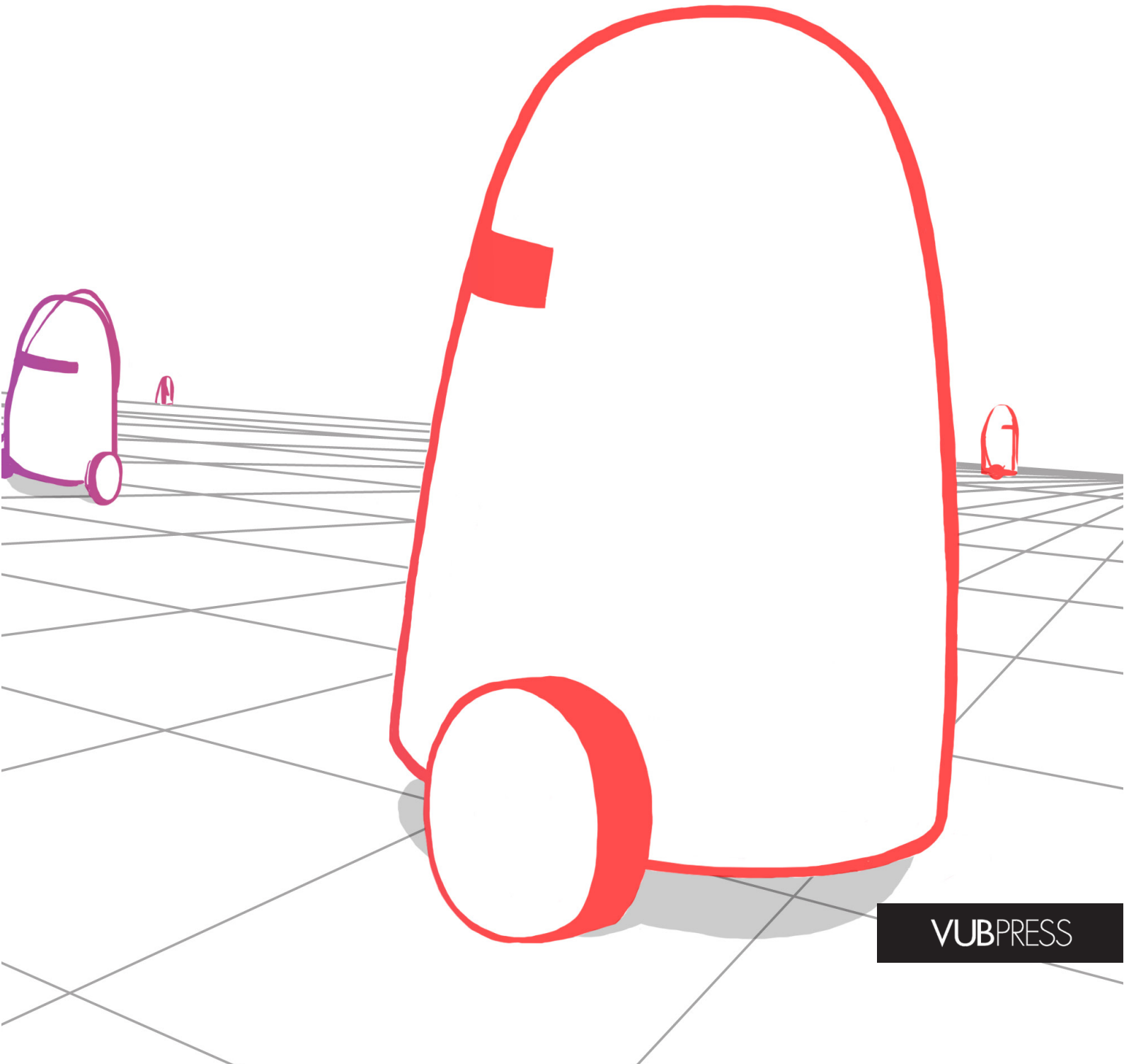
Sparse Interactions in Multi-Agent Reinforcement Learning

Yann-Michaël De Hauwere

Supervisor: Prof. Dr. Ann Nowé



VUBPRESS

Vrije Universiteit Brussel

Faculty of Sciences
Department of Computer Science
Computational Modeling Lab

Sparse Interactions in Multi-Agent Reinforcement Learning

Yann-Michaël De Hauwere

Dissertation Submitted for the degree of Doctor of Philosophy in Sciences

Supervisor: Prof. Dr. Ann Nowé

Print: Silhouet, Maldegem

Cover design by Pjotr Cornelis

2011 Uitgeverij VUBPRESS Brussels University Press
VUBPRESS is an imprint of ASP nv (Academic and Scientific Publishers nv)
Ravensteingalerij 28
B-1000 Brussels
Tel. ++32 (0)2 289 26 50
Fax ++32 (0)2 289 26 59
E-mail: info@vubpress.be
www.vubpress.be

ISBN 978 90 5487 920 6
NUR 984
Legal Deposit D/201111.161/083

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).

Angels fly so high as to be beyond our sight,
but they are always there, looking out for us

Committee Members:

Prof. Dr. Ann Nowé
Vrije Universiteit Brussel
(supervisor)

Prof. Dr. Olga De Troyer
Vrije Universiteit Brussel
(Commitee chair)

Prof. Dr. Bernard Manderick
Vrije Universiteit Brussel

Prof. Dr. Luc Steels
Vrije Universiteit Brussel

Prof. Dr. Bram Vanderborght
Vrije Universiteit Brussel

Dr. Karl Tuyls
Maastricht University

Dr. Marco Wiering
University of Groningen

Samenvatting

Reinforcement leertechnieken (RL) werden reeds succesvol toegepast in heel uiteenlopende onbekende omgevingen met een hoge graad van onzekerheid. Ook voor domeinen waarin meerdere agenten actief zijn, is RL een interessant paradigma. In dergelijke domeinen zijn er echter ook verschillende nieuwe problemen. De agenten handelen autonoom en hebben mogelijk tegenstrijdige doelstellingen die zij wensen te bereiken. Een voor de hand liggende oplossing is om de agenten te voorzien van de nodige informatie over de toestand van de andere agenten, samen met hun handelingen en beloningen. Dit laat de agenten toe om equilibriumpunten in de toestandsruimte te leren, maar brengt een hoge kost met zich mee. De agenten leren in de gecombineerde toestand-actie ruimte wat, gezien de grootte van deze ruimte, het leerproces aanzienlijk vertraagt.

In deze doctoraatsverhandeling argumenteren wij dat in situaties waar de agenten maar zelden met elkaar interageren, het interessanter is voor het leerproces om de agenten onafhankelijk van elkaar te laten leren en elkaar enkel in rekening te brengen als dit nodig blijkt. In de toestanden waar agenten elkaar niet beïnvloeden zijn de *toestandovergangsfunctie* en de *beloningsfunctie* onafhankelijk van de toestand of actie van de andere agenten in het systeem. In deze situatie kunnen de andere agenten in de omgeving genegeerd worden en een RL techniek voor één enkele agent kan toegepast worden. Wanneer deze vereiste over de onafhankelijk echter niet geldt, hebben we te maken met een multi-agent coördinatie probleem en is een multi-agent leertechniek vereist. Een belangrijke vraag hierbij is hoe gedetecteerd kan worden wanneer de agenten interageren.

In deze thesis introduceren wij nieuwe leertechnieken die dergelijke interacties kunnen detecteren en, gebruik makend van deze informatie, ofwel een onafhankelijke leertechniek gebruiken, ofwel een multi-agent techniek. Het eerste algoritme, genaamd *2Observe*, maakt gebruik van de ruimtelijke relatie die bestaat in de toestandsruimte om de verzameling van toestanden waarin agenten elkaar beïnvloeden

te leren. Deze techniek is gebaseerd op generalized learning automata en kan deze relatie benaderen. De tweede techniek, dat we *CQ-learning* noemen, gebruikt de onmiddellijke beloningen die een agent krijgt om de invloed van andere agenten in bepaalde toestanden te detecteren. Door het uitvoeren van statistische test op significante verschillen tussen deze beloningen is het mogelijk om de relevante toestandsinformatie van andere agenten in interacties te bepalen. Het laatste algoritme, *FCQ-learning*, is een uitbreiding van dit idee, maar laat toe om coördinatieproblemen te anticiperen, verschillende tijdsstappen voor deze de kop op steken, en hierop te reageren. Dit resultaat wordt bereik door de statistische tests uit te voeren op de som van de onmiddellijke en toekomstige beloningen.

Ten slotte demonstreren we ook hoe, gebruik makend van 2Observe en CQ-learning, ervaring omtrent coördinatieproblemen kan veralgemeend worden en gedeeld worden tussen agenten en omgevingen. Deze methoden zijn de eerste technieken die dergelijke kennis kunnen doorgeven in multi-agent systemen.

Abstract

Reinforcement learning has already been widely used in unknown domains with a high degree of uncertainty. Also for domains in which multiple agents are acting together it is an interesting paradigm. In these domains however several additional problems arise. Agents behave autonomously and might have conflicting goals. A straightforward approach is to allow agents to always observe the state information of the other agents, as well as their actions and rewards they receive. This allows the agents to learn to reach equilibrium points in the environment, but it comes at a high cost. Agents are learning in the joint state-action space which considerably slows down the learning process.

In this dissertation we argue that in settings where the interactions between agents are sparse, an efficient learning approach is to allow the agents to learn individually and only take into account the other agents when necessary. In the former case, agents are not influencing each other in a particular state. Hence, the state transition function and the reward function are independent of the state and action of any other agent acting in the environment. In this case, the learning can be reduced to single agent reinforcement learning and the agent can safely ignore the other agents in the environment. In the latter case, when this independency requirement does not hold, we are dealing with a multi-agent coordination problem and a multi-agent learning approach is required. A key question is how to determine when interaction occurs.

We propose novel approaches which are capable of learning in which states such sparse interactions occur and based on this information use either a single agent approach or a multi-agent approach. The first algorithm, called 2Observe, exploits spatial dependencies that exist in the joint state space to learn the set of states in which sparse interactions occur. This approach is based on generalised learning automata that can approximate these dependencies in the state space. The second algorithm, called CQ-learning, uses the immediate reward signal to determine the

influence of other agents in certain states. By performing statistical tests on these immediate rewards, the relevant state information of other agents during sparse interactions can be determined. The last algorithm, called FCQ-learning, extends on this idea, but also allows to anticipate coordination issues, several timesteps before they actually occur and as such dealing with the issue in a timely fashion. This is achieved by performing the statistical tests on the sum of immediate and future rewards.

Finally, we also introduce some methods to generalise knowledge about coordination problems and demonstrate how experience can be shared between agents and environments using 2Observe and CQ-learning. These methods are the first in their kind to provide knowledge transfer about coordination experience in multi-agent systems.

Acknowledgments

Jaren onderzoek, gevolgd door maanden van het schrijven van deze thesis is alleen maar mogelijk met de aanmoediging en steun van familie, collega's en vrienden.

Als eerste wens ik mijn promotor, Ann Nowé te bedanken. Zij bood me de kans aan om dit onderzoek aan te vangen. Haar wetenschappelijke begeleiding en haar beeld van een doctoraat hebben in grote mate bijgedragen tot dit onderzoek. Bedankt Ann, om me de mogelijkheid te geven om naar conferenties te gaan en me mijn eigen onderwerp te laten vinden. Voornamelijk AAMAS in 2009, dat echt wel een keerpunt is geweest in mijn onderzoek.

I would like to thank the members of the examination committee Olga De Troyer, Bernard Manderick, Luc Steels, Bram Vanderborght, Karl Tuyls and Marco Wiering for finding the time to participate in this dissertation and for their insightful comments and suggestions.

The members of the Computational Modeling Lab whom I have had the honour and pleasure to work with, also deserve a special word of gratitude. Thank you for making CoMo such a nice environment! Whether it is with research, Wii, ping-pong, a nice BBQ, letting me observe the strange game of Magic or just have a nice chat, I thank you for the fun atmosphere: Abdel, Allan, Anne, Bert, Cosmin, David, Other David, Emily, Feng, Jonatan, Katja, Kris, Maarten, Matteo, Mihail (Mike ;-)), Mohamed, Nyree, Ruben, Pasquale, Saba, Sven, Steven, Yailen, Yann-Aël and Yifei.

Stijn and Peter. Ik ben jullie niet vergeten. Al het bovenstaande geldt ook voor jullie. Peter, bedankt voor het nalezen en meewerken aan het onderzoek in dit doctoraat en voor de leuke momenten op de conferenties waar we samen naartoe zijn geweest. Stijn, heel erg bedankt voor het nalezen van dit doctoraat. Ondanks je vrij hoge workload heb ik zelden meegemaakt dat iemand zoveel hoofdstukken op zo

korte tijd kon nalezen. Bedankt voor de grammaticale en inhoudelijke opmerkingen!

Ook buiten de VUB zijn er verschillende mensen die ik dankbaar ben. Maarten, bedankt voor vele uren judo die we dit laatste jaar samen hebben beleefd met dat prachtige resultaat aan het einde. Hoewel het bij momenten afzien was, deed het deugd om even weg te zijn van de lerende agenten! Yana, heel erg bedankt voor je steun in die momenten waar de werkdruk erg hoog lag. Bedankt om er te zijn!

Ten slotte wens ik mijn familie te bedanken. Voor hun steun en voor hun goede wil om mijn onderzoek te begrijpen. Voor naar me te luisteren op die momenten dat ik voornamelijk tegen mezelf aan het babbelen was en probeerde mijn eigen resultaten te begrijpen. Mijn zus verdient hier ook een speciaal bedankje voor haar eindeloos enthousiasme in het mee-organiseren van mijn doctoraatsreceptie! En uiteraard bedank ik mijn familie om me aan te moedigen om aan dit doctoraat te beginnen. Ik heb er geen seconde spijt van gehad.

Bedankt!!!

Brussels, June 2011
Yann-Michaël

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Artificial Intelligence, IT'S HERE.
– Business Week cover, July 9, 1984 –

The research presented in this dissertation is concerned with the application of reinforcement learning in domains where multiple agents are acting. In this chapter we will explain on a high level what agents are, what reinforcement learning is about and how these are connected. Next we will focus on the additional problems that arise when such agents are acting in the same environment and influencing each other. This will lead us to the problem statement of this dissertation. Finally, we give an outline of the research presented in the different chapters of this dissertation.

1.1 Agents in their environment

Everyone will agree upon the fact that computers are capable of performing complex calculations or solving planning problems, such as trajectory planning, much faster than a human ever will. But for a computer system being able to perform these operations, a programmer implemented the desired functionality the system is supposed to possess [Wooldridge (1999)]. However, despite many hours of debugging and testing, it is impossible to foresee every possible parameter to which the system may be exposed. Whenever such unanticipated situations occur, the application will most often not respond as desired and possibly even malfunction. Hence it is logical to create computer systems that have a higher degree of *autonomy* and are capable of *decision making* without human intervention. Such computer systems are called *agents*.

The field of Artificial Intelligence (AI) is broad and agents are being used widely throughout it. Although being such a widespread term, no general consensus exist about what exactly constitutes an agent. The definition by Jennings et al. represents best the view about an agent that is adopted in this dissertation:

Definition 1. *[Jennings et al. (1998)]: An agent is a computer system that is situated in some environment, and that is capable of autonomous action taking in this environment in order to meet its design objectives.*

This definition is very open in the sense that it does not define precisely what the environment should look like, how autonomous action should be viewed or what the design objectives of the agent are. This definition however captures all the requirements to which the notion of an agent used throughout this dissertation must satisfy. An agent is observing the environment and acting upon these observations through which it can change the environment. It does so in a closed loop with the environment, as shown in Figure 1.1, and attempts to accomplish its design objectives.

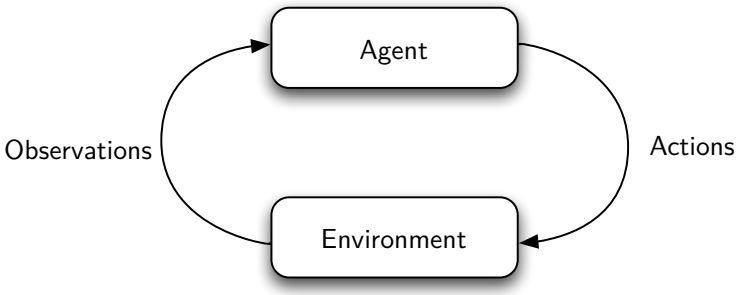


Figure 1.1: Agent in a loop performing actions in its environment, based on observations it has from this environment.

These design objectives could go from something very simple, such as keeping the temperature in a car constant at 22° Celsius. The observation the agent has about the environment is the current temperature inside the car. Its actions are to lower or increase the climate control. Or these objectives could be very complex, such as personal recommendations on online shopping sites.

The environment described above for controlling the temperature in a car is a very simple one. A much more complex is a nuclear reactor which has to be kept within its safe operating temperature range. Both environments are concerned with the temperature, but are very different in terms of parameters, observation and control. In [Russel & Norvig (2010)] the following axes to classify environments along are proposed:

- *Observability*
  An agent can have complete access to all the information about the state the

system is in at each point in time. Such systems are called fully observable. Alternatively, certain required data cannot be measured because there are no sensors available, or these are malfunctioning. Such environments are called partially observable.

- *Number of agents*
  This relates to the number of agents that are acting in the environment. A computer program solving Sudoku is an example of a single-agent environment, whereas a computer program solving the game of chess is doing so in a multi-agent environment. On a much higher scale, freight logistics are also an example of a huge and very complex multi-agent system. Other 'objects' in the environment are considered to be agents if they attempt to maximise some performance measure.

- *Agents' intentions*
  Agents act in their environment in order to complete their design objectives. If these objectives are the same for all agents, the environment is said to be cooperative. If however an agent attempts to optimise its own performance measure and minimises the performance of the other agents, the environment has a competitive nature.

- *Transition probabilities*
  If the next state of the system is completely determined by the current state and the action executed by the agent, the environment is deterministic. If these transitions occur with some measurable probability, the environment is said to be stochastic. If, in an extreme case, these transitions do not have any probabilities attached to them, the environment is said to be non-deterministic.

- *Execution flow*
  In episodic environments, there is no dependency between the episodes. This means that the actions taken by an agent are not affected by its actions in the previous episode. Alternatively, the system can be sequential. In such systems, the current decision could affect all future decisions.

- *Evolution of the environment*
  If the environment does not change while the agent is deliberating over which action to take, it is called a static environment. A more complex situation, in which the environment does change and hence the speed at which the agents have to select an action becomes important, is called a dynamic environment.

- *State information*
  The information the agent has about the environment could be discrete or continuous. Similar, the time could advance continuously or in fixed time

steps and the actions the agent has to perform could also be continuous or could be an action from a predetermined discrete set of actions.

- *Knowledge*
  This refers to the knowledge the agent has about the dynamics of the environment. Is the agent aware of the outcome of performing a particular action, or is the environment unknown and does the agent have to learn the effect of its actions?

Before explaining our learning approach to multi-agent systems, we will first describe the assumptions of the research presented throughout this dissertation. The environments used in this dissertation are partially observable. Multiple agents are acting in the same initially unknown environment and agents do not have all the information about the other agents. The observations the agents have about the current state the system is in, are exact. Similar, if agents do observe each other, these observations are exact i.e. we do not consider possibly noisy observations. The transitions the agents experience are deterministic. However, since agents do not always have full information about each other, some transitions may seem non-deterministic to the agents if agents are influencing each other 's actions. The testbed we used in our experiments are gridworld environments in which agents might have a different goal location. This would indicate that agents have conflicting interests. However, the goal is also to avoid collisions between agents. Hence, our setting is partially competitive and partially cooperative. In order to reach their respective goals, agents have to take a sequence of actions. As we will explain shortly, the agents do this repeatedly, so our setting is both sequential as episodic. Finally, agents are learning in discrete state spaces and select one of their actions from a discrete set at every timestep.

So far, we have only considered agents using a simple mechanism to act, based on the observations. In a sense, the agent responsible for the climate control in a car, is nothing more than a simple if-then-else statement:

Algorithm 1 Climate control agent

```
1: Observe current temperature t
2: if t ≤ 22° then
3:     Increase temperature of climate control
4: else
5:     Decrease temperature of climate control
6: end if
```

In the next section we will amplify the notion of autonomous agents, by explaining the requirements which agents must satisfy in order to be called *intelligent agents*.

1.2 Intelligent agents

> There is a popular cliché... which says that you cannot get out of computers any more than you put in. Other versions are that computers only do exactly what you tell them to, and that therefore computers are never creative. The cliché is true only in the crashingly trivial sense, the same sense in which Shakespeare never wrote anything except what his first schoolteacher taught him to write –words.
>
> [Dawkins (1986)]

This section is loosely based on the overview paper of Wooldridge and Jennings [Wooldridge & Jennings (1995)] and on the book chapter by Wooldridge [Wooldridge (1999)] in [Weiss (1999)]. In these works, an agent is identified as an entity capable of *flexible* autonomous actions in order to meet its design objectives. This flexibility can stand for three things:

- *reactivity:* The ability to respond in a timely fashion to changes that occur in the environment the agent perceives, in order to satisfy its design objectives.
- *pro-activeness:* The ability to take initiatives in order to achieve the goals of its design objectives.
- *social ability:* The ability to interact with other agents and possibly humans, in order to accomplish its design objectives

These three properties are all intended to accomplish the design requirements of the agent. Pro-activeness is the most straight forward of these properties. Every program, every procedure a programmer writes is intended to accomplish some post-conditions, based on some pre-conditions. This pro-activeness assumes that the pre-conditions remain valid and do not change until the post-conditions are reached. This requirement is reasonable for static and fully known environments, but in complex environments, where multiple agents are acting simultaneously, it is usually not met. In such environments, an agent must react continuously to changes in the environment or to the behaviour of other agents, while still accomplishing its goal. It is clear that a good trade-off between these two objectives is necessary. The final property of an intelligent agent is the capability of understanding the goals of other agents and acting on this understanding in order to reach fruitful cooperation.

Multi-agent systems (MAS) are a commonly accepted approach for modeling dynamical systems in which several agents are interacting because control and/or data is decentralised. Jennings et al. define a MAS as follows:

Definition 2. *A **Multi-Agent System** is a loosely coupled network of agents that cooperate or compete with the goal of solving problems that are beyond the individual capabilities of each agent*

In this dissertation we are concerned with MAS, in which agents are learning which actions to take in order to reach their design objectives. The learning approach used is reinforcement learning (RL). In the next section we will explain this learning paradigm and show its advantages compared to other learning approaches. We will also demonstrate why RL agents satisfy all the requirements of an intelligent agent, as described above.

