0.4/0.5**

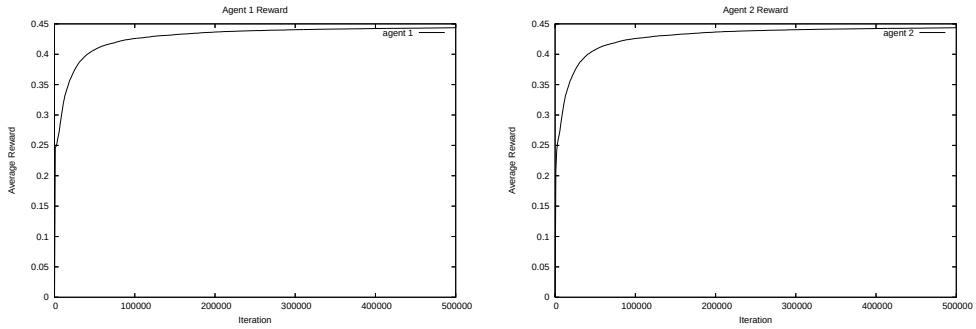
Table 6.3: Approximating limiting game at the agent level for the Markov game in Table 6.2. Equilibria are indicated in bold.

(std. dev: 0.00199), with an average payoff difference at the finish of 0.00001.

In a second experiment we apply the algorithm to a somewhat larger Markov game given by the grid world shown in Figure 6.6(a). This problem is based on the experiments described in [GHS03]. Two agents start from the lower corners from the grid and try to reach the goal location (top row center). When the agents try to enter the same non-goal location they stay in their original place and receive a penalty -1. The agents receive a reward when they both enter the goal location. The reward they receive depends on how they enter the goal location,however. If an agent enters from the bottom he receives a reward of 100. If he enters from the sides he receives a reward of 75. A state in this problem is given by the joint location of both agents, resulting in a total of 81 states for this 3×3 grid. Agents have four actions corresponding to moves in the 4 compass directions. Moves in the grid are stochastic and have a chance of 0.01 of failing[4]. The game continues until both agents arrive in the goal location together, then agents receive their reward and are put back in their starting positions. As described in [GHS03], this problem has pure equilibria corresponding to the joint policies where one agent prefers a path entering the goal from the side and the other one enters from the bottom. These equilibria are asymmetric and result in one agent always receiving the maximum reward, while the other always receives the lower reward.

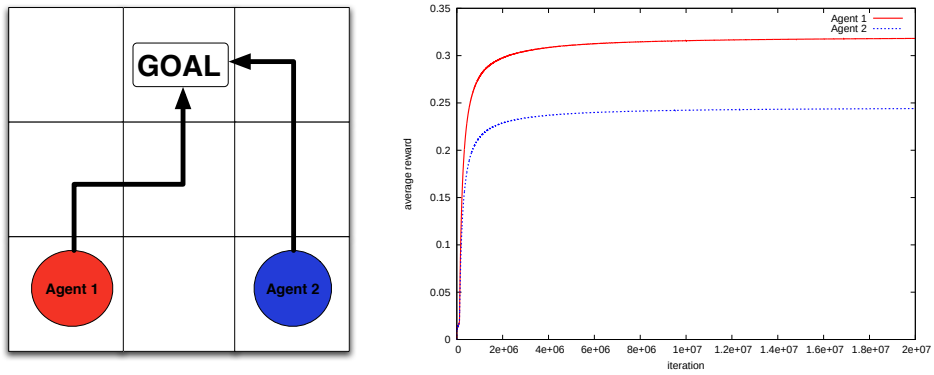
In order to apply the LA algorithms all rewards described above were scaled to lie in $[0, 1]$. The periodic Markov game algorithm was applied as follows. Each time the agents enter the start state (both agents in their starting position), they communicate and exchange their average reward up to that point. Using this information the agents can then calculate the average

[4]when a move fails the agent either stays put or arrives in a random neighboring location.



(a) (b)

Figure 6.5: Typical run of the periodic policy algorithm on the Markov game in Table 6.2.(a) Average reward over time for agent 1. (b)Average reward over time for agent 2.



(a) (b)

Figure 6.6: (a) Deterministic equilibrium solution for the grid world problem. (b) Average reward over time for 2 agents converging to equilibrium.

reward the other agent received during the last episode. The automata are updated using the average reward for that episode (i.e. from start to goal). When the PLA have converged this system results in agents taking turns to use the optimal route. Results for a typical run are shown in Figure 6.7, with the same settings as described above. From this figure it is clear that agents equalize their reward, both receiving an average reward that is between the average rewards for the 2 paths played in an equilibrium. For comparison purposes we also show the rewards obtained by 2 agents using the original Markov game algorithm of Chapter 4 to converge to one of the deterministic equilibria.

6.5 Discussion and Related Work

In the periodic policy system presented here, we assume that all agents are cooperative and are willing to sacrifice some reward in order to help other agents. In systems where agents cannot be trusted or are not willing to cooperate, methods from computational mechanism design could be used to ensure that agents' selfish interests are aligned with the global system utility. Another possible approach is considered in [dJT08], where the other agents can choose to punish uncooperative agents, leading to lower rewards for those agents.

The system presented here relies on agents communicating periodically to share information about their rewards. It should be noted that we assume that agents have limited communication abilities, meaning that this approach can be implemented using fully independent agents, provided that agents can observe the rewards of all agents in the system at the start of each new episode. Settings used by traditional multi-agent algorithms based on the Q-learning algorithm such as NashQ [HW98] or the Correlated Q-learning [GHS03] algorithm do not only assume full knowledge of rewards at each simulation step but also assume knowledge of the action choice as well.

Note also that in the system presented here agents learn to correlate on the joint actions they play. In [GHS03] an approach was presented to learn correlated equilibria. A deeper study on the relation between periodic policies and correlated equilibrium still needs to be done. The main difference we put forward here is that a periodic policy was proposed as a vehicle to reach fair reward divisions among the agents.

In [ZGL06] the concept of cyclic equilibria in Markov Games was proposed. These cyclic equilibria refer to a sequence of policies that reach a

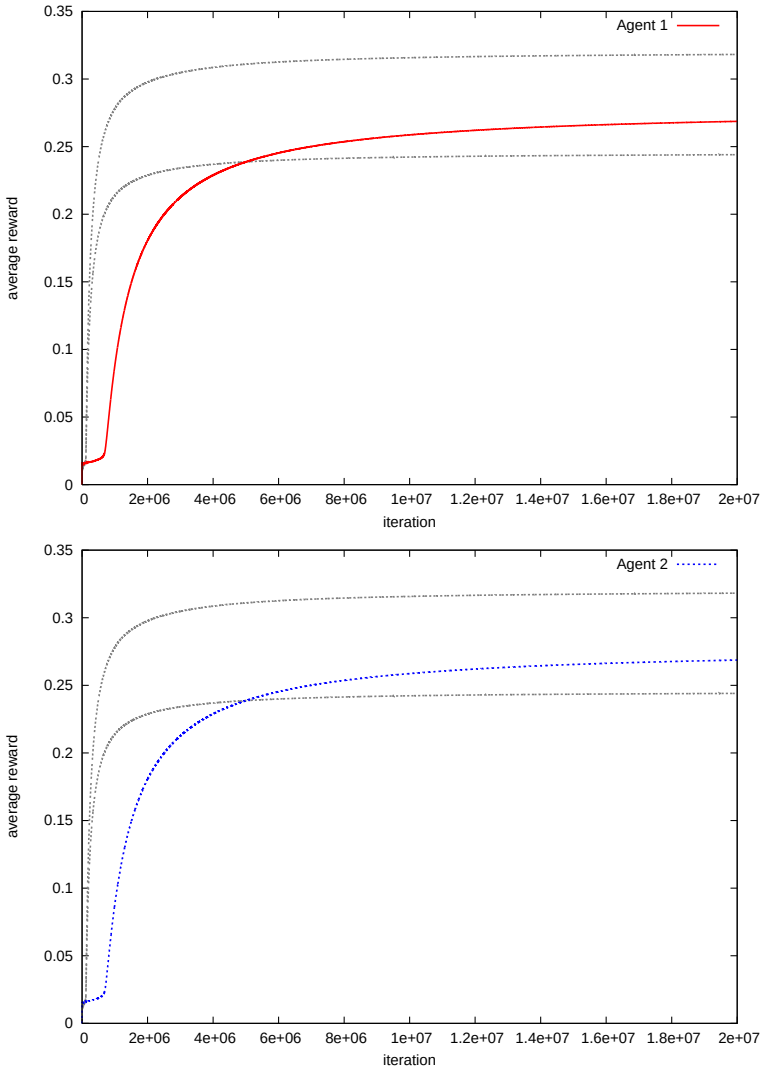


Figure 6.7: Results of the homo egualis learning in the grid world problem. The coloured lines give the average reward over time for both agents. Grey lines give the rewards for agents playing one of the deterministic equilibria.

limit cycle in the game. However again, no link was made with individual agent preferences and how they compare to each other.

Chapter 7

Partial Observability in Markov games

One of the key assumptions made in the preceding chapters, is that agents face a problem that is described by a Markov game, and consequently that the underlying state transitions have the Markov property. When agents can observe the actual system states, this assumption might be unrealistic, as in a multi-agent system Markovian transitions often require the state to contain detailed information on the local situations of all agents in the system. Thus, by assuming agents know the system state, we are implicitly assuming that they have knowledge about the other agents.

In this chapter we still consider Markov games, but we no longer assume that agents have perfect knowledge of the system state. These problems are known as *partially observable* systems or problems with *hidden state*. In a partially observable Markov game the system state contains one or more state variables which cannot be observed by an agent learning in the system. This means that while the underlying system still has the Markov property, the problem may appear non-Markovian to the agents, since they cannot observe all relevant state information. We will focus on the case where the hidden variables are related to the local states of other agents present in the system.

As a testbed for our approach we focus on grid world and mobile robot problems. These problems are a popular benchmark tool for learning systems and are ubiquitous in RL literature. While in a single agent problem, the system state can be described simply by the location of the agent in the world, this is no longer the case in a multi-agent setting. When describing a grid world problem as a Markov game, the result is a state space which

consists of the joint location of all agents in the system and as such, is exponential in the number of agents. This also means that by letting the agents learn in the Markov game state space, we are implicitly assuming that the agents have full knowledge of the position of all other agents in the world, at all times. Not only is this assumption unrealistic for many problems, often it is also unnecessary.

In this chapter we study the case where the agents only have information on the state variables pertaining to their own situation. Contrary to other approaches in partially observable environments (e.g [LCK95]), we do not let agents attempt to estimate the global system state by making inferences based on their observations. Agents in our system learn directly in their observation space. By this we mean that they directly apply their learning algorithm to the non-Markovian problem. We will show that under some simplifying assumptions, agents using the MG-ILA algorithm can still converge to an equilibrium between the policies they can learn.

This chapter will proceed as follows. In the next section we define our learning setting and specify our assumptions. In Section 7.2 we apply our limiting game analysis to this setting and draw a comparison with the fully observable case. We then provide some theoretical results on the application of the MG-ILA algorithm to this setting in Section 7.3, and experimentally demonstrate these results in Section 7.4 Finally, we provide an overview of related approaches and possible extensions. References for this chapter are: [VVN08b, VVN07a, VVN07c]

7.1 Partially Observable Markov Games

We start out by defining our partially observable problem setting. We assume that the problem with full information is still a Markov game, but that agents receive incomplete information on the system state:

Definition 25 *A partially observable Markov game (POMG) is a tuple* $(n, o_{1,\ldots,n}, S, A_{1,\ldots,n}, R_{1,\ldots,n}, T)$ *such that:*

- *the tuple* $(n, S, A_{1,\ldots,n}, R_{1,\ldots,n}, T)$ *is a Markov game as defined in Definition 15*
- *the state space* S *can be factored* $S = \times_i X_i$*, so that the states* $s(t)$ *can be represented by a finite set of values* $\{X_1(t), \ldots, X_M(t)\}$*. Here each* X_i *is a discrete variable with a finite domain* $dom(X_i)$*.*

- $o_k \in \mathcal{P}(\{X_1, \ldots, X_M\})$, *where* $\mathcal{P}(A)$ *denotes the power set of* A*, is the set of state variables that can be observed by agent* k*.*

The definition above describes a *factored* state space representation. The states are described by a set of variables (e.g. agents' x and y coordinates) and each different assignment of values to these variables $\{X_1 = x_1, \ldots, X_M = x_m\}$ where $x_j \in dom(X_j)$, corresponds to a new state s. Additionally, we assume that agents can observe only the values assumed by a subset of all state variables. Note that the set o_k of observable variables is individual to each agent k.

The setting defined here corresponds to the problems detailed in [MV09]. It should be noted that it departs from other partially observable problems literature, for example see [LCK95], where we have a set of observations O and a probability $Pr\{o \mid s\}$ to make observation $o \in O$ when the system is in state $s \in S$. Here we do not consider such stochastic observations, thus for a state $s^i = \{x_1^i, \ldots, x_M^i\}$, agent k will always make the same observation $\cup_{\{j|X_j \in o_k\}}\{x_j^i\}$. This means that agents always make the same observation in the same state, but different states might lead to the same observation, if these states differ only in the values of state variables that cannot be observed. In such cases the agent will not be able to distinguish between these system states.

In order to distinguish between the different notions of states, we shall refer to the full system state of the Markov game as the *joint state s(t).* The individual agent observations will be called *local states*, or in the case of our grid world examples, *locations*. We will focus on the case where we have a state variable $X_k(t)$ corresponding to the local state of each agent k, which can be observed only by that agent. In addition to the ergodicity assumption (Assumption 2 in Chapter 4) also made in the previous chapters, we make following simplifying assumption for our partially observable Markov games:

Assumption 3 $X_k(t+1)$*, the kth component of s(t+1), depends only on* $X_k(t)$ *and* $a_k(t)$*.*

This assumption implies that the transition dynamics of our full Markov game can be decoupled in terms of the local states and actions of the agents. So given joint states $s = (x_1, \ldots, x_M)$, $s' = (x'_1, \ldots, x'_M)$ and a joint action $\vec{a} = (a_1, \ldots, a_n)$ we can write the transition probabilities as:

$$T(s(t+1) = s' \mid s(t) = s, \vec{a}(t) = \vec{a}) = \prod_k Pr\{X_k(t+1) = x'_k \mid a(t) = a_k, X_k(t) = x_k\} \tag{7.1}$$

In our grid world setting it means that an agent's next location is determined only by it's current location and the action it chooses, i.e. it is not possible for another, unseen agent to move agent k around the grid. While we require that transitions are independent of the actions of other agents, this does not mean that we only allow deterministic transitions. Local state transitions are in general still stochastic, but he transition probabilities do not depend on actions of other agents or values of other state variables.

It should also be noted that while we assume that the state transitions can be decoupled in terms of the local agent states, this need not be true for the rewards. We still consider rewards which are influenced by the other agents, i.e. which depend on the transitions of the joint state $s(t)$, not just the local state $X_k(t)$. If the rewards can also be decoupled, the agents do not influence each other in any way, and are effectively solving n MDPs in parallel. In this case we can simply rely on the traditional RL approaches from Chapter 3 to find the optimal solution for each agent.

7.2 Grid problems

We will demonstrate our approach to POMGs, using grid world problems. This simple learning setting, abstracts the problem that a group of mobile robots faces when moving around in a shared environment in order to complete individual objectives. In Chapter 6 we described this setting as a Markov game to test our ILA approach. We will now examine the issues that arise when agents can no longer observe the full Markov game joint state.

First we examine the grid world problems using the Markov game framework, as used in the Chapter 6. In the next section we will show how the partial observability restriction influences our ability to learn in this setting. Below we introduce a small grid world problem which we will use as a first example.

Example 11 *As an example we consider the 2-agent coordination problem depicted in Figure 7.1. The game consists of only two grid locations $l1$ and $l2$. Two agents A and B try to coordinate their behaviour. Each time step both agents can*

state	R1
s^1 ={l1,l1}	(0.01,0.01)
s^2 ={l2,l1}	(0.5,0.5)
s^3 ={l1,l2}	(1.0,1.0)
s^4 ={l2,l2}	(0.01,0.01)

(a)

state	R2
s^1 ={l1,l1}	(0.5,0.5)
s^2 ={l2,l1}	(1.0,0.0)
s^3 ={l1,l2}	(0.0,1.0)
s^4 ={l2,l2}	(0.1,0.1)

(b)

Table 7.1: Reward functions for 2 different Markov Games. Each function gives a reward (r_1, r_2) for agent 1 and 2 respectively. Rewards are based on the joint locations of both agents after moving. (a) Function $R1$ results in a Team Game with identical payoffs for both agents. (b) Function R2 specifies a conflicting interest Markov game.

take one of 2 possible actions. If an agent chooses action 'Stay' (S) it stays in the same location, if it chooses action 'Move'(M) it moves to the other grid location. The transitions in the grid are stochastic, with action M having a probability of 0.9 to change location and a probability of 0.1 to stay in the same location and visa versa for action S. The agents receive a reward that depends on their joint location after moving. Table 7.1 gives 2 reward functions $R1$ and $R2$ for 2 different learning problems in the grid of Figure 7.1. For both problems, column 1 specifies the joint state, while column 2 gives the reward the agents receive for reaching these states, under reward functions $R1$ or $R2$. Reward function $R1$ defines a common interest problem, where agents need to coordinate their location in order to maximise their reward. Reward function $R2$ on the other hand, defines a conflicting interest problem. Here both agents attempt to reach a state in which they are in different locations, but agent 1 prefers configuration $\{l1, l2\}$, i.e. agent 1 is in location $l1$ and agent 2 in location $l2$, while agent 2 gets its highest reward if the situation is reversed.

7.2.1 Full Markov game view

The problem in Example 11 can be represented as a Markov game by considering the product space of the locations and actions. The state variables here consist of the locations of each agent, i.e. with 2 locations $l1$ and $l2$ and 2 agents we get 2 state variables X_1 and X_2, both having the same domain $\{l1, l2\}$. A state in the Markov game then consists consists of the locations of both agents, e.g. $s^1 = \{l1, l1\}$ when both agents are in grid location 1. The result is a 4 state Markov game.

The actions that can be taken to move between the states are the joint

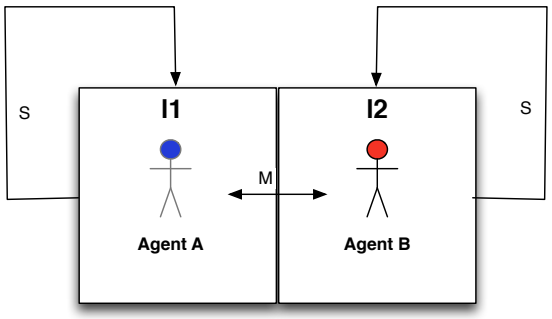


Figure 7.1: Small grid world problem described in Example 11.