1.3 Learning from experience

Many problems currently exist that remain unsolved. Not because computers do not have enough computational power or memory to solve them, but because it is not feasible to determine what the program should do due to the complexity of the problem. Imagine for instance an automated manufacturing system. In such systems, many parameters exist to control the flow of the manufacturing process. Moreover, many of these parameters may even be unknown to the people designing an agent to solve such systems, so calculating the optimal solution may be impossible. Machine learning (ML) is an interesting paradigm for learning a solution to such problems. Tom Mitchell defined this paradigm as follows [Mitchell (1997)]:

Definition 3. *A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P improves with experience E.*

ML can be divided into three categories. The first, are supervised learning approaches. These are a generalised method for approximating unknown target functions, by providing sample input-output pairs from the function to be learned. This method is a good approach for problems such as face recognition, which cannot be calculated or programmed by hand and where a human can provide correct labels for the samples. For the example of the automated manufacturing system however, the lack of examples of the target function to be learned still poses a problem. Second, we have unsupervised learning approaches which attempt to minimise some error on the output function. Clustering is a popular technique within this category, but without an error function to minimise, this is also not suitable for the manufacturing system. Finally, in between both approaches, there is reinforcement learning. RL is an approach to mitigate the issues of supervised and unsupervised approaches in certain problems. RL learns a solution through efficient use of the experience from trial-and-error interaction with the environment. An RL agent is only given a goal to achieve, based on the information of the current state

of the system. Each decision the RL agent takes, results in a reward that evaluates this decision. Hence, the goal of the agent is to optimise its long term performance, which is measured by the total reward the agent accumulates over time. A RL agent may for instance be asked to optimise the manufacturing process, where the utility is based on the amount of finished products in a given time range. This is a very intuitive way of representing problems and very similar to how humans do it.

Imagine the problem of learning a child to ride a bicycle. A parent can show many times how to do it, but explain how to react to every possible sensation one receives from riding a bike is impossible. So supervised learning approaches are unsuitable. In practice, when a child learns to ride a bicycle it does so by trial-and-error and learns (sometimes the painful way) how hard it should push on the pedals or how far to turn the handle bars. The reward signal is the time the child can remain on the bike, without having to put foot on the ground (or falling over).

A RL agent is learning, based on the current state of the environment, which actions to take (i.e. it is reactive) in order to accumulate the highest reward (i.e. it is pro-active) over time. Moreover, this reward signal can be based on the performance of a group of agents (i.e. agents have to exhibit social behaviour). In Figure 1.2 we show a schematic representation of a single reinforcement learning agent in the environment.

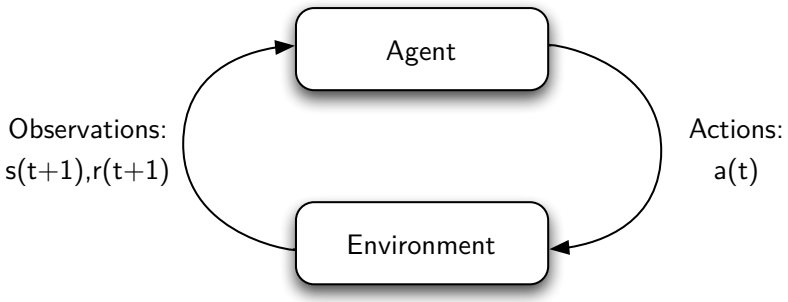


Figure 1.2: RL agent acting in an environment. For every action a(t) at timestep t it is informed of the next state of the system s(t+1) and receives a reward for its action r(t+1).

Interactions between the agent and the environments take place in discrete time steps $t = 1, 2, \ldots$. At every time step t the agent selects an action $a(t)$, which is the input of the environment. This action results in an internal change in the environment, after which the agent observes the new description of the state of the system $s(t+1)$ and the immediate reward $r(t+1)$ which is the result of its action. The agent is learning a policy, which is a mapping from each state the

system can be in to an action, in order to maximise the total reward it accumulates over time. This mapping is learned by maintaining estimates of the immediate and future rewards an agent receives for an action in a particular state. After several trials of a particular action, these estimates become more and more accurate to the true value of that action. This is why RL is often referred to as learning from experience.

In this dissertation we are concerned with multiple agents acting in the same environment. As a result, the changes of the environment at every timestep are the result of the actions of all the agents. If an agent cannot observe the actions of the other agents, it becomes difficult to learn an optimal policy, since the environment changes randomly from the viewpoint of this agent. This property leads us to the problem statement we will address in this dissertation and which is explained in the following section.

1.4 Problem statement

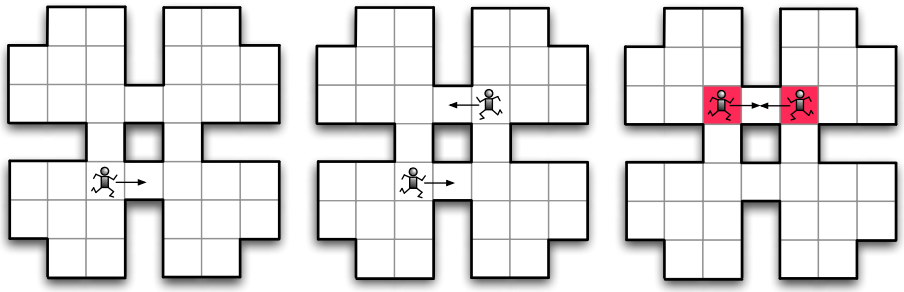
When multiple agents are acting in the same environment, the payoffs every agent receives, together with the transition the system undergoes are dependent on the states and actions of all agents. Imagine an environment with red and blue blocks where an agent is trying to stack all red blocks. At the same time, another agent's goal is to stack all blocks in the environment regardless of their colour. We will refer to the first agent as Agent 1, and to the second as Agent 2. If Agent 1 does not observe the actions of Agent 2, but only the result of them through its observation of the stack of blocks, it will loose a lot of time removing the blue blocks from the stack. Similar, Agent 2 will not understand why there are blocks being removed from the stack. Hence, the environment both agents observe is non-stationary. In this example, agents can work together and find a solution that satisfies both their design objectives. For instance, Agent 2 could wait until Agent 1 has finished stacking the red blocks, before putting the blue blocks on top. This is however only possible if agents are aware of each others' actions and intentions.

Different learning approaches exist, with regard to such set-ups. These approaches can be categorised based on the information available to the agent. If agents have full access to the entire system information, together with the actions performed by all agents and the rewards they received, it is possible to model the other agents and to estimate the rewards for the combination of actions of all agents. As such, agents can learn over these joint actions and learn a good policy. However, assuming that agents have access to all this information, induces two problems. The first one is that the system is less distributed. Agents still learn independently and select their own actions, but with a large overhead in terms of costly communication in order to provide all agents with the information about each others' actions. The second problem concerns the size of the state-action space in which

agents are learning. For the example described above, this state-action space is the combination of the possible location of the blocks, with all possible combinations of the actions of the agents. This means, that the size of the state-action space is exponential in the number of agents which considerably slows the learning process.

Other approaches attempt to solve this problem, where agents only use their own local information and select their actions independently. They only experience the presence of other agents through their rewards and through the transitions the system undergoes. From the point of view of such an agent, it may even seem that it is alone in the environment. Due to the lack of information, these approaches require careful exploration in order to reach good results [Verbeeck (2004), Vrancx (2010)].

In this dissertation we adopt a third possibility to mitigate this problem of either always observe the information of other agents or never observe it. Our goal is to introduce a flexible approach which allow agents to learn in which situations, agents *need* state information from other agents, and in which situations their own local state information is sufficient. This leverages the need for choosing only one approach which might be very slow to learn (in the case of always using full observations), or suboptimal for the problem at hand (in the case of independent learners). We represent this principle using a gridworld example in Figure 1.3. Agents have four actions at their disposal: `NORTH`, `EAST`, `SOUTH`, `WEST` which takes them one cell up, right, down or left respectively. The grid is bounded, so agents cannot exit the grid. Taking an action that would cause it to leave the grid, results in the agent remaining in the same location. Finally, each cell can only contain a single agent.



(a) Single-agent (b) Multi-agent, not interacting (c) Multi-agent, **interacting**

Figure 1.3: Sparse interactions in multi-agent environments.

In (a) we show a single agent acting in an environment. This environment is stationary, as its transitions are the result of its own actions only. Figure 1.3(b)

shows multiple agents acting in the same environment. The selected actions of the agents are indicated by arrows. Since these actions are not interfering, the agents can act independently and do not have to observe each other. The transition the agents will make is the sole product of their respective actions. In Figure 1.3(c) this is no longer the case. If agents perform their selected actions, they will interfere as it is assumed that no two agents can be in the same cell at the same time. In this situation the transition the agents experience will not be stationary since it is dependent on the location and the action of the other agent. Such situations are called *sparse interactions*. It should be clear that knowing when these sparse interactions occur, is beneficial for both the learning speed of the agent since it can learn most of the time using only local state information, and for the communication required between the agents since they only need information of each other in a small set of states.

This leads us to the research question which expresses the main motivation of the research presented throughout this dissertation:

When should agents observe the state information of other agents in order to avoid coordination problems?

We provide two approaches for answering this question. In the first one, which is presented in Chapter 4, we assume that agents can always observe each other through sensory input, even when the agents cannot influence each other. Our approach learns to what extent agents should take each other into consideration and coordinate with each other. In the second approach we argue that, when agents follow a certain policy, agents will only influence each other in certain states where their policies are interfering. In Chapter 5 we present an algorithm that is capable of learning these states and allows agents to locally adapt their policies.

1.5 Contributions

Throughout this dissertation we will show that the following contributions have been made:

- We present an overview of single agent reinforcement learning, in which we describe what the elements of a reinforcement learning problem are and the theoretical framework on which it can rely. We also provide a discussion of the most important issue in reinforcement learning, i.e. the exploration-exploitation dilemma. We give an overview of the most well-known solution methods and we discuss the additional issues, as well as benefits that come with multiple agents learning in the same environment.

- We give a taxonomy of multi-agent learning research, based on the number of strategic interactions between the agents.

- We argue that multi-agent learning research should particularly focus on strategic interactions between agents. More specifically, that agents should learn when to interact with other agents and when it is safe to ignore each other. To this end we describe a two layer framework that captures this key idea and which also represents the main intuition behind the research presented in this dissertation.

- We describe an extension to decentralised Markov decision processes which captures local interaction states. The interaction states are represented by a function which takes the current state of an agent as its input and returns the interaction states in which coordination with other agents should occur.

- Next to introducing a formal framework to capture local interactions between dependent states we also introduce an algorithm, which we call 2Observe, that learns the function to return with which agents interactions should occur.

- We show the need for sparse interactions in multi-agent reinforcement learning since both extreme approaches, i.e. never observe other agents or always observe other agents, suffer from suboptimal behaviour and/or long learning times.

- We introduce an algorithm, called CQ-learning, that learns in which states coordination is required between agents by performing statistical tests on the immediate reward signal.

- One of the most important features of RL is the capability of dealing with a delayed reward signal. Therefore, we extend CQ-learning to FCQ-learning which can detect coordination problems, several time steps ahead before this problem is reflected in the immediate reward signal. This algorithm is the first in its kind to be able to deal with a delayed reward signal using sparse interactions.

- In single agent RL the concept of transferring past experience to unseen environments has gained much interest because of the significant improvement it offers in the learning speed in these previously unseen environments. We use this concept of transfer learning within a multi-agent concept to re-use coordination experience. In a multi-agent context this experience can be re-used for other agents and across environments.

1.6 Outline of the dissertation

This section provides an overview of the research presented in this dissertation.

In Chapter 2 we present an in-depth overview of the necessary single agent reinforcement learning background needed to understand the research in later chapters. First we explain the theoretical framework used in RL research, called a Markov decision process. We then focus on action selection mechanisms and the exploration-exploitation trade-off common to many reinforcement learning problems. We also introduce model-based methods, known as dynamic programming, methods that attempt to learn a model of the transition and the reward function, as well as model-free methods. For these latter, we go in detail on Q-learning and learning automata as these algorithms will form the basis of the research presented in subsequent chapters.

In Chapter 3 we explain the concepts of RL in multi-agent settings. We present a thorough overview of several multi-agent frameworks, such as Markov games and decentralised Markov decision processes, before introducing the framework used throughout the remainder of this dissertation: the decentralised sparse interaction Markov decision process. This framework models the interactions that occur between agents in a particular set of states, under the assumption that the other agents follow their respective policies. In this chapter we also present several multi-agent learning approaches, together with the state of the art of MARL research with sparse interactions.

Next, in Chapter 4 we introduce our two layer approach to detecting sparse interactions between agents. We present a special kind of decentralised sparse interaction Markov decision process called a decentralised local interaction Markov decision process. This framework represents the interaction states of the agents as a function of the current local state of an agent. We introduce 2Observe, which is capable of approximating this function and solve the underlying decentralised local interaction Markov decision process. This chapter presents an overview of the work published in [De Hauwere et al. (2008)a], [De Hauwere et al. (2008)b], [De Hauwere et al. (2009)a] and [De Hauwere et al. (2010)c].

In Chapter 5 we introduce two variants of Coordinating Q-learning (CQ-learning). This algorithm is capable of solving a decentralised sparse interaction Markov decision process. To do so, it learns the set of states in which interactions between agents are reflected in the immediate reward signal and extends its state space representation in these states to include the state information of the other agents in the interaction. We also present some notions on the convergence of this algorithm by analysing the dynamics of the system by means of evolutionary game theory. The approach and results discussed in this chapter have been reported in [De Hauwere

et al. (2010)b] and [De Hauwere et al. (2011)a].

In Chapter 6 we present an extension to the work introduced in Chapter 5. Future Coordinating Q-leaning (FCQ-learning) is concerned with learning interaction states even though the effect of interactions is only reflected in the reward signal, several timesteps ahead. At the end of this chapter we give an empirical evaluation of this algorithm to some gridworld environments that exhibit this kind of interactions. Our research on future interactions has been published in [De Hauwere et al. (2011)b] and [De Hauwere et al. (2011)c].

In Chapter 7 we present two extensions on the algorithms introduced in chapters 4 and 5. These extensions allow for the transfer of coordination experience between agents and environments in order to speed up the learning process, as well as improve the final solution. Knowledge transfer has recently gained a lot of attention in single agent RL, but the work presented in this chapter is the first in its kind for transferring knowledge in environments where multiple agents are acting. These extensions on 2Observe and CQ-learning are also reported in [De Hauwere et al. (2009)b], [De Hauwere et al. (2010)a], [De Hauwere et al. (2010)b] and [Vrancx et al. (2011)].

We conclude this dissertation with a summary of the presented research and give some directions for future research in Chapter 8.

A graphical overview of this outline is given in Figure 1.4. Arrows indicate the recommended sequence of reading. Parallel chapters can be read in chronological order but they do not depend on each other although small cross references are possible.

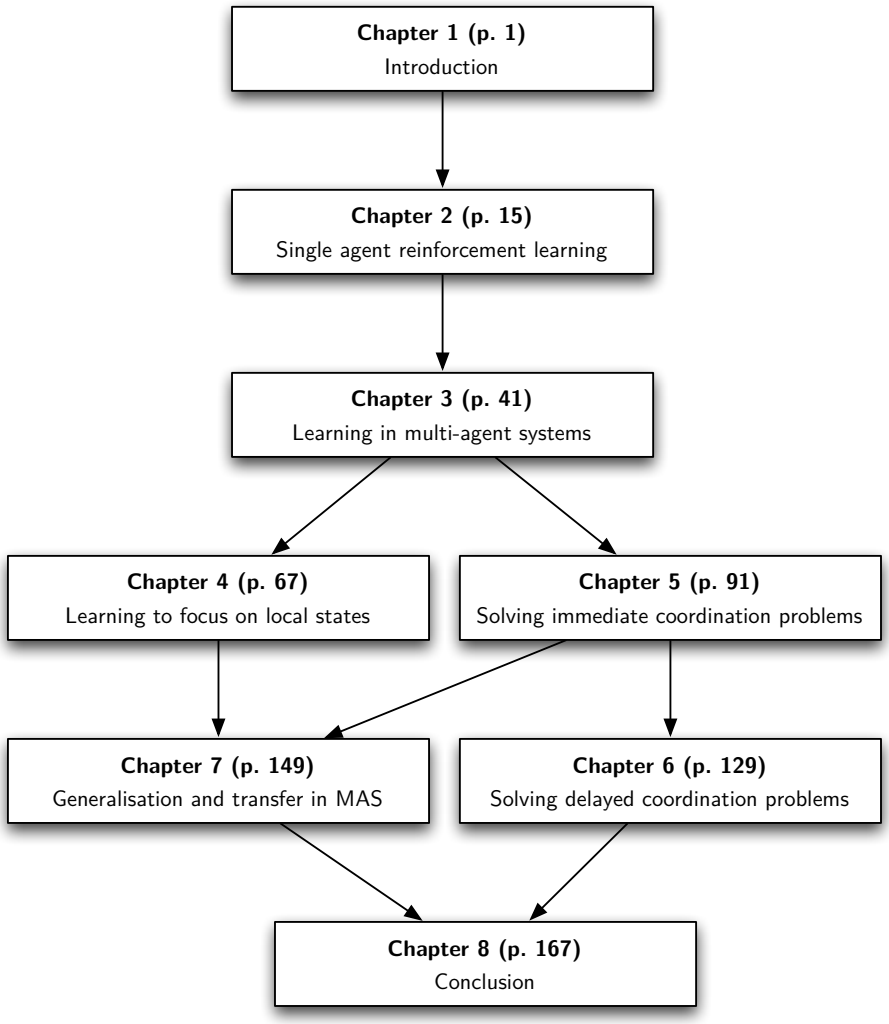


Figure 1.4: Graphical overview of the outline of this dissertation. Arrows indicate a dependency on the order in which the chapters should be read.

Chapter 2

Single agent reinforcement learning

I hear and I forget. I see and I remember. I do and I understand.
– Confucius, 551BC-479BC –

In this chapter we address the problem a single agent is facing when it has to take decisions in a dynamic environment consisting of a finite set of states. It has to learn which actions to take, resulting in both short and long term changes. Short term changes consist in an evolution of the system to a next state, accompanied by a feedback to the agent. The long term changes on the system are which future states the agent will observe, together with future rewards it might obtain for following a certain course of actions. These changes are used by the agent to learn which actions it should take for a given state. So an agent learns from its experience. This process of sequential decision making under uncertainty can be formalised by a *Markov Decision Process* (MDP) [Puterman (1994)]. This is a theoretical framework which models the dynamics of the system, in which the agent is learning. We will describe this framework in Section 2.1.

Reinforcement Learning (RL) attempts to learn an optimal policy for such an MDP, without knowledge about the MDP itself [Sutton & Barto (1998)]. It does so, purely based on the experience that is gathered from acting in the environment. This experience consists of the transitions the system undergoes, together with a reinforcement signal the agent receives, both in the short and long run. RL finds its origin from two major research tracks. One track originates in psychology, more specifically animal learning by trial and error. The other pillar on which RL is based is control theory.

Edward Thorndike wrote the following in 1911, which is known as the *Law of Effect*:

> Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or close followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond.
>
> [Thorndike (1911)]

This represents one of the key ideas of RL in which an agent's goal is to learn optimal actions in an unknown environment through trial and error, steered by a numerical reward. To put it in Thorndike's words: the actions taken in situations that yielded positive reinforcements should result in a strengthening in the bond to select those actions and vice versa. Two major issues exist in RL. Imagine a situation in which the reinforcement is only given after several actions. For instance, in the game of chess, a reinforcement is only given after winning or losing the game. This is called a *delayed reward*. Backpropagating this reward signal is a necessity, to ensure that an agent can take correct decisions, even though the immediate effect of this decision might be sub-optimal.
The other main issue in RL is called the *exploration-exploitation trade-off*. Informally, this trade-off determines when an agent should stop looking for a better action given a certain situation, and start exploiting its best known action so far. It is clear that exploiting too early in the learning process could result in sub-optimal policies, whereas delaying the exploitation of good actions will result in a worse overall performance of the agent. Hence this issue is also known as the exploration-exploitation dilemma. We will describe some of the approaches for dealing with these issues in this chapter.

Control theory is the second track on which RL is based. This research area, which uses mathematics for engineering applications, is concerned with the behaviour of dynamical systems. One of the pioneers of this field, Richard Bellman, laid the foundations of *Dynamic Programming* (DP), on which many RL-algorithms are built [Bellman (1957)]. DP is a way of computing optimal policies for an environment, given a perfect model of this environment. Such a model is represented as an MDP.

Throughout this chapter we will first describe MDPs and the elements which make a problem a reinforcement learning problem. This general introduction will be followed by a discussion and evaluation about some techniques for selecting actions

in RL problems. Finding a good balance in which action to select, allows for a balanced trade-off between exploring new actions and possibly finding better ones, and exploiting actions which are currently considered to be good. In Section 2.5 we will give an overview of solution methods for both DP and RL. We will make a distinction between techniques that directly adapt the policy and techniques that change their estimates for the values of states and actions from which they compute a policy. Finally, we will present two extensions to the MDP-framework. The first models situations in which not all state features in the system are known to the agent. This extension is referred to as a *Partially Observable Markov Decision Process*. The second, captures certain dependencies that exist between state variables in the transition function and is called a *Factored Markov Decision Process*.

Important references for this chapter, which give a detailed description of the field of MDPs is [Puterman (1994)] and DP is explained in depth in [Bertsekas (1995)a, Bertsekas (1995)b, Bertsekas & Tsitsiklis (1996)]. Important references for RL are [Barto et al. (1989)], [Kaelbling et al. (1996)] and [Sutton & Barto (1998)].

2.1 Markov decision processes

Informally a Markov decision process is a model for sequential decision making. At every time step t an agent must select an action a, given knowledge about the current state of the system $s(t)$. This action will result in a transition of the system from state $s(t)$ to state $s(t+1)$ for which the agent receives a scalar reward $r(t+1)$. The transition to state $s(t+1)$ is only dependent on the current state $s(t)$ and the action $a(t)$ selected by the agent. In other words, the effect of this action and the future evolution of the system is independent of the history of the system. This independence relation is known as the Markov property[1].
Formally, a Markov decision process can be described as follows:

Definition 4. *A **Markov decision process** is a tuple (S,A,T,R), 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* $s \in S$*,*
- $T : S \times A \times S \rightarrow [0, 1]$ *is the transition function,* $T(s, a, s')$*, specifying the probability of going to each state* s'*, from state* s *after action* a *is performed,*
- $R : S \times A \times S \rightarrow \mathbb{R}$ *is the reward function,* $R(s, a, s')$*, specifying the expected reward for the transition of* s *to* s' *after action* a *is performed.*

and an MDP obeys the Markov Property.

[1] Named after Andrey Andreyevich Markov (1856 - 1922). A Russian mathematician, best known for his work on the theory of stochastic processes

Definition 5. *A system is said to possess the **Markov Property** if the future state transitions of the system are independent of the previous states, given the current state:*

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

The goal of an agent is to learn a policy which maximises the expected future reward an agent receives. This future reward can be defined in terms of the expected average reward over time, or, as we will focus on in this dissertation, in terms of discounted future rewards. In discounted MDPs rewards received at later time steps are multiplied by a discount factor $\gamma \in [0, 1]$ to represent the current value of these future rewards.
An illustrative example of an MDP is the Stochastic Taxi Domain problem. This is a stochastic extension to the taxi domain from [Dietterich (2000)].

Example 1: *Consider the stochastic taxi domain problem depicted in Figure 2.1. The taxi starts in a randomly chosen location and has to pick up a passenger at one of the locations marked in red, green, yellow or blue. That passenger wishes to be transported to one of the other coloured locations. The taxi must go to the location of the passenger (chosen randomly), pick up the passenger, go to its desired destination (also chosen randomly) and put down the passenger there.*
The actions the agent can take, in this case the taxi driver, are four navigation actions that move the taxi one square `North`, `East`, `South`, `West`, *from its current location, a* `Pickup` *action and a* `Putdown` *action. If a* `Pickup` *action is performed at a location where a passenger was waiting, the passengers enters the taxi. Similar, a passenger exits the taxi after a* `Putdown` *action at its destination. An illegal* `Pickup` *action is trying to pick up a passenger in a location where there is no passenger. An illegitimate* `Putdown` *action is trying to put down a passenger when the taxi is empty, or when the passenger is not yet at its destination.*
The dashed line on the second row represents a stop light. The taxi has a probability of 0.7 *of moving through to the next cell and a probability of* 0.3 *of remaining in the same location. The taxi can avoid this uncertainty by taking a longer route going through the passage on the fourth row.*
Every action receives a reward of -1*, except delivering the passenger at its destination, which yields a reward of* $+20$*. If an illegitimate* `Pickup` *or* `Putdown` *action is attempted a penalty of* -10 *is given.*
The state of the system is described by the location of the passenger (in the environment or in the taxi), the location of the taxi and the requested destination of the passenger.

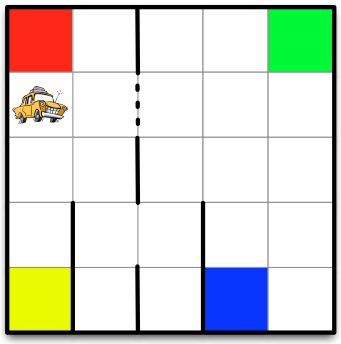


Figure 2.1: The stochastic taxi domain problem. Adapted from [Dietterich (2000)].

2.2 The reinforcement learning problem

Reinforcement learning is characterised by the problem that needs to be learned, rather than by a particular set of algorithms. As such, any algorithm that can solve a RL problem is a valid RL solution method. In this section we will begin by describing what makes a learning problem a RL problem, before going over some of the most well known solution methods in the following section.

In RL problems, agents receive a reward signal for performing actions. The learner must discover which actions yield the most reward, by trial and error. It must discover a mapping from its input (its information about the environment), to an action, without any information from a teacher about what is good and what is bad. The only feedback it receives is a numerical reward signal. This is different from most machine learning techniques in which the learner is explicitly told which actions to take. In these so-called supervised learning approaches, the learner observes a set of input-output pairs and learns a mapping between them. It is also different from clustering approaches, which are usually referred to as unsupervised learning. RL is classified between these two learning techniques.
The numerical reward signal the agent observes, might not just be the cause of the last action taken, but might originate from an action taken several timesteps in the past or from a whole sequence of actions. This principle, known as delayed rewards, together with stochastic events in the environment, are the two characterising attributes of RL problems and these make such problems both interesting and challenging.

We depict the model of RL problems graphically in Figure 2.2. An agent is

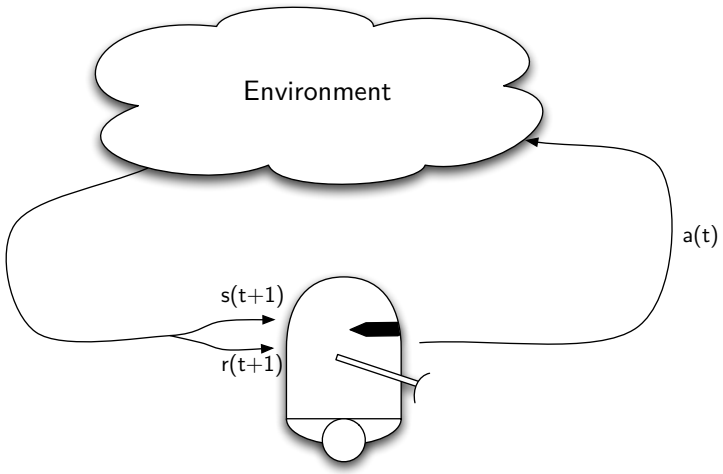


Figure 2.2: The standard reinforcement learning model, adapted from [Kaelbling et al. (1996)].

interacting with its environment by selecting an action $a(t)$, based on its current information of the state environment $s(t)$ at every discrete time step $t : 0, 1, \ldots$. The response of the environment consists of a description of the new state $s(t+1)$ and an immediate numerical reward $r(t+1)$ representing how good (or bad) the last performed action was. Transitions to the next state can be stochastic. This means that the next state $s(t+1)$ and reward $r(t+1)$ are random variables given with respect to a stochastic transition function T and a stochastic reward function R. It is commonly assumed that these probability functions are stationary, i.e. they do not change over time. In the following chapter we will discuss multi-agent RL settings, and argue why this assumption is not necessarily met.

Definition 6. *In a **stationary environment**, the probabilities of making state transitions or receiving specific reinforcement signals do not change over time.*

If we go back to the taxi domain from Example 2.1 we can see that this problem contains all the characteristics of a RL problem. The reward signal that indicates whether the actions taken by the agent were good or bad is delayed since the only positive reward given in the system is obtained for putting down the passenger at its destination. Moreover, there is no teacher telling the taxi agent what its next action should be. The agent must learn on its own an optimal policy for picking up a passenger and putting him down at its destination, with respect to the total cumulative reward. This leads us to describing the elements of a reinforcement learning system. Besides the agent and the environment, one can identify three main subelements:

- a **policy** π. An agent's policy π is a function that maps states to a tuple of

probabilities $\vec{p}$ for all actions an agent can take in that state. If S is the set of states of the environment, $\pi(s, a)$ denotes the probability of taking action $a \in A$ in state $s \in S$ at time step t. 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 a stochastic policy.

- a **reward function** R defining the quality of the action chosen by the agent. It maps a state-action-state tuple to a single numerical reward.
- a **value function** $V^{\pi}(s)$ specifying the value of a state. It indicates how good a particular state s is in the long run according to the current policy π. This value indicates the total amount of reward an agent can expect to accumulate in the future, starting from that state and following policy π. Even if the immediate rewards that can be obtained in a particular state s are low, the value of the state can still be high if it is followed by states that yield high rewards.

The policy an agent adopts in a RL problem will be changed, based on the rewards it receives. The goal of a RL algorithm is to maximise the total accumulated reward it receives for taking actions in the environment. For instance if an agent found an action in a certain state that results in a better payoff than the action currently generated by the policy, the policy will be adapted to select this new action with a higher probability. The choice whether to select currently known good actions, or try out other actions is known as the exploration-exploitation dilemma. This dilemma is discussed in the next section. A RL system may also contain a predictive model of the environment. This model imitates the environment. For example, a model can predict the probability for observing state $s(t + 1)$ after performing action $a(t)$ in state $s(t)$. In Section 2.5 we will on one hand present RL techniques that use such a model to learn a policy and on the other hand describe approaches that directly learn a policy without a model.

2.3 Action selection mechanisms

To explain the exploration-exploitation dilemma and introduce some of the most common action-selection mechanisms we will use the n-armed bandit problems as an example. In such problems, named after a slot machine or one-armed bandit, the agent is faced with a choice among n different actions. The n-armed bandit is a very simple RL problem, in which there are no delayed rewards, and only 1 state. Every action results in a reward chosen from a stationary probability distribution. The objective is to maximise the expected total reward over some period of time. Each action selection is called a play. The expected values of the actions, i.e. the reward they return is unknown to the agent. Otherwise it would be straightforward to determine the best action and always select that action. An agent can maintain

estimates over the actions and update these estimates over time. The choice the agent has to make is when to start selecting the currently best action. Choosing this best action is called *greedy action selection*. If this is done too early, the value estimates of the agent might still be inaccurate and a suboptimal action might be selected. If this is done too late, there might not be enough plays remaining to collect a high reward over the entire time period. The reward through exploration is lower in the short run, but might be higher on the long term after having discovered the actions that yield the highest rewards. In order to select these actions an estimate about the value of an action is maintained and updated at every timestep. A straightforward way of maintaining such an estimate is the sample average method. The estimate of the reward for action a at the t-th timestep $Q_t(a)$ is given by:

$$Q_t(a) = \frac{r(1) + r(2) + \ldots + r(k_a)}{k_a},$$

where k_a is the number of times action a has been selected prior to t and $r(i)$ is the reward received for action a at timestep i. In this dissertation we will be using ϵ-*greedy* and *softmax action selection* strategies described below.

An ϵ-greedy selection strategy is a simple alternative to behaving greedily all the time. The selection mechanism will select the best action most of the time, but with a small probability ϵ it will select an action at random, uniformly, independent of the action-value estimates. The main advantage of this approach is that as the number of plays increases, all actions are still being explored. In the following chapter we will describe why this is important.

Another technique for selecting actions is to map the estimates of the values of every action to a probability distribution and then selecting actions probabilistically according to this distribution. This is called softmax action selection. The most commonly used distribution is the Boltzmann distribution. The probability of selecting action a at the t-th play is given by:

$$Pr(a) = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}, \tag{2.1}$$

where τ is a positive, non-zero parameter called the temperature, n the number of actions and $Q_t(a)$ is the estimate of the value of action a at timestep t. High temperatures causes all actions to be (nearly) equiprobable, whereas the lower the temperature the more greedy the action selection will be. The parameter τ can be seen as how greedy the selection mechanism chooses its actions. By decreasing this parameter over time, the selection mechanism will shift from more exploration in the initial phase of the task (learning), to more exploitation at the end[2] (earning). This action selection mechanism overcomes one of the major drawbacks of the ϵ-greedy approach. When using ϵ-greedy, the probability of choosing the next to best action

[2] Note that with an ϵ-greedy selection mechanism it is also possible to reduce ϵ over time

is equal to the probability of choosing the worst action. Since in a softmax approach the probability of selecting an action is a grading function of their estimates, the probability of choosing a very bad action is very small.

By over- or underestimating the initial values of actions, the dynamics of the exploration process can be changed. For a more in-depth survey of a variety of such ad-hoc action selection techniques we refer to [Thrun (1992)b, Thrun (1992)a, Wiering (1999)]. In recent years, other techniques have emerged, aiming to steer the exploration in RL problems by means of the incorporation of domain knowledge [Grzes (2010)] or by providing the agents with additional reward signals to allow them to learn faster in large environments [Grzes & Kudenko (2009)].

2.4 Evaluation of action selection mechanisms

In the previous section we briefly described greedy, ϵ-greedy and softmax action selection mechanisms. In this section we will evaluate the behaviour of these mechanisms on 2 and 10-armed bandit problems. We performed 1000 tasks of each 1000 plays and present the average result over the tasks. For every task, the true value of an action, $Q^*(a)$, was generated from a normal distribution with mean 0 and variance 1. The rewards for every task were generated from a normal distribution with mean $Q^*(a)$ and variance 1. The agent is updating the estimates of the value of its actions using the sample average method.

2.4.1 2-armed bandit problem

We first show the behaviour of the action selection mechanisms using the 2-armed bandit problem. One of the actions is the optimal one, whereas the other is suboptimal. In Figure 2.3 we demonstrate the average probability of selecting the optimal action for the different schemes. We have added an agent following a completely random policy as a reference ($\epsilon = 1.0$). All ϵ-greedy schemes need less than 200 plays to reach a probability of selecting the optimal action between 85% and 95%. Note that when these schemes select a random action, they are just as likely to select the optimal action as they are to select the suboptimal action.

For the softmax schemes using a Boltzmann distribution, we clearly see the effect of the temperature parameter τ in the resulting probabilities. At the beginning of the task, τ was set to 1000 and decayed at every play according to following formula:

$$\tau = 1000 * decay^{play}$$

We used 0.90,0.95 and 0.99 as the decay factor. From Figure 2.3 we see that the faster we let the temperature decay, the earlier in the task it starts to play greedy and plays the action with the highest $Q_t(a)$. Before the temperature decreases,

this scheme acts similar to a completely random action selection mechanism and is exploring between its possible actions. All softmax schemes reach a higher probability of selecting the optimal action than the ϵ-greedy schemes. This is due to the fact that after some time the temperature parameter has become so small that the scheme is acting completely greedy.

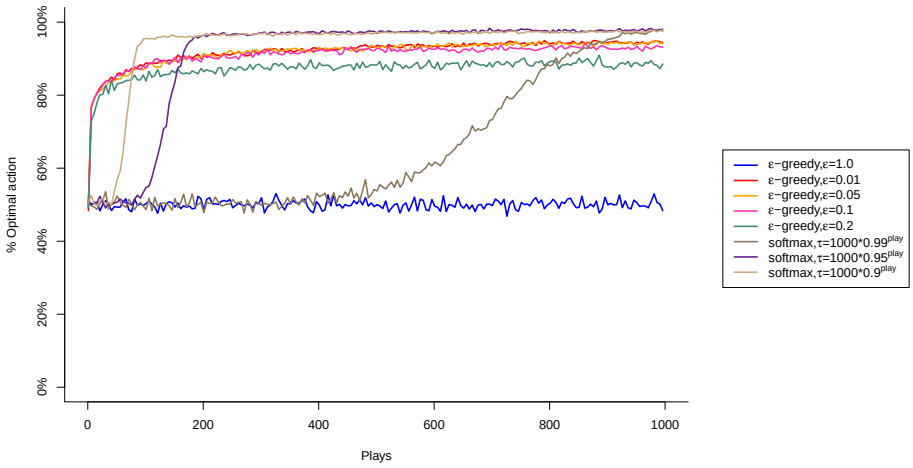


Figure 2.3: Average probability that the agent selects the optimal action under various action selection mechanisms in the 2-armed bandit problem

In Figure 2.4 we show the reward the agents collect over the entire task. As in the figure showing the probability of the optimal action, we also see here that the ϵ-greedy schemes improve faster than the softmax approaches, but are outperformed by these in the long run.

In a 2-arm setting we could also apply ϵ-greedy schemes with a decreasing ϵ. As such, more control is obtained over the level of exploration at the beginning of the task. In Table 2.1 we show the total reward the agents collected per task. For this task, ϵ-greedy with $\epsilon = 0.1$ and softmax with initial temperature at 1000 and a decay rate of 0.9 perform almost on par, even though the softmax method selects the optimal action more often. This is due to the fact that before the temperature decreases, this scheme selects the suboptimal action in 50% of the plays. The ϵ-greedy method exploits the optimal action earlier in the task and as such makes up for not selecting the optimal action all the time. It should be clear that the performance of every action selection scheme is very specific to the task at hand as will be shown in the next chapter.

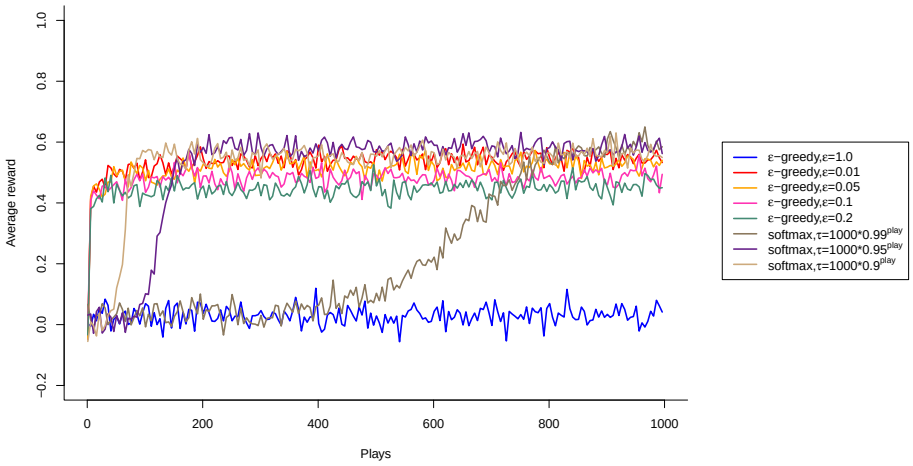


Figure 2.4: Average reward the agent collects over time under various action selection mechanisms in the 2-armed bandit problem

scheme	Accumulated reward
ϵ-greedy,$\epsilon = 1.0$	-6.561
ϵ-greedy,$\epsilon = 0.01$	532.464
ϵ-greedy,$\epsilon = 0.05$	531.823
ϵ-greedy,$\epsilon = 0.1$	556.065
ϵ-greedy,$\epsilon = 0.2$	445.134
softmax,$\tau = 1000 * 0.99^{play}$	206.646
softmax,$\tau = 1000 * 0.95^{play}$	486.821
softmax,$\tau = 1000 * 0.9^{play}$	**560.463**

Table 2.1: Total accumulated reward per task in the 2-armed bandit problem

2.4.2 10-armed bandit problem

In the 2-armed bandit problems of the previous section, only little exploration is needed to identify the better of both actions. In this section we investigate how these action selection mechanisms behave when the optimal action has to be selected from 10 actions. Figure 2.5 show the probabilities of selecting the optimal action for the different schemes. We clearly see that softmax outperforms the ϵ-greedy methods and that a decay factor of 0.95 reaches the highest probability of selecting the optimal action over 1000 plays. In Figure 2.6 we see a similar situation as in the

2-armed bandit problem. The ϵ-greedy methods perform better in the beginning, but are outperformed by the softmax methods in the end.

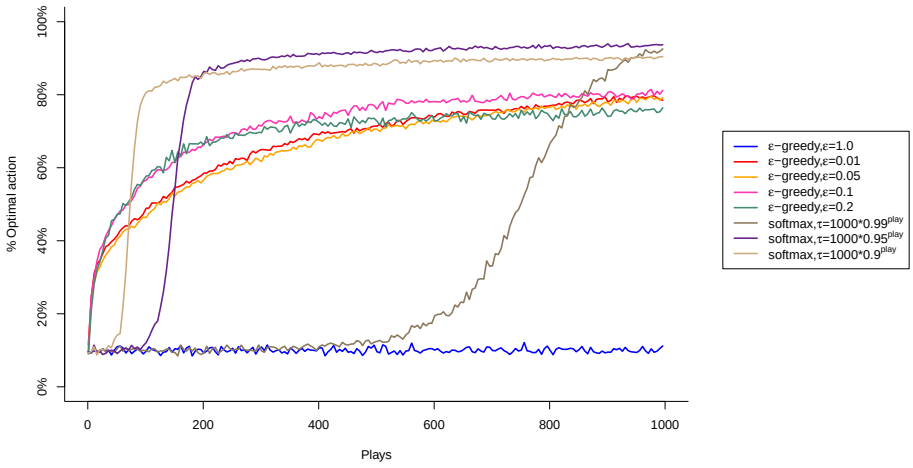


Figure 2.5: Average probability that the agent selects the optimal action under various action selection mechanisms in the 10-armed bandit problem

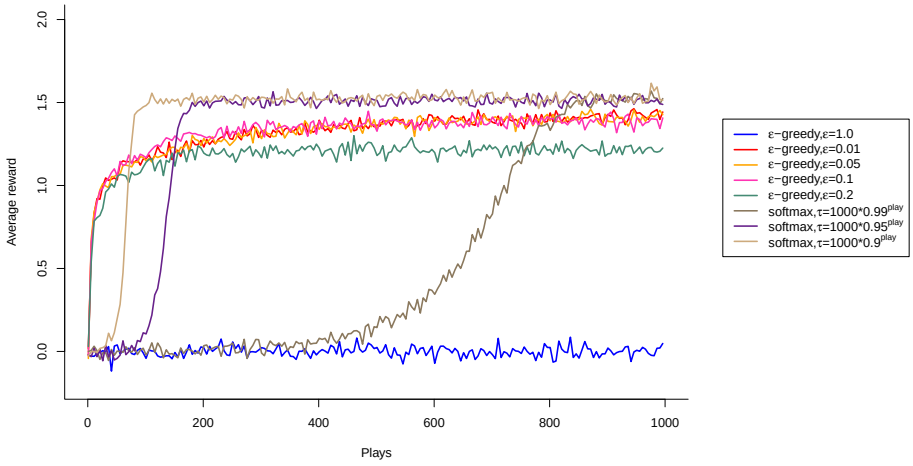


Figure 2.6: Average reward the agent collects over time under various action selection mechanisms in the 10-armed bandit problem

The total accumulated reward, represented in Table 2.2, shows the benefits of finding a correct trade-off between exploration and exploitation. A softmax scheme, using a decay factor of 0.9, reached the highest cumulative reward over the entire task. The rapid increase from the ϵ-greedy approaches in the first 100 plays is not enough to reach the same cumulative reward as the softmax approach.

scheme	**Accumulated reward**
ϵ-greedy,$\epsilon = 1.0$	3.535
ϵ-greedy,$\epsilon = 0.01$	1324.319
ϵ-greedy,$\epsilon = 0.05$	1349.819
ϵ-greedy,$\epsilon = 0.1$	1298.816
ϵ-greedy,$\epsilon = 0.2$	1190.268
softmax,$\tau = 1000 * 0.99^{play}$	511.756
softmax,$\tau = 1000 * 0.95^{play}$	1333.369
softmax,$\tau = 1000 * 0.9^{play}$	**1409.146**

Table 2.2: Total accumulated reward per task in the 10-armed bandit problem

From these experiments we can conclude that different parameters for the exploration strategies result in very varying outcomes. It should be clear that even in this simple setting, a good balance must be found between exploration and exploitation. Too little exploration leads to suboptimal results at the end of the learning process, whereas too much exploration leads to bad performance during the learning process and long learning times. The best choice of action selection mechanism is problem dependent and there is no known way to automatically select the value of ϵ in ϵ-greedy methods [Auer et al. (2002)], or at which rate to decrease the temperature for the Boltzmann exploration strategy. In the following chapter we will illustrate that this issue is even more pressing when multiple agents are acting in the same environment.

2.5 Solution methods

In the previous sections we already introduced the theoretical framework on which RL relies, the characterising attributes of RL problems, as well as some mechanisms for selecting actions, given some estimates for the values of these actions. In this section we provide an overview of solution methods to RL problems, with multiple states and (possibly) delayed rewards. Two main categories of approaches exist. On one side we have *model-based* techniques. These techniques use an explicit model of the environment to calculate the optimal policy. This model can either be available to the agent and dynamic programming methods can be applied, or

the agent can learn this model while selecting actions. The latter can increase the learning speed compared to traditional RL techniques and is applied in for instance the *dyna architecture* and *prioritized sweeping*.