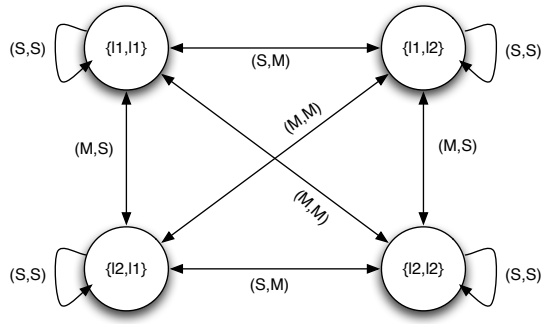


Figure 7.2: Markov game representation of the grid world problem of Example 11.

actions resulting from the individual actions selected by both agents. The transition probabilities for the joint states of the grid world problems are simply products of the local agent state transition probabilities. For instance, in joint state $\{l1, l1\}$ with joint action $\{S, S\}$ chosen, the probability to stay in state $\{l1, l1\}$ is $0.9 \times 0.9 = 0.81$. The probabilities corresponding to move to states $\{l1, l2\}$, $\{l2, l1\}$ and $\{l2, l2\}$ are $0.09, 0.09$ and 0.01 respectively. The transition probabilities for all states and joint action pairs can be calculated this way. With the transition function and the rewards known, we can use Equation 4.3 of Chapter 4 to calculate the expected average rewards. The full Markov game representation of Example 11 can be seen in Figure 7.2.

Using the representation above, we can apply the MG-ILA algorithm to the grid world Markov game and assure convergence to a Nash equilibrium between agent policies. However, this representation is feasible only for small problems as both the number of states and the number of agent policies grow exponentially with the number of agents in the system. Even for the very small example considered here, we have a 4-state Markov game with $2^4 = 16$ policies for each agent. Moreover, the assumption that an agent always exactly knows where all other agents are is often not realistic. Therefore, we will now consider a more feasible learning setting in which each agent only knows its own position in the grid. We will see that despite this limitation, agents can still coordinate their behaviour using the MG-ILA algorithm.

7.2.2 Partial Observability

In this section we no longer assume that agents observe the joint state of the Markov game. Instead we assume that each agent learns using an individual local state space, i.e. its location in the grid. It should be noted that while the agents cannot observe the locations and movements of the other agents, these factors can still have an impact on their rewards.

The main difference with the application of the MG-ILA algorithm is that we do not assume that agents can observe the complete system state. Instead, each agent learns directly in its own observation space, by associating a learning automaton with each distinct local state it can observe. Since an agent does not necessarily observe all state variables, it is possible that it associates the same LA with multiple states, as it cannot distinguish between them. For example, in the 2-location grid world problem of Example 11, an agent associates a LA with both locations it can occupy, while the full system state consists of the joint locations of all agents. As a consequence, this local state representation does not allow the agents to learn all policies that are possible in the full Markov game. Consider for example the automaton associated by agent A with location $l1$. This automaton is used in state $s^1 = \{l1, l1\}$ as well as in state $s^3 = \{l1, l2\}$. Therefore it is not possible for agent A to learn a different action in state s^1 and s^3. This corresponds to the agent associating actions with locations, without modelling the other agents. Contrary to the full Markov game case, agents cannot condition their action on the locations of other agents. Furthermore, since both agents have different observation spaces, they observe different local states. In our example, agent 2 is able differentiate between states $s^1 = \{l1, l1\}$ and $s^3 = \{l1, l2\}$, since they differ in its own position, which

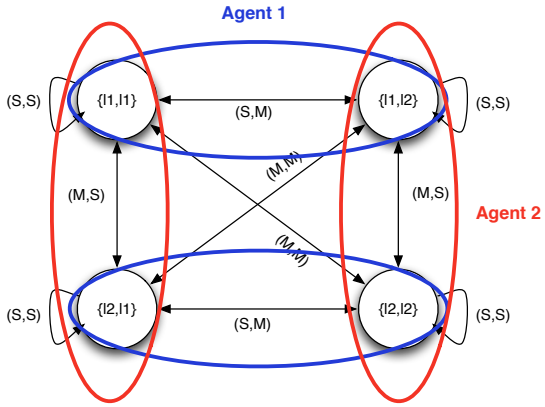


Figure 7.3: Observation spaces for both agents have in the Markov game of Example 11. Coloured circles indicate states which are indistinguishable to the corresponding agent.

is the only state variable it can observe. It is not able however, to tell states $s^1 = \{l1, l1\}$ and $s^2 = \{l2, l1\}$ apart, while these do appear as separate (local) states to agent 1. Figure 7.3 shows the observation spaces for both agents with respect to the Markov game representation of Example 11.

The definition of the update mechanism works in the same way as in the MG-ILA model, the difference is that here agents update their local states instead of the global system state. This will give the following: $LA(k,i)$, active for agent k in location l_i is not informed of the one-step reward $R(s(t), \vec{a}^i, s(t+1))$ resulting from choosing joint action $\vec{a}^i = (a^i_1, \ldots, a^i_n)$ with $a^i_k \in A_k(s^i)$ in $s(t)$ and leading to state $s(t+1)$. Instead, automaton $LA(k,i)$ updates its probabilities when location l_i is visited again. This update can now take place in a state which differs from the original $s(t)$, since location l_i might also be observed by the agent in other joint states. Automaton $LA(k,i)$ receives the same two pieces of data as in the full Markov model: the cumulative reward for agent k up to the current time step and the current global time. From these, automaton $LA(k,i)$ computes the incremental reward generated since this last visit and the corresponding elapsed global time The environment response or the input to $LA(k,i)$ is then taken to be: $\beta_k = \frac{\rho_k(l^i, a^i_k)}{\eta^i(l^i, a^i_k)}$ where $\rho_k(l^i, a^i_k)$ is the cumulative total reward generated for action a^i_k in location l_i and $\eta^i(l^i, a^i_k)$ the cumulative total time elapsed. Contrary to the original MG-ILA algorithm, different

agents may now update automata in different states due to the fact that they have different observations. Additionally, a single probability update may now affect an agent's policy in multiple joint states. In the following, we will show that even when the agents have only knowledge of their own location, in some situations it is still possible to find an equilibrium point of the underlying limiting game. Provided that Assumption 2 still holds, we can still look at the behaviour of MG-ILA as an automata game. The main difference is that we no longer have a play corresponding to each deterministic joint policy in the Markov game. Since the agents now map actions to observations, they will assign the same action to all states in which they make the same observation. As such, the agents can not learn a policy which requires them to play a different action in those states. Since the agents cannot express all joint policies, we cannot be sure that they converge to a Nash equilibrium of the underlying Markov game, as they may not be able to learn the required policies. We will show however, that they still converge to a Nash equilibrium of the limiting agent game, which constitutes an equilibrium between the policies the agents can actually express.

In Table 7.2 we give the limiting games for the agent view of Example 11 for both reward functions $R1$ and $R2$. Reward function $R1$ results in a common interest game with a suboptimal equilibrium giving a payoff of 0.4168 and the optimal equilibrium resulting in a payoff of 0.8168. Reward function $R2$ results in a conflicting interest limiting game with a single equilibrium giving a reward of 0.176 to both players. In this game several plays exist that give both players a higher payoff than the equilibrium play. Note that the underlying limiting game in case of full observability would be different. In that case, an agent policy is described based on joint state-information rather than single agent locations. This means an agents' policy is described by a 4-tuple, i.e. for each joint state of the corresponding Markov game an action is given. This would result in an underlying limiting game of size $2^4 \times 2^4$. The corresponding automata limiting games can be seen in Table 7.3. From these tables it is clear that in the grid world problem of Example 11 the equilibria of both views agree. In the next section we will show that this must always be the case.

7.3 Theoretical Analysis

We will now show that the MG-ILA model applied directly to the observation space of the agents, converges to an equilibrium between agent policies. This theorem is the analogue of Theorem 7 of Chapter 4. It should be

		agent 2			
	policy	[S,S]	[S,M]	[M,S]	[M,M]
agent 1	[S,S]	0.38	0.28	0.48	0.38
	[S,M]	0.48	0.1432	**0.8168**	0.48
	[M,S]	0.28	**0.4168**	0.1432	0248
	[M,M]	0.38	0.28	0.48	0.38

		agent 2			
	policy	[S,S]	[S,M]	[M,S]	[M,M]
agent 1	[S,S]	(0.4,0.4)	(0.68,0.28)	(0.12,0.52)	(0.4,0.4)
	[S,M]	(0.28,0.68)	(0.496,0.496)	(0.064,0.864)	(0.28,0.68)
	[M,S]	(0.52,0.12)	(0.864,0.064)	**(0.176,0.176)**	(0.52,0.12)
	[M,M]	(0.4,0.4)	(0.68,0.28)	(0.12,0.52)	(0.4,0.4)

Table 7.2: Limiting games for the reward functions given in Table 7.1. (Top)Common interest game with both an optimal and a suboptimal equilibrium. (Bottom) Conflicting interest game with a dominated equilibrium. Equilibria are indicated in bold.

noted that although we find an equilibrium in the limiting agent game, we cannot guarantee convergence to a Nash equilibrium of the underlying full Markov game. This is due to the fact that the agents can only express a subset of possible policies, and as such may not be able to learn the equilibrium policies.

Theorem 10 *Consider a Partially Observable Markov Game* $\Gamma = (o_{1,\ldots,n}, n, S, A_{1,\ldots,n}, R_{1,\ldots,n}, T)$*, satisfying the ergodicity requirement of Assumption 2. If Assumption 3 also holds, any deterministic joint policy* $\vec{\pi}$ *that is a Nash equilibrium in the multi-agent view, is also a pure equilibrium policy for the LA-view and vice versa.*

Proof Outline:

As was the case in the full Markov game model, a joint action either in the agent view or in the automata view corresponds to a (here observation based) deterministic joint policy for the agents. The idea is to prove that this LA game has the same equilibrium points as the limiting game the agents are playing. It is easy to see that an equilibrium in the agent view limiting game is also an equilibrium of the LA-game, since in any situation where it

$\vec{\pi}$	$J^{\vec{\pi}}$
(S,S,S,S)	0.38
(S,S,S,M)	0.28
(S,S,M,S)	0.48
(S,S,M,M)	0.38
(S,M,S,S)	0.48
(S,M,S,M)	0.1432
(S,M,M,S)	**0.8168**
(S,M,M,M)	0.48
(M,S,S,S)	0.28
(M,S,S,M)	**0.4168**
(M,S,M,S)	0.1432
(M,S,M,M)	0.28
(M,M,S,S)	0.38
(M,M,S,M)	0.28
(M,M,M,S)	0.48
(M,M,M,M)	0.38

(a)

$\vec{\pi}$	$J_1^{\vec{\pi}}$	$J_2^{\vec{\pi}}$
(S,S,S,S)	0.4	0.4
(S,S,S,M)	0.68	0.28
(S,S,M,S)	0.12	0.52
(S,S,M,M)	0.4	0.4
(S,M,S,S)	0.28	0.68
(S,M,S,M)	0.496	0.496
(S,M,M,S)	0.064	0.864
(S,M,M,M)	0.28	0.68
(M,S,S,S)	0.52	0.12
(M,S,S,M)	0.864	0.064
(M,S,M,S)	**0.176**	**0.176**
(M,S,M,M)	0.52	0.12
(M,M,S,S)	0.4	0.4
(M,M,S,M)	0.68	0.28
(M,M,M,S)	0.12	0.52
(M,M,M,M)	0.4	0.4

(b)

Table 7.3: Limiting automata games for the reward functions given in Table 7.1. (a) Common interest game with both an optimal and a suboptimal equilibrium. (b) Conflicting interest game with a dominated equilibrium. Equilibria are indicated in bold.

is impossible for a single agent to improve its reward, it is also impossible for a single automaton to improve its reward.

Now assume that we can find an equilibrium joint policy $\vec{\pi} = (\pi_1, \ldots, \pi_n)$ in the LA-game that is not an equilibrium point of the agent view. In this case an agent k and a new policy π'_k can be found that produces more reward for agent k than policy π_k and differs from π_k in the actions of at least 2 states or LA, belonging to k, while all other agents keep their policies fixed.

As was the case for the full Markov game, we can now formulate an MDP describing the situation agent k faces. Consider the MDP $(dom(X_k), A_k, R_k, T_k)$. Here X_k is the state variable observable by agent k. This means $dom(X_k)$ is the set of all possible observations the agent can make in the Markov Game, i.e. the states of the MDP are exactly the local states of the agent in the Markov game. The action set A_k is kept the same as in the Markov game. The transition function $T_k(x, a, x')$ gives the probabilities $Pr\{X_k(t+1) = x' \mid X_k(t) = x, a_k(t) = a\}$. Under Assumption 3 this probability depends only on the previous state of the MDP and the action of agent k. Since the policies of all other agents are kept fixed, we can obtain the reward function R_k for the MDP by taking expectations with regard to the other state variables and agents. Provided that all other agents are playing a fixed policy, this gives a stationary reward function for the MDP.

Under Assumptions 2 and 3, the MDP described above corresponds to an ergodic MDP. Since the states of this MDP are exactly the observable local states of the Markov game, the agent assigns a learning automaton to each state of the MDP and the problem faced by agent k reduces to the MDP-ILA case described in Chapter 3. In [WJN86] it is shown that a better policy can be constructed by changing the action in only one state of the MDP. This means that a single automaton of agent k can change its action to receive a higher reward than is obtained using π_k. However, this contradicts the original assumption of $\vec{\pi}$ being an equilibrium point of the LA game. This leads to a contradiction and shows that an equilibrium point from the LA game is also be an equilibrium point in the agent limiting game.

□

Corollary 3 *When the MG-ILA algorithm of Chapter 4 is applied to the local states of a partially observable Markov game satisfying Assumptions 2 and 3, local convergence towards pure equilibria between possible agent policies is established.*

This result is a direct consequence of Theorem 10 above, which shows that equilibria between the LA are equilibria between agent policies and Theorem 3 of Chapter 2 which establishes the local convergence of LA using L_{R-I} to a pure equilibrium point.

Corollary 4 *When the MMDP-ILA algorithm of Chapter 6 is applied to the local states of a partially observable Markov game with identical payoffs to all agents and satisfying Assumptions 2 and 3, convergence towards the optimal pure equilibrium between expressible agent policies is established.*

This is again a direct consequence of Theorem 10 above, this time combined with the convergence properties of PLA noted in Theorem 4 of Chapter 2. It should again be noted that the optimality of this algorithm holds *only over those policies that can be expressed by the agents*, and does not necessarily correspond to global optimality in the full underlying MMDP.

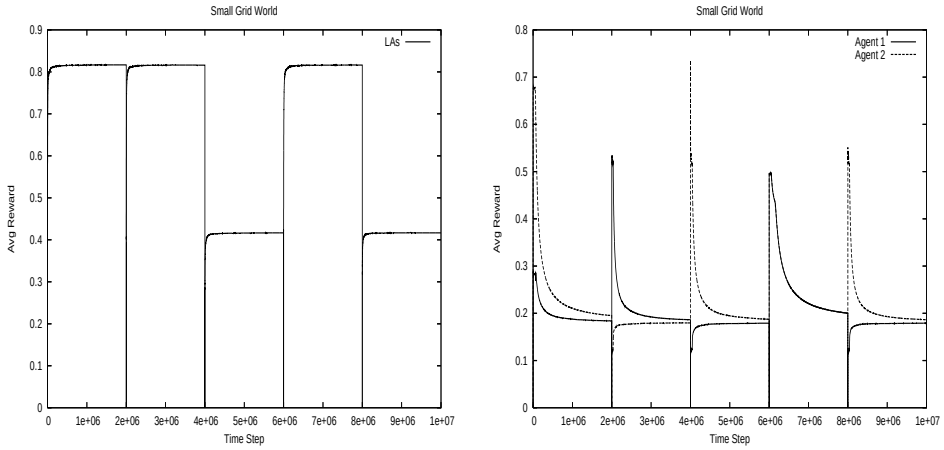
7.4 Experiments

We now experimentally demonstrate the results that can be obtained by applying the MG-ILA and MMDP-ILA algorithms directly in the observation spaces of the agents. We start out by giving results on the small grid games described above. We then move to some benchmark robot navigation problems. Finally, we give results for an agent coordination problem inspired by the mobile ad-hoc networking domain.

7.4.1 Small grid world

Figure 7.4(a) and (b) show the results obtained with the L_{R-I} update scheme in the Markov games using reward function $R1$ and $R2$ respectively. Since we are interested in the value the agents converge to, we again show a single typical run, rather than an average over multiple runs. To show convergence to the different equilibria we restart the agents every 2 million time steps, with action probabilities initialised randomly.

We can observe that in the game with reward function $R1$ agents move to either the optimal or the suboptimal equilibrium of the underlying limiting game given in Table 7.2(Top), depending on their initialization. Using $R2$ the agents always converge to the same, single equilibrium of the limiting game of Table 7.2(Bottom). Even when the agents start out using policies that give a higher payoff, over time they move to the equilibrium.



(a) (b)

Figure 7.4: *Results for the grid world problem of Figure 7.1. (a) Average reward over time for both agents using identical rewards of $R1$ (b) Average reward over time for both agents, using reward function $R2$. Both experiments used $\lambda = 0.05$*

	Eq.	Exp. Rew	Avg (/20)	Total (/12500)	Avg Time
L_{R-I}	([M,S],[S,M])	0.41	1.73(3.39)	1082	1218.78(469.53)
$\alpha_r = 0.1$	([S,M],[M,S])	0.82	15.62(4.62)	9762	
L_{R-I}	([M,S],[S,M])	0.41	0.82(3.30)	515	11858.85(5376.29)
$\alpha_r = 0.01$	([S,M],[M,S])	0.82	19.14(3.34)	11961	
PLA	([M,S],[S,M])	0.41	0.08(0.33)	50	21155.24(8431.28)
	([S,M],[M,S])	0.82	19.81(0.59)	12380	

Table 7.4: Results of L_{R-I} and PLAs on the small grid world problem with reward function $R1$. Table shows the average convergence to each equilibrium, total convergence over all trials and average time steps needed for convergence. Standard deviations are given between parentheses. PLA settings were $b = 0.1, \sigma = 0.2, K = L = n = 1$

Next, we demonstrate the ability of the MMDP-ILA algorithm to learn the optimal joint agent policy. Table 7.4 shows a comparison of PLA with L_{R-I} on the grid world problem with reward function $R1$. For these experiments, each automaton was initialised to play action S with a probability of $0.18, 0.35, 0.5, 0.65,$ or 0.82. This gives a total of 625 initial configurations for the 4 automata in the grid world problem. For each configuration, 20 runs were performed, resulting in a total of 12500 runs for each algorithm. Table 7.4 gives the average number of times the algorithms converged to each of the equilibria, the total equilibrium convergence over all runs and the average amount of time steps needed for all LA to converge. A learning automaton was considered to have converged if it played a single action with a probability of 0.98 or more. Each run was given a maximum of 250000 time steps to allow the automata to converge. It is immediately clear from Table 7.4 that the PLA converge to the optimal equilibrium far more often, but on average take more time to converge.

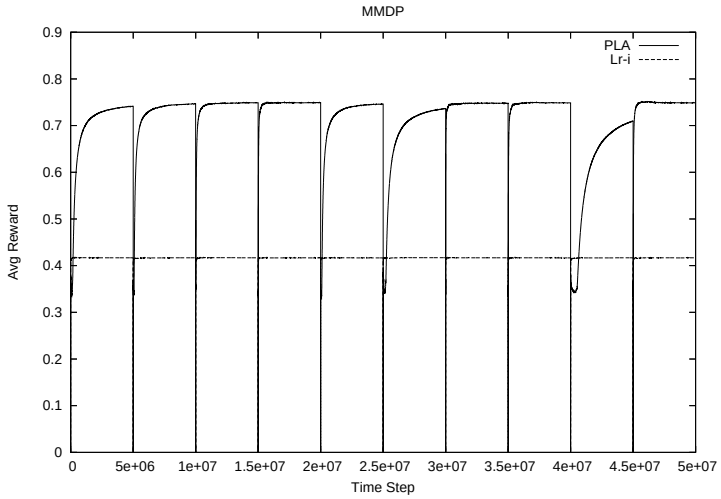
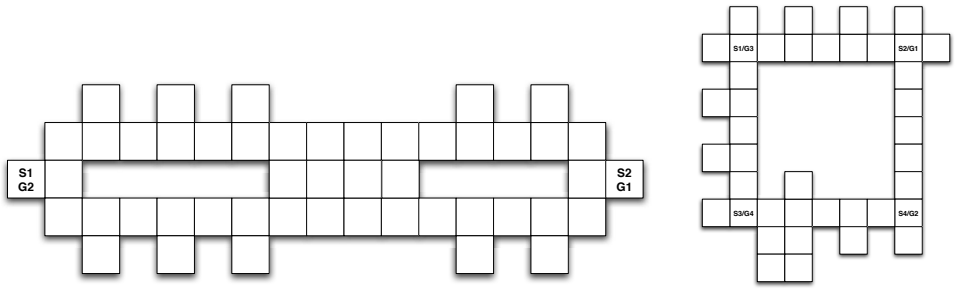


Figure 7.5: *Comparison of PLA with L_{R-I} on the grid world problem of Figure 7.1, using reward function $R1$. The automata were initialized to play their suboptimal equilibrium action with probability 0.82. Settings were $\alpha_r = 0.01$ for L_{R-I} and $b = 0.1, \sigma = 0.2, K = L = n = 1$ for the PLA.*

Figure 7.5 shows results on an initial configuration for which the L_{R-I} automata always converge to the suboptimal equilibrium. The PLA, however, are able to escape the local optimum and converge to the globally



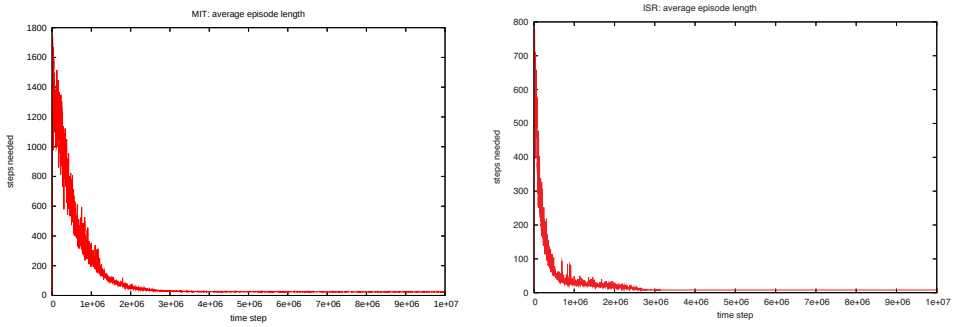
(a) (b)

Figure 7.6: Environments for the robot navigation problems. Si and Gi labels mark starting and goal locations for agent i. (a) The MIT environment (2 agents) (b) The ISR environment (4 agents)

optimal equilibrium point. Due to the random noise added to the update scheme, the PLA do receive a slightly lower pay-off than is predicted in Table 7.2(top).

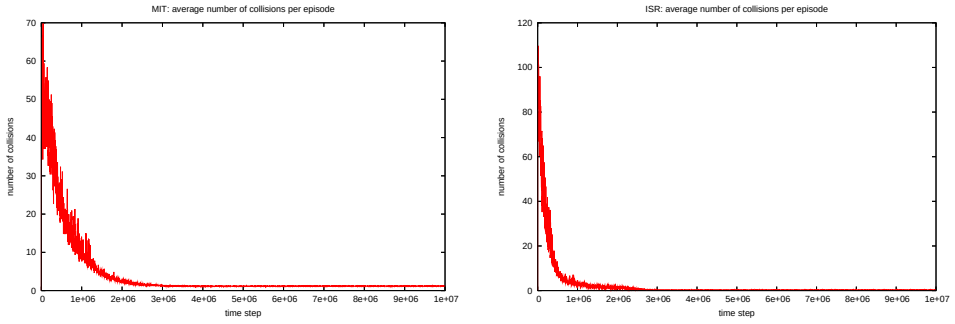
7.4.2 Robot Navigation

We also show some results on 2 robot navigation problems inspired by benchmark problems from partially observable MDP literature. The environments for both problems are given in Figure 7.6(a) and (b). The MIT environment has a total of 49 locations, giving 2401 joint states when used with 2 agents, as we do here. The ISR environment has 43 locations, which results in over 3 million possible joint states for the 4 agents we use. In both problem settings the agents' objective is to get from their starting position to an individual goal location, without colliding with the other agents. The agents only observe their own location and can move in the 4 compass directions N, E, S, W (except when such a move leads outside the grid). Agent rewards are normalised to lie in the interval $[0, 1]$. Agents get a reward of 1 for reaching their goal, a 0 reward when they move into the same location as another agent (i.e. collide) and a reward 0.02 for any move which does not result in a collision. Transitions in the grid are stochastic with agents having a 0.9 probability of moving in the desired direction and a 0.1 probability of going to another, random neighbouring location. When the agent arrives in its goal location, it is automatically transferred back to its starting position.



(a) (b)

Figure 7.7: Average number of steps an agent needs to get from its starting location to the goal. Results averaged over 20 runs.(a) The MIT environment (2 agents) (b) The ISR environment (4 agents)



(a) (b)

Figure 7.8: Average number of collisions per start to goal episode. Results averaged over 20 runs. (a) The MIT environment (2 agents) (b) The ISR environment (4 agents)

The results for both environments are summarised in Figures 7.7 and 7.8. Figure 7.7 shows the average number of time steps an agent needs to get from the starting position to its goal. Figure 7.8 gives the average amount of collisions agents have per episode, where we define an episode as the steps needed to get from the start to the goal. The plots give a moving average over the last 5000 time steps. All results are averaged over 20 runs. These results were generated by applying the MG-ILA algorithm directly to the local state space of the agents. Learning rates were 0.02 for the MIT environment and 0.01 for the ISR environment. Example solutions found by the agents are shown in Figures 7.9 and 7.10. These figures show the preferred paths the agents take from their starting location to their goal, these are. the paths the agents would follow if no moves fail. From these results it is clear that the agents are able to find a path to their goal, while minimising the number of collisions, despite the fact that they are not even aware of the presence of other agents. In the ISR environment agents learn to move around the square environment, taking the shortest paths to their goals. In the MIT environment 2 solutions frequently arise. In the first solution, both agents prefer the shortest possible path to the goal, with one agent choosing the upper hallway and the other one using the lower hallway. In the second solution agents' paths cross in the centre and change from using the upper hallway to the lower one (or the other way around). While in this second solution the agents do not follow the shortest path to the goal, neither agent can improve on this solution by unilaterally changing its policy. When a single agent switches to a shorter path to the goal, half the path overlaps with that of the other agent and the risk of collisions increases. Both agents need to switch policies together to arrive at the better solution in Figure 7.10(a). This agrees with our expectation that the agents find an equilibrium between their policies, but not necessarily an optimal solution.

7.4.3 Ad-Hoc Networking

Finally, we present an agent coordination problem based on experiments in [CHK04a, CHK04b]. In this setting the agents function as mobile network nodes. The agents must attempt to connect a stationary source and sink node. Each agent has a transmission range of 1 grid cell in any direction (north, south, east, west or diagonally). Agents automatically establish a network link when they are in transmission range of each other. Two nodes in the network (i.e. either agents, sink or source) are connected when there exist a path of linked network nodes between them. The agents receive a

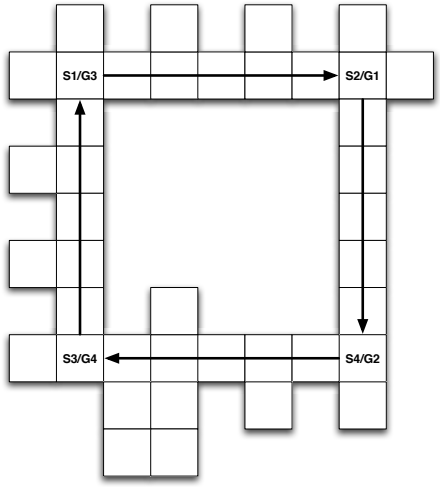
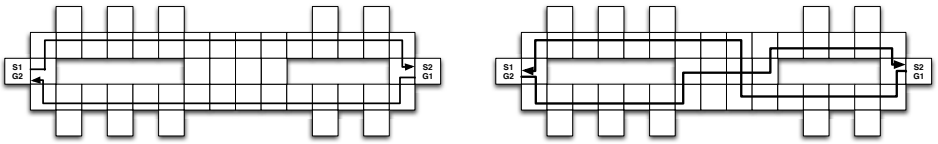
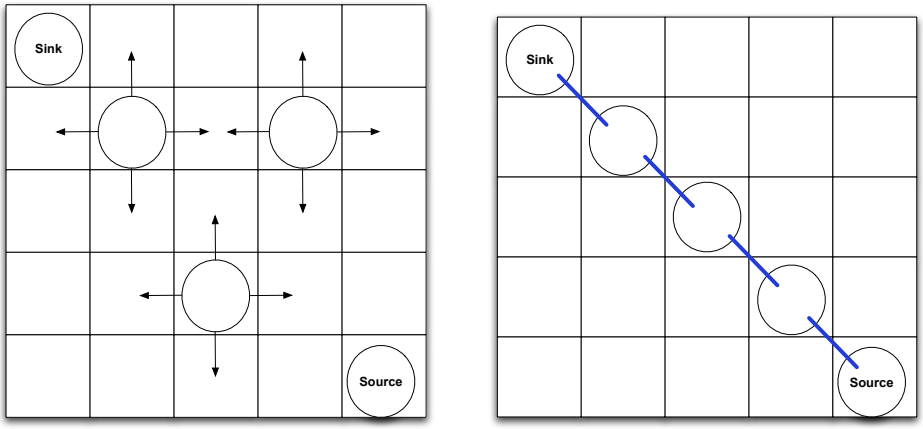


Figure 7.9: Example solution found by the MG-ILA algorithm in the ISR environment with 4 agents.



(a) (b)

Figure 7.10: Examples of typical solutions found by the MG-ILA algorithm applied to the MIT environment with 2 agents. (a) Both agents take the shortest path to their respective goal (b) Agents' paths intersect in the centre of the environment.



(a) (b)

Figure 7.11: Mobile ad-hoc networking domain. (a) Setup: 3 mobile network nodes must connect a stationary source and sink node. (b) Solution network.

common reward of 1 when a connection between source and sink is made and 0 otherwise. In the example presented here, the connection can only be established when the agents stand on the grid's diagonal. The problem setting and solution are visualised in Figure 7.11.

Agents are again limited to observing only their own location. They have no knowledge of other agents' locations or even of other agents that are in transmission range. In each location an agent can move in the 4 compass directions or can choose to stay in the current location. No penalty is given when agents move into the same location. Transitions are stochastic, with each move having a 0.01 chance of ending up in a random neighbouring location. Figure 7.12 gives the average reward over time the agent receives when applying the MG-ILA algorithm with learning rate 0.02. Results are averaged over 20 runs. Despite the stochastic environment and lack of information agents manage to keep the connection established over 90% of the time.

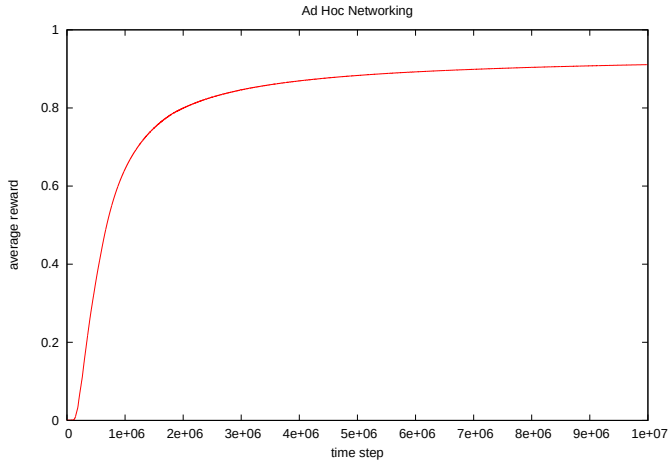


Figure 7.12: Average reward over time in the ad hoc networking domain. Result averaged over 20 runs.

7.5 Related Work

Most approaches concerned with partially observable work within the partially observable MDP (POMDP) or decentralised POMDP frameworks. These approaches often rely on a form of (history based) state estimation in combination with a reinforcement learning or dynamic programming algorithm. A few systems, however attempt to directly apply learning in the observation space of the agents, as is done in this chapter.

In [SJJ94] a (single agent) learning approach is considered which also works directly with observation rather than states. Similarly, in [PM97] the single agent MDP-ILA algorithm described in Chapter 3 is applied directly in a POMDP setting in which rewards and transitions can depend on the process history. It is shown that MDP-ILA is still optimal with regard to expressible policies, provided that the entire history contains sufficient information to make the problem Markovian again. Based on these observations the authors extend their work in [PM98] to use limiting game analysis in order to study the limitations of direct (single agent) reinforcement learning in history dependent problems. In [CHK04a, CHK04b] multi-agent partially observable problems are studied. The agents are not assumed to have a full view of the world. All agents contribute to a collective global reward function, but since domain knowledge is missing, independent agents use filtering methods in order to try to recover the underlying true reward sig-

nal from the noisy one that is observed. This approach is shown to outperform single agent Q-learning in a number of demonstration mobile robot settings.

In this dissertation we only look at 2 extreme possibilities for learning in Markov games. Agents either have access to the full model, or as was the case in this chapter, they have no information on the other agents whatsoever. In reality the best approach will likely be somewhere in between, balancing the growth of the state space with the need for information necessary for the agents to achieve good solutions. Recently, such hybrid approaches have gained attention in the multi-agent community [DVN10]. In [MV09] an extended Q-learning is introduced. This algorithm learns by default in the single agent, local state perspective. The action set of the agents is extended with an additional, *coordinate* action, which allows to agent to expand its state space to the joint state view. The idea is that the agents will learn for themselves in which situations an expanded state space allowing coordination with other agents is beneficial.

7.6 Conclusion

In this chapter we study the behaviour of individual agents learning in a shared environment, when the agents cannot observe the other agents. We show that this situation can again be analysed by considering a limiting game, obtained by considering agent policies as single actions. We demonstrate that when agents are fully ignorant about the other agents in the environment and only know their own local state, a network of learning automata can still find an equilibrium point of the underlying limiting game, provided that the Markov game is ergodic and that the agents do not interfere each others transition probabilities. We have shown that local optimum points of the underlying limiting games are found using MG-ILA algorithm with the L_{R-I} update scheme. The parameterised learning automata used in MMDP-ILA enables us to find global optimum points in case of team Markov Games.

Chapter 8

Modelling Stigmergy

The concept of *stigmergy* was first introduced by entomologist Paul Grassé [Gra59] to describe indirect interactions between termites building a nest. Generally stigmergy is defined as a class of mechanisms that mediate animal to animal interaction through the environment. The idea behind stigmergy is that individuals coordinate their actions by locally modifying the environment rather than by direct interaction. The changed environmental situation caused by one animal, will stimulate others to perform certain actions. This concept has been used to explain the coordinated behaviour of termites, ants, bees and other social insects [TB99].

Recently the notion of stigmergy has gained interest in the domains of multi-agent systems and agent based computing [BDT99, PL04, VK01, VN02, HM99, MM88]. Algorithms such as Ant Colony Optimisation (ACO) [DS04] model aspects of social insect behaviour to coordinate agent behaviour and cooperation. The concept of stigmergy is promising in this context, as it provides a relatively simple framework for agent communication and coordination. One of the main problems that arises, however, is the difficulty of determining the global system behaviour that will arise from local stigmergetic interactions.

In this chapter we analyse the dynamics of stigmergetic interactions using the limiting game techniques, developed in preceding chapters. We directly translate the results on MDPs and Markov games obtained for (multi-)agent learning to convergence proofs for ant like algorithms in classical optimisation problems. The analysis is done from the viewpoint of ant agents and colonies who communicate indirectly via stigmergetic interactions. More specifically we examine the case where a colony of agents coordinates its actions by sharing local pheromone information in the envi-

ronment. We show that the pheromone information in one location can be seen as a strategy of that location in an limiting game played between all locations of the environment. The global system performance is then determined by the payoff achieved in this game. The long term behaviour of the system can be predicted by analysing the dynamics of the pheromone update in the limiting *pheromone game*.

In a second stage we extend the model to a situation where multiple pheromone signals and multiple colonies of agents are present in the same environment. In this case agents not only need to optimise their own reward function, but also need to coordinate with other colonies. The resulting problem can be approached at two different levels: one can still look at the pheromone strategies in the approximating game, but it is also possible to look at interactions on the colony level. We will examine the relations between both levels.

The remainder of this chapter is organised as follows. The next section describes some background material on stigmergy and pheromone based agent interactions. Section 8.2 describes the model we use to describe stigmergetic interactions. Results for the single colony setup and multi colony setup are derived. These theoretical results are experimentally demonstrated in Section 8.4.1 on a simple biologically inspired optimisation problem and on a small routing instance, a successful ACO application. We conclude the chapter with a final discussion. References for this chapter are: [VN02, VVN07b, VVNar]

8.1 Stigmergetic Algorithms

Several different approaches have been proposed to apply stigmergy to multi-agent systems. A commonly used method is to let agents communicate by using artificial pheromones. Agents can observe and alter local pheromone values which guide action selection. This system has been used in optimisation [DBT00] and manufacturing control [VK01, Bru99], among others. An example of this type of algorithm is given in the next subsection.

Other algorithms are based on termite and wasp nest building behaviour [TB95] or ant brood sorting [BHD94, HM99]. In these systems an individual's actions (e.g. building, depositing dirt, picking up brood) modify the local environment and cause reactions of other workers (i.e building more, moving brood, etc.).

In most of the algorithms based on the systems mentioned above a set of common elements can be isolated:

- The agent environment is subdivided in a number of discrete locations, which agents can visit.
- Each location contains local information that can be accessed and updated by agents visiting that location.
- Agents can perform actions in the location they visit. The probability of an action is determined by the agent, based on the local information available at the time of the visit.
- An interconnection scheme between locations is defined, allowing agents to travel between locations.

We will try to accommodate these recurring elements using our learning automata framework. We will see that 2 possible settings emerge. First we can consider the simplest class of pheromone based algorithms. In these algorithms a colony of agents all cooperate to optimise a common objective. Agents share information and coordinate through the use of pheromones. In [VN02] this setting was modelled using the MDP-ILA framework described in Chapter 3. It was demonstrated that we can view this setting as a single agent MDP in which the cooperating agents provide a form of parallel exploration. Below we will demonstrate how this mapping allows us to apply the limiting game analysis to pheromone based algorithms.