On the other side we have approaches which do not require an explicit model of the dynamics of the system. These approaches are called model-free approaches. Within these we can make the distinction between techniques that learn estimates for the values of the actions, and subsequently derive a policy from these estimates, and techniques that directly learn a policy and update this by means of a quality measure. We will describe these approaches in Section 2.5.2

2.5.1 Model based approaches

A model in RL means some form of knowledge of the probabilities of the state transition function $T(s, a, s')$ and the reward function $R(s, a, s')$. This could for instance be a lookup table or a decision tree. When such a model is known in advance, a policy π can be computed from it using dynamic programming.

2.5.1.1 Dynamic programming

The term dynamic programming (DP) encompasses a collection of algorithms that can be used to compute an optimal policy π^*, if the agent has access to the underlying model of the environment, represented as a Markov decision process (see Section 2.1). Such policies can be derived by calculating the value of a state.

Definition 7. *The **value of a state** s, given a policy π is the total amount of reward an agent can expect to accumulate in the future, starting from that state s following that policy.*

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left\{R_t | s(t) = s\right\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r(t+k+1) | s(t) = s\right\} \end{aligned} \tag{2.2}$$

In a similar way, we can define the value of taking action a in state s:

Definition 8. *The **value of action** a **in state** s, given a policy π is the total expected amount of reward the agent can expect to accumulate in the future, starting from state s, taking action a and thereafter follow policy π.*

$$\begin{aligned} Q^{\pi}(s, a) &= E_{\pi}\left\{R_t | s(t) = s, a(t) = a\right\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r(t+k+1) | s(t) = s, a(t) = a\right\} \end{aligned} \tag{2.3}$$

Rewriting the state-value function of Equation 2.2 as a recursive function gives

us the Bellman equation for V^π:

$$\begin{aligned} V^\pi(s) &= E_\pi \{r(t+1) + \gamma V^\pi(s(t+1)) | s(t) = s\} \\ &= \sum_a \pi(s,a) \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^\pi(s')\right] \end{aligned} \tag{2.4}$$

This equation expresses the recursive relation between the value of a state and the value of its successor states. It calculates a weighted average over all possible successor states, based on the probability of occurring. When solving RL problems, one wants to find the optimal policy, i.e. the policy which maximises Equation 2.4. This *optimal value function* V^* for all states $s \in S$ is defined as:

$$V^*(s) = \max_\pi V^\pi(s) \tag{2.5}$$

Similar to this equation we can define the *optimal state-value function* Q^* for all states $s \in S$ and for all actions $a \in A$:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \tag{2.6}$$

So $Q^*(s,a)$ describes the expected return for taking action a in state s according to a policy π and following an optimal policy after that:

$$Q^*(s,a) = E_\pi\{r(t+1) + \gamma V^*(s(t+1)) | s(t) = s, a(t) = a\}$$

The value of a state under the optimal policy is equal to the expected return for the best action from that state. This is called the Bellman optimality equation for V^* and can be derived as follows:

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\ &= \max_{a \in A(s)} E_{\pi^*} \{R_t | s(t) = s, a(t) = a\} \\ &= \max_{a \in A(s)} E_{\pi^*} \left\{ \sum_{k=0}^{\infty} \gamma^k r(t+k+1) | s(t) = s, a(t) = a \right\} \\ &= \max_{a \in A(s)} E_{\pi^*} \left\{ r(t+1) + \gamma \sum_{k=1}^{\infty} \gamma^k r(t+k+1) | s(t) = s, a(t) = a \right\} \\ &= \max_{a \in A(s)} E \{r(t+1) + \gamma V^*(s(t+1)) | s(t) = s, a(t) = a\} \\ &= \max_{a \in A(s)} \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right] \end{aligned} \tag{2.7}$$

In a similar way, the Bellman optimality equation for Q^* is given by:

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right] \tag{2.8}$$

Given the transition function T and the reward function R, algorithms that solve Equation 2.7 are known as dynamic programming algorithms. We will briefly review the two most well-known DP algorithms: policy iteration and value iteration.

Policy Iteration

The policy iteration algorithm will use the state-value function to calculate the value of the entire policy, by calculating the value of all the states. This step is known as the policy evaluation (PE) step. Using the newly calculated values of the states, in the policy improvement (PI) step, the algorithm will attempt to find a better action in all states. As long as the algorithm finds better actions, it will iterate over these two steps as shown in Figure 2.7.

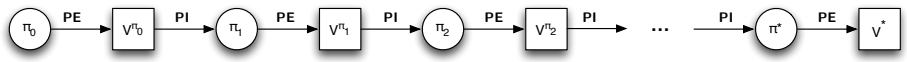


Figure 2.7: Alternation between policy evaluation and policy improvement steps, where each step yields a better policy, until termination with the final policy π^*

Since the number of deterministic policies is finite for a finite MDP, i.e. $|A|^{|S|}$ and each policy π_i is an improvement over the previous policy π_{i-1}, this algorithm is guaranteed to find the optimal policy in at most an exponential number of iterations [Puterman (1994)]. The pseudocode for this algorithm is given in Algorithm 2.

Value Iteration

Another approach to find the optimal policy is to calculate the optimal value function directly as is done in the value iteration algorithm. Value iteration does not need a separate policy evaluation step, and as such is computationally less expensive than policy iteration. Value iteration uses a simple backup operation to calculate the optimal value function iteratively:

$$V^{\pi'}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi}(s')\right] \tag{2.9}$$

It can be shown that this process converges to the optimal value function V^* [Bellman (1957), Bertsekas (1976)]. As a stopping criterion the difference between two successive value functions is commonly used. If this difference is arbitrarily small, the optimal policy can be derived from the value function, by selecting the greedy action for every state. The complete outline of the algorithm is given in Algorithm 3.

These algorithms are a way of solving an MDP when the underlying transition and reward functions are known. In the following section we will discuss RL algorithms in which these functions are unknown to the agent.

2.5.2 Model free approaches

In this section we will describe the RL counterparts of value and policy iteration. In RL it is common that the agents do not have access to the underlying model of the

Algorithm 2 Policy Iteration

Initialisation:
$V(s) \in \mathbb{R}$ and $\pi(s) \in A(s)$ arbitrarily $\forall s \in S$
repeat
  Policy evaluation:
  repeat
    $\Delta \leftarrow 0$
    for all $s \in S$ **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V(s')\right]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
  until $\Delta < \theta$
  Policy improvement:
  policy-stable $\leftarrow$ `true`
  for all $s \in S$ **do**
    $b \leftarrow \pi(s)$
    $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V(s')\right]$
    if $b \neq \pi(s)$ **then**
      policy-stable $\leftarrow$ `false`
    end if
  end for
until *policy-stable*

problem they are trying to solve, i.e. the transition and reward functions cannot be used explicitly but can only be experienced implicitly through interaction with the environment. This concept of learning from experience is known as direct learning. As with DP, we can also distinguish between approaches that learn the values of the states and derive a policy from these values subsequently and approaches that change the policy directly. We will begin by describing the Q-learning algorithm, which is one of the most popular and well-known RL algorithms and is an example of an approach that learns the values of the states. The research described in Chapters 5 and 6 is based on this algorithm. In Section 2.5.2.2 we will describe Learning Automata (LA). These simple reinforcement learning units have strong theoretical convergence guarantees and can be combined into complex networks to learn policies for the problem at hand. LA, which fall in the category of policy iteration algorithms, form the basis of the research described in Chapter 4.

Algorithm 3 Value Iteration

Initialise V arbitrarily $\forall s \in S$
repeat
  $\Delta \leftarrow 0$
  for all $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$
Output a deterministic policy π, such that
$\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right]$

2.5.2.1 Q-learning

One of the most popular approaches for solving RL problems is *Q-learning* [Watkins (1989), Watkins & Dayan (1992)]. The goal of Q-learning is to approximate the optimal state-action values as given in Equation 2.8. Since we are dealing with model-free approaches, the values for $T(s,a,s')$ and $R(s,a,s')$ are not available. For this reason, Q-learning uses estimates for the values of future states in its updating process. This principle is known as bootstrapping. The current estimates of the state-action values are known as Q-values. Every estimate $\hat{Q}(s,a)$ is the learner's current hypothesis about the actual value of $Q^*(s,a)$ for a state s and action a. These Q-values are updated according to following update rule:

$$\hat{Q}(s,a) \leftarrow (1-\alpha(t))\hat{Q}(s,a) + \alpha(t)\left[r + \gamma \max_{a'} \hat{Q}(s',a')\right] \tag{2.10}$$

where $\alpha(t) \in [0,1]$ is the learning rate and $\gamma \in [0,1]$ is the discount rate. A complete listing of the algorithm is given in Algorithm 4.

By iteratively updating these estimates $\hat{Q}$, these values will converge to Q^* under some general conditions [Tsitsiklis (1994)]:

- all state-action pairs are visited infinitely often,
- a suitable evolution for the learning rate is chosen.

Clearly, visiting all state-action pairs infinitely often is impracticable for real-world applications of Q-learning. In such applications, the amount of exploration is usually reduced after the Q-values have converged (i.e. changes are smaller than some Δ). Techniques for tuning this exploration rate have already been discussed in Section 2.3.

Algorithm 4 Q-learning

1: Initialise $\hat{Q}(s, a)$ arbitrarily
2: **for** each episode **do**
3: Initialise s
4: **for** each step in the episode **do**
5: Choose a from s using policy derived from $\hat{Q}$ (cfr. Section 2.3)
6: Take action a, observe r,s'
7: $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha[r + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$ (cfr. Equation 2.10)
8: $s \leftarrow s'$
9: **end for**
10: **end for**

Because Q-learning is learning the value of state-action pairs and deriving a policy from these values, this algorithm is classified in the family of value iteration algorithms. In the following section we describe learning automata which is an example of a reinforcement learning approach that belongs to the policy iteration algorithms.

2.5.2.2 Learning automata

Learning Automata (LA) are adaptive decision makers which are capable of acting in highly uncertain stochastic environments. They have their roots in mathematical psychology and were used in the fifties to describe human behaviour from psychological and biological viewpoints [Bush & Mosteller (1955)]. In the 1960's LA were used for engineering research [Tsetlin (1962)]. Nowadays, they are commonly used in control problems for their low computational complexity such as in [Ahamed et al. (2002)]. In its most basic form, a learning automaton is connected to its environment in a feedback loop [Narendra & Thathachar (1989)]. This feedback loop is represented graphically in Figure 2.8

Each time step t the automaton selects an action $a(t) \in A$. This results in an output of the environment $r(t + 1) \in B$. Using this response, the automaton will update its internal probability distribution $\vec{p}$ over its action, according to its update scheme U. This probability distribution is its policy.

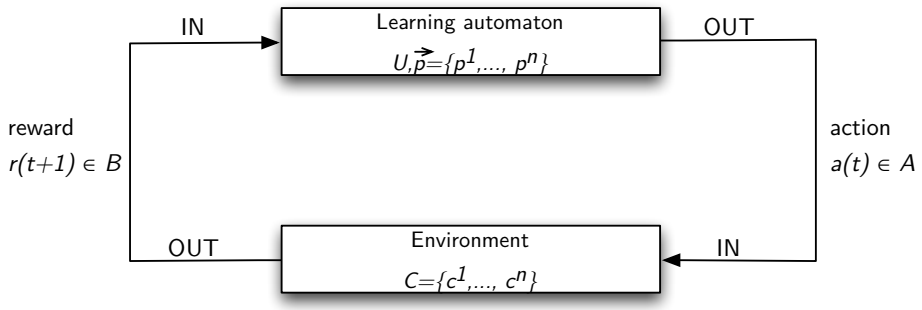


Figure 2.8: Situated learning automaton in a feedback loop with the environment.

Formally a learning automaton is described as follows:

Definition 9. *A **Variable Structure Learning Automaton** can be defined as the quadruple $\{ A, B, \vec{p}, U\}$, where:*

- *A is the action set of the automaton,*
- *B is the response set of the environment,*
- *$\vec{p}$ is a probability distribution over the actions,*
- *U is the update scheme used to update $\vec{p}$.*

and the environment in which the automaton is acting:

Definition 10. *An **Environment** is a triple $\{ A, C, B\}$, where:*

- *A is a finite set of possible inputs to the environment with $|A| = n$,*
- *C is a vector containing reward probabilities where each element $c^i \in C$ corresponds to one action $a^i \in A$ and is defined as $c^i = Pr[B = 1|a(t) = a^i]$, which is the probability of success if action a^i was selected at timestep t,*
- *B is a finite set representing the possible outputs of the environment.*

Note that the input set of the automaton, is the same as the output set of the environment and the output set of the automaton is the same as the input set of the environment.

Depending on the response set B, we distinguish different models for the environment:

- P-model: In this model, the response set is binary: $B = \{0, 1\}$. 1 denoting success and 0 denoting failure.

- Q-model: In this model, the response is an element of a finite set: $B = \{b^1, \ldots, b^m\}$ with $\forall i$, where $0 \leq b^i \leq 1$.
- S-model: In this model, the response is a real value in the interval $[0, 1]$.

The update scheme U follows the law of effect and will increase the probabilities of actions that resulted in a favorable response and decrease the probabilities for actions that resulted in an unfavorable response. The general scheme when action a^i was chosen at time step t is given by:

$$p^i(t+1) = p^i(t) + \alpha r(t+1)(1 - p^i(t)) - \beta(1 - r(t+1))p^i(t) \quad (2.11)$$

with $p^i$the probability of selecting action a^i

$$p^j(t+1) = p^j(t) - \alpha r(t+1)(p^j(t)) + \beta(1 - r(t+1))\left(\frac{1}{n-1} - p^j(t)\right) \quad (2.12)$$

$\forall a^j \in A$, where $a^j \neq a^i$ and $p^j$the probability of selecting action a^j

where n is the total number of actions, $r(t+1) \in B$ is the response received for performing action a^i at time step t, α is the reward parameter and β is the penalty parameter. Three common update schemes are defined in literature:

- Linear Reward-Inaction (L_{R-I}) if $\beta = 0$ in Equations 2.11 and 2.12,
- Linear Reward-Penalty (L_{R-P}) if $\alpha = \beta$ in Equations 2.11 and 2.12,
- Linear Reward-ϵ-Penalty ($L_{R-\epsilon P}$) if $\beta << \alpha$ in Equations 2.11 and 2.12.

When used to solve MDPs, multiple learning automata can be connected to form a network, assigning one automaton to each state. When the automaton in state s is activated, it selects an action $a \in A(s)$, according to the probability vector $\vec{p}$ associated with state s.The updates are performed using the accumulated reward since the last visit to state s. The algorithm for optimising the expected average reward over time is described in [Wheeler Jr & Narendra (1986)] and for optimising the future discounted reward in [Witten (1977)].

2.5.3 Learning models

In Section 2.5.1 we have described approaches for calculating the optimal policy when the models of the transition and the reward functions are known to the agent. In the previous section, we discussed methods for learning an optimal policy when this model is not known. Sutton's dyna architecture provides a way to integrate both of these approaches by simultaneously learning by trial-and-error in the real world and by performing updates based on a learned model of the world [Sutton (1990), Sutton (1991)]. It uses experience to build a model of the transition function and of the reward function. This model is then used simultaneously with real experience, to update the policy. A graphical representation of this process is shown

in Figure 2.9. In [Sutton (1990)] two actual algorithms based on the Dyna architecture are given: Dyna-PI and Dyna-Q. The basic idea of the former is to alternate real world experience with hypothetical experiences. In Dyna-Q, the Q-learning algorithm is executed in the real world to generate experience, while, at the same time, k additional updates are performed using experience gathered from the model. This requires about k times the computation of Q-learning per time step, but, as shown in [Sutton (1990)], the optimal policy is reached a lot sooner than with the model-free Q-learning algorithm (Algorithm 4) for deterministic environments.

In later work, Moore and Atkeson developed prioritized sweeping, which stores probabilities on state transitions in a priority queue to concentrate the computational effort on the states with the largest probability of reaching an end state [Moore & Atkeson (1993)].

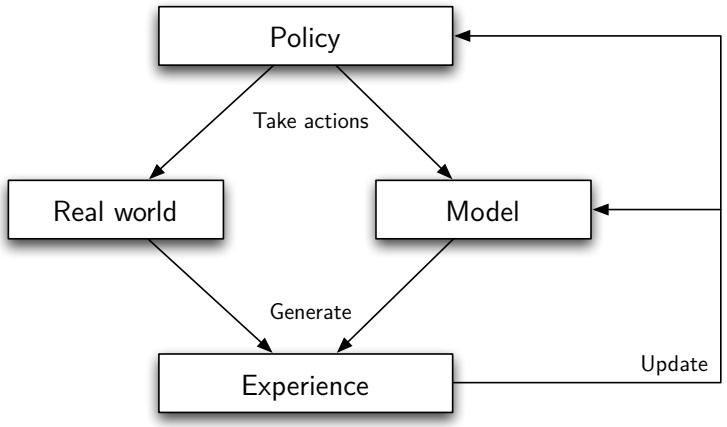


Figure 2.9: Dyna architecture: Simultaneously use experience to update the policy and the model of the world, by taking actions in both the real world as in the learned model.

2.6 Other MDP frameworks

All the algorithms and approaches we described above used the Markov Decision Process as the underlying framework. As explained, an MDP consists of a set of distinct states and the agent is informed about the state it is currently in. In practical applications, such as robotics, this is however not always realistic. In an extension of an MDP, called a *Partially Observable Markov Decision Process* (POMDP) the agent is not informed of the state the system is currently in, but is rather given a set of observations of the state which provide a hint of the state the system is currently in [Astrom (1965)].

Based on these observations, the agent can calculate a belief state. A belief

state is a probability distribution over states, representing the probability of being in a certain state s, given a set of observations. Formally we can describe a POMDP as follows:

Definition 11. *A **Partially Observable Markov Decision Process** is a tuple (S, A, O, T, Ω, R), where:*

- *S, A, T and R are the same as in an MDP,*
- *$O : o^1, \ldots, o^N$ is a finite set of observations,*
- *$\Omega : S \times A \times O \to [0, 1]$ is the observation function $\Omega(o, s, a)$, specifying the probability of observing o, given state s after taking action a.*

POMDPs offer a challenging problem for RL, because the environment the agents experience does not obey the Markov Property. An agent has no complete perception of the state of the environment. A simple example of a POMDP in which such problems occur is a continuous state problem which is being discretised too coarse. Another example is when critical state information, that is required for a Markovian description of the state, cannot be observed by the agent. A more thorough overview of POMDPs together with some solution methods can be found in [Littman et al. (1998)]. For a more in depth study of learning to act optimally in partially observable domains, we refer to [Cassandra (1998)].

Sometimes it may be beneficial to represent states by means of a set of properties instead of looking at states as atomic entities. This allows for the exploitation of certain dependencies that exist between some of these state variables. This framework is known as a *factored Markov decision process* (FMDP) [Boutilier et al. (1995)].

In an FMDP system states are described using a set of random variables $X = \{X_1, \ldots, X_n\}$ where each state variable X_i can assume values in a finite domain $Dom(X_i)$. Each possible system state corresponds to a value assignment $x_i \in Dom(X_i)$ for every state variable X_i.
The state transition function in such systems can compactly be described by a *Dynamic Bayesian Network* (DBN). This is a two-layer directed acyclic graph G_a where the nodes are $\{X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$. Such a graph is maintained for every action a and captures the dependencies between the state variables at time step t (X_i) and the resulting state variables after the transition at timestep $t + 1$ (X'_i). In this graph, the parents of X'_i are denoted by $Parents_a(X'_i)$. With every node $X'_i \in G_a$, a conditional probability distribution
$CPD^a_{X_i}(X'_i | Parents_a(X'_i))$ is associated quantifying the DBN. This method benefits from the dependencies that exist (or don't exist) between the variables of the network.

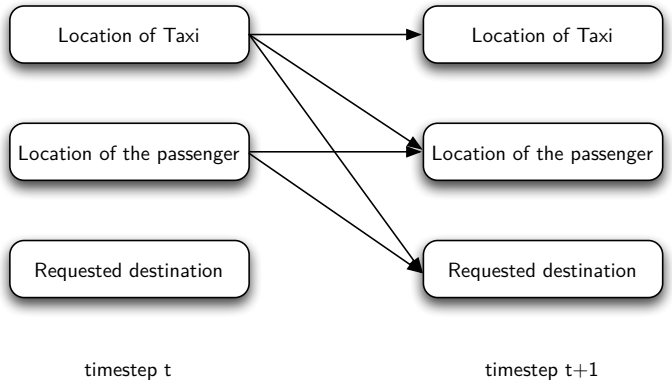


Figure 2.10: DBN for the `Pickup`-action of the stochastic taxi domain problem (see Example 2.1). At the moment the taxi executes this action it is not carrying a passenger.

Figure 2.10 shows the dependencies between the variables at timestep t and timestep $t+1$ when the `Pickup`-action is executed in the stochastic taxi domain problem (see Example 2.1). The current context is that the taxi is not yet carrying a passenger. The taxi remains in the same location so this variable is only dependent on its value in the previous timestep. The passenger's location can change from being in one of the coloured cells to being inside the taxi, if the taxi is at the location of the passenger and the passenger was not already in the taxi. The same goes for the desired destination of the passenger. This will only change if the taxi is at the same location as the passenger and the taxi is not already carrying a passenger. Several works aimed at learning and exploiting this structure have been published. Key references for this area of RL research are [Koller (1999)], [Guestrin et al. (2004)] and [Degris et al. (2006)].

2.7 Summary

This chapter presented a brief overview of reinforcement learning. We started this chapter by describing the underlying framework of RL: the Markov decision process. Second, we explained how reinforcement learning is defined by the problem an agent faces, rather than by a particular set of algorithms. We showed different methods for selecting actions in unknown environments and gave a comparison of their performance in n-armed-bandit problems, which can be seen as stateless RL problems. We continued by introducing multi-stage reinforcement learning and described solution methods for situations in which the model was known to the agent: dynamic programming methods such as value and policy iteration. Situations in which the model is unknown to the agent: Q-learning and learning automata.

And situations in which the agent simultaneously learns a model of the world, as well as a policy: the dyna architecture.

At the end of this chapter we shortly described some other frameworks that are common in a RL context. In the POMDP framework an agent receives observations and maintains beliefs over the actual system states it is in. We explained that care had to be taken in such systems since the environment the agents are experiencing is no longer assured to obey the Markov property. In a sense we could see a multi-agent system as a POMDP in which the agent only has its own information and does not have complete observability over the entire system state (i.e. the information of all the agents).

In this chapter we assumed that the state and action space are discrete. For the interested reader in continuous environments, we refer to [Busoniu (2008)] and [Busoniu et al. (2010)].