In the more complex non-cooperative setting, ant agents can have different and possibly conflicting goals. This can be viewed as ant agents that belong to different colonies. The ant agents cooperate with members of their own colony, but compete against agents belonging to different colonies. We will propose an extension of the original model to incorporate this possibility, and show that this situation now maps to the MG-ILA framework.

8.1.1 Cooperative Stigmergetic Algorithms

We start out by considering the basic cooperative single colony setting. In this section we describe an algorithm called S-ACO (Simple Ant Colony Optimisation). The algorithm was proposed in [DS04] to study the basic properties of ant colony optimisation algorithms. It contains all the basic mechanisms used in algorithms that employ artificial pheromones.

The goal of the S-ACO algorithm is to find the minimum cost path between 2 nodes in a weighted graph. The algorithm uses a colony of very simple ant-like agents. These agents travel through the graph starting

from the source node, until they reach the destination node. In all nodes a pheromone value is associated with each outgoing edge. When an agent arrives in the node it reads these values and uses them to assign a probability to each edge. This probability is used to select an edge in order to travel to the next node. When all agents have reached the goal state they return to the source node and the pheromone value τ_{ij} on each edge ij is updated using the following formula:

$$\tau_{ij} \leftarrow \rho\tau_{ij} + \Delta\tau_{ij} \tag{8.1}$$

The pheromone update described above consists of two parts. First the pheromones on the edges are multiplied by a factor $\rho \in [0, 1]$. This simulates pheromone evaporation and prevents the pheromones from increasing without bound. After evaporation a new amount of pheromone $\Delta\tau$ is added. Typically $\Delta\tau$ contains a small pheromone contribution from each ant that used the corresponding edge. The amount of new pheromones is determined by the paths found by the ant agents. Edges that are used in lower cost paths receive more pheromones than those used in high cost paths. By using this system the ant agents are able to coordinate their behaviour until all agents follow the same shortest path.

8.1.2 Non-cooperative Stigmergetic algorithms

The algorithm described in the previous section is one of the simplest stigmergetic algorithms possible. All agents have the same goal and they alter their environment (by leaving pheromones) to share information and coordinate their actions. The agents do not influence each other's reward, however. Each agent builds it own path, and the feedback it gets is based solely on the cost of this path. The paths followed by other agents do not influence this feedback.

In this chapter we also examine more complex problems where agents can have different goals and directly influence each other's reward. Examples of these algorithms can be found in multi-pheromone algorithms. These systems use not one, but several pheromone gradients to guide agents. One such system was proposed in [NVV04] to let different colonies of agents find disjoint paths in a network. Another multi-pheromone system was proposed in [PL04]. Here the different pheromones guide agents to different locations. We will introduce a model which incorporates multiple colonies, with goals that can conflict. This model was inspired by the AntNet routing algorithm [DCD98]. In this algorithm ant agents in the

system have different goals, as they need to find routes to multiple destinations. Therefore, AntNet uses different pheromones, corresponding to the possible destination nodes. Ants finding routes to the same destination can be seen as a separate colony, sharing pheromone information with each other, but not with ants that have a different destination. However, the colonies in AntNet cannot simply be treated as several single colonies learning in parallel. Since all colonies use the same network links, the performance of a colony is influenced by the strategies of other colonies. Network routes that are heavily used by other colonies will become less attractive, since they can suffer higher delays. This means that the situation incorporates 2 different dynamics: on the one hand we have the indirect communication and cooperation between agents belonging to the same colony, on the other hand we have a competition for resources (in this case the use of the network links) between agents belonging to different colonies. As such, to analyse these types of multi-colony systems we also need to model interactions between colonies. In Section 8.3 we will show how we can extend the original model and map this situation to the Markov game setting considered in earlier chapters.

8.2 Model of Cooperative Stigmergetic Algorithms

In this section we describe the basic model, which was originally introduced by Verbeeck and Nowé [VN02]. The purpose of this model is to capture the essence of the pheromone dynamics in systems similar to Ant Colony Optimisation algorithms. To this end we include only the basic elements of stigmergetic coordination. The model abstracts pheromone communication as a set of location based strategies which are shared among multiple agents. Currently, the model is limited to pure pheromone interactions and additional optimisation tools such as heuristic information and local search improvements are not modeled. The description given here is based on the Markov Decision Process model, described in Chapter 3.

We consider an environment consisting of a set of discrete locations $L = \{l^1, \ldots, l^N\}$. Each location l has a set of possible outgoing links that can be followed, which we alternatively call actions $A(l) = \{a_1^l, \ldots, a_r^l\}$ that can be performed in that location. Further we have a transition function $T(l, a, l')$, which gives the probability to move from location l to l' when taking action $a \in A(l)$ and a reward function $R(l, a, l')$ which gives the reward for this transition. In this environment a set or *colony* of ant agents learns a policy π which maps each location $l \in L$ to an action $a^l \in A(l)$. The goal of the

colony is to learn a policy which maximises the average reward over time:

$$J^{\pi} \equiv lim_{T\to\infty}\frac{1}{T}E\left[\sum_{t=0}^{T-1} R^{\pi}(l(t), l(t+1))\right]. \tag{8.2}$$

The description above essentially describes an average reward MDP (L, A, R, T). The only difference with the standard model is the presence of multiple agents. However, in the stigmergy model these are not independent agents. The actual policy learning is stored in the environment, external to the agents and is shared by all agents.

Ant agents belonging to a colony collaboratively learn a single policy. To do this, they base their action selection in location l on a shared pheromone signal τ^l that associates a value τ_i^l with each action $a_i^l \in A^l$. A pheromone learning system then consists of 2 components: an update rule which governs changes in pheromones based on received reinforcements and an action selection function, which is used to generate action probabilities from the pheromones. For a location l these probabilities are determined from the local pheromone signal τ^l, by applying the normalization function g, so that $Pr\{a(t) = a\} = g(a, \tau)$. As is common in pheromone based systems, we update local pheromones using a reward signal over an entire episode, rather than using the immediate reward $R(l, a, l')$. More precisely, when an ant returns to a previously visited location, it updates the local pheromones using the following estimate β of the global average reward:

$$\beta = \frac{\Delta r}{\Delta t} \tag{8.3}$$

Here Δr is the reward gathered since the last visit, and Δt is the number of time steps since this last visit.

We focus on a system were τ^l is a probability vector and $g(a_i^l, \tau^l) = \tau_i^l$. This means that ant agents directly update the action probabilities and we have for all locations l: $\sum_j \tau_j^l = 1$ and $0 \leq \tau_j^l \leq 1\ \forall j$. To update the values we use the linear reward-inaction (L_{R-I}) scheme, which is repeated below:

$$\tau_i \leftarrow \tau_i + \lambda\beta(1 - \tau_i) \quad \text{if } a_i \text{ is the action taken} \tag{8.4}$$

$$\tau_j \leftarrow \tau_j - \lambda\beta\tau_j \quad \text{if } a_j \neq a_i \tag{8.5}$$

We assume that the reinforcement β has been normalised to lie in $[0, 1]$. The constant $\lambda \in [0, 1]$ is called the *learning rate* and determines the influence of pheromone deposit β. It is similar to the evaporation rate often used in pheromone based systems. The reward-inaction update system conforms to the traditional idea behind pheromone updates that the probability of an action is increased relative to the quality of solutions in which it was used. The update corresponds to the S-ACO update with an evaporation $\rho = (1 - \lambda\beta)$ and a feedback $\Delta\tau = \lambda\beta$. The main difference is that we keep pheromones normalised to probabilities. The similarities between L_{R-I} and the S-ACO update, (which was also used in the first ACO algorithm *Ant System* [DMC96]) are discussed in [VN02, NV99]. One advantage of using this scheme is that convergence results from LA theory, detailed in Chapter 2 can be transferred to pheromone based systems.

The model described here was used in [VN02] to show optimality for the pheromone system with L_{R-I} update, using the convergence results for the MDP-ILA algorithm [WJN86], also described in Chapter 3. In the next subsection we will show how this model can be used more generally, to analyse a pheromone update scheme in terms of their behaviour on a strategic game which approximates the optimisation problem under study. We will then continue by extending the model to allow for multiple colonies of agents, and demonstrating the game theoretic analysis that is possible in this extended case.

8.2.1 Analysis of the Cooperative Model

We now demonstrate how the use of the model above allows us to approximate the behaviour of the pheromone system by a limiting game. The stigmergy model described in the previous section allows us to apply the analysis that was used in previous chapters to pheromone systems.

As before, a critical assumption we need to make here is that the Markov chain of locations generated by a single ant agent is ergodic under all possible policies, i.e. Assumption 1 from Chapter 3. This means the process converges to a stationary distribution over the locations and for each policy π we have a probability distribution d^π over the locations:

$$d^\pi = \{d^\pi(l_1), \ldots, d^\pi(l_n)\} \text{ with } \sum_{l \in L} d^\pi(l) = 1, \text{ and } d^\pi(l) > 0, \forall l$$

where $d^\pi(l)$ represents the probability of the ant agent being in location l. This probability is independent of the time-step and starting location.

Under this ergodicity assumption, it is possible to approximate the single colony model by a game as follows. Consider all locations $l \in L$ of the environment as a players. The action set for each player is exactly the action set A^l of the corresponding location. The pheromone vector τ^l (together with function g) represents the current strategy for this player. Since we have one player for each location, a play in this game maps every location to an action and represents a pure policy π over all locations. The payoff that players receive for this play in the game is the expected average reward J^π for the corresponding policy π. Using the stationary distribution over the locations d^π this reward can be written as follows:

$$J^\pi = \sum_l d^\pi(l) \sum_{l' \in L} T^\pi(l, l') R^\pi(l, l') \tag{8.6}$$

Where T^π and R^π are the expected transition probabilities and rewards under policy π. As each play (or policy) gives the same reward for all players, the game is called an identical payoff game or a team game. Since our model is essentially an MDP, we have the optimal equilibrium guarantee from Theorem 6.

Since the L_{R-I} update scheme was proven to converge to pure Nash equilibria in repeated games the above theorem guarantees that the ants will converge to the optimal policy, i.e. the policy that maximises J. In [VN02] Verbeeck and Nowé showed that these results still hold in the case of multiple cooperative ant agents updating the same action probability vectors. In this setting ant agents are responsible only for sampling the reward and triggering updates. The actual learning and intelligence is stored in the pheromone vector update in each location $l \in L$.

Of course the above model can be used with other pheromone update systems. The limiting games described above, depend only on the problem and not on the pheromone update used. Any combination of a local pheromone update with a normalisation to action probabilities could be treated as a learning strategy in this approximating limiting game. In order to predict the outcome such a system will obtain, we need to study its dynamics on the approximating game. As an example, we examine the pheromone update in Equation 8.7, which is used in Ant Colony System [DS04]:

$$\begin{aligned} \tau_i \quad &\leftarrow \quad (1 - \lambda)\tau_i + \lambda\beta \\ &\qquad \text{if } a_i \text{ is the action taken} \end{aligned} \tag{8.7}$$

We shall examine the dynamics of this system when used with a Boltzmann normalisation:

$$g(a_i^l, \tau^l) = \frac{e^{\tau_i^l/T}}{\sum_{a_j^l \in A^l} e^{\tau_j^l/T}} \tag{8.8}$$

This distribution function assigns each action a probability based on the associated pheromone value and a parameter T, called the temperature. This parameter determines the amount of influence a difference in pheromones has on the action probabilities. Higher values cause actions to become to equiprobable and lower values result in greater differences in probabilities. The repeated game dynamics of the update in Equation 8.7 when used with a Boltzmann normalization are studied in [Tuy04]. In Section 8.4.1 we give an example of the behaviour of this system, as an alternative to the L_{R-I} update.

Example 12 *As an example consider the very simple, biologically inspired optimisation problem depicted in Figure 8.1. A colony of ant agents is trying to optimise the rate at which they collect food. Starting from their nest at location l_1 they can collect food from 2 sources with source 1 having an expected payoff of 1.0 units and source 2 giving an average of 0.55 units of food. From location l_1 they can choose either to proceed to location l_2 (action 1) or to go directly to source 2 (action 2). From location l_2 they can go to either source 1 (action 1) or source 2 (action 2) directly. Transitions are assumed to be stochastic with ants having a 0.9 probability of arriving at the chosen location and a 0.1 probability to make the transition associated with the other action. When an ant agent reaches a food source it receives a payoff depending on the food source and returns to the nest at location l_1. The agents have to decide whether to exploit the closer food source or to concentrate on the richer but also further source.*

The game approximating the problem in Example 12 is shown in Table 8.1. Since we have 2 locations with 2 possible actions each, the game is a 2 player, 2 action game. The row and column actions here are the possible actions in location l_1, l_2, respectively. The game has 1 pure Nash equilibrium corresponding to the policy which plays action 1 in both locations and thus prefers food source 1. This equilibrium indeed corresponds to the optimal policy. Experimental results on this problem are provided in Section 8.4.1.

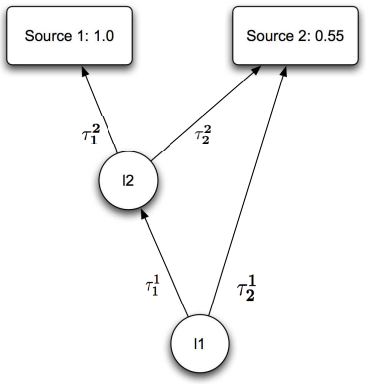


Figure 8.1: Example problem

		l2	
		a1	a2
l1	a1	**0.32**	0.20
	a2	0.28	0.26

Table 8.1: Approximating game for the problem shown in Figure 8.1. Players correspond to problem locations. The game payoffs are the expected averages over time of the amount of food collected under the corresponding policies. The unique equilibrium is shown in bold.

8.3 A Model of Non-cooperative Stigmergetic Algorithms

In this section we extend the previous model by adding ant agents belonging to different colonies. In a multi-colony system, each colony uses its own pheromone signal, where pheromones are now only shared by ants belonging to the same colony. Different colonies correspond to different objectives, represented by different reward functions. Therefore they are suited to solve multi-objective optimisation problems. It should be noted that the model we consider here is mainly based on routing and load-balancing applications such as AntNet, rather than the combinatorial multi-objective problems treated for example in [AW09]. We now give a description of this extended model, which is based on the framework of Markov games, introduced in Chapter 4.

Consider m objectives and thus m colonies present in the problem. The environment still consists of a set of discrete locations $L = \{l_1, \ldots, l_n\}$ with a set of outgoing links or actions $A(l) = \{a_1^l, \ldots a_r^l\}$ that can be followed[1]. However, in each location l, m different pheromone vectors $\tau^{l,c},\ c : 1 \ldots m$ are now present, representing the action probabilities for an ant agent of colony c in location l. Transitions to new locations can still be defined by $T(l, a, l')$ as being the probability of one ant going to location l' by taking action a in location l independent of colony type c. As before, the pheromone probability vectors $\tau^{l,c}$ are updated by ant agents walking around in the environment using the reward gathered over the entire episode as specified by Equation 8.3. However, each colony c now has a different reward function, the value of which can be influenced by the behavior of another colony. The expected average reward for a single colony now depends on the joint policies $\vec{\pi} = (\pi_1, \ldots, \pi_m)$ of all colonies. Denote $\vec{\pi}_{-c}$ for the joint policy $\vec{\pi}$ for all colonies minus the policy of colony c, then $R^c(l, a, \vec{\pi}_{-c}, l'), c : 1 \ldots m$ specifies the reward for an ant agent of colony c, taking action a in location l, moving to location l' with $\vec{\pi}_{-c}$ representing the interference of ant agents not in colony c. The goal of each colony c is to learn a policy π_c which maximises the average reward over time:

$$J^{\vec{\pi},c} \equiv lim_{T\to\infty} \frac{1}{T} E\left[\sum_{t=0}^{T-1} R^{\pi,c}(l(t), l(t+1), \vec{\pi}_{-c})\right] \tag{8.9}$$

Since the problem is multi-objective, optimising for one colony might

[1] For ease of notation we assume that each colony has the same action set in every location.

lead to lower rewards for other colonies. In the next subsection we explain that this setting can now be analysed in terms of the colonies' behaviour on two different strategic games.

8.3.1 Analysis of the Non-cooperative Model

Again we make the ergodicity assumption. Consider one ant-agent from colony c moving in the environment with other ant agents from all other colonies excluding colony c. We assume that the Markov chain generated by this ant agent is ergodic, meaning that again a stationary distribution over the locations exist and the average reward over time for colony c can be calculated as follows:

$$J^{\vec{\pi},c} = \sum_{l} d^{\pi,c}(l) \sum_{l' \in L} T^{\pi}(l,l') R^{\pi,c}(l,l',\vec{\pi}_{-c}) \tag{8.10}$$

with $d^{\pi,c}(l)$ the stationary distribution of one ant agent of colony c over the locations under policy π. The resulting problem can now be viewed at two levels. Instead of making the locations the players of the game, we can look at the game between the different pheromone strategies, i.e. the pheromone vectors. Since m vectors are present in each location, there are m players for each location. A play in this game results in an action for each colony in every location and thus represents a pure joint policy. The game will no longer be a identical payoff game since players belonging to different colonies follow different reward functions. Players within one colony c do receive the same payoff $J^{\vec{\pi},c}$ (given by Equation 8.10) for a play $\vec{\pi}$. In total we will have a game consisting of $n \times m$ players, with n the number of locations and m the different number of colonies, each having r actions. We refer to this game as the *pheromone* game, since each player represents a pheromone strategy in a single location. This game can be seen to correspond to the automata view we have considered in Markov games.

At another level we look at the expected rewards resulting from the interaction between colonies. We do this by considering all the possible pure policies for the colonies as actions in a single large game. In this game the players are the colonies and the payoff is the expected reward a colony obtains for a certain combination of policies from all colonies. In total we have now a game of m players each having $r \times n$ actions. We refer to this game as the *colony* game. This view corresponds to the agent view of earlier chapters.

From the description above it is clear that each play in both the pheromone and the colony game maps an action to each location for every colony. Thus a play in either game represents a pure joint policy over all locations. Furthermore for every play in the colony game we have a corresponding play in the pheromone game and vice versa. We can now show the following relation between both games:

Theorem 11 *Let Γ_1 denote the colony game where each player (or colony) c has action set $A^1 \times \ldots \times A^n$ and receives payoff $J^{\vec{\pi},c}$ for play $\vec{\pi} \in (A^1 \times \ldots \times A^n)^m$. Let Γ_2 denote the pheromone game, where every player (or pheromone vector) in location l^i has action set $A(l^i)$ and receives payoff $J^{\vec{\pi},c}$ for a play $\vec{\pi} \in \prod_{i:1\ldots n}(A(l^i))^m$. Then a joint policy $\vec{\pi}$ is a pure Nash equilibrium of Γ_1 if and only if the corresponding joint policy $\vec{\pi}$ is a pure Nash equilibrium of Γ_2.*

Proof:

if: In a Nash equilibrium no player can improve his payoff by unilateral deviation of the current strategy profile. So in a Nash equilibrium of the colony game, no player has a policy which gives a higher payoff provided that the other players stick to their current policy. A player (or pheromone vector) of colony c switching its action in the pheromone game corresponds to a single player (here colony c) in the colony game switching to a policy which differs in just 1 location. As the players representing pheromone vectors belonging to colony c in the pheromone game receive the same payoff as the corresponding colony c in the colony game, the pheromone player cannot improve its payoff.

only if: Suppose a play $\vec{\pi}$ exists which is a Nash equilibrium in the pheromone game, but the corresponding play is not a Nash equilibrium in the colony game. This means that in the colony game we can find at least one player or colony c which can switch from its current policy π_c to a better policy π'_c while other players keep their policy constant. But the situation where other colonies keep their policies fixed corresponds to a 1 colony problem without any external influences for the reward function. So for colony c where the other colonies play their fixed strategy $\vec{\pi}_{-c}$, the reward function is given by taking the expectation of reward R with respect to the fixed $\vec{\pi}_{-c}$ and the resulting expected average payoff is given by $J^{\pi_c} = J^{(\vec{\pi}_{-c},\pi_c),c}$. According to Theorem 6 this situation can be represented by a game Γ with a

unique, optimal equilibrium. Since $J^{(\vec{\pi}_{-c},\pi_c),c} < J^{(\vec{\pi}_{-c},\pi'_c),c}$, π_c cannot be this equilibrium and a policy must exist which achieves a higher payoff but differs in only 1 location[2]. But since this policy would also receive a higher payoff then π_c in the pheromone game, $\vec{\pi} = (\vec{\pi}_{-c}, \pi_c)$ cannot be a Nash equilibrium of this game which leads to a contradiction.

The relation between both games allows us to predict colony behavior based on the convergence properties of the local pheromone update in the pheromone game. For instance, from the theorem above and the pure equilibrium convergence of L_{R-I} we immediately get following result:

Corollary 5 *Consider an optimisation problem with location set L and C colonies optimizing individual reward functions. If all colonies use the L_{R-I} update with a sufficiently small learning rate, the system will converge to a pure Nash equilibrium between the policies of the colonies.*

Example 13 *We demonstrate these results on an extended version of the example in Figure 8.1. Instead of a single colony optimizing its food collection, we now consider 2 colonies gathering food in the same environment. Each colony needs to optimise its own foraging, but also has to coordinate with the other colony. When both colonies select the same food source, they have to share and thus will receive a lower payoff. So instead of a fixed average payoff for each source, an ant reaching a food source now receives a payoff which is also based on the number of ants from other colonies at that source. More specifically when arriving at a food source we give the ant a reward of*
$(1.0 - \frac{q_s}{q_{tot}}) * p_s$ where q_s is the number of ants from other colonies at the food source, q_{tot} is the total number of ants from other colonies in the system and p_s is the average payoff for the source (i.e. 1.0 and 0.55 for source 1 and 2, respectively).

In Table 8.2 we show the colony game for the 2 colony problem described in Example 13. Since we have 2 colonies this game has 2 players. The actions for these players are the possible pure policies for the corresponding colony. Since both colonies have 2 locations with 2 possible actions, they have 4 pure policies or actions in the game. The payoffs for the game are the expected average payoffs for the resulting joint policy, as defined in Equation 8.10. Since these payoffs differ for the colonies the approximating game is not a team game, as was the case in the single colony

[2]if such a policy did not exist π_c would be an equilibrium of Γ.

		Colony 2			
		[a1,a1]	[a1,a2]	[a2,a1]	[a2,a2]
Colony 1	[a1,a1]	0.235, 0.235	0.295, 0.184	**0.288, 0.254**	0.297, 0.246
	[a1,a2]	0.184, 0.295	0.149, 0.149	0.128, 0.205	0.122, 0.183
	[a2,a1]	**0.254, 0.288**	0.205, 0.128	0.176, 0.176	0.169, 0.152
	[a2,a2]	0.246, 0.297	0.183, 0.122	0.152, 0.169	0.142, 0.142

Table 8.2: Colony game approximating the multi-colony version of Figure 8.1. Equilibria are indicated in bold.

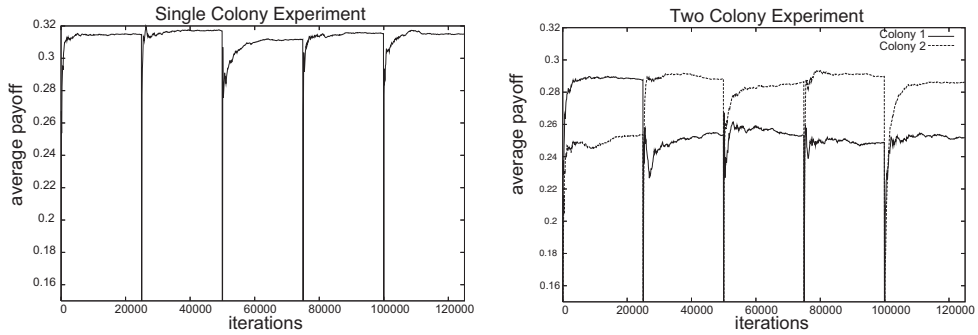
problem. Payoffs in the table are determined by calculating the stationary distribution of locations corresponding to the different policies and using these values in Equation 8.10.

The 2 pure Nash equilibria are indicated in bold. These equilibria correspond to a situation where 1 colony exploits source 1, but the other one selects source 2 in location 1. It should be noted that these equilibria give different payoffs for the colonies, with one colony receiving an average payoff of 0.288 and the other one receiving only 0.254. This results in different preferences of the colonies for both equilibria. Furthermore, the equilibria do not give the maximum possible reward for either colony, but they are Pareto optimal.

The pheromone game corresponding to this problem is shown in Table 8.3. This game consists of 4 players $p^{l,c}$, corresponding to the pheromone vectors $\tau^{l,c}$. Since two actions are present in each location, all players have two possible actions. Players corresponding to the same colony receive the same payoff for a play. These payoffs also correspond to the values achieved by the colonies in the game of Table 8.2. Equilibria of the game are indicated in bold. It can easily be verified that these equilibria correspond to the same joint policies as those of the colony game. Of course, while these games provide an interesting point of view for these small examples, calculating them for more realistic scale problems would is typically not feasible. Fortunately, the theoretical results above assure the equilibrium convergence of our model, *without the need to explicitly calculate these games.*

$(p^{1,1}, p^{1,2}, p^{2,1}, p^{2,2})$	$J^{\vec{\pi},1}, J^{\vec{\pi},2}, J^{\vec{\pi},1}, J^{\vec{\pi},2}$
(a1,a1,a1,a1)	0.235,0.235,0.235,0.235
(a1,a2,a1,a1)	**0.288,0.254,0.288,0.254**
(a1,a1,a2,a1)	0.184,0.295,0.184,0.295
(a1,a2,a2,a1)	0.128,0.205,0.128,0.205
(a2,a1,a1,a1)	**0.254,0.288,0.254,0.288**
(a2,a2,a1,a1)	0.176,0.176,0.176,0.176
(a2,a1,a2,a1)	0.246,0.297,0.246,0.297
(a2,a2,a2,a1)	0.152,0.169,0.152,0.169
(a1,a1,a1,a2)	0.295,0.184,0.295,0.184
(a1,a2,a1,a2)	0.297,0.246,0.297,0.246
(a1,a1,a2,a2)	0.149,0.149,0.149,0.149
(a1,a2,a2,a2)	0.122,0.183,0.122,0.183
(a2,a1,a1,a2)	0.205,0.128, 0.205,0.128
(a2,a2,a1,a2)	0.169,0.152,0.169,0.152
(a2,a1,a2,a2)	0.183,0.122,0.183,0.122
(a2,a2,a2,a2)	0.142,0.142,0.142,0.142

Table 8.3: Pheromone game approximating the multi-colony version of Figure 8.1. Column 1 lists possible plays, with column 2 giving the expected payoff for each player resulting from a play. Equilibrium plays are indicated in bold.



(a) (b)

Figure 8.2: Results for L_{R-I} update on the example of Figure 8.1. (a) Single Colony Experiment (b) Two colony experiment. Settings where $\lambda = 0.001$ and 100 ants per colony.

8.4 Experiments

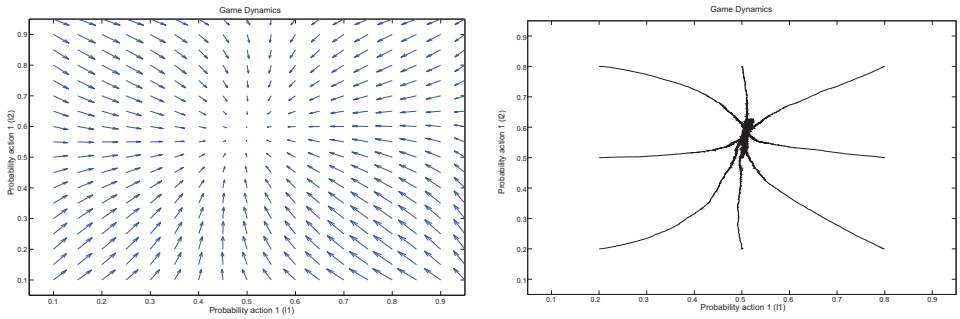
8.4.1 A Simple Example

We first demonstrate the single colony case. Figure 8.2(a) shows the average reward over time obtained by a single colony using the L_{R-I} update. To demonstrate the convergence we do not show an average over multiple runs, but rather a single typical run where we allow the average reward to converge and then randomly reinitialise the pheromone vectors. The figure shows the average reward over time obtained by a single ant of the colony at each time step, for a total of 25000 time steps for each initialisation. A single time step here corresponds to the selection of 1 action by all ants in the system, i.e each ant makes 1 transition between locations. In each case the system converged to the predicted equilibrium $(1, 1)$ and after 25000 time steps the average reward approached the predicted value to within 0.005.

Results for the two colony problem described in Section 8.3 can be seen in Figure 8.2(b). Here the system converged to either one of the equilibria in Table 8.3, with one colony receiving an average payoff of 0.254 and the other one receiving 0.288. Again the obtained values after 25000 iterations can be seen to closely approximate the predicted values. The eventual equilibrium reached depends on the initialisation of the pheromone vectors.

Additionally, we give an example of how this analysis can be used with another pheromone update system. Again we apply this update on the problem in Figure 8.1. Since this is the same problem as studied above, the approximating limiting game is still the game given in Table 8.1. However, we will demonstrate that a different pheromone update can give very different outcomes. We give results for the pheromone update in Equation 8.7, together with a Boltzmann normalisation. As was also described in Chapter 5, the learning dynamics for this scheme can be studied by determining a continuous time approximation using ordinary differential equations (ODEs) [Tuy04]. This continuous time limit of the dynamics gives an adapted version of the continuous time replicator dynamics. Using these ODEs, we can predict the outcome of the update on a strategic game by determining the stationary points of the system and investigating their stability. A similar approach was used in [SPT94] to show the equilibrium convergence of the L_{R-I} update in strategic games.

Rather than explicitly calculating the stationary points for the system, we visualise the dynamics using a direction plot. Figure 8.3(a) shows a plot of the predicted evolution of the action probabilities for both locations



(a) (b)

Figure 8.3: (a) Predicted dynamics on the game in Table 8.1. (b) Experimental results on the problem in Figure 8.1.

when using a Boltzmann normalization with $T = 0.2$. This plot was obtained by analysing the dynamics of the replicator equations on the approximating game in Table 8.1. The visualisation immediately shows that for these settings the update scheme will not reach the optimal pure equilibrium, but rather will go to a mixed outcome. Figure 8.3(b) shows the results for multiple runs with different pheromone initialisations on the single colony version of the problem in Figure 8.1. The ants used a Boltzmann function with $T = 5$ for action selection and the pheromone update in Equation 8.7 with $\lambda = 0.001$, with the colony consisting of 100 ants. The observed evolution of action probabilities on the single colony foraging experiment closely follow these predicted dynamics.

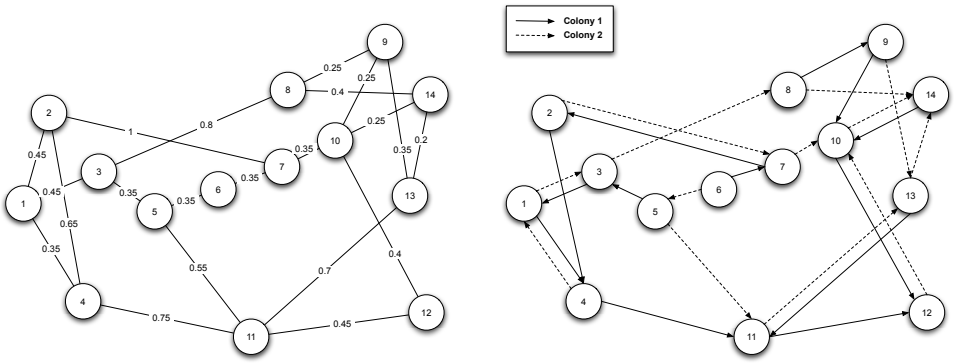
8.4.2 Ant Routing Model

As a final demonstration we give some results for our model on a larger problem. For this experiment we model a system inspired by the AntNet routing algorithm [DCD98], also described above. The algorithm uses ant inspired behaviour to route packets in communication networks. The system builds routing tables by periodically sending out ants to explore the network. These ant packets are sent by nodes to random destinations and use the same queues as network traffic. Ants select edges in the network probabilistically based on pheromone tables which associate a pheromone value with each edge for each destination node. After arriving at a destination ants are sent backwards to update pheromone tables in visited nodes, based on the total delay for the path they followed. Data packets in the net-

work are then routed by using a greedy policy with the pheromone tables built by ants.

We model this system using the multi-colony approach described in Section 8.3. Since pheromone values in AntNet depend on the destination nodes, this problem maps to a system with one colony for each destination in the network. Ants belonging to different colonies start from random nodes and travel through the network until they reach the destination node associated with their colony. We demonstrate this approach on the NSFNET network shown in Figure 8.4(a). This former North American backbone network was one of the first networks on which AntNet was demonstrated. We limit our experiments to 2 colonies which build routing tables to route packets towards destination nodes 12 and 14. Ants from both colonies start from all other network nodes and travel to the network until they reach their destination where they receive a reward $+1$. When ants traverse a network edge they suffer a delay as indicated in Figure 8.4(a). So in order to maximise the average reward over time, ants need to minimise the delay from each node to the destination. To simulate the influence of heavy network load slowing down the network, ants receive a penalty when both colonies use the same edge. If an ant uses an edge which more than 50% of ants from the other colony also prefer, they suffer a delay of 100 for that edge instead of the normal delay.

Explicitly calculating the approximating games in this case becomes impractical as it would result in a game with 2×14 players, resulting in a large number of plays to evaluate. When we use an update like L_{R-I}, however, we can still give guarantees based on the convergence properties of the update scheme in games. We know that the update converges to a Nash equilibrium between colony policies. This means that each colony will prefer the minimum delay routes to their destination, with respect to the current policy of the other colony. A colony will thus prefer lower delay routes and avoid sharing edges since this results in high delays. Note that these results represent equilibria (i.e. local optima) but not necessarily (Pareto) optimal results for this problem. In the next section we discuss some possibilities which could be used to improve the outcome based on this equilibrium convergence behavior. A typical solution found by the L_{R-I} system is shown in Figure 8.4(b). Average results over 20 runs are given in Table 8.4. For comparison purposes the table also lists the results obtained when both colonies use Dijkstra shortest path routing to find the minimum cost path to their destination, but do not take into account which edges are used by the other colony. From Table 8.4 it is clear that the pheromone based routing achieves relatively low delay routes. The antnet routing model shares an



(a) (b)

Figure 8.4: (a) NFSNet former backbone (after [DCD98]). Each link represents 2 directed edges, numbers indicate delay of an edge. (b) Example solution found by AntNet routing model using L_{R-I} update ($\lambda = 0.001$) with 2 colonies routing paths to nodes 12, 14.

average of 0.3 edges over 20 trials, compared to 6 edges shared in shortest path routing scheme.

8.5 Discussion

In this chapter we have shown how the behavior of pheromone based stigmergetic systems can be analysed in terms of the dynamics of the pheromone update in an approximating game. While the explicit calculation of these games becomes cumbersome or even impossible in large problems, this analysis can still be applied by studying the dynamics of the pheromone update. For instance, as we have shown for the case of the L_{R-I} update scheme, convergence to a pure Nash equilibrium is a sufficient condition to assure optimal convergence in the single colony case and convergence to a Nash equilibrium between colony policies in the multiple colony case. These properties do not depend on the underlying game, and can therefore be assured in other problems without the need to calculate the approximating game.

In the multi-colony case one could argue that while Nash Equilibrium convergence is an interesting property, it is not necessarily the desired result for an optimisation approach. In a Nash equilibrium agents play mu-

start node	colony 1	colony 2	Dijkstra 1	Dijkstra 2
1	41.2 (21.3)	41.6 (20.0)	116	33
2	44.0 (22.5)	49.7 (27.15)	208	205
3	30.8 (5.0)	27.8 (6.12)	27	24
4	38.6 (20.2)	47.3 (28.3)	109	214
5	28.1 (11.1)	31 (2.1)	20	212
6	40.1 (33.6)	34.7 (20.3)	208	205
7	43.5 (35.6)	45.6 (35.4)	108	105
8	19.3 (3.9)	9.1 (2.7)	113	8
9	15.3 (5.6)	11.3 (1.0)	108	105
10	8.0 (0.0)	5.0 (0.0)	8	5
11	11.4 (7.2)	18.4 (1.2)	9	114
12	0 (0.0)	13.0 (0.0)	0	13
13	21.8 (1.8)	4.7 (2.9)	113	100
14	13.7 (3.1)	0 (0.0)	13	0

Table 8.4: Results obtained by AntNet model in NFSNET experiment. Columns 2 and 3 give the average delay (standard deviation) to destination nodes 12 and 14, respectively (results averaged over 20 runs). For comparison purposes columns 4 and 5 give the delays that result from shortest paths based routing to both destinations, without taking into account delays caused by sharing edges.

tual best replies, and as such it represents a local optimum. Nash equilibria do not guarantee equal payoffs for the players, however, and can result in unfair solutions. Moreover, multiple equilibria can exist, with players having different preferences for these equilibria, resulting in a selection problem. Furthermore equilibria can be Pareto dominated. This means that solutions exist where all players receive at least the same reward and one or more players do strictly better (this is for instance the case in the well-known prisoner's dilemma game). Especially in the case of multi-objective optimisation problems the goal is often to find a Pareto optimal (i.e. non-dominated) solution.

The (pure) Nash equilibrium convergence of a learning rule, however, can be used as a basis for designing more complex systems which exhibit the desired properties. In team games, where all players receive the same payoffs and a globally optimal equilibrium always exists, MMDP-ILA using the parameterised LA update could be employed to obtain optimality. In conflicting interest games the turn-taking techniques of Chapter 6 can find periodic solutions which equalise the average payoff over time.

8.6 Conclusion

The analysis developed in this chapter offers a new framework in which the outcomes of pheromone based agent coordination can be studied. Unfortunately, it is difficult to use this model with existing ACO type optimisation algorithms, since these systems typically incorporate additional techniques, such as heuristic information and local search. Further research is needed to determine how these methods influence the game structure obtained for the pheromone dynamics. While it may not be possible to give explicit convergence guarantees for these extended systems, we believe that our framework can aid in the design of these algorithms, as it allows to better understand the dynamics of the pheromone interactions, and ensure that these interactions lead to desirable outcomes.