The next chapter will describe the additional complexities when multiple agents are acting in the same environment and how RL is being used in such settings.

Chapter 3

Learning in multi-agent systems

Life can only be understood backwards; but it must be lived forwards.

– Soren Kierkegaard, 1813-1855 –

3.1 Introduction

Telecommunications, economics, mobile robots, traffic simulation, ... are all examples of systems in which decentralisation of data and/or distribution of control is either required or inherently present. As such the use of multiple interacting agents is either a necessity to solve the problem at hand, or the most optimal way to do so. Such systems are called *Multi-Agent Systems* (MAS). In MAS we can make a distinction based on several dimensions [Kaminka (2004)]:

- **Agent design:** Agents can be heterogeneous or homogeneous.
- **Perception:** The view the agents have on the environment is usually distributed. They can all have their own view and state information about the world, they can share information between each other, or they might all have the same perception of the environment due to a central information system (f.i. an overhead camera).
- **Goals:** In a MAS, the decision making is decentralised. This means that each agent makes its own decisions in order to achieve its goals. When dealing with multiple agents, these agents can either act as a team, working together, trying to achieve the same common goal, or they can each have their own goals and conflicting interests. In the former it is important to coordinate

actions in order to achieve the goal as best as possible using all available resources. In the latter, it is important to take fairness into consideration and attempt to find a good solution for the entire system and not just for a single agent.

- **Communication and interaction:** Agents can have many interactions with each other, communicating their own preferences, actions, learned models, ... or agents could act completely independent.

These distinctions in MAS could follow from the system in which the agents are acting, or could be a design choice depending on the application for which MAS are a solution.

Despite the added complexity that comes naturally with multi-agent systems, we can also list several benefits [Bond & Gasser (1988), Sycara (1998)]:

- **Speedup:** Due to the parallel computation, multi-agent systems can solve certain problems faster than a centralised sequential system.
- **Robustness:** Since several agents are acting in the same environment, possibly achieving the same goal, there is no single point of failure present in the system. If one agent fails, the system might undergo a performance drop, but the entire system will not fail.
- **Scalability:** Similar to how one failing agent might only have a negative impact on the performance of the system, adding additional agents to the systems might improve the global performance of the system.
- **Costs:** As with most technological advances, the cost of several systems is usually lower than the cost of one system with twice the power.
- **Maintainability:** It is easier to maintain a system, built up of several independent entities rather than a monolithic one.

In the previous chapter we focused on the theoretical framework and solution methods for learning optimal behaviour through trial-and-error interactions with the environment with only a single agent present. However, when multiple learners simultaneously apply reinforcement learning in a shared environment, the traditional approaches often fail.

In the multi-agent setting, the assumptions that are needed to guarantee convergence are often violated. Already in the most basic case where agents share a stationary environment and need to learn a strategy for a single interaction, many new complexities arise. Even when agent objectives are aligned and all agents try to maximise the same reward signal, 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, an equilibrium between agent strategies is usually looked for. In such an equilibrium, no agent can improve its payoff when the other agents keep their actions fixed.

When, in addition to multiple agents, we assume a dynamic environment which requires multiple sequential decisions, the problem becomes even more complex. Agents do not only have to coordinate, they also have to take into account the current (and possible future) state of their environment containing other agents. This problem is further complicated by the fact that agents typically have only limited information about the system. In general, they may not be able to observe actions or rewards of other agents, even though these actions have a direct impact on their own rewards and the environment in which they are acting. 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 some limited cases, the agents have access to all this information, but actually learning in a joint state-action space makes the problem much harder to solve, both computationally and in terms of the coordination required between the agents. In order to develop a successful multi-agent approach, all these issues need to be addressed. Figure 3.1 depicts a typical model of Multi-Agent Reinforcement Learning (MARL). Multiple agents are connected to an environment, which transitions to a new state, based on the combined effect of the actions of all the agents. It informs the agents of this new state, together with a (individual) reward signal for the past joint action.

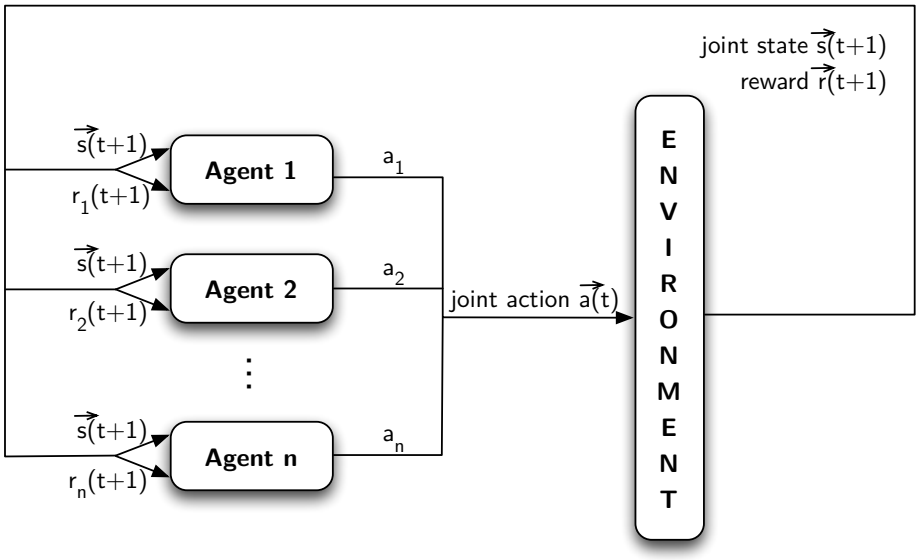


Figure 3.1: Standard model of multi-agent reinforcement learning [Nowé et al. (2011)].

The need for adaptive multi-agent systems, combined with the complexities of dealing with interacting learners has led to the development of the field of multi-agent reinforcement learning, which is built on two basic pillars: the reinforcement learning research performed within AI (See Chapter 2), and the interdisciplinary work on game theory. While early game theory focused on purely competitive games, it has since developed into a general framework for analyzing strategic interactions. It has attracted interest in fields as diverse as psychology, economics and biology. With the advent of multi-agent systems, it has also gained importance within the AI community. For a more broad overview of MAS and additional references, we refer to [Wooldridge (2002)] and [Vlassis (2009)]. For MARL we refer the interested reader to the overview papers by Shoham et al. [Shoham et al. (2003)] and Busoniu et al. [Busoniu et al. (2008)].

In the remainder of this chapter we will present a collection of approaches to model a MAS. Each of these frameworks is different in terms of the rewards the agents receive, how the actions are selected, and the view the agents have over of the system. After introducing these theoretical frameworks that are capable of representing the complexities of learning in MAS, we will shortly describe some well known existing MARL algorithms. After having introduced this background of multi-agent systems we will explain the current trend in MARL research. This new focus in MARL captures the dependencies that exist between agents in certain regions of the state space, and attempts to reduce the state-action space in which the agents have to learn. These dependencies are called sparse interactions. Finally, we will elaborate on novel algorithms that are able to learn and exploit these interactions.

3.2 Overview of multi-agent frameworks

The multi-agent systems considered in this chapter are characterised by strategic interactions between the agents. By this we mean that the agents in these systems are autonomous entities, which have individual goals and independent decision making capabilities, but which also are influenced by each others' decisions. We distinguish this case from the approaches that can be described as distributed or parallel reinforcement learning. In such systems multiple learners collaboratively learn a single policy. This includes approaches dividing the learning state space among agents [Steenhaut et al. (1997)], systems where multiple agents update the policy in parallel [Mariano & Morales (2001)] and swarm based techniques [Dorigo & Stützle (2004)]. Many of these systems can be treated as advanced exploration techniques for standard reinforcement learning and are still covered by single agent theoretical frameworks, such as the framework described in [Tsitsiklis (1994)], since the assumptions made there explicitly allow agents to use temporally outdated Q-values. The convergence of the algorithms remain valid as long as outdated information is eventually updated. For example, it allows to use outdated Q-values in

the max-operator in the right hand side of the standard Q-learning update rule (see Equation 2.10). This is particularly interesting if the Q-values belong to different agents exploring different parts of the environment. The systems covered by this chapter, however, go beyond the standard single agent theory, and as such require a different framework to model certain properties.

Over the years many taxonomies for MAS have been published. In the most recent one problems are classified according to four agent modelling dimensions: model of the agent and model of other agents, learning or non-learning agents, individual or group input, and cooperative or conflicting interests [Guttmann (2009)]. In this dissertation we are concerned with the interactions that occur between learning agents. An overview of multi-agent learning research based on strategic interactions between agents is given in Figure 3.2. The techniques listed are categorised based on the amount of information they require during the learning phase. On the horizontal axis is the information contained in the state space. On the vertical axis is shown how actions are being selected in the system. Near the origin, agents always select their actions independently, whereas at the top of the figure, agents select their actions in conjecture with each other. So, at the bottom left we have techniques that completely ignore other agents and only observe their own state information and select their actions without coordinating with other agents. These techniques have been described in Chapter 2. While designed for use in single agent learning, agents could also apply these techniques in a shared environment. However, as such, the influence of other agents is completely ignored and their convergence is not guaranteed. We will discuss this issue in Section 3.2.1.

At the top right we list techniques that use full joint state information and select their actions together either implicitly, or explicitly. Agents select their actions, based on some knowledge about the other agents, or they coordinate to select an action. At the bottom right, we list some techniques that use full joint state observability, but select their actions independently. These algorithms do not observe the actions of other agents. All these approaches lie at an extreme point, i.e. always observe the state and/or action information of other agents or never observe it. The main focus of this dissertation lies in between these approaches. In the following chapters we will describe several algorithms, that only use full joint state observability in situations where this complete state information is required, and select actions independently. These algorithms are marked in bold and red in the figure and are our own contributions. 2Observe will be explained in Chapter 4, CQ-learning will be described in Chapter 5 and FCQ-learning in Chapter 6. At the end of this chapter we describe two other related algorithms that also exploit the dependencies that exist among the agents in certain states: Utile Coordination [Kok et al. (2005)] and more recently Learning of Coordination [Melo & Veloso (2009)].

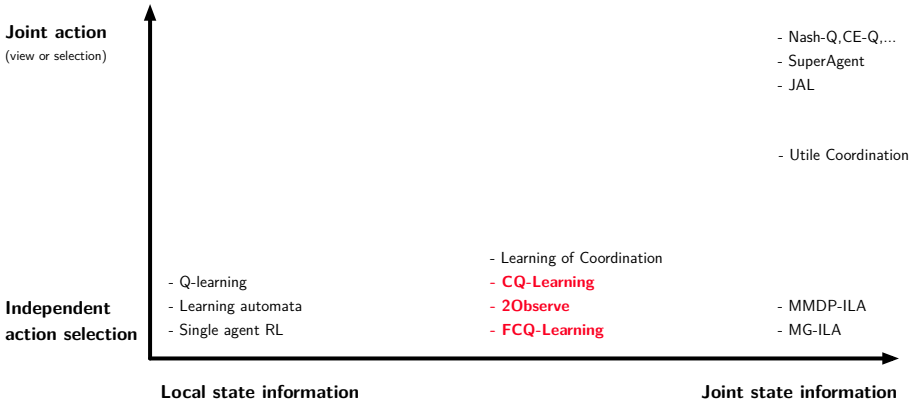


Figure 3.2: An overview of multi-agent research categorized by the strategic interactions.

We will begin by giving a thorough explanation of why the single agent MDP is no longer suitable for situations where multiple agents are acting in the same environment. This analysis is followed by the introduction of a framework for modelling multi-agent learning problems, called *Markov Games* in Section 3.2.2. This framework generalises the Markov Decision Process (MDP) setting usually employed for single agent RL. It considers both interactions between agents and a dynamic environment. Section 3.2.3 describes the current state of the art in MARL research, which takes the midground between independent learning techniques and Markov game techniques operating in the full joint-state joint-action space.

3.2.1 Single agent MDPs

The main problems of using the single agent MDP for environments where multiple agents are present, is that the Markov property no longer holds and that environments seem to be non stationary to the agent. Algorithms based on the MDP framework, such as independent Q-learning may still be applied, but the results may vary, depending on the number of interactions with other agents. To illustrate this we performed some experiments with multiple agents in a small environment where interactions occur very often. The environment is illustrated in Figure 3.3 [Greenwald & Hall (2003)]. Two agents have to reach the same goal, marked by the coloured dot, without colliding into any walls or into each other from their start position, marked by the **X**. Agents are colliding with each other if they attempt to move to the same cell at the same time. The state information for each agent, contains only their current location. A penalty of -10 is given for colliding with the other agent, -1 for every action and $+20$ for reaching the goal.

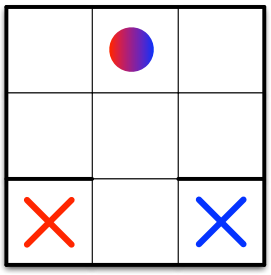
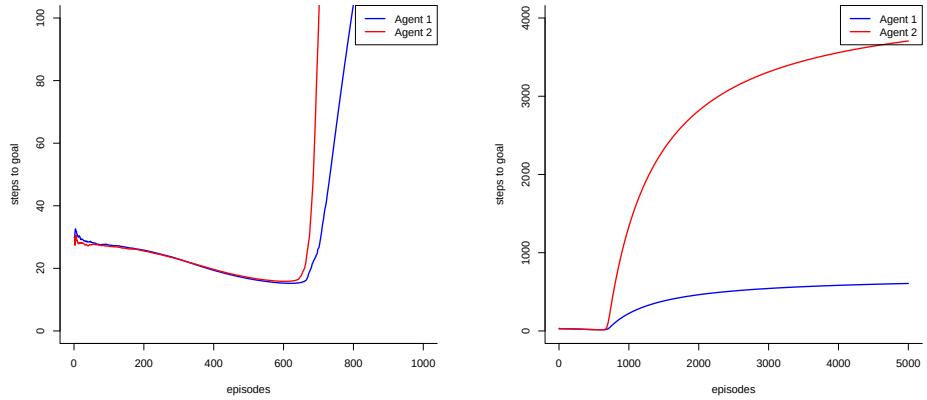


Figure 3.3: Grid game 2 from [Greenwald & Hall (2003)]. A game with a high number of strategic interactions between the agents.

The algorithm used for this experiment was Q-learning, with a learning rate of 0.1 and a softmax exploration strategy with a decreasing temperature. This selection strategy was selected because it obtained the best results in our bandit experiments in single agent environments (See Section 2.4). The value for the temperature was as follows: $\tau = max(100 * 0.99^{timestep}, 0.05)$.
50 independent runs of the algorithm were performed and the results presented here are the averages over these runs.

In Figure 3.4 we show the number of steps both agents needed to complete an episode, i.e. reach the goal state. On the left hand side of this figure we see that initially the learning process is optimising the performance of both agents and the number of steps to reach the goal is decreasing. So slowly, agents are converging to the shortest path towards the goal. As can be seen from Figure 3.3, the shortest path to the goal of both agents coincides. Thus, as learning progresses, the agents start colliding more often and learn not to take the action that will lead them directly to the goal. On the right hand side we illustrate this effect over $5,000$ episodes. Since agents are still exploring a little, they still manage to reach the goal eventually. In RL however, it is common to turn off the exploration after some time and only play the policy to which the agents have converged. In this very simple scenario this would however have a terrible impact, since neither agent would ever reach the goal anymore.

Agent 1 needs considerably less steps than Agent 2. This can be explained by the fact that the Q-value of its action that would lead it in the direction to the goal is relatively still a bit better than the actions that would lead towards a boundary in the grid and hence is selected slightly more often by the softmax strategy. Using an ϵ-greedy strategy would allow agents to have a similar performance and, depending on the value of ϵ, even lead to much better solutions. These results are shown in Figure 3.5 with $\epsilon = 0.9$. Agents learn the same Q-values, but their exploration rate remains constant over time. Moreover, the action that leads them in the direction of the goal is selected with an equal probability as the other suboptimal actions.



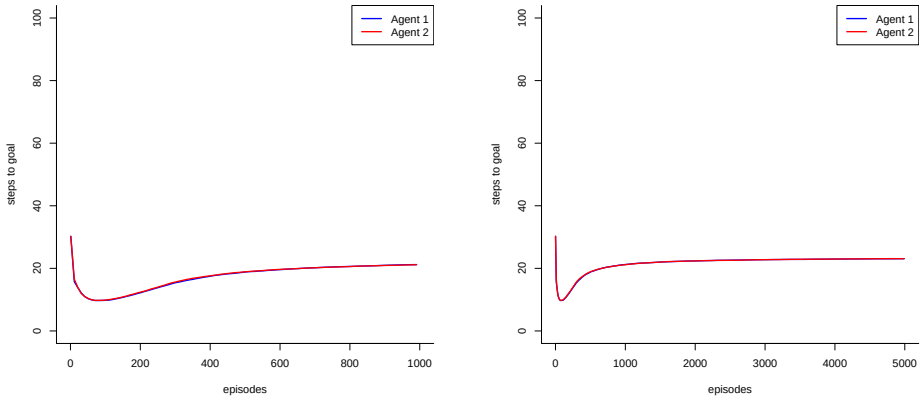
(a) (b)

Figure 3.4: The number of steps needed for both agents to complete an episode in the Grid game 2 environment using a softmax exploration strategy with $\tau = max(100 * 0.99^{timestep}, 0.05)$. (a) shows the first $1,000$ episodes, (b) shows the results over $5,000$ episodes.

One could add a `wait` action to allow one agent to go through the small passage before the other, but since agents cannot differentiate on the position of the other agent, this will result in both agents converging to their `wait` action in their initial position and never reach the goal.

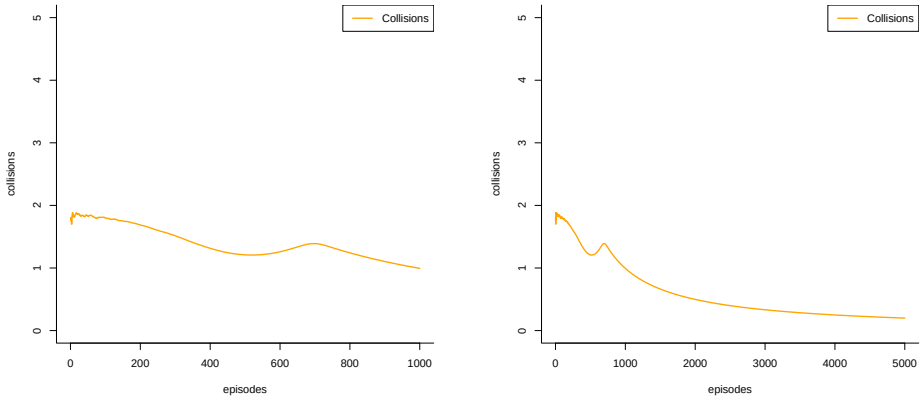
Figure 3.6 shows the average number of collisions that occurred during the learning process using a softmax strategy. On the left hand side we see the progress over the first $1,000$ episodes. Initially this number was decreasing, until the agents started to converge towards the shortest path around the 700th episode, where there is a sudden increase in the number of collisions. This resulted in low Q-values for the only action that leads towards the goal. Both agents preferred the three actions that caused them to bump into a wall, since the penalty for this was lower than for bumping into each other. On the right hand side we show the results over $5,000$ episodes. The number of collisions eventually goes to zero, but this comes with a huge overhead in the number of steps the agents take to go to the goal in this very small environment. If the agents would play purely greedy at the end, they would never collide anymore, but they would also no longer reach the goal.

When using an ϵ-greedy strategy agents still collide and never learn to completely avoid collisions. The results for $\epsilon = 0.9$ are shown in Figure 3.7. What agents gain in terms of the number of steps, they loose again in terms of the number of collisions. Overall an ϵ-greedy strategy is less extreme than the softmax strategy. For this



(a) (b)

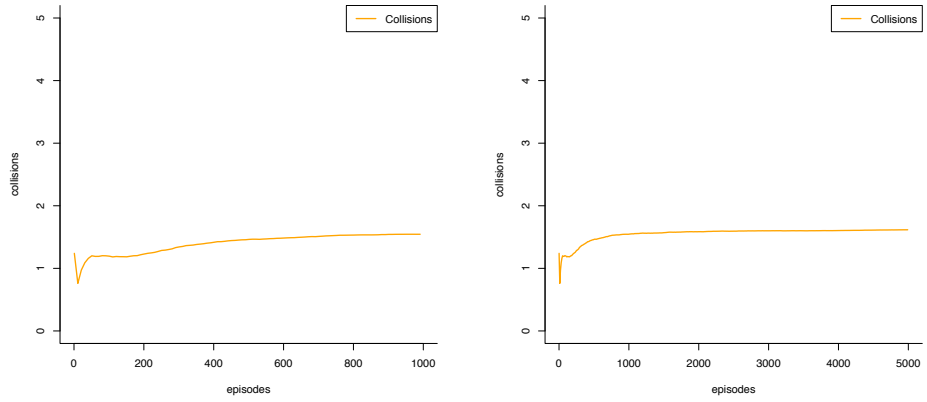
Figure 3.5: The number of steps needed for both agents to complete an episode in the Grid game 2 environment using an ϵ-greedy exploration strategy ($\epsilon = 0.9$). (a) shows the first $1,000$ episodes, (b) shows the results over $5,000$ episodes.



(a) (b)

Figure 3.6: The number of collisions per episode in the Grid game 2 environment using a softmax exploration strategy with $\tau = max(100 * 0.99^{timestep}, 0.05)$. (a) shows the first $1,000$ episodes, (b) shows the results over $5,000$ episodes.

reason we will use this exploration strategy when using independent learners in this dissertation.



(a) (b)

Figure 3.7: The number of collisions per episode in the Grid game 2 environment using an ϵ-greedy exploration strategy ($\epsilon = 0.9$). (a) shows the first 1000 episodes, (b) shows the results over 5000 episodes.

These experiments clearly show that using single agent algorithms in settings where the behaviour of agents influences the rewards of the other agents, is far from optimal. Different results can be obtained with other exploration strategies and action selection mechanisms, but the behaviour of such single agent algorithms remains unpredictable and completely unsuitable for this kind of multi-agent environments. Such environments require more informed algorithms that use more information about the other agents in the environment.

3.2.2 Markov games and Decentralised MDPs

As shown above, an agent cannot simply ignore the presence of other agents and thus we can no longer rely on single agent MDPs as a framework. An extension of the single agent MDP to the multi-agent case is straightforward and can be defined by Markov games. In a Markov game (MG), actions are the joint result of multiple agents choosing an action independently, based on the state information of the entire system (i.e. all the agents).