Chapter 9

Conclusion

Reinforcement learning is a powerful tool, which allows a single agent to learn optimal behaviour in a previously unknown and possibly stochastic environment. These results are obtained solely through experimentation using trial-and-error interactions with the environment. Convergence guarantees are available under some relatively straightforward assumptions on the environment and the amount of experimentation allowed to the agent. These properties make RL an interesting focus for unsupervised learning systems.

However, when we moving to a decentralised, multi-agent learning setting, the preconditions for convergence of RL algorithms are generally no longer satisfied. In this setting both agent rewards and the global system state depend on the actions taken by all agents. As such, the learning agents not only have to deal with the stochastic environment, but also have to take interactions with other agents into account.

In this dissertation we introduced a new framework for reinforcement learning in multi-agent settings. We have also provided convergence results for these algorithms for a wide range of settings. These results are obtained using a repeated normal form game approximation of the learning dynamics. This analysis also provides a new framework for analysing learning in the Markov game framework using classic game theory. In the following, we give a detailed overview of the contributions presented in this work as well as some possible future research directions.

9.1 Contributions

Below we give an overview of our main contributions to the current state of multi-agent learning research:

- *Demonstrate the importance of LA in MARL:* Throughout this dissertation we show that LA are valuable tools for the design of multi-agent learning algorithms. In this we go beyond previously considered repeated game [Ver04] and tree-structured game [Pee08] settings.
- *MDP game view:* we promote a unified view of single and multi-agent reinforcement learning, by approximating both settings with the problem of finding an equilibrium in a normal form game. Both these settings can then be analysed in a game theoretic framework. In the single agent setting we obtain a common interest game in which equilibria represent optimal policies. In the multi-agent setting we get a possibly conflicting interest game in which equilibria represent equilibrium points between agent policies.
- *MG-ILA algorithm:* We introduce a new algorithm for learning in general sum Markov games. This algorithm is among the most flexible current approaches in terms of information requirements, but still offers broad convergence guarantees in self-play.
- *Multi-level Markov game analysis:* We introduced a novel framework for analysing learning in Markov games using approximating normal form game representations. We showed that this analysis can be applied at different levels, highlighting different complexities that arise in learning.
- *MMDP-ILA:* We introduced an algorithm capable of finding optimal joint policies in common interest Markov games. Currently the only algorithm matching these results [WS03] has significantly greater information requirements.
- *Correlated policies:* We introduced a flexible system using minimal communication, which allows agents to correlate their policy play in order to alternate between playing different agents' preferences. This system provides a simple method to compose complex non-stationary solutions. It also allows a departure from the traditional focus on equilibrium points in conflicting interest problems. Using this approach agents can also reach a fair solution which avoids possible low equilibrium payoffs.

- *Analysis of multi-state learning dynamics:* we introduced the first approach for analysing and visualising learning dynamics in multi-state problems. Our approach extends current research that uses evolutionary game theory to study reinforcement learning in repeated games and allows the same methods to be applied to multi-state problems.

- *Convergence in hidden state problems:* We demonstrated that the algorithms introduced in this work can be applied in settings with even more limited information available to the agents. Even in these settings we obtain convergence to an equilibrium between agent policies. These results are achieved using the basic algorithms, without relying on complex state estimation techniques.

- *Modelling stigmergy:* We provide a multi-colony model of pheromone communication based on shared updating of learning automata. This allows a more formal treatment of stigmergy based algorithms using the tools offered by classical game theory. It also aids in the design of these algorithms, as the methods developed in this dissertation can be used to analyse the global consequences of low-level stigmergetic interactions.

9.2 Future work

In closing we list some points that warrant further attention. We start with the items we believe offer the most opportunity for expanding our framework: scaling MG-ILA to large-scale and possibly continuous problems. The key to achieving this, is to further develop the techniques used in Chapter 6 and Chapter 7 in order to create general state and policy abstraction for MG-ILA:

- *Adaptive resolution for state representations:* The 2 approaches with regard to the visibility of state variables that are used in this dissertation, represent extremes. On the one hand we consider full visibility, on the other hand in Chapter 7 we give agents access only to those variables also present in a single agent setting. Current MARL research focuses on intermediate approaches [MV09, DHVN09, DVN10]. These approaches start with a minimal state space, but expand when needed for achieving better results. Since we have shown that LA are able to converge to equilibrium in both extreme cases, they offer an interesting approach for use in these adaptive state space algorithms.

This would entail first using an algorithm to learn the best state space resolution to use (either by splitting large states or aggregating simple states). This could then be followed by assigning a learning automaton to each of the resulting states and applying MG-ILA as before.

- *Alternative objectives for correlated policies:* Equalising agent rewards is but one of many possible objectives that can be considered in the correlated policies framework of Chapter 6. The policies learned by the parameterised LA can serve as a set of basic solutions that can be composed into more advanced solution concepts. In [GHS03] the equalising approach is considered (under the name egalitarian objective) and compared with several alternatives such as maximise the sum of agent rewards (utilitarian objective) or maximise the maximum of agent rewards (republican objective). These different objectives could also be used in our framework to determine joint policy switches, but their properties need to be investigated in more detail. Another possible route is to allow the agents to individually learn a strategy over the set of basic policies. This would result in a setting similar to the options framework used in single agent RL [SPS99].

- *Generalise partial state observations:* In Chapter 7, we considered the case in which the set of state variables could be perfectly decomposed in one local state for every agent. On could of course consider more general factored systems, in which agents view different, possibly non-disjoint sets of state variables. The applicability and convergence of MG-ILA in this case remains to be investigated.

In addition to the points above, some smaller adaptations could be made to the framework, in order to increase its applicability:

- *MG-ILA for discounted Markov games:* The algorithms in this work all extend the MDP-ILA algorithm [WJN86], which was shown to be optimal in average reward MDPs. However, as mentioned in Chapter 3, similar results exist for discounted MDPs [Wit77]. These results could serve as the basis for a discounted version of MG-ILA

- *Relax ergodicity assumption:* Throughout this dissertation we required the Markov chain of system states to be ergodic under all (joint) policies. This assumption could be relaxed in order to allow a wider range of problems. First, one could consider the general recurrent case, in which all states are still recurrent, but we no longer require

them to be aperiodic. The difficulty in this setting is that the limits used for the average expected reward, may not exist in the case of periodic chains. This can be dealt with, however by considering the corresponding Cesaro limits [Put94]. Secondly, we could consider the unichain case. Here we still have a unique stationary distribution for each policy, but we allow some states to have stationary probability of 0. These transient states are only visited during the initial learning phases and accordingly they do not contribute to the long term average reward. While we cannot guarantee the convergence of automata in transient states, most other results are expected to carry over to this case. Extensions to both these settings should be possible, as both remain close to the ergodic case. On the other hand, the full multi-chain problem remains an open problem. Several issues make learning in this setting difficult. Chief among these are the non-constant expected rewards for stationary policies and the failure of stationary equilibria to exist. To the best of our knowledge no learning approaches exist to deal with these issues.

Finally, we note some more theoretical aspects that could be investigated further:

- *Investigate ILA relationship to discounted Replicator Dynamics:* A recent and intriguing result from MARL literature [Akc09], shows that the continuous time replicator dynamics can be used as the basis for an algorithm capable of finding Nash equilibria in discounted general sum Markov games. Unfortunately, the current approach requires knowledge of the exact dynamical system, which then needs to be numerically simulated. One possible avenue of research for MG-ILA in discounted Markov games, is to investigate whether the relation between the RD and LA can be exploited to implement a similar algorithm, without the need for knowing the exact dynamics. If a suitable automata feedback scheme could be devised, this could lead to general Nash convergence in discounted Markov games, without the limits inherent to the current approach.

- *Alternative LA update schemes:* The ILA algorithms in this dissertation all relied on either the L_{R-I} or the PLA update system. However, a large body of work on LA updates and their properties exists [TS04]. Especially interesting are ϵ-optimal schemes such as pursuit automata which offer behaviour similar to reward-inaction but may exhibit faster convergence.

- *Mixed equilibria:* The MG-ILA algorithm is able to converge to pure equilibria. However, there is no guarantee that such equilibria exist. As such it should be investigated if it is possible to achieve convergence to mixed equilibrium points. An interesting possibility here is the Reward-ϵPenalty update scheme, described in Chapter 2. This scheme is known to be able to approximate mixed equilibria in 2 player zero-sum automata games. Currently it is not clear, however, if these properties extend to multi-automata systems.

- *Regret based learning:* Recently, the notion of regret based learning has received interest in the multi-agent learning community. Regret is defined as for not playing the best strategy in hindsight. In repeated games, regret based algorithms have been shown to be able to converge to either the set of correlated equilibria [HMC01] or general Nash equilibria [FY06, GL07]. Both these learning rules are applicable in situations similar to those of LA and could possibly be used as update rules in MG-ILA. An interesting research topic would be to see if a regret based ILA algorithm could achieve similar results in Markov games.

- *Multi-state dynamics:* In Chapter 5 we visualised the dynamics of simple 2-state, 2-agent problems. While the link with the replicator dynamic can still be used in larger problems, the visualisation of the dynamics becomes difficult. As such the possibility of extending state based visualisations remains an important question. Alternative methods for investigating the basins of attraction of different equilibria, such as used in [PTL08] could be employed in these cases.

Appendix A

Dynamical Systems

In this appendix we introduce some basic notions of dynamical systems. We only consider differential systems, i.e. systems that can be described by a set of ordinary differential equations (ODE). These systems play an important role in Machine learning as they are often used to study the dynamics of learning algorithms. Examples are the ODE approach used to demonstrate the convergence of learning automata [SPT94] and the replicator dynamics from evolutionary game theory which are also used to study reinforcement learning in repeated games[Tuy04].

This appendix is based on [Str01], [HSD04] and [Man88].

A.1 Terminology

A dynamical system consists of a *state space* or *phase space* S and a vectorfield F on this space. The state space S is the set of points describing all possible configurations of the system. Each point in this space gives a complete characterisation of the system. The vectorfield F associates with each point $x \in S$ a vector, describing the evolution of the system in time. In an n-dimensional state space, the system can be described by n ordinary differential equations:

$$\begin{aligned} \frac{dx_1}{dt} &= f_1(x_1, \ldots, x_n) \\ \ldots & \\ \frac{dx_n}{dt} &= f_n(x_1, \ldots, x_n) \end{aligned} \tag{A.1}$$

which can also be written as:

$$\vec{x}' = F(\vec{x}) \tag{A.2}$$

where $\vec{x} = (x_1 \ \ldots \ x_n)^T$ and F is the vector valued function $F(\vec{x}) = (f_1(\vec{x}), \ldots, f_n(\vec{x}))$.

Definition 26 (trajectory) *The solution for a system $\vec{x}' = F(\vec{x})$ with initial condition $\vec{x}_0$ is a function $X(t)$ describing the succession of states through time, starting at $\vec{x}_0$. This function is also called the trajectory starting at $\vec{x}_0$*

Under some general conditions on the function F it can be shown that there exists a unique trajectory $X(t)$ starting at state $\vec{x} \in S$.

Definition 27 (fixed point) *A fixed point (also called critical, stationary or equilibrium point) of the system A.2 is a point $\vec{x}^* \in S$ for which $F(\vec{x}^*) = 0$.*

Fixed points correspond to constant solutions of the system, i.e. $X(t) = \vec{x}^*$, $\forall t$.

Definition 28 (stability) *A fixed point $\vec{x}^* \in S$ is called stable (or Lyapunov stable) if all trajectories $X(t)$ that start at a point $X(0)$ sufficiently close to $\vec{x}^*$, remain close to it for all time $t > 0$. If $\vec{x}^*$ is stable and all trajectories starting near $\vec{x}^*$ approach $\vec{x}^*$ as time goes to infinity, i.e. $X(t) \to \vec{x}^*$ as $t \to \infty$,then $\vec{x}^*$ is called asymptotically stable. A fixed point that is not stable is called unstable.*

A.2 Linear Systems

Linear systems are an important part of dynamical systems theory, since these systems are well understood and can be solved analytically. Furthermore, linear systems can be used to locally approximate nonlinear systems in the neighbourhood of a given point. This provides an important tool for the analysis of nonlinear systems, which in general cannot be solved. The general form for a linear system is:

$$\begin{aligned} \frac{dx_1}{dt} &= a_{11}x_1, \ldots, a_{1n}x_n \\ \ldots & \\ \frac{dx_n}{dt} &= a_{n1}x_1, \ldots, a_{nn}x_n \end{aligned} \tag{A.3}$$

which can also be written as:

$$\vec{x}' = A\vec{x} \tag{A.4}$$

where A is the matrix $(a_{ij}), i = 1 \ldots n, j = 1 \ldots n$. Note that the origin is always a fixed point of these systems.

A special set of solutions for linear systems are given by the eigenvectors of the matrix A. Let $\vec{v}_1$ be a non-zero vector such that $A\vec{v}_1 = \lambda_1\vec{v}_1$, i.e. $\vec{v}_1$ is an eigenvector of A with eigenvalue λ_1. Then

$$X(t) = e^{\lambda_1 t}\vec{v}_1 \tag{A.5}$$

is a solution of the system in Equation A.4. For each t this solution corresponds to a scalar multiple of the vector $\vec{v}_1$ and all points $X(t)$ lie on the line from the origin through $\vec{v}_1$. Therefore, the solution $e^{\lambda_1 t}\vec{v}_1$, is also called a *straight line solution* of the system. Below we will show how this result can be used to determine the general solution of the system. We start out by considering planar systems.

A.2.1 Planar systems

In a planar system (i.e. a 2-dimensional system), we can determine the 2 straight line solutions of the system by finding both eigenvalues and corresponding eigenvectors of the matrix A. Suppose the matrix A is given by:

$$\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

Then the eigenvalues of A are λ_1 and λ_2 with corresponding eigenvectors $v_1 = (1\ 0)^T$ and $v_2 = (0\ 1)^T$. This results in the straight line solutions $e^{\lambda_1 t}\vec{v}_1$ and $e^{\lambda_2 t}\vec{v}_2$. The *general solution* of the planar linear system can then be found by combining both solutions:

Theorem 12 (Hirsch et al.) *Suppose matrix A has a pair of real eigenvalues $\lambda_1 \neq \lambda_2$ with associated eigenvectors $\vec{v_1}$ and $\vec{v_2}$. Then the general solution of the system $\vec{x}' = A\vec{x}$ is given by:*

$$X(t) = c_1 e^{\lambda_1 t}\vec{v}_1 + c_2 e^{\lambda_2 t}\vec{v}_2 \tag{A.6}$$

Given the general solution, we can now find the trajectory with initial condition $X(0) = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$ by solving:

$$c_1\vec{v}_1 + c_2\vec{v}_2 = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

to determine the values for c_1 and c_2.

The special cases where eigenvalues are equal or complex can be treated similarly. For a matrix

$$A = \begin{pmatrix} \alpha & \beta \\ -\beta & \alpha \end{pmatrix}$$

which has complex eigenvalues $\alpha \pm i\beta$ the general solution is given by:

$$X(t) = c_1 e^{\alpha t} \begin{pmatrix} cos\beta t \\ -sin\beta t \end{pmatrix} + c_2 e^{\alpha t} \begin{pmatrix} sin\beta t \\ cos\beta t \end{pmatrix} \tag{A.7}$$

When A has the form

$$\begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix}$$

both eigenvalues are equal to λ and we have a single linearly independent eigenvector $(1\ 0)$. In this case the general solution is given by:

$$X(t) = c_1 e^{\lambda t} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + c_2 e^{\lambda t} \begin{pmatrix} t \\ 1 \end{pmatrix} \tag{A.8}$$

In general when given a linear system, where the matrix A is invertible, one can first find a transformation T:

$$\vec{y}' = (TAT^{-1})\vec{y} \tag{A.9}$$

such that the matrix (TAT^{-1}) has one of the following forms:

$$\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}, \quad \begin{pmatrix} \alpha & \beta \\ -\beta & \alpha \end{pmatrix}, \quad \begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix}$$

This is also called the *canonical form* of A. To find this form we need to calculate T, which is the matrix whose columns are the the eigenvectors of A. The transformed system can then easily be solved using the methods above. If $Y(t)$ is a solution for the of the system in Equation A.9, the solution for the original system is given by $TY(t)$.

A.2.2 Classification of Planar Systems

Using the methods in the previous section we can make a complete classification of the possibilities that can occur in a planar, linear dynamical system. This classification relies on 2 factors: whether the eigenvalues are real

or complex and the sign of the (real parts of the) eigenvalues. The eigenvalues of the matrix A can be found by solving the *characteristic equation* $det(A - \lambda I) = 0$. For a 2-dimensional matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

this results in the equation:

$$\lambda^2 - \tau\lambda + \Delta = 0$$

where

$$\begin{aligned} \tau &= trace(A) = a + d \\ \Delta &= det(A) = ad - bc \end{aligned} \tag{A.10}$$

This results in the solutions:

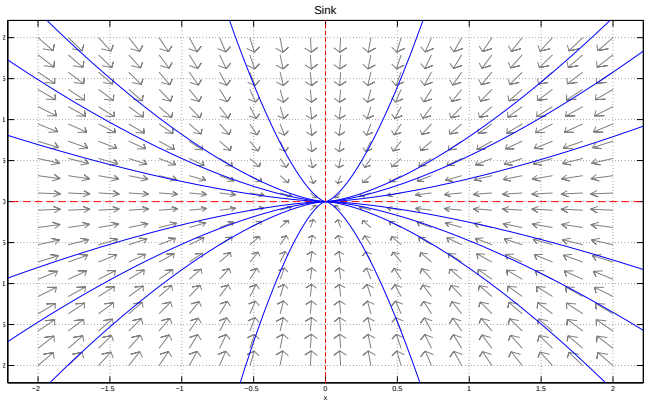
$$\lambda_1 = \frac{\tau - \sqrt{\tau^2 - 4\Delta}}{2},\ \lambda_2 = \frac{\tau - \sqrt{\tau^2 + 4\Delta}}{2}$$

Thus the eigenvalues are determined by the trace τ and determinant Δ of the matrix A. We can distinguish the following cases:

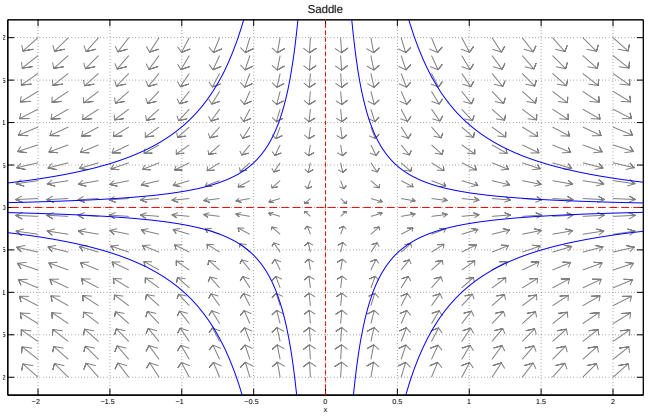
- $\tau^2 - 4\Delta > 0$: **2 distinct, real eigenvalues λ_1 and λ_2**. We can further refine this case by examining the signs of both eigenvalues. From the straight line solution in Equation A.5 we can see that the solutions either tend to the origin when $\lambda < 0$ or away from it when $\lambda > 0$.This gives 3 possible cases for the 2 eigenvalues:

  1. $\lambda_1 < \lambda_2 < 0$
  2. $\lambda_1 < 0 < \lambda_2$
  3. $0 < \lambda_1 < \lambda_2$

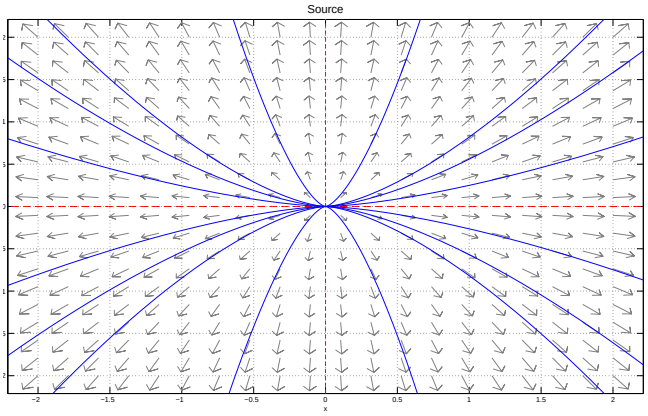
  In the first case all solutions tend to the origin as $t \to \infty$. The origin is therefore called a sink. In Figure A.1 (a) we give an example phase portrait for this situation. If instead the eigenvalues have opposite sign (case 2), the solutions along the line corresponding to the negative eigenvalue (the stable line) tend to the origin, while those along the line corresponding to the positive eigenvalue (the unstable line) tend away from it. All other solutions come from ∞ in the direction of the stable line and tend to ∞ in the direction of the unstable line.



(a)

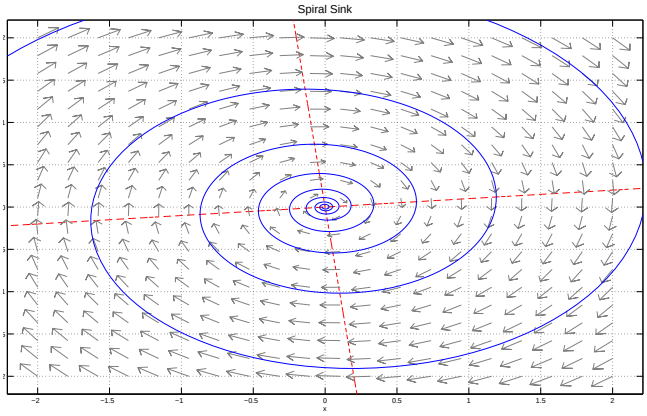


(b)

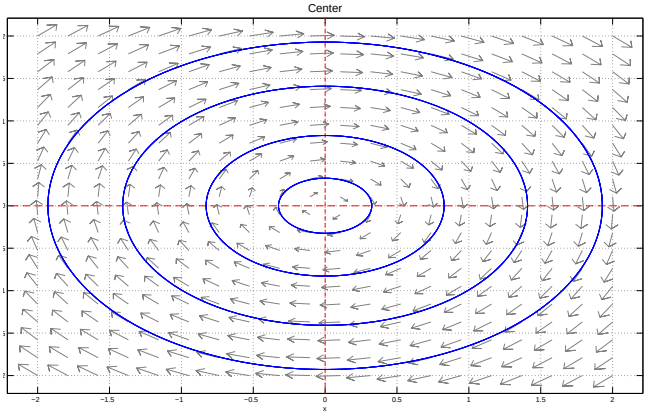


(c)

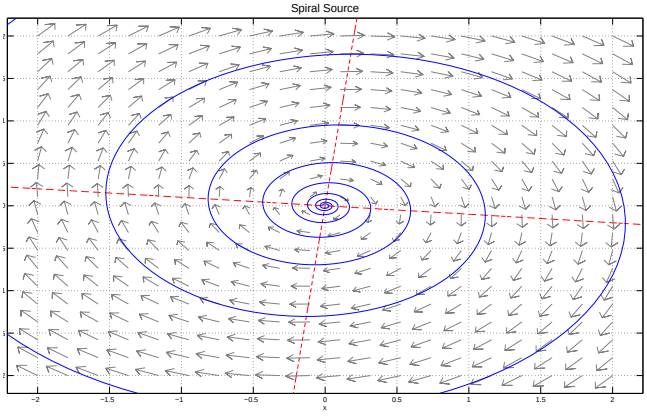
Figure A.1: Phase plots for systems with 2 distinct real eigenvalues.



(a)



(b)



(c)

Figure A.2: Phase plots for systems with imaginary eigenvalues.

The result is *a saddle*, shown in Figure A.1(b). In the final case, both eigenvalues are strictly positive, giving the opposite of the first case. All solutions now tend away from the origin, which is called *a source*. The corresponding phase portrait is given in Figure A.1(c).

- $\tau^2 - 4\Delta = 0$: **a single real, repeated eigenvalue λ.** This corresponds to the boundary cases, which signify the transition from real to complex eigenvalues.
- $\tau^2 - 4\Delta < 0$: **complex eigenvalues with nonzero imaginary part.** We can again subdivide this case, now based on the sign of the real part of the eigenvalues. The real part for both eigenvalues is given by $\tau/2$. We can now consider the following cases:

  1. $\tau < 0$
  2. $\tau = 0$
  3. $\tau > 0$

  In the first case, solutions again tend to the origin, now resulting in a *spiral sink*, visualised in Figure A.2(a). In the second case solutions neither tend to or away from the origin. Instead they circle around it, resulting in periodic solutions. The result is called *a center* and is shown in Figure A.2(b). In the last case trajectories again tend away from the origin which is now called a *spiral source.* This is shown in Figure A.2(c).

The entire classification above can be summarised in a single figure by plotting the parabola $\tau^2 = 4\Delta$ in the $\tau\Delta-$plane. This is shown in Figure A.3. The regions are labelled with the resulting type of system for corresponding values of τ and Δ.

Example 14 (Linear system) *(based on Example 5.21 in [Str01])*
As an example of a planar linear system we will consider the system:

$$\begin{aligned} x' &= x + y \\ y' &= 4x - 2y \end{aligned}$$

with initial condition $(x_0, y_0) = (2, -3)$. The corresponding matrix A for this system is:

$$\begin{pmatrix} 1 & 1 \\ 4 & -2 \end{pmatrix}$$

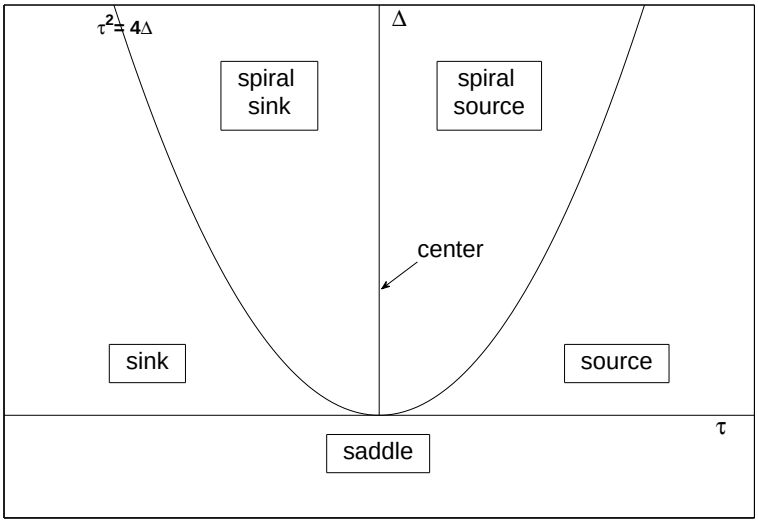


Figure A.3: Classification of planar linear dynamic systems.

which has characteristic equation $\lambda^2 + \lambda - 6 = 0$*. This results in 2 distinct real eigenvalues* $\lambda_1 = 2$ *and* $\lambda_2 = -3$*. Since the eigenvalues have different signs the origin is a saddle. The corresponding eigenvectors are:*

$$v_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ and } v_2 = \begin{pmatrix} 1 \\ -4 \end{pmatrix}$$

This gives straight line solutions $X(t) = e^{2t} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ *and* $X(t) = e^{-3t} \begin{pmatrix} 1 \\ -4 \end{pmatrix}$*, with general solution:*

$$X(t) = c_1 e^{2t} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + c_2 e^{-3t} \begin{pmatrix} 1 \\ -4 \end{pmatrix}$$

To determine c_1 *and* c_2 *we solve:*

$$\begin{pmatrix} 2 \\ -3 \end{pmatrix} = c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} + c_2 \begin{pmatrix} 1 \\ -4 \end{pmatrix}$$

which results in $c_1 = 1$ *and* $c_2 = 1$ *and finally gives the solution:*

$$X(t) = e^{2t} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + e^{-3t} \begin{pmatrix} 1 \\ -4 \end{pmatrix}$$

Figure A.4 visualises this system and the solution above.

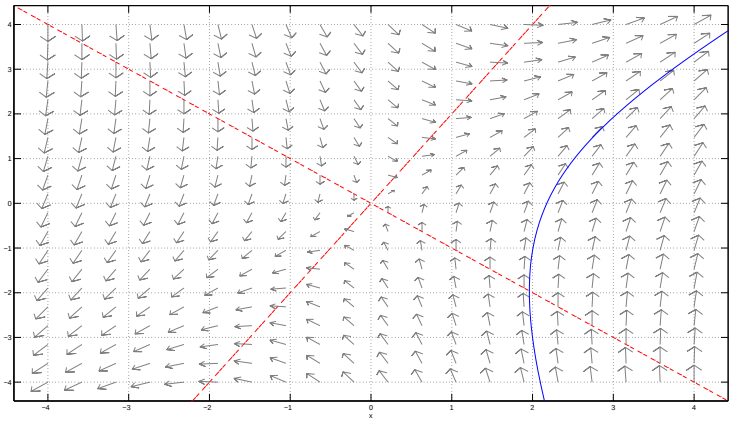


Figure A.4: Vector field and solution for the system described in Example 14

A.2.3 Higher dimensions

The results for linear systems, described in the previous sections, can be extended to higher dimensions. Again we put matrix A in its canonical form and solve the resulting system. In an n-dimensional system, the general solution for the linear system then is described in the following theorem:

Theorem 13 (Hirsh et al.) *Consider the system $\vec{x}' = A\vec{x}$, where A has distinct eigenvalues $\lambda_1, \ldots, \lambda_k \in \mathbb{R}$ and $\alpha_1 + i\beta_1, \ldots, \alpha_r + i\beta_r$,. Let T be the transformation to put A in its canonical form:*

$$TAT^{-1} = \begin{pmatrix} \lambda_1 & & & & & & & \\ & \ddots & & & & & & \\ & & \lambda_k & & & & & \\ & & & B_1 & & & & \\ & & & & \ddots & & & \\ & & & & & \ddots & & \\ & & & & & & B_r \end{pmatrix}$$

where

$$B_j = \begin{pmatrix} \alpha_j & \beta_j \\ -\beta_j & \alpha_j \end{pmatrix}$$

Then the general solution of the system is $TY(t)$, with:

$$Y(t) = \begin{pmatrix} c_1 e^{\lambda_1 t} \\ . \\ . \\ . \\ c_k e^{\lambda_k t} \\ a_1 e^{\alpha_1 t} cos\beta_1 t + b_1 e^{\alpha_1 t} sin\beta_1 t \\ -a_1 e^{\alpha_1 t} sin\beta_1 t + b_1 e^{\alpha_1 t} cos\beta_1 t \\ \\ . \\ . \\ . \\ a_r e^{\alpha_r t} cos\beta_r t + b_r e^{\alpha_r t} sin\beta_r t \\ -a_r e^{\alpha_r t} sin\beta_r t + b_r e^{\alpha_r t} cos\beta_r t \end{pmatrix}$$

We can now again look at the eigenvalues to determine the stability of the origin. The subspace spanned by the eigenvalues with negative real part is called the stable subspace. Within this subspace the solutions tend to the origin. the eigenvalues with positive real part span the unstable subspace, where solutions tend away from the origin.

Alternatively, we can write the general solution for A.4 with initial condition $X(0) = \vec{x}_0$ as the matrix exponential $e^{tA}\vec{x}_0$, where

$$e^{tA} = \sum_{i=0}^{\infty} \frac{(tA)^i}{i!} \tag{A.11}$$

A.3 Nonlinear Systems

Contrary to their linear counterparts, most nonlinear systems of ODEs cannot be solved analytically. Instead of solving these systems explicitly, we typically try to characterise their behaviour in terms of *attractors*. Attractors are subsets of S, such that all neighbouring trajectories tend to this set as $t \to \infty$. An asymptotically stable fixed point is one example of an attractor.

An important tool for this investigation is the linear stability analysis of equilibrium points. Consider a general nonlinear system $\vec{x}' = F(\vec{x})$. We

denote by $DF_{\vec{x}_0}$ the Jacobian of F, evaluated at $\vec{x}_0$. We can then write the linear system:

$$\vec{y}\,' = DF_{\vec{x}_0}\vec{y} \tag{A.12}$$

This system is called the *linearised system near* $\vec{x}_0$. An important result from dynamical systems theory states that if $\vec{x}_0$ is a hyperbolic equilibrium point[1] of the system $\vec{x}\,' = F(\vec{x})$, then the flow of the system near $\vec{x}_0$ resembles that of the linearised system in Equation A.12. This result allows us to investigate the stability of nonlinear equilibrium points, by analysing the corresponding linearised system.

Example 15 (Linearisation) *(Based on example 6.3.1 from [Str01])*
Consider following nonlinear system:

$$\begin{aligned} x' &= x^3 - x \\ y' &= -2y \end{aligned}$$

A straightforward calculation shows that this system has 3 *fixed points: (-1,0), (0,0) and (1,0). We will now investigate the stability of these points by considering the linearised system. The general Jacobian for this system is:*

$$DF = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix} = \begin{pmatrix} 3x^2 - 1 & 0 \\ 0 & -2 \end{pmatrix}$$

Evaluated at the 3 *fixed points this gives:*

$$DF_1 = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}, DF_2 = \begin{pmatrix} -1 & 0 \\ 0 & -2 \end{pmatrix}, DF_3 = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$$

From this we immediately see that the points at $(\pm 1, 0)$ *are saddles and thus unstable, while the origin is a sink and stable. This can be confirmed by inspecting the phase plot of the system in Figure A.5.*

It can be shown that in 2 dimensions the only possible attractors are fixed points (*point attractors*) or closed orbits which correspond to *periodic attractors*. In higher dimensions other possibilities may arise. Examples are a torus (*quasi-periodic attractor*) and fractal sets (*strange or chaotic attractors*). Figure A.6 shows the Lorenz attractor which is an example of chaotic attractor.

[1] an equilibrium $\vec{x}_0$ is called hyperbolic if all eigenvalues of the corresponding $DF_{\vec{x}_0}$ have nonzero real parts.

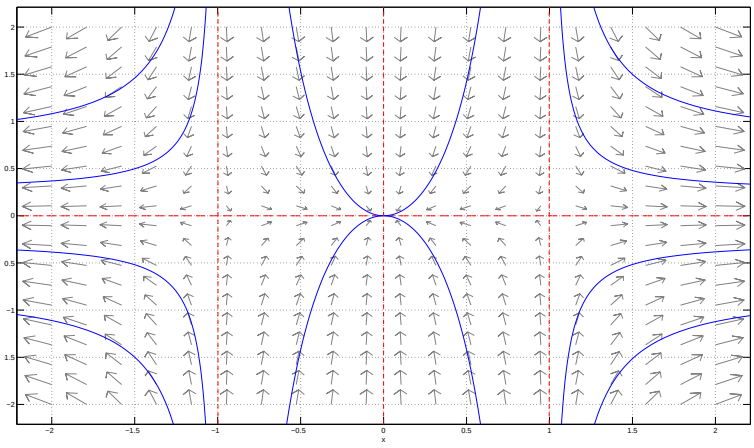


Figure A.5: Phase plot for the system described in Example 15

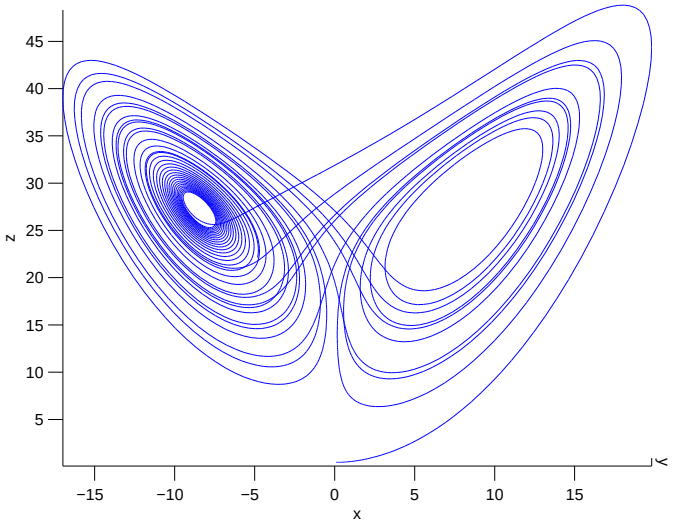


Figure A.6: The Lorenz attractor.

Bibliography

[Akc09] N. Akchurina. Multiagent reinforcement learning: algorithm converging to Nash equilibrium in general-sum discounted stochastic games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2 (AAMAS2009)*, pages 725–732. IFAAMAS, 2009.

[AS93] E. Altman and A. Shwartz. Time-Sharing Policies for Controlled Markov Chains. *Operations Research*, 41(6):1116–1124, 1993.

[Aum74] R.J. Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1(1):67–96, 1974.

[AW09] D. Angus and C. Woodward. Multiple objective ant colony optimisation. *Swarm Intelligence*, 3(1):69–85, 2009.

[Ban68] A. Banos. On pseudo-games. *The Annals of Mathematical Statistics*, pages 1932–1945, 1968.

[BBDS08] L. Busoniu, R. Babuska, and B. De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172, March 2008.

[BDT99] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence:From Natural to Artificial Systems*. Oxford University Press, New York, 1999.

[Bel57] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[BHD94] R. Beckers, O. Holland, and J.L. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. *Artificial Life IV*, 181:189, 1994.

[BM55] R.R. Bush and F. Mosteller. *Stochastic models for learning*. Wiley New York, 1955.

[Bou96] C. Boutilier. Planning, Learning and Coordination in Multiagent Decision Processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 195–210, Renesse, The Netherlands, 1996.

[Bow05] M. Bowling. Convergence and No-Regret in Multiagent Learning. In *Advances in Neural Information Processing Systems 17 (NIPS)*, pages 209–216, 2005.

[BP03] B. Banerjee and J. Peng. Adaptive policy gradient in multiagent learning. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems (AAMAS)*, pages 686–692, New York, NY, USA, 2003. ACM.

[Bru99] S. Brueckner. *Return from the Ant: Synthetic Ecosystems for Manufacturing Control*. PhD thesis, PhD Dissertation, Humboldt-Universitat Berlin, Germany (2000), 1999.