Definition 12. *A **Markov game** is a tuple* $(n, S, A_{1,\ldots,n}, T, R_{1,\ldots,n})$*:*

- 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*.*

- $T : S \times A_1 \times \ldots \times A_n \to \mu(S)$ *the transition function.*
- $R_k : S \times A_1 \times \ldots \times A_n \times S \to \mathbb{R}$*, the reward function of agent* k*.*

Note that $A_k(s^i)$ is now the action set available in state s^i to 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 current state s^i, next 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. A special case of this framework in which all agents share the same reward function is called a *Multi-agent Markov Decision Process* (MMDP) [Boutilier (1996), Claus & Boutilier (1998)]. This special case is used to model fully cooperative multi-agent tasks.

As was the case in MDPs, agents try to optimise some measure of their future expected rewards. In this dissertation we are concerned with systems that try to maximise their future discounted reward. The main difference to single agent RL is that now these criteria also depend on the policies of other agents. This results in the following definition for the expected discounted reward[1] for agent k under a joint policy $\vec{\pi} = (\pi_1, \ldots, \pi_n)$, which assigns a policy π_k to each agent k:

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

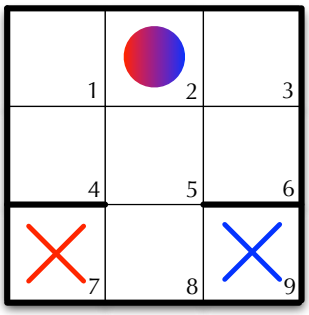
It is clear that learning in a Markov game introduces several new issues over learning in MDPs with regard to the optimal policy that should be learned. In an MDP, it is possible to prove that, given some basic assumptions, an optimal deterministic policy that maximises the reward, always exists. In Markov games however, it is in general impossible to maximise this criterion for all agents simultaneously since they can all have a different reward function. Therefore, we typically rely on equilibria as the solution concept for these problems. The best response equilibrium for instance calculates the best course of actions to take, given the policy of the other agents. If all agents adopt this strategy, they are playing according to a pure *Nash equilibrium*. This means that for none of the agents another policy exists which gives a higher expected future reward, provided that the other agents keep their policies fixed. However, this only holds if, like in a single agent MDP, it is sufficient to consider only those policies which deterministically map each state to an action. It is however possible in MG that the equilibria can only be reached using

[1] Another measure of the future expected reward is for instance the average reward method. For agent k this is defined as:

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

stochastic policies. As such, it is not sufficient to let agents map a fixed action to each state: they must be able to learn a mixed strategy. The situation becomes even harder when considering other reward criteria, such as the average reward, since then it is possible that no equilibria in stationary strategies exist [Gillette (1957)]. This means that in order to achieve an equilibrium outcome, the agents must be able to express policies which condition the action selection in a state on the entire history of the learning process. Fortunately, one can introduce some additional assumptions on the structure of the problem to ensure the existence of stationary equilibria [Sobel (1971)].

In a Markov Game it is assumed that the agents have continuous implicit communication, since each agent is aware of the complete state (including inherently internal state information of other agents), as well as the actions performed by the agents at every time step. This is sometimes called a superagent view of the entire system. Having access to all this information is appealing from an equilibrium learning point of view, but is rarely the case in practical situations. Moreover, such a view looses one of the most appealing features of multi-agent systems, i.e. their decentralised control. A more general framework, with which it is possible to model the state information the agents have access to is a *Decentralised Markov Decision Process* (DEC-MDP) [Bernstein et al. (2002), Becker et al. (2004)]. In this model, at each step, each agent receives local information and chooses an action based on this local information. It cannot observe the actions of the other agents or their rewards. Similar to a MG however, the transitions and rewards received depend on the vector of actions of all agents and the combination of the information of all agents, uniquely describes the state the system is currently in. The difference between a MG and a DEC-MDP is illustrated in Figure 3.8. On the left hand side is the MG view on the system, in which agents observe the state and actions of all agents in the system. On the right hand side is the DEC-MDP view in which agents only observe their own state and actions. The underlying dynamics of a MG and a DEC-MDP are the same, but the information provided to the agent is different. In a DEC-MDP, the agents share the same reward function, so in the example depicted, Agent 1 receives a penalty because Agent 2 collided against a wall.



Markov Game

Observations of Agent 1 = Observations of Agent 2

State	=	⟨7,9⟩
Actions	=	⟨EAST,NORTH⟩
Rewards	=	⟨0,-1⟩
New state	=	⟨8,9⟩

DEC-MDP

Observations of Agent 1

State	=	⟨7⟩
Action	=	⟨EAST⟩
Reward	=	⟨-1⟩
New state	=	⟨8⟩

Observations of Agent 2

State	=	⟨9⟩
Action	=	⟨NORTH⟩
Reward	=	⟨-1⟩
New state	=	⟨9⟩

Figure 3.8: Left hand side: MG view of the system. Right hand side: DEC-MDP view of the system.

Formally a DEC-MDP is defined as follows:

Definition 13. *A Decentralised Markov Decision Process is a tuple* $(n, S, A_{1,\ldots,n}, T, R, \Omega, O)$*, with:*

- n *the number of agents in the system,*
- $S = S_0 \times S_1 \times \ldots \times S_n$ *the set of joint states, where* S_k *indicates the set of states of agent* $k = 1, \ldots, n$ *and* S_0 *contains all external features*[2]*,*
- A_k *= the action set of agent* k*,*
- $T : S \times A_1 \times \ldots \times A_n \to \mu(S)$ *the transition function.*
- $R : S \times A_1 \times \ldots \times A_n \times S \to \mathbb{R}$ *the reward function,*
- $\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_n$ *a finite set of joint observations, where* Ω_k *is the set of observations for agent* k*,*
- $O : S \times A \times S \times \Omega \to \mathbb{R}$ *the observation function.* $O(o_i|s, (a_1, \ldots, a_n), s')$ *is the probability of making observation* $o_i \in O$ *when taking joint action* $(a_1, \ldots, a_n)$ *in state* s *and transitioning to state* s' *as a result.*

[2] We factored the state space to separate the state features of the agents and the external state features [Becker et al. (2003)].

We can also define the following property of a DEC-MDP:

Definition 14. *A DEC-MDP is fully joint observable if there exists a mapping $J : \Omega_1 \times \ldots \times \Omega_n \rightarrow S$, such that whenever $O(o_k|s, (a_1, \ldots, a_n), s')$ is nonzero for all agents k, then $J(o_1, \ldots, o_n) = s'$.*

If a DEC-MDP only has local full observability every agent can only determine its own local state unambiguously, from its local observations. To fully understand what this means, we must explain what we mean by system state and local state.

Definition 15. *We assume that the state space S can be factored as $S = S_0 \times S_1 \times \ldots \times S_n$. Each **system state** s is a tuple $(s_0, s_1, \ldots, s_n)$, with $s_i \in S_i, i = 0, \ldots, n$. This system state contains all the information regarding the current internal state in which the agents are $(S_i, i = 1, \ldots, n)$, as well as the external information about the environment(S_0).*
*The **local state** of an agent k contains all the internal information about the current state of the agent (S_k), as well as the external information about the environment (S_0).*

If we go back to our example of Figure 3.8, S_0 is empty, S_1 contains the locations of agents 1. Similar, S_2 contains the locations of agents 2. The system state is defined as $\langle s_1, s_2 \rangle$. So a MG always uses the system, whereas in the DEC-MDP view, both agents only have access to their local states $\langle s_1 \rangle$ and $\langle s_2 \rangle$ respectively. Since agents do not share their observations (which are the same as their local states in this example), this DEC-MDP only has local full observability.

In this dissertation we adopt the following notation: s is the system state and contains all the information about all agents and the environment. s_k is the local state of agent k and contains the internal information of agent k, together with the information about the environment. s_{-k} contains the system state information without the information of agent k. Similar, a is the joint action, a_k the action of agent k and a_{-k} the joint action without the action of agent k. s_K is the joint state information for a set of agents K, containing their internal information, as well as the environmental information. Note that if $|K| = n$, then $s_K = s$.

A special case of DEC-MDPs in which agents have individual reward functions is called a Decentralised Markov Game (DEC-MG) [Aras et al. (2004)].

Definition 16. *A Decentralised Markov Game is a tuple $(n, S, A_{1,\ldots,n}, T, R_k, \Omega, O)$, where $n, S, A_{1,\ldots,n}, T, \Omega$ and O are defined as in Definition 13 and $R_k : S \times A_1 \times \ldots \times A_n \times S \rightarrow \mathbb{R}$ is the reward function of agent k. $R_k(s, (a_1, \ldots, a_n), s')$ is a real number representing the reward obtained by agent k from taking joint action $(a_1, \ldots, a_n)$ in state s and transitioning to state s'.*

Another special case is a *Transition Independent DEC-MDP* (TI-DEC-MDP). A TI-DEC-MDP is a subclass of the general DEC-MDP in which the state transitions are only dependent on the local state and action of an agent and not on the entire system state and joint actions. This formalism was introduced in [Becker et al. (2004)] to exploit a certain independence between the agents. Note that the agents still share the same reward function. This independence relaxes the complexity requirements to NP-complete in finite horizon settings, where general DEC-MDPs are shown to be NEXP-complete [Goldman & Zilberstein (2004)].

If we assume that each agent has full observability and that the DEC-MDP is fully joint observable, the DEC-MDP becomes a MMDP (or a MG in the case of a DEC-MG). However, in many of these joint states, the outcome of the transition function is only dependent on the action of one agent and independent of the local state/action of the other agents. This means that in many states, we are actually dealing with a TI-DEC-MDP. Situations where this independence assumption does not hold, are called sparse interactions. In the following section we discuss a formal framework for modelling these interactions. This framework lies between DEC-MDPs and TI-DEC-MDPs and models the transition and reward dependencies that occur in certain regions of the state space. As such this allows for a sparse representation of the joint state space, which is then only considered during those sparse interactions.

3.2.3 Sparse interactions

In many multi-agent environments, agents only need to interact with each other in specific regions of the state space because in these regions the agent experience influence of other agents, i.e. they are not independent of each other. These independencies that occur in certain states are *sparse interactions* [Melo & Veloso (2010)b].

Definition 17. *In a DEC-MDP, an agent k is independent of agent l in a state $s \in S$ if the following conditions hold at state s:*

- *The transition probabilities for the local state of agent k at s do not depend on the state/action of agent l:*

$$\begin{aligned} & P[s_k(t+1) = v_k \quad | \quad s(t) = u, a(t) = b] \\ = & P[s_k(t+1) = v_k \quad | \quad s_{-l}(t) = u_{-l}, a_{-l}(t) = b_{-l}] \end{aligned}$$

- *It is possible to decompose the reward function, such that the reward signal both agents experience is independent of each others actions at state s.*

So intuitively this means that an agent k depends on the state information or the action of another agent l in a particular state s if either the transition

function cannot be decoupled from the state information or action of agent l or the reward cannot be decomposed such that agent k does not experience any influence from the actions of agent l at s. An important remark concerning this definition must be made. First, these interactions are dependent on a particular agent in a particular state. This means that state s is not necessarily an interaction state for any other agent than agent k, although it certainly can be. Moreover, if there is a dependency between agent k and agent l at state s, it does not mean that there is also a dependency between agent l and agent k at state s. This relation is not symmetrical.

Given these rules for independence, sparse interactions are defined as follows:

Definition 18. *In a DEC-MDP a set of agents K interact at state s if the following conditions simultaneously hold:*

- *If agent $k \in K$ and agent k depends on agent l in state s, then agent $l \in K$*
- *If agent $l \in K$ and $\exists$ agent k, such that agent k depends on agent l, then agent $k \in K$*
- *There is no strict subset $K' \subset K$, such that the above conditions hold for K'.*

If a set of agents K interact in state s, we refer to s_K as an interaction state for the agents in K.

Following from the definition of interaction states, we can now define interaction areas:

Definition 19 (Interaction area)**.** *An **interaction area** S^I is a set of joint states for which the following conditions hold:*

1. *$S^I \subset S_K$ for some set of agents K, where S_K is the joint state space of the agents in K,*
2. *$\exists s \Rightarrow s \in S^I$ and s is an interaction state for the agents in K,*
3. *For any $s^i \in S^I, a^i_k \in A_k$ and $s^j \notin S^I$,*

$$P[S_K(t+1) = s^j | S_K(t) = s^i, \vec{a}_K(t) = \vec{a}^i_K] = \prod_{k \in N} T(s^i_k, a_k, s^j_k)$$

4. *the set S^I is connected. This means that for every pair of states $x, y \in S^I$, there exists a sequence of actions, such that y is reachable from x (or vice versa), through a set of states $U \in S^I$.*

The introduction of such interaction states leads to a specialisation of a DEC-MDP, in which this local dependency between agents is represented. This specialisation is called a *Decentralised Sparse Interaction MDP* (DEC-SIMDP). Intuitively

such a DEC-SIMDP is a collection of single agent MDPs for states that are not a member of an interaction area and a collection of MMDPs containing the states of the interaction areas.

Definition 20 (DEC-SIMDP)**.** *A N-agent Decentralised Sparse Interaction MDP with a set of L interaction areas $\{S_1^I, \ldots, S_L^I\}$ is a tuple of :*

$$\Gamma = \left(M^k, (M^{I,l}, S^{I,l})\right), \text{with } k \in 1, \ldots, L \text{ and } l \in 1, \ldots, L$$

where

- *each M^k is an MDP, $M^k = (S_k, A_k, T_k, R_k)$, that individually models agent k in the absence of other agents,*
- *each $(M^{I,l}, S^{I,l})$ is a MMDP that represents a local interaction between K agents in the states of $S^{I,l}$. For this joint state space representation holds: $S_K^I \subset S$: $M^{I,l} = (|K|, S_K, A_{1,\ldots,|K|}, T, R^I)$, where R^I represents the reward function for the agents in the interaction.*

Once again, we could extend this definition to a Decentralised Sparse Interaction Markov Game (DEC-SIMG) for the situation where agents do not share the same reward function.

3.2.4 Discussion

In Table 3.1 we list the information that is contained in the system state and in the local state for the multi-agent taxi domain problem given in Example 2 [Ghavamzadeh & Mahadevan (2004)].

Example 2: *Consider the multi-agent taxi domain problem of [Ghavamzadeh & Mahadevan (2004)] shown in Figure 3.9. The environment is inhabited by two taxis. As in Example 2.1, passengers appear at one of the coloured cells and wish to be transported to one of the other cells. The goal is to increase the throughput of the system, i.e. carry as many passengers from their origin to their destination in a predefined amount of time. Multiple agents can obtain a better result than one agent alone, but if both agents try to pick up the same passenger, precious time is lost.*
The state variables for this domain are the locations of the taxis (25 possibilities each), the status of the taxi (carrying a passenger or not), the status of the pickup locations (passenger waiting or not for each coloured cell), the desired destination of the possible passenger for one of the coloured cells (one of the other three or n.a.) and the destination where the taxi is heading (one of the coloured cells or n.a.). The complete size of the state space of this problem is $25^2 \times 2^2 \times 5^2 \times 2^4 \times 4^4 = 256,000,000$.

The actions the agents can take are the same as for the single agent variant: four navigation actions that move the taxi one cell `North`, `East`, `South`, `West`, *from its current location, a* `Pickup` *action and a* `Putdown` *action.*

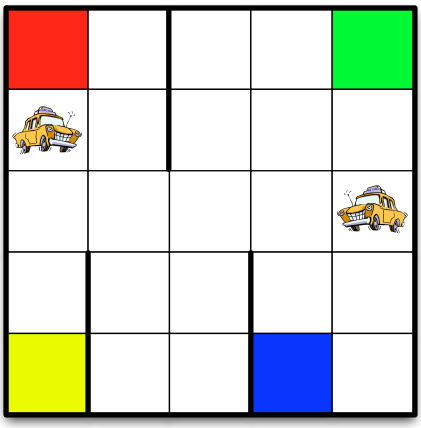


Figure 3.9: The multi-agent Taxi domain problem [Ghavamzadeh & Mahadevan (2004)].

The numbers listed are based on a situation in which two taxis are present in the environment and only one passenger. If we model this problem as an MDP, the agent could observe over one million distinct states. However, since only a single agent is modelled, but two agents are changing the environment, the environment becomes non-stationary and this description is not Markovian. Modeling it as an MG, results in a total of 256 million states, since this contains the location, status, and destination of both taxis. If the problem is modeled as a DEC-MDP, the local state is not necessary equal to the system state. The agent could for instance only have information about its own location, next to the position and destination of the passengers, or, if the system is fully joint observable, it could have access to the same information as in an MG. We did not list any exact numbers but only a lower and upper bound for the last framework we discussed, DEC-SIMDPs, since for this framework the number of states depends on the interaction area. One could for instance define the interaction area to contain only those states in which both taxis their status is empty. It is reasonable to assume that if a taxi is already transporting a passenger, it should not know about the actions and intentions of the other taxis in the system. On the other hand if the taxis are empty, they could coordinate in order to decide which agent should pick up which passenger. This would prevent the loss of time and resources if both taxis were to try to pick up the passenger and at the same time reduce the state space in which the agents are trying to learn,

since they only observe the other agent in those situations where they need to.

In the next section we will first give a short overview and references to some solution methods for multi-agent problems modelled as an MG. Second, we will describe some algorithms that explicitly exploit the sparse interactions that exist in the multi-agent environment.

State information		MDP	MG	DEC-MDP	DEC-SIMDP
System state	Taxi position	▪	▪	▪	▪
	Taxi status	▪	▪	▪	▪
	Taxi destination	▪	▪	▪	▪
	Other taxi pos		▪	▪	▪
	Other taxi status		▪	▪	▪
	Other taxi dest		▪	▪	▪
	Passenger pos	▪	▪	▪	▪
	Passenger dest	▪	▪	▪	▪
#States $\times 10^3$		1,024	256,000	256,000	256,000
Local state	Taxi position	▪	▪	▪	▪
	Taxi status	▪	▪	▪	▪
	Taxi destination	▪	▪	▪	▪
	Other taxi pos		▪	■	▲
	Other taxi status		▪	■	▲
	Other taxi dest		▪	■	▲
	Passenger pos	▪	▪	▪	▪
	Passenger dest	▪	▪	▪	▪
#States $\times 10^3$		1,024	256,000	1,024 or 256,000	Between 1,024 and 256,000

Table 3.1: Table representing what information is contained in the system state and in the local state for the multi-agent taxi domain problem under different multi-agent models. ▪ indicates that this information is completely encapsulated in this state, ■ indicates that this information is either always available in this state or never, ▲ indicates that this information is sometimes available in this state.

3.3 Solution methods

Even though some successes are known for single agent RL in multi-agent environments [Sen et al. (1994), Claus & Boutilier (1998)], as shown in the previous section care has to be taken when applying such methods to multi-agent settings. We followed this discussion by an overview of several multi-agent frameworks. Most multi-agent learning algorithms are based on the MG framework. This means that they have full information about the complete system state. In this framework we distinguish algorithms that learn actions independently and algorithms that learn joint actions.

Examples of the former are algorithms based on learning automata, such as *MMDP-ILA* and *MG-ILA* for cooperative and conflicting interest environments, respectively. MMDP-ILA is guaranteed to converge to global optimal points of the MMDP [Vrancx et al. (2007)], whereas MG-ILA will converge to a pure equilibrium point between the agent policies [Vrancx et al. (2008)]. Other examples of algorithms that independently learn actions are Policy Search [Peshkin et al. (2000)] and Policy Gradient [Könönen (2003)b]. Neither of these provide the same convergence guarantees as MMDP-ILA or MG-ILA.

Examples of the latter are mostly variants of Q-learning for multi-agent settings that attempt to learn an equilibrium policy or make assumptions about the strategies of the other agents. *Nash-Q* is an example of an algorithm that learns an equilibrium policy [Hu & Wellman (1998)]. It observes the rewards for all agents and keeps estimates of Q-values, not only for the learning agent, but for all agents. As such, the joint action selection in each state can be seen as a game, where the entries in the payoff matrix are the Q-values of the agents for the joint action. Nash-Q will then assume that all agents in the game play according to a Nash Equilibrium of this game. This principle can also be applied in combination with other equilibrium concepts, such as correlated equilibria [Greenwald & Hall (2003)] or the Stackelberg equilibrium [Könönen (2003)a].

Algorithms that make assumptions about the policy of the other agents are for instance *minimax Q-learning* and *friend-or-foe Q-learning*. Minimax Q-learning is restricted to zero-sum games and attempts to learn the best response to the opponents' best action [Littman (1994)]. This approach was later expanded to general-sum games in which the agent tries to determine whether the other agents are friends or foes [Littman (2001)a]. Based on this distinction either the max or the min operator is used in the update rule of the Q-values.

Other well known Q-learning variants are *Team Q-learning*, in which agents individually learn the optimal joint action for every state [Littman (2001)b], and *Joint-Action Learners* [Claus & Boutilier (1998)], which also learns these joint actions, but also maintains beliefs about the strategies of the other agents for its exploration strategy.

All the techniques above are aimed at coordinating actions, either explicitly or implicitly. To do so, they always observe the entire joint-state space. As mentioned previously, this state space is exponential in the number of agents (so is the action space) and thus intractable for most problems. The trend in multi-agent reinforcement learning research seems to be to explore when these other agents have to be observed. In the following section we introduce two novel approaches that adopt this principle.

3.3.1 Sparse multi-agent view

In Section 3.2.3 we have introduced the DEC-SIMDP framework which models the dependencies that exist between agents in a subset of the entire joint state space. Several planning algorithms that exploit these sparse interactions exist [Spaan & Melo (2008), Melo & Veloso (2010)a]. We will however not focus on these planning algorithms, but describe two learning approaches: *Utile Coordination* (UC) [Kok et al. (2005)] and *Learning of Coordination* (LoC) [Melo & Veloso (2009)].

3.3.1.1 Utile Coordination

Kok & Vlassis are interested in using a sparse representation for the joint action space of the agents. More specifically they are interested in learning joint-action values for those states where the agents explicitly need to coordinate. In many problems, this need only occurs in very specific contexts [Guestrin et al. (2002)b]. *Sparse Tabular Multiagent Q-learning* has a list of states at its disposal in which coordination is necessary. In these states, agents select a joint action, whereas in all the uncoordinated states they all select an action individually [Kok & Vlassis (2004)b]. By replacing this list of states by coordination graphs (CG) it is possible to represent dependencies that exist only between some agents [Guestrin et al. (2002)a, Kok & Vlassis (2004)a, Kok & Vlassis (2006)]. This technique is known as *Sparse Cooperative Q-learning* (SCQ) Figure 3.10 shows a graphical representation of a simple CG for a given situation where the effects of the actions of agent 4 depend on the actions of agent 2 and the actions of agent 2 and 3 both depend on the actions of agent 1, so the nodes represent the agents, while an edge defines a dependency between two agents. If agents transitioned into a coordinated state, they applied a variable elimination algorithm to compute the optimal joint action for the current state. In all other states, the agents select their actions independently.

In later work, the authors introduced *Utile Coordination* [Kok et al. (2005)]. This is a more advanced algorithm that uses the same idea as SCQ, but instead of having to define the CGs beforehand, they are being learned online. This is done by maintaining statistical information about the obtained rewards conditioned on the states and actions of the other agents. As such, it is possible to learn the context specific dependencies that exist between the agents and represent them in a CG.

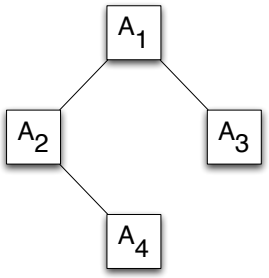


Figure 3.10: Simple coordination graph. In the situation depicted the effect of the actions of agent 4 depends on the actions of agent 2 and the actions of agent 2 and 3 both depend on the actions of agent 1.

This technique is however limited to fully cooperative MAS.

Care has to be taken in the update rule of the Q-values when dealing with sparse interactions. Since the system can transition between all combinations of coordinated and uncoordinated states. The authors used the following approach. When transitioning between two uncoordinated states, the independent action Q-values can easily be updated. Similar for the transition between coordinated states, the joint Q-values are used in the bootstrapping process. When moving from a coordinated state to an uncoordinated state, the independent Q-values are used for the backup of the joint Q-values. Conversely, when the agent goes from an uncoordinated state to a coordinated state, the joint Q-values are used in the backup process of the independent Q-table.

The primary goal of these approaches is to reduce the joint-action space by only learning (or computing) joint actions in interaction states. However, all the algorithms described above, always employ a complete multi-agent view of the entire joint-state space to select their actions, even in states where only using local state information would be sufficient. As such, the state space in which they are learning is still exponential in the number of agents, and its use is limited to situations in which it is possible to observe the entire joint state.

3.3.1.2 Learning of Coordination

Spaan and Melo approached the problem of coordination from a different angle [Spaan & Melo (2008)]. They introduced a new model for multi-agent decision making under uncertainty called *interaction-driven Markov games* (IDMG). This model contains a set of interaction states which lists all the states in which coordination should occur.

In later work, Melo and Veloso [Melo & Veloso (2009)] introduced an algorithm where agents learn in which states they need to condition their actions on the local

state information of other agents to reduce the joint state space in which agents are learning. As such, their approach can be seen as a way of solving an IDMG where the states in which coordination is necessary is not specified beforehand. To achieve this they augment the action space of each agent with a pseudo-coordination action (`COORDINATE`). This action will perform an active perception step. This could for instance be a broadcast to the agents to divulge their location or using a camera or sensors to detect the location of the other agents. This active perception step will decide whether coordination is necessary or if it is safe to ignore the other agents. Since the penalty of miscoordination is bigger than the cost of using the active perception, the agents learn to take this action in the interaction states of the underlying IDMG. This approach solves the coordination problem by deferring it to the active perception mechanism.

The active perception step of LoC can consist of the use of a camera, sensory data, or communication to reveal the local state information of another agent. As such the outcome of the algorithm depends on the outcome of this function. Given an adequate active perception function, LoC is capable of learning a sparse set of states in which coordination should occur. Note that depending on the active perception function, this algorithm can be used for both cooperative as conflicting interest systems.

As in Utile Coordination, the authors use a variation on the standard Q-learning update rule:

$$Q_k^C(s,a) \leftarrow (1-\alpha(t))Q_k^C(s,a) + \alpha(t)\left[r_k + \gamma \max_{a'} Q_k(s'_k, a'_k)\right] \tag{3.2}$$

Where Q_k^C represents the Q-table containing states in which agent k will coordinate and Q_k contains the state-action values for its independent states. So the update of Q_k^C uses the estimates of Q_k. This represents the one-step behaviour of the `COORDINATE` action. This allows for a sparse representation of Q_k^C, since there is no direct dependency between the states in this joint Q-table. The pseudo code for this technique is given in Algorithm 5. π_e stands for a policy that ensures enough exploration (such as an ϵ-greedy policy[3]) and π_g stands for the greedy policy. $\hat{A}_k$ is the action set of agent k, without the `COORDINATE` action. We will refer to this technique in the remainder of this dissertation as LoC (Learning of Coordination).

[3] See Section 2.3

Algorithm 5 Learning of Coordination (LoC)

1: Initialise Q_k and Q_k^C;
2: Set $t = 0$;
3: **while forever do**
4: Choose $A_k(t)$ using π_e
5: **if** $A_k(t) =$ `COORDINATE` **then**
6: **if** $ActivePercept() =$ `TRUE` **then**
7: $\hat{A}_k(t) = \pi_g(Q_k^C, S(t))$;
8: **else**
9: $\hat{A}_k(t) = \pi_g(Q_k^*, S_k(t))$;
10: **end if**
11: Sample $R_k(t)$ and $S_k(t+1)$;
12: **if** $ActivePercept =$ `TRUE` **then**
13: `QLUpdate`$(Q_k^C; S(t), \hat{A}_k(t), R_k(t), S_k(t+1), Q_k)$;
14: **end if**
15: **else**
16: Sample $R_k(t)$ and $S_k(t+1)$;
17: **end if**
18: `QLUpdate`$(Q_k; S(t), \hat{A}_k(t), R_k(t), S_k(t+1), Q_k)$;
19: $t = t + 1$
20: **end while**

where `QLUpdate`$(Q; s; a; r; s'; Q')$ is equivalent to

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q'(s', a')) \tag{3.3}$$

3.4 Summary

In this chapter we describe the problems an agent faces when learning in a multi-agent environment. If the agents are acting in the same environment, using only a local view of the state space, the environment seems non-stationary to the agents. A naive solution is to include all the information about the other agents. Such an approach can be modelled using the Markov games framework. This framework allows for multi-agent extensions of Q-learning, such as Nash Q-learning or Correlated Q-learning, to learn equilibrium policies. However, providing the agents with all the information about all the agents results in an exponential explosion of the size of the state space in which agents are learning and is unsuitable for all but the smallest environments with a limited number of actions. Next to being computationally intensive, none of these techniques have convergence guarantees. Techniques based on LA, such as MMDP-ILA and MG-ILA, have these guarantees

under some general assumptions, but are still learning in a complete joint state space. Other frameworks, that are more realistic from a multi-agent point of view are based on decentralised Markov decision processes. In this framework, agents have a local view of the environment, and the combination of the views of the agents constitute the state in which the system is currently.

An important extension on which we focus is a decentralised sparse interaction Markov decision process, which models the interactions and dependencies that occur between certain agents in the environment. Agents have a local view in most states, but can observe other agents in particular regions of the state space, called interaction areas, in which the state transition function does not only depend on an agent's own local state and local action. We describe two algorithms, Utile Coordination and Learning of Coordination which exploit these interaction dependencies. The former uses coordination graphs to represent these dependencies and selects actions using a variable elimination algorithm. The latter uses joint state information in those situations where an external perception mechanism informs the agent when coordination is beneficial. This could for instance be a broadcast to detect the presence of other agents, or using an overhead camera.

In the next chapter we propose a new approach that is based on the sparse interaction model. We assume that interaction areas can be calculated from the current local state the agent is in. The algorithm we propose, called 2Observe, uses generalized learning automata to learn these interaction states.

Chapter 4

Learning to focus on local states

Do what you can, with what you have, where you are.
– *Theodore Roosevelt, 1858-1919* –

In the previous chapter we discussed several multi-agent reinforcement learning frameworks, classified by the number of strategic interactions and the level of observability the agents have of the entire system. We described the most complete form, called Markov games, in which agents have information about the complete system state and action set and reward signals of all agents at their disposal. We also described the DEC-MDP framework, which models the situation in which agents only have a local view of the dynamics of the system at their disposal. Given only such partial observability of the dynamics of the entire system, the agent is possibly experiencing the environment as non-Markovian. In many environments however, it is possible to define the state information required for the agent to experience a Markovian environment under certain assumptions. This is modelled by DEC-SIMDPs, which provide a basis for learning and exploiting the sparse interactions that exist among agents. In this chapter we introduce a framework in which agents can learn independently of each other, depending only on local state information in those states where the global state does not provide any additional useful information. The framework is very similar to a DEC-SIMDP, but instead of having a list of the states that are part of an interaction area, the interaction areas are described by an interaction function. This allows not only for a compact representation of the interaction area, but aims to exploit certain spatial relations that exist in the joint state space. Throughout this chapter we will give an example of situations that comply with this model, as well as describe a framework for modeling these relations. Furthermore, we will present *2Observe*. This is an algorithm, capable of

approximating this interaction function. Finally, we will validate this approach using experiments in various gridworld environments.

4.1 Two layer framework

As explained in Section 3.2.3 in the previous chapter, agents should only rely on global state information, in those situations where the transition the system undergoes and the rewards the agents experience is not only dependent on the local state information of the agent performing the action. As such, an agent can learn to act optimally in a greatly reduced state space. The driving question to exploit these sparse interactions is *'When is an agent experiencing influence from another agent?'*. Answering this questing, allows an agent to know when it can select its actions independently (i.e. the state transition function and reward function are only dependent on its own action) or when it must coordinate with other agents (i.e. the state transition function and the reward function is the effect of the joint action of multiple agents). This leads naturally to a decomposition of the multi-agent learning process into two separate layers. The top layer will learn when it is necessary to observe the state information about other agents and select whether pure independent action selection is sufficient, or whether some form of coordination between the agents is required. The bottom layer contains a single agent learning technique, to be used when there is no risk of influence by other agents, and a multi-agent technique, to be used when the state transition and reward the agent receives is dependent of the current state and actions of other agents. Figure 4.1 shows a graphical representation of this framework.

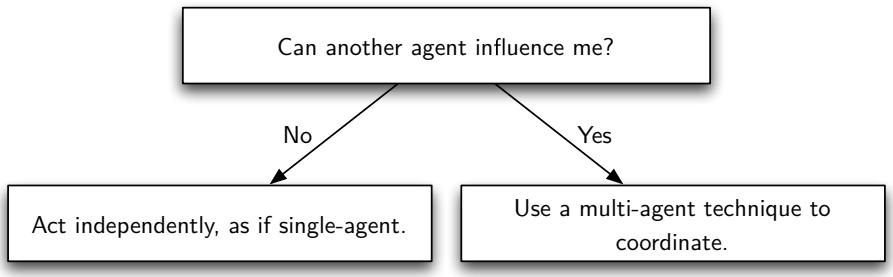


Figure 4.1: Two layer framework for modeling problems in which a function can be approximated that returns the list of states in which interaction is beneficial.

In the previous chapter we have introduced two algorithms, called Utile Coordination and Learning of Coordination. Both these techniques are implementations of the framework described above and explained in Section 3.3.1. Utile Coordina-

tion uses statistical tests on the top level to determine in which states there is a net improvement in the reward signal for a particular joint action performed by the agents. If the rewards an agent receives are better for a particular joint action in a state, this indicates that agents can improve their performance by coordinating with each other in this state. The multi-agent approach that is used is a variable elimination algorithm. The agents will calculate the best course of action in that state, based on the actions of the other agents. If no difference in the reward signal is detected for different joint actions, the agent selects an action independently but still uses joint state information to do so.

Learning of Coordination assumes that an active perception mechanism is available on the top level. This mechanism could be anything from an overhead camera, to communication or even a predefined list of states, that informs the agent if coordination is necessary in the current local state of an agent. If coordination is needed, this mechanism also informs the agent about the relevant state information upon which it should select an action. Otherwise, an action is selected using only local state information. Compared to Utile Coordination, this algorithm does not require full observability over the entire joint state-action space, but relies heavily on this active perception mechanism.

Note that both these approaches have a different view on reducing the joint state-action space in which agents are learning. UC reduces the joint action space by only selecting a joint action in those states where coordination is required. LoC reduces the joint-state space, by only using joint state information when coordination is required. In this chapter we describe our implementation of this two-layer framework, that attempts to reduce the state space in which agents are learning, without additional requirements such as an active perception mechanism. We use mobile robots with proximity sensors as an application of this framework. An example of a robot system with such sensors is the Khepera III robot by K-Team [K-Team (2009)] (see Figure 4.2).

This robot has ultrasonic sensors measuring the distance between objects and the front and sides of its base. These sensors have a range going from 20cm to 4m. Given an unknown environment, the distance at which the robot should detect objects to avoid collisions is related to its current speed (and possibly the speed of other robots in the environment). Given the reduced speed at which these robots are operating, the sensors detect obstacles far in advance, long before an action to deal with these obstacles is required. In Figure 4.3 we illustrate a simplified version of this issue which we adopt for our experiments. In (a) the robots use the entire range of their sensors and even detect remote obstacles that are not influencing the robot. (b) shows the relevant range at which obstacles should be observed[1]. (c) shows the same range, but with the presence of another robot in this range.

[1] For this example we assume that robots move 1 cell per time step

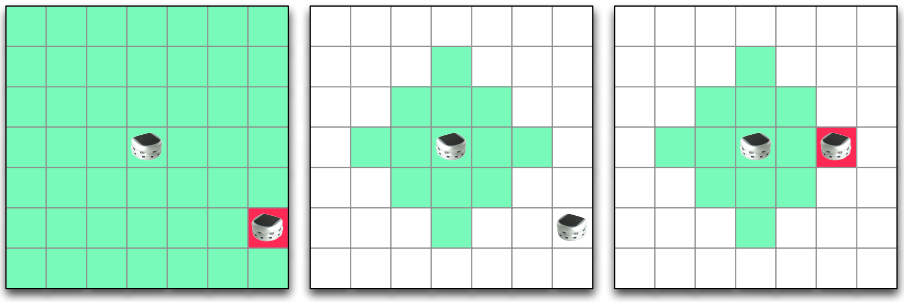
Figure 4.2: Khepera III robot by K-Team

The observed region in Figures 4.3(b) and 4.3(c) indicate what the agent should learn at the top level. If there is no other robot in this region (Figure 4.3(b)) the agent can act independently. If another robot is present (Figure 4.3(c)) coordination is required to ensure that both robots do not collide in the next time step. The entire joint state space contains the information if the sensors were to detect everything within their range. Since much of this information will not be important, our goal is to reduce the range of the sensors to a level that contains only the relevant information. Also note that although we will speak about collisions throughout this dissertation, this does not actually imply a physical collision between the robots. Robots have various other sensors to detect imminent collisions and avoid them. The purpose here is not so much to replace this feature, rather than to avoid having to use it.

We will first formalise this agent-centric view of the interaction area before describing a solution method for this type of multi-agent problems with sparse interactions.

4.2 Decentralised Local Interactions Markov Decision Process

We assume that there exists a relationship between the current local state an agent is in, and the (possible) interaction area of which this state is part. As described above, for mobile robots, this interaction area is the spatial region around the current location of the robot. This means that an interaction area can be represented as a function I, which, given the current local state $s_k(t)$ returns the interaction area for that state. It is based on the state information of this interaction area that the agent



(a) entire range (b) relevant range with no other robot (c) relevant range with one other robot

Figure 4.3: States observed by the ultrasonic sensors of the robot if (a) the entire range is used, (b) only the relevant range is used but no other robot is present, (c) only the relevant range is used and another robot is present.

should then select an action. Problems for which this assumption holds, are modeled as a *Decentralised Local Interactions Markov Decision Process* (DEC-LIMDP). A DEC-LIMDP shows close resemblance to a DEC-SIMDP, with the major difference being that the list of states in which interaction can occur, is now replaced by the interaction function I.

Definition 21. *A **Decentralised Local Interaction Markov Decision Process** is a tuple* $(n, S, A_{1,\ldots,n}, I_{1,\ldots,n}, T, R_{1,\ldots,n})$*, with:*

- n *the number of agents in the system,*
- $S = S_0 \times S_1 \times \ldots \times S_n$ *the set of joint states, where* S_k *indicates the set of states of agent* $k = 1, \ldots, n$ *and* S_0 *contains all external features of the environment,*
- A_k *= the action set of agent* k*, with* $k = 1, \ldots, n$
- $I_k : S_k \rightarrow \prod^M S^M$ *= the interaction function.* $I_k(s_k^i)$ *returns a set, which contains the local state,* $s_k^i \in S_k$ *of agent* k*, and which may, additionally, contain state information about other agents* M *that are part of the interaction.*[2]
- $T : S \times A_1 \times \ldots \times A_n \times S \rightarrow \mathbb{R}$ *the transition function.* $T(s'|s, (a_1, \ldots, a_n))$ *is the probability of the outcome state* s' *when the joint action* $(a_1, \ldots, a_n)$ *is taken in state* s*, with* $s, s' \in S$ *and* $(a_1, \ldots, a_n) \in A_1 \times \ldots \times A_n$*,*

[2] In states in which interaction with other agents is not necessary, the following holds: $I_k(s_k^i) = \{s_k^i\}$

- $R_k : S \times A_1 \times \ldots \times A_n \times S \rightarrow \mathbb{R}$ *the reward function.* $R_k(s, (a_1, \ldots, a_n), s')$ *is a real number representing the reward obtained from taking joint action* $(a_1, \ldots, a_n)$ *in state* s *and transitioning to state* s'*, with* $s, s' \in S$ *and* $(a_1, \ldots, a_n) \in A_1 \times \ldots \times A_n$

Given a DEC-LIMDP, the answer to the question when an agent experiences influence from another agent is exactly the function I. So an agent should attempt to approximate its function I in order to determine when to observe the interaction area or when its local state information is sufficient.

Such a DEC-LIMDP is also an implementation of the two layer framework of Figure 4.1. If the output of the interaction function I is the same as the local state information of the agent, the agent will not experience any influence from any other agent and can act independently (bottom left in Figure 4.1). If the output of this interaction function contains more information than just the local state information of the agent, the agent should act upon this information, and use a multi-agent technique capable of dealing with these external influences (bottom right of Figure 4.1).

4.3 Generalized learning automata

As described above, agents should approximate their interaction function I_k, in order to learn when to act using more information than just its own local state.

Generalized learning automata (GLA) are simple associative reinforcement learning units that learn a mapping from given inputs or contexts to actions. They are widely used for classification tasks and have appealing theoretical convergence guarantees. This makes them suitable for approximating the interaction function and to be used for the top level in our implementation of the two-layer framework. We will begin by defining the necessary background of learning automata, required for the remainder of this chapter.

4.3.1 Definitions

The LA we considered in Section 2.5.2.2 only use the reinforcement they receive from the environment as input. The automaton learns the optimal action that results in the maximum expected reward. In a pure RL setting, an agent however attempts to learn which actions yields the highest expected (delayed) reward, in the context of the state of the environment. Using traditional LA, this can be achieved by creating networks of LA, where each automaton is responsible for learning the optimal action for one state of the environment. An alternative is to use an additional input to the automaton, in the form of a context vector representing the state

of the environment. Such an automaton is called a *Generalized learning automaton* [Thathachar & Sastry (2004)]. At each time step the GLA receives an input which describes the current system state. Based on this input and its own internal state the unit then selects an action. This action serves as input to the environment, which in turn produces a response for the GLA. Based on this response the GLA then updates its internal state. Such a situated GLA is represented in Figure 4.4.

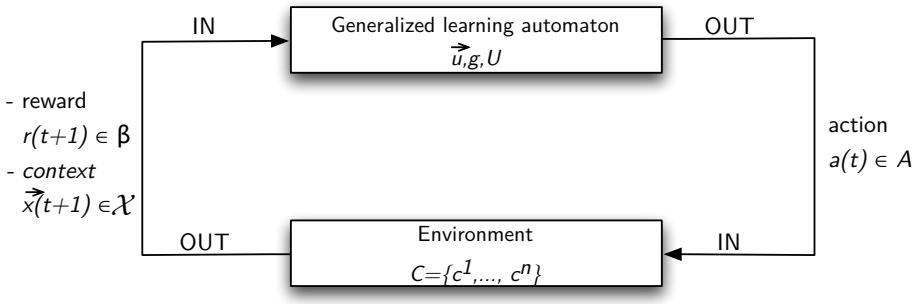


Figure 4.4: Situated generalized learning automaton in a feedback loop with the environment.

Definition 22. *Formally, a **generalized learning automaton** can be represented by a tuple* $(\mathcal{X}, A, \beta, \vec{u}, g, U)$*, where:*

- $\mathcal{X}$ *is the set of inputs to the GLA. Individual context vectors are denoted by* $\vec{x} \in \mathcal{X}$*,*
- $A = \{a^1, \ldots, a^r\}$ *is the set of outputs or actions the GLA can produce,*
- $\beta \in [0, 1]$ *denotes the feedback the automaton receives for an action,*
- $\vec{u}$ *is a real vector representing the internal state of the unit,*
- g *is the probability generating function*

$$P\{a(t) = a | \vec{u}, \vec{x}\} = g(\vec{x}, a, \vec{u}) \tag{4.1}$$

where g *has to satisfy following conditions:*

$$g(\vec{x}, a, \vec{u}) \geq 0 \quad \forall \vec{x}, a, \vec{u}$$
$$\sum_a g(\vec{x}, a, \vec{u}) = 1 \quad \forall \vec{x}, \vec{u}$$

- U *is a learning algorithm which updates* $\vec{u}$*, based on the current value of* $\vec{u}$*, the given input, the selected action and response* β*.*

In general, the internal state $\vec{u}$ of a GLA is a set of r vectors for each of the r actions: $\vec{u} = \{\vec{u}_1, \ldots, \vec{u}_r\}$.

The earliest algorithm capable of performing updates on these internal state vectors is called REINFORCE [Williams (1992)], which uses the following update rule:

$$U: \quad \vec{u}_i(t+1) \quad \leftarrow \quad \vec{u}_i(t) + \lambda\beta(t+1)\frac{\partial ln\, g(\vec{x}(t),a(t),\vec{u}(t))}{\partial \vec{u}_i} \quad (i = 1, \ldots, r) \quad (4.2)$$

where $\vec{u}_i$ is the i^{th} component of $\vec{u}$.

A basic property of the REINFORCE algorithms is the stochastic gradient following property. However , this algorithm may result in unbounded behaviour because $\vec{u} \to \infty$. An analysis of this phenomenon together with a modification of this algorithm to ensure boundedness is introduced in [Phansalkar et al. (1990)]. This modified update scheme is as follows:

$$\begin{aligned} U: \quad \vec{u}_i(t+1) \quad &= \quad \vec{u}_i(t) + \lambda\beta(t+1)\frac{\partial ln\, g(\vec{x}(t),a(t),\vec{h}(\vec{u}(t)))}{\partial \vec{u}_i} \\ &\quad +\lambda K_i(h_i(\vec{u}_i(t)) - \vec{u}_i(t)) \qquad\qquad (i = 1, \ldots, r) \end{aligned} \quad (4.3)$$

where $\vec{h}(\vec{u}) = [h_1(\vec{u}_1), h_2(\vec{u}_2), \ldots h_r(\vec{u}_r)]$, with each h^i defined as:

$$h_i(\eta) = \begin{cases} L_i & \eta \quad \geq L_i \\ 0 & |\eta| \quad \leq L_i \\ -L_i & \eta \quad \leq -L_i \end{cases} \quad (4.4)$$

In this update scheme λ is the learning rate and $L_i, K_i > 0$ are constants. The update scheme can be explained as follows. The first term added to the parameters is a gradient following term, which allows the system to locally optimise the action probabilities. The next term uses the $h_i(\vec{u}_i)$ functions to keep the parameters $\vec{u}_i$ bounded within predetermined boundaries $[-L_i, L_i]$. In [Thathachar & Sastry (2004)] it is shown, that the adapted algorithm described above, converges to local maxima of $f(\vec{u}) = E[\beta|\vec{u}]$, showing that the automata find a local maximum over the mappings that can be represented by the internal state in combination with the function g.