[BS97] T. Börgers and R. Sarin. Learning through reinforcement and Replicator Dynamics. *Journal of Economic Theory*, 77:1–14, 1997.

[BS98] A.G. Barto and R.S. Sutton. *Reinforcement Learning: an introduction*. MIT Press, Cambridge, MA, 1998.

[BV01a] M. Bowling and M. Veloso. Convergence of Gradient Dynamics with a Variable Learning Rate. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pages 27–34, 2001.

[BV01b] M. Bowling and M. Veloso. Rational and Convergent Learning in Stochastic Games. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1021–1026, 2001.

[CB98] C. Claus and C. Boutilier. The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In *Proceedings*

of the Fifteenth National Conference on Artificial Intelligence, pages 746–752. AAAI Press, 1998.

[CHK04a] Y.H. Chang, T. Ho, and L.P. Kaelbling. All learning is local: Multi-agent learning in global reward games. *Advances in neural information processing systems*, 16, 2004.

[CHK04b] Y.H. Chang, T. Ho, and L.P. Kaelbling. Mobilized ad-hoc networks: A reinforcement learning approach. In *Proceedings of the 1st International Conference on Autonomic Computing*, 2004.

[Cro73] J.G. Cross. A stochastic learning model of economic behavior. *The Quarterly Journal of Economics*, pages 239–266, 1973.

[DBT00] M. Dorigo, E. Bonabeau, and G. Theraulaz. Ant algorithms and stigmergy. *Future Gener Comput Syst*, 16(8):851–871, 2000.

[DCD98] G. Di Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9(2):317 – 365, 1998.

[DHVN09] Y-M. De Hauwere, P. Vrancx, and A. Nowé. Learning what to observe in multi-agent systems. In *The 21st Benelux Conference on Artificial Intelligence*, Eindhoven, The Netherlands, 2009.

[dJGH+04] H. de Jong, J.C. Gouze, C. Hernandez, M. Page, T. Sar, and J. Geiselmann. Qualitative simulation of genetic regulatory networks using piecewise-linear models. *Bull Math Biol.*, 66(2):301–340, 2004.

[dJT08] S. de Jong and K. Tuyls. Learning to cooperate in public-goods interactions. In *Proceedings of the 6th European workshop on Multi-Agent Systems (EUMAS 2008)*, Bath, UK, 2008.

[dJTV08] S. de Jong, K. Tuyls, and K. Verbeeck. Fairness in multi-agent systems. *Knowledge Engineering Review*, 23(2):153–180, 2008.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26(1):29–41, 1996.

[DS04] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Books, 2004.

[DVN10] Y-M. De Hauwere, P. Vrancx, and A. Nowé. Learning Multi-Agent State Space Representations. In *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 715–722, Toronto, Canada, 2010.

[EGW06] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(1):503, 2006.

[FS99] Ernst Fehr and Klaus M Schmidt. A theory of fairness, competition, and cooperation. *Quarterly journal of Economics*, pages 817–868, 1999.

[FVV97] J.A. Filar, K. Vrieze, and OJ Vrieze. *Competitive Markov decision processes*. Springer Verlag, 1997.

[FY06] D.P. Foster and H.P. Young. Regret testing: learning to play Nash equilibrium without knowing you have an opponent. *Theoretical Economics*, 1(3):341–367, 2006.

[GHS03] A. Greenwald, K. Hall, and R. Serrano. Correlated Q-learning. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 242–249, 2003.

[Gil57] D. Gillette. Stochastic Games with Zero Stop Probabilities. *Ann. Math. Stud*, 39:178–187, 1957.

[Gin00] H. Gintis. *Game theory evolving*. Princeton University Press, 2000.

[GK73] L. Glass and S.A. Kauffman. The logical analysis of continuous non-linear biochemical control networks,. *J.Theor.Biol.*, 39(1):103–129, 1973.

[GL07] F. Germano and G. Lugosi. Global Nash convergence of Foster and Young's regret testing. *Games and Economic Behavior*, 60(1):135–154, 2007.

[GLP02] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *Proceedings of the ICML-2002 The Nineteenth International Conference on Machine Learning*, pages 227–234, 2002.

[Gra59] P.P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1):41–80, 1959.

[HK66] A. J. Hoffman and R. M. Karp. On Nonterminating Stochastic Games. *Management Science*, 12(5):359–370, 1966.

[HM99] O. Holland and C. Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artificial Life*, 5(2):173–202, 1999.

[HMC01] S. Hart and A. Mas-Colell. A reinforcement procedure leading to correlated equilibrium. *Economic Essays: A Festschrift for Werner Hildenbrand*, pages 181–200, 2001.

[How60] R.A. Howard. *Dynamic Programming and Markov Processes*. Technology Press of Massachusetts Institute of Technology, 1960.

[HSD04] M.W. Hirsch, S. Smale, and R.L. Devaney. *Differential equations, dynamical systems, and an introduction to chaos*. Academic Press, 2004.

[HSS79] J. Hofbauer, P. Schuster, and K. Sigmund. A note on evolutionary stable strategies and game dynamics. *Journal of Theoretical Biology*, 81(3):609, 1979.

[HTR09] D. Hennes, K. Tuyls, and M. Rauterberg. State-coupled replicator dynamics. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 789–796. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[HW98] J. Hu and M.P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, volume 242, page 250, 1998.

[HW03] J. Hu and M.P. Wellman. Nash Q-learning for general-sum stochastic games. *The Journal of Machine Learning Research*, 4:1039–1069, 2003.

[JSW98] Nicholas R Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38, 1998.

[Kon03a] V. Kononen. Asymmetric multiagent reinforcement learning. In *IEEE/WIC International Conference on Intelligent Agent Technology, 2003. (IAT2003)*, pages 336–342, 2003.

[Kon03b] V. Kononen. Gradient based method for symmetric and asymmetric multiagent reinforcement learning. *Lecture notes in computer science*, pages 68–75, 2003.

[KtHBV05] J.R. Kok, P.J. 't Hoen, B. Bakker, and N. Vlassis. Utile coordination: Learning interdependencies among cooperative agents. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 29–36, 2005.

[KTTP08] M. Kaisers, K. Tuyls, F. Thuijsman, and S. Parsons. Auction analysis by normal form game approximation. In *Proceedings IEEE conference on Intelligent Agent Technology (IAT2008)*, Sydney, Australia, 2008.

[KV04] J.R. Kok and N. Vlassis. Sparse Cooperative Q-learning. In *Proceedings of the International Conference on Machine Learning*, pages 481–488. ACM, 2004.

[LCK95] M. Littman, A. Cassandra, and L.P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings International Conference on Machine Learning*, pages 362–370, 1995.

[Li03] J. Li. *Learning average reward irreducible stochastic games: analysis and applications*. PhD thesis, University of South Florida, 2003.

[Lit94] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann, 1994.

[Lit01a] M. Littman. Friend-or-foe Q-learning in general-sum games. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 322–328. Morgan Kaufmann, 2001.

[Lit01b] M. Littman. Value-function reinforcement learning in Markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.

[LR00] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proc. 17th International Conf. on Machine Learning*, 2000.

[LRD07] J. Li, K. Ramachandran, and T.K. Das. A Reinforcement Learning (Nash-R) Algorithm for Average Reward Irreducible Stochastic Games. Under Review, Journal of Machine Learning Research, 2007.

[Mah96] S. Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22(1):159–195, 1996.

[Man88] B. Manderick. Basic notions in and applications of the theory of differential dynamical systems. Technical Report AIMEMO8 8-4, Artificial Intelligence lab, Vrije Universiteit Brussel, Brussels, Belgium, 1988.

[MM88] F. Moyson and B. Manderick. The Collective Behaviour of Ants: an Example of Self-Organisation in Massive Parallelism. In *Proceedings of the AAAI Spring Symposium on Parallel Models of Intelligence*, Stanford, California, 1988. AAAI.

[MS02] S. Mannor and N. Shimkin. The Steering Approach for Multi-Criteria Reinforcement Learning. *Advances in Neural Information Processing Systems*, 2:1563–1570, 2002.

[MV09] F.S. Melo and M. Veloso. Learning of Coordination: Exploiting Sparse Interactions in Multiagent Systems. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.

[Nas50] J.F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, pages 48–49, 1950.

[Nor98] J Noris. *Markov Chains. Cambridge series in statistical and probabilistic mathematics*. Cambridge University Press, 1998.

[NPV01] A Nowé, J. Parent, and K. Verbeeck. Social Agents Playing a Periodical Policy. In *Proceedings of the 12th European Conference on Machine Learning*, volume LNAI 2168, pages 382 – 393, Freiburg, Germany, 2001. Springer-Verlag.

[NT89] K.S. Narendra and M.A.L. Thathachar. *Learning automata: an introduction*. Prentice-Hall, NJ, USA, 1989.

[NV99] A Nowé and K. Verbeeck. Formalizing the Ant Algorithms in term of Reinforcement Learning. In *LNAI: Proceedings of the 5th European Conference on Artificial life*, volume 1674, pages 616 – 620, Lausanne, Switserland, 1999. Springer-Verlag.

[NVV04] A Nowé, K. Verbeeck, and P. Vrancx. Multi-type Ant Colony: The Edge Disjoint Paths Problem. *Lecture Notes in Computer Science: ANTS 2004*, 3172:202 – 213, 2004.

[OR99] M.J. Osborne and A. Rubinstein. *A course in game theory*. MIT press, 1999.

[Pee08] M. Peeters. *Solving Multi-Agent Sequential Decision Problems Using Learning Automata*. PhD thesis, Computational Modeling Lab, Vrije Universiteit Brussel, Belgium, 2008.

[PKKM00] L. Peshkin, K. Kim, L.P. Kaelbling, and N. Meuleau. Learning to cooperate via policy search. In *Uncertainty in Artificial Intelligence*, pages 489–496. Morgan Kaufmann, 2000.

[PL04] L. Panait and S. Luke. A pheromone-based utility model for collaborative foraging. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems. (AAMAS 2004)*, pages 36–43, New York, NY, USA, 2004. ACM.

[PM97] M.D. Pendrith and M.J. McGarity. An analysis of non-Markov automata games: Implications for reinforcement learning. Technical Report UNSW-CSE-TR-9702, School of Computer Science and Engineering, University of New South Wales, Sydney, Australia, 1997.

[PM98] M.D. Pendrith and M.J. McGarity. An analysis of direct reinforcement learning in non-Markovian domains. In *Proc. 15th International Conf. on Machine Learning*, pages 421–429, 1998.

[PTL08] L. Panait, K. Tuyls, and S. Luke. Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *The Journal of Machine Learning Research*, 9:423–457, 2008.

[PTRD08] M. Ponsen, K. Tuyls, J. Ramon, and K. Driessens. The Evolutionary Dynamics of Poker. In *Proceedings European Conference on Complex Systems, ECCS'08*, 2008.

[Put94] M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[RGK09] E. Rodrigues Gomes and R. Kowalczyk. Dynamic analysis of multiagent Q-learning with epsilon-greedy exploration. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 369–376, New York, NY, USA, 2009. ACM.

[Sam98] L. Samuelson. *Evolutionary games and equilibrium selection*. The MIT Press, 1998.

[Sch93] A. Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth International Conference on Machine Learning*, volume 298305, 1993.

[Sha53] L.S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953.

[SJJ94] S.P. Singh, T. Jaakkola, and M.I. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of the Eleventh International Conference on Machine Learning*, 1994.

[SKM00] S. Singh, M. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 541–548, 2000.

[Smi82] J.M. Smith. *Evolution and the Theory of Games*. Cambridge University Press, 1982.

[Sob71] M.J. Sobel. Noncooperative Stochastic Games. *The Annals of Mathematical Statistics*, 42(6):1930–1935, 1971.

[SPG07] Y. Shoham, R. Powers, and T. Grenager. If Multi-Agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377, 2007.

[SPS99] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

[SPT94] P.S. Sastry, V.V. Phansalkar, and M.A.L. Thathachar. Decentralized learning of Nash equilibria in multi-person stochastic games with incomplete information. *IEEE Transactions on Systems, Man and Cybernetics*, 24(5):769–777, 1994.

[SSH94] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *Proceedings of the National Conference on Artificial Intelligence*, pages 426–426. Wiley & Sons, 1994.

[Str01] S.H. Strogatz. *Nonlinear dynamics and chaos: With applications to physics, biology, chemistry, and engineering*. Perseus Books, 2001.

[TB95] G. Theraulaz and E. Bonabeau. Modelling the collective building of complex architectures in social insects with lattice swarms. *Journal of Theoretical Biology*, 177(4):381–400, 1995.

[TB99] G. Theraulaz and E. Bonabeau. A brief history of stigmergy. *Artificial Life*, 5(2):97–116, 1999.

[Tes04] G. Tesauro. Extending Q-learning to general adaptive multi-agent systems. *Advances in neural information processing systems*, 16, 2004.

[tHT04] P.J. 't Hoen and K. Tuyls. Analyzing Multi-agent Reinforcement Learning Using Evolutionary Dynamics. In *15th European Conference on Machine Learning (ECML04), LNAI Volume 3201*, pages 168–179, Pisa,Italy, 2004. Springer.

[TP95] M.A.L. Thathachar and V.V. Phansalkar. Learning the global maximum with parameterized learning automata. *IEEE Transactions on Neural Networks*, 6(2):398–406, 1995.

[TS04] M.A.L. Thathachar and P.S. Sastry. *Networks of learning automata: Techniques for online stochastic optimization*. Kluwer Academic Publishers, 2004.

[Tse61] ML Tsetlin. On the behavior of finite automata in random media. *Automation and Remote Control*, 22(10):1210–1219, 1961.

[Tsi94] J.N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202, 1994.

[TtHV06] K. Tuyls, P.J. 't Hoen, and B. Vanschoenwinkel. An evolutionary dynamical analysis of multi-agent learning in iterated games. *Autonomous Agents and Multi-Agent Systems*, 12:115–153, 2006.

[Tuy04] K. Tuyls. *Learning in multi-agent systems: An evolutionary game theoretic approach*. PhD thesis, Vrije Universiteit Brussel, Brussels, Belgium, 2004.

[TVL03] K. Tuyls, K. Verbeeck, and T. Lenaerts. A selection-mutation model for q-learning in multi-agent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 693–700. ACM New York, NY, USA, 2003.

[TW09] K. Tuyls and R. Westra. Replicator Dynamics in Discrete and Continuous Strategy Spaces . In *Multi-Agent Systems: Simulation and Applications*, pages 215–240. CRC Press, 2009.

[Ver04] K. Verbeeck. *Coordinated Exploration in Multi-Agent Reinforcement Learning*. PhD thesis, Computational Modeling Lab, Vrije Universiteit Brussel, Belgium, 2004.

[VK01] P. Valckenears and M. Kollingbaum. Multi-agent coordination and control using stigmergy applied to manufacturing control. *Mutli-agents systems and applications*, pages 317–334, 2001.

[VN02] K. Verbeeck and A. Nowé. Colonies of learning automata. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 32 (6):772 – 780, 2002.

[VNP04] K. Verbeeck, A. Nowé, and M. Peeters. Multi-agent coordination in tree structured multi-stage games. In *Proc. 4th Symposium on Adaptive Agents and Multi-agent Systems (AISB 2004)*, volume 2, pages 63–74, Leeds,UK, 2004.

[VNPT07] K. Verbeeck, A. Nowé, J. Parent, and K. Tuyls. Exploring selfish reinforcement learning in repeated games with stochastic rewards. *Autonomous Agents and Multi-Agent Systems*, 14(3):239–269, 2007.

[VNVP08] K. Verbeeck, A. Nowé, P. Vrancx, and M. Peeters. Multi-automata learning. In C. Weber, M. Elshaw, and N.M. Mayer, editors, *Reinforcement Learning: Theory and Applications*, pages 167–185. I-Tech Education and Publishing, 2008.

[vS34] H. von Stackelberg. *Marktform und Gleichgewicht*. Julius Springer, Vienna, 1934.

[VTWN08] P. Vrancx, K. Tuyls, R. Westra, and A. Nowé. Switching dynamics of multi-agent learning. In *International joint conference on Autonomous Agents and Multi-Agent Systems (AAMAS'08)*, 2008.

[VVN07a] K. Verbeeck, P. Vrancx, and A. Nowé. Networks of learning automata and limiting games. In K. Tuyls, S. de Jong, M. Ponsen, and K. Verbeeck, editors, *Adaptive Learning Agents and Multi-Agent Systems Workshop 2007*, pages 171–183, Maastricht, The Netherlands, 2007. MICC/IKAT. ISSN 0922-8721, number 07-04.

[VVN07b] P. Vrancx, K. Verbeeck, and A. Nowé. Analyzing stigmergetic algorithms through automata games. *Lecture Notes in BioInformatics: Knowledge Discovery and Emergent Complexity in Bioinformatics*, 4366:145–156, 2007.

[VVN07c] P. Vrancx, K. Verbeeck, and A. Nowé. Limiting games of multi-agent multi-state problems. In K. Turner, S. Sen, and L Panait, editors, *Workshop on Adaptive and Learning Agents 2007, 6th International Conference on Autonomous Agents and Multi-Agents Systems (AAMAS 2007)*, Hawaii, USA, 2007.

[VVN07d] P. Vrancx, K. Verbeeck, and A. Nowe. Optimal Convergence in Multi-Agent MDPs. *Lecture Notes in Computer Science: KES 2007*, 4694:107–114, 2007.

[VVN08a] P. Vrancx, K. Verbeeck, and A. Nowe. Decentralized learning in Markov games. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 38(4):976–981, 2008.

[VVN08b] P. Vrancx, K. Verbeeck, and A. Nowé. Networks of learning automata and limiting games. *Lecture Notes in Artificial Intelligence, ALAMAS III*, 4865:224–238, 2008.

[VVNar] P. Vrancx, K. Verbeeck, and A. Nowé. Analyzing the dynamics of stigmergetic interactions through pheromone games. *Theoretical Computer Science. Special Issue on Swarm Intelligence: Theory*, To Appear.

[WD92] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[Wei97] J.W. Weibull. *Evolutionary game theory*. The MIT press, 1997.

[Wei99] Gerhard Weiss. Multiagent systems: A modern approach to distributed artificial intelligence. *artificial intelligence*, 1999.

[Wil92] R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[Wit77] I.H. Witten. An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34(4):286–295, 1977.

[WJN86] R. Wheeler Jr and K.S. Narendra. Decentralized learning in finite Markov chains. *IEEE Transactions on Automatic Control*, 31(6):519–526, 1986.

[Woo09] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[WR04] M. Weinberg and J.S. Rosenschein. Best-response multiagent learning in non-stationary environments. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 2, pages 506–513, Washington, DC, USA, 2004. IEEE Computer Society.

[WS03] X.F. Wang and T. Sandholm. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. *Advances in neural information processing systems*, 15:1571–1578, 2003.

[ZGL06] M. Zinkevich, A. Greenwald, and M. Littman. Cyclic equilibria in Markov games. *Advances in Neural Information Processing Systems*, 18:1641, 2006.

[Zin03] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Machine Learning International Conference-*, volume 20(2), page 928, 2003.

Index

Author Index