In the remainder of this chapter we use the following set-up for the GLA. With every action $a^i \in A$ the automaton can perform, it associates a vector $\vec{u}_i$. This results in an internal state vector $\vec{u} = [\vec{u}_1^T \ldots \vec{u}_r^T]$ (where T denotes the transpose). With this state vector we use the Boltzmann distribution as probability generating function:

$$g(\vec{x}, a^i, \vec{u}) = \frac{e^{-\frac{\vec{x}^T \vec{u}_i}{\tau}}}{\sum_j e^{-\frac{\vec{x}^T \vec{u}_j}{\tau}}} \quad (4.5)$$

with τ a parameter that represents the temperature. Of course, since this function is fixed in advance and the environment in general is not known, we have no guarantee that the GLA can represent the optimal mapping. For instance, when using the function given in Equation 4.5 with a 2-action GLA, the internal state vector represents a hyperplane. This plane separates context vectors which give a higher probability to action 1 from those which prefer action 2. If the sets of context vectors where different actions are optimal, are not linearly separable the GLA cannot learn an optimal mapping.

To allow a learner to better represent the desired mapping from context vectors to actions, we can utilise systems composed of multiple GLA units. For instance the output of multiple 2-action GLAs can be combined to allow learners to build a piecewise linear approximation of regions in the space of context vectors. In general, we can use systems which are composed of feedforward structured networks of GLA. In these networks, automata on one level use actions of the automata on the previous level as inputs. If the feedforward condition is satisfied, meaning that the input of a LA does not depend on its own output, convergence to local optima can still be established [Phansalkar & Thathachar (1995)].

4.4 GLA for multi-agent state space aggregation

Our goal is to use GLA to determine in which states an agent experiences an influence of other agents. Similar to the ideas used for single agent reinforcement learning problems with large state spaces, we are trying to generalise policies over similar states. In multi-agent systems additional problems arise however. System control is distributed and agents might have conflicting goals, which results in an additional requirement for coordination between the agents. Solving this problem in a centralised way would result in an exponential growth of the state-action space in the number of agents. This motivates the need for a decentralised approach using a generalisation over the state space. As a basic tool for our approach we use GLA as described in Section 4.3. As explained, these automata have the advantage that they are computationally simple and can be combined into larger networks to offer greater flexibility to accurately approximate regions in the state space. Even when multiple GLA are networked, strong theoretical convergence properties exist [Phansalkar & Thathachar (1995)].

Figure 4.5 shows the general agent learning set-up. Each time step t a vector $\vec{x}(t)$ giving a factored representation of the current system state is generated. This vector is given to each individual agent as input. The agents internally use a set of GLA to select an action corresponding to the current state. The joint action $\vec{a}(t)$ of all agents serves as input to the environment, which responds with a feedback $\beta(t+1)$ that agents use to update the GLA. One of the main advantages of this approach

is that convergence guarantees exist for general feedforward GLA structures. In this chapter we study common interest problems where each agent uses one or more GLA. As such, this system can be viewed as a single large network of GLA, thus ensuring convergence to a local optimum.

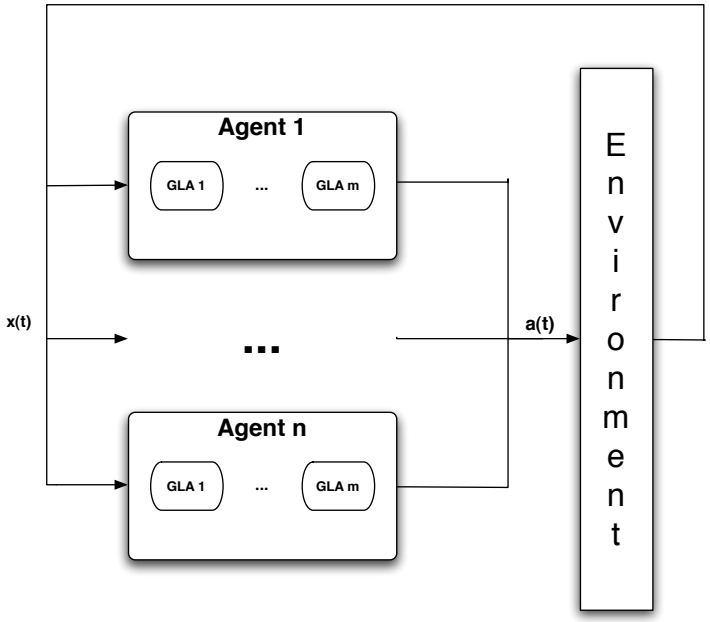


Figure 4.5: Learning set-up. Each agent receives a factored state representation as input. The GLA of the agents select the action to be performed.

What follows is a demonstration of the capabilities of GLA in a number of relatively simple experiments.

4.4.1 Agents using a single GLA

Our basic experimental set-up is shown in Figure 4.6. Two agents A and B move on a line between $[-1, 1]$. Each time step both agents select action `left (L)` or `right (R)`, move and then receive a reward based on their original joint location and the joint action they chose. Each agent then updates using only the reward signal and the joint location, without any knowledge of the action selected by the other agent.

If Agent A is to the left of Agent B, the optimal action for both agents is to take action `Left`. If Agent A is to the right of Agent B, the highest reward is obtained if both agents select action `Right`. If the absolute value of the distance between the agents is less than 0.5, the highest reward is obtained if both agents move apart

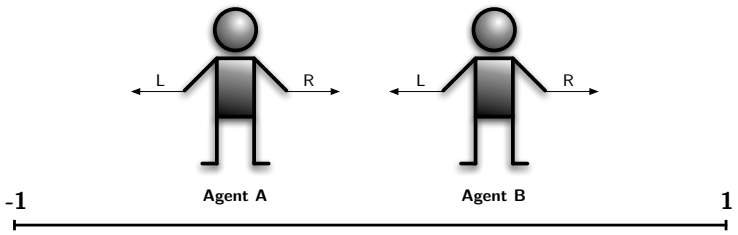


Figure 4.6: Experimental set-up. Two agents move around on a line between positions -1 and 1. Each time step both agents take a step left or right.

from each other. The reward scheme is as follows:

1. $+1$ if both agents choose action `Left`, if Agent A is to the left of Agent B, 0 otherwise.
2. $+1$ if both agents choose action `Right`, if Agent A is to the right of Agent B, 0 otherwise.
3. $+1$ if both agents choose a different action, if the absolute value of the distance between both agent is less than 0.5, 0 otherwise.

This means that there are three regions in the state space in which a different action is optimal for the agents. This is shown in Figure 4.7. In region 1 Agent A is left of Agent B. In the second, Agent A is to the right of Agent B. The third region encapsulates all the states where the absolute value of the distance between the two agents is less than 0.5.

For this experiment each agent uses a single GLA with 2 actions corresponding to the agent's actions L and R. Each time step we give both agents an input vector $\vec{x} = [x1\ x2\ 1]$, where $x1$ is the position of agent A and $x2$ is the position of agent B. The third component is always set to 1, as is usually done with context vectors. This is called the bias in pattern recognition tasks or neural networks and allows the internal state to better represent the target function [Mitchell (1997)]. The GLA use a vector $\vec{u}_i = [\vec{u}_1\ \vec{u}_2\ \vec{u}_3]$ for each action i. The learning process of a GLA can then be seen as moving the lines $(\vec{u}_1 - \vec{u}_2)^T\vec{x} = 0$ and $(\vec{u}_1 - \vec{u}_3)^T\vec{x} = 0$ which separates regions in the state space where the GLA prefers action `L` from those where it prefers action `R`. Typical results obtained with this system can be seen in Figure 4.8. This result was obtained running the experiment for 100.000 iterations. Each iteration consists of a single action choice and update for both agents. After each move and subsequent learning update, the agents were reset to new random positions and the game was restarted. This was done to avoid the undersampling problem which occurs easily when dealing with such large state spaces.

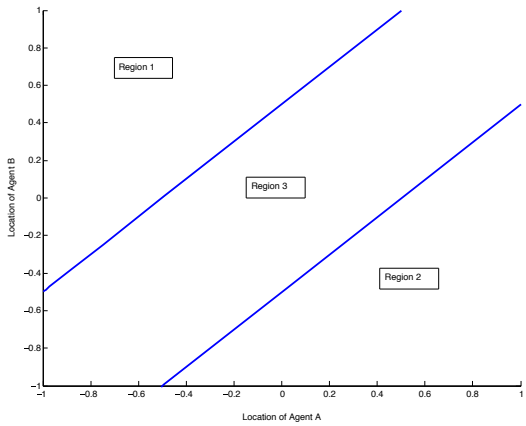
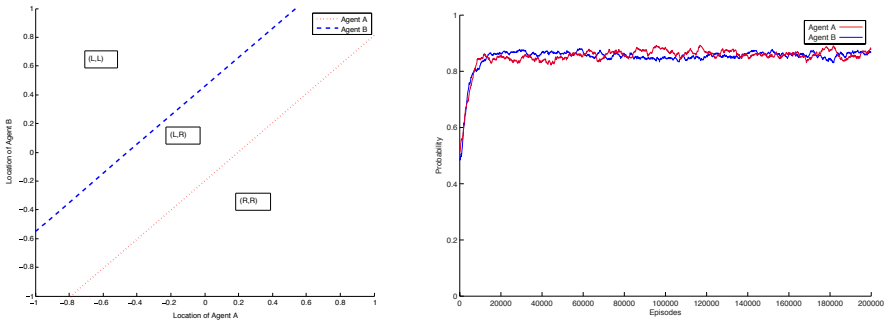


Figure 4.7: State space regions for experiment 1.



(a) (b)

Figure 4.8: Experimental results for the first experiment. (a) Typical Results for approximations of the regions in the state space learnt by agents. Lines separate regions where agents prefer different actions. Joint actions with highest probability are given in each region.(b) Probabilities of optimal action in region 1 for both agents. Parameter settings where $\lambda = 0.005$,$K_i = L_i = 1$ $T = 0.5$

Since GLA take context vectors as input, it is possible to present the state information in different forms to the agent. Figure 4.9 shows a comparison of the average reward obtained, with three distinct ways of information. We compared the use of the joint location, as described above, to an absolute distance metric $(AbsoluteValue(Pos(AgentA) - Pos(AgentB)))$ and a deictic distance metric $(Pos(AgentA) - Pos(AgentB))$. This experiment was run without tuning of the exploration of the Boltzmann action selection method. Hence, the results we de-

scribe here are only meant as a criterion to compare the influence of the information given in the context vectors and not as an absolute performance evaluation of the GLA.

The absolute distance metric clearly performs the worst due to the inability of making a distinction between different positions of the other agent (left or right). When using a deictic representation, the GLA obtain higher rewards than when the joint location was used. This seems contradictory to the work published in [Finney et al. (2002)], in which deictic representations worsen performance. However, the research described in that paper also introduces partial observability when using deictic representations. This is not the case in our scenario.

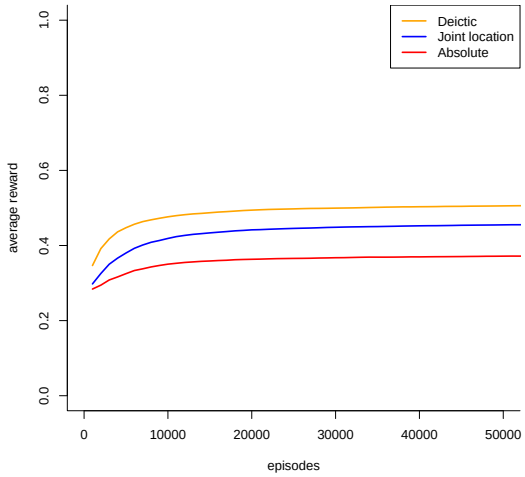


Figure 4.9: Comparison of the influence of the state information given to the GLA.

4.4.2 Agents using multiple GLA

In the second experiment we examine a situation where the different regions in the state space are not linearly separable. In such a case the agents cannot exactly represent the optimal mapping, but rather have to approximate it using different hyperplanes. We use the same set-up as in the previous experiment, but now we consider two regions, as given in Figure 4.10.

The reward scheme for this setup is the following.

1. Region 1, delimited by the inside of the parabola: A reward of $+0.9$ is given, when both agents choose action `Left (L)`, 0.1 otherwise.

2. Region 2, delimited by the outside of the parabola: A reward of $+0.5$ is given, when both agents choose action `Right (R)`, 0.1 otherwise.

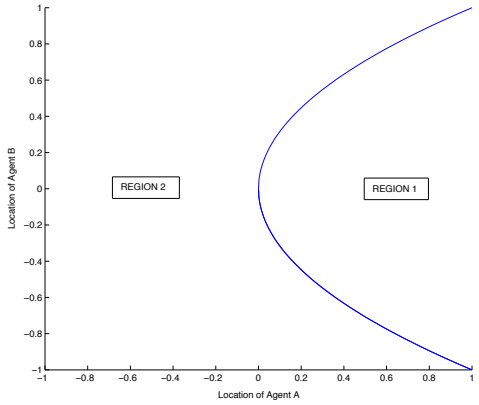
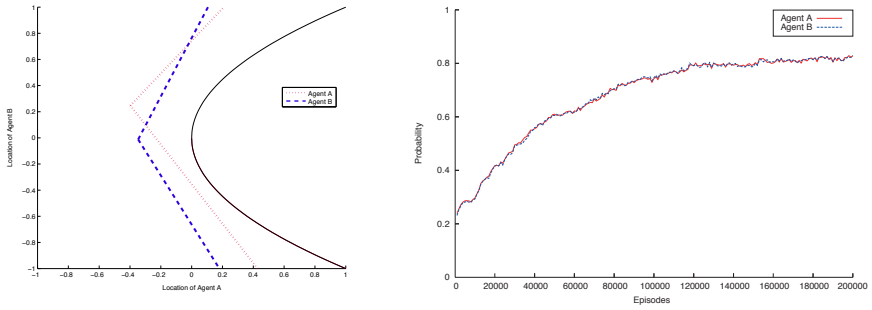


Figure 4.10: State space regions for Experiment 2.

Both agents use a system consisting of 2 GLA, connected by an `AND` operation. The agents have 2 actions: `L` and `R`. Both GLA also have 2 actions. If the automata both choose the same action, the agent performs action `L`, otherwise it performs action `R`. Figure 4.11(a) shows typical results for the boundaries that the agents learn to approximate the parabola. Figure 4.11(b) shows for both agents the evolution of probability of the optimal action L in region I. The probabilities in this plot were obtained by generating 100 points in the region with uniform probability and calculating the average probability over these points during learning.

While it can be seen from the results in Figure 4.11 that the agents are able to approximate the desired regions, this experiment also demonstrated the limits of GLA. As was mentioned in the previous section the GLA are only guaranteed to converge to a local optimum. This means that the agent can get stuck in suboptimal solutions. Such a situation was observed when the reward for the optimal action in region 2 is increased. In this case it is possible for the agents to get stuck in a situation where they both always prefer the optimal action for region 2 and neither agent has a good approximation of the region inside the parabola. Since the rewards of both agents are based on their joint action, no agent can get out of this situation on its own. The agents can only improve their pay-off by switching actions inside region 1 together. In such a situation with multiple local optima, the final result obtained by the agents depends on the initialisation of the automata.



(a) (b)

Figure 4.11: Experimental results for the second experiment. (a) Typical results for approximations for parabola learnt by agents. (b) Probabilities of optimal action in region I for both agents (average over 100 runs). Parameter settings were $\lambda = 0.005$,$K_i = L_i = 1$, $T = 0.5$

4.5 Learning the interaction function

In the previous section we have shown that GLA are useful for aggregating states in multi-agent settings, and are able to learn these aggregations online, using a reinforcement signal. This makes GLA prime candidates for using at the top level of the framework described in Section 4.1. Our goal is to move away from approaches that make a very black and white distinction between observing the other agents or not observing them. In the previous section we have shown that GLA are capable of learning boundaries in the state space and select different actions based on these boundaries. In mobile robotics, the application we use for our approach, we can represent the fact whether an agent should observe other agents, as a boundary around its own location. This means that if another agent is too close, it is not safe anymore to just ignore it. What follows is a description of the *2Observe* algorithm, which uses a GLA to learn the boundary that delimits when an agent is too close to be ignored. The approach proposed here bears some resemblance to the adaptive resolution methods used in single agent RL. There the learning agent uses statistical tests to determine when a greater granularity in the state space representation is needed. Examples of these kind of systems are the G-learning algorithm [Chapman & Kaelbling (1991)] and the U-tree algorithm [McCallum (1995)]. A more detailed description of these approaches is given in Chapter 5.

4.5.1 2Observe

In the previous section we used GLA in an ad-hoc manner, learning on only one layer. In this section we will implement the two layer framework from Section 4.1. We have compared different representations for giving the GLA information about

the distance between the robots. A deictic representation proved to yield the best results for these kind of settings. Hence, in this section we provide the GLA with the *Manhattan distance* between the two agents if they are within each others *line-of-sight* (i.e. no walls between them). At the top level of the two layer framework, we employ GLA to learn, based on the rewards they receive and the provided distance, how close the other agent can be before there is a possibility of colliding.

The main advantage of using GLA is that the first layer of the algorithm can learn to determine which technique on the second level must be chosen, without explicitly storing estimates or samples of visited state-action pairs. All necessary information is encoded in the parameters of the GLA, which are typically much smaller in number than the states about which information needs to be kept. The possibility of combining GLA into a larger network which can learn more complex representations, also gives us the flexibility to adapt the complexity of the first layer to the needs of the problem at hand. Moreover, since there are no absolute state estimates being used, these GLA are independent of the current location of the agent.

At the second level of the learning algorithm two possible methods can be used, depending on the outcome of the first layer of the algorithm. If this first layer determines that other agents can safely be ignored 2Observe uses independent Q-learning. Otherwise, a simple form of coordination through communication is used.

If the GLA have learnt to coordinate their actions in a particular state they first observe if a collision will occur if both agents would just choose their preferred action. If this is the case, one agent is selected randomly to perform its action, while the other selects an alternative action. If no collision will occur, both agents can select their action independently (see Algorithm 4). One could chose to always play this coordination mechanism, since it is ensured to be collision free, but that would imply a strong dependency on fault-free communication between the agents, which is undesirable or even impossible in many systems. The algorithm is more formally described in Algorithm 6.

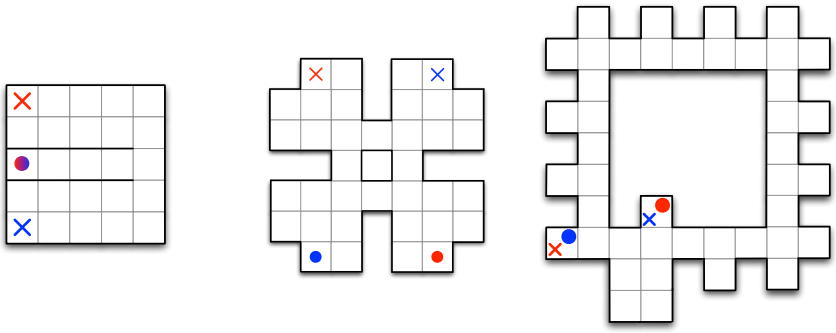
Algorithm 6 2Observe

1: For each agent and every $\langle s_k, a_k \rangle$ pair, initialise the table entry for $\hat{Q}(s_k, a_k)$ to zero,
2: Initialise the GLA for each agent arbitrarily,
3: **loop**
4: Observe the current state s and the distance between the agents δ
5: **if** GLA of both agents select to coordinate, based on δ **then**
6: agents use coordination mechanism to select the actions to avoid a collision
7: **else**
8: Each agent k independently selects action a_k, based on their action selection strategy
9: **end if**
10: Observe r_k^q and r_k^c, where r_k^q is the reward for the actions selected by the Q-learning algorithm and r_k^c is the reward for the action selected by the GLA of agent k.
11: Observe the new state s'_k
12: Update the Q-table entry of $\langle s_k, a_k \rangle$ wit r_k^q according to Equation 2.10
13: $s_k \leftarrow s'_k$
14: Update the GLA for input δ with r_k^c according to Equation 4.3
15: **end loop**

s_k is the local state information of the agent k, a_k the action selected by agent k, r_k^c the reward for the selected action of the GLA at the top level for agent k, r_k^q the reward for selected action of the Q-learning algorithm on the second layer for agent k.

4.5.2 Experimental results

To validate our approach, we apply it to various gridworld problems. These are a simplified version of the problems mobile robots face when navigating in unknown environments and contain all the difficulties of much harder problems [Sutton & Barto (1998)]. It is also widely used in the RL community and thus it provides a good testbed for comparing and evaluating our algorithm to other RL techniques. We compare the results of 2Observe with independent Q-learning agents, with the MMDP framework, as well as with a DEC-MDP approach. Figure 4.12 shows a graphical representation of the gridworlds we used. Figure 4.12(a) shows the *TunnelToGoal* environment, Figure 4.12(b) the *2-robot game* and Figure 4.12(c) the *ISR* environment. The agents both have to reach their respective goals, indicated with the coloured dots [3], starting from their initial positions marked by the coloured **X**, while avoiding to bump into walls and into each other. Agent 1 is indicated in red, agent 2 is indicated in blue. The agents have four actions at their disposal (`N`,`E`,`S`,`W`), which moves them respectively up, right, down or left for 1 cell.



(a) (b) (c)

Figure 4.12: Different gridworlds in which we experimented with our algorithm. In (a) both agents have the same goal state (G), whereas in (b) and (c) the agents have different goals marked by coloured dots. The initial positions are indicated with a **X**. We refer to the different gridworlds as follows: (a) *TunnelToGoal*, (b) *2-robot game*, (c) *ISR*. (b) and (c) are variations of the games in [Melo & Veloso (2009)]

Before discussing the results of the different techniques, we analyse the state-action spaces used by the different approaches in these 2-agent gridworlds. An overview is given in Table 4.1. The independent Q-learners do not take any information about the other agents into account, resulting in an individual state space of the number of locations in the grid, and 4 possible actions per state. The joint

[3] In Environment (a), the agents share the same goal location indicated with a dot in both colours

		TunnelToGoal	2-robot game	ISR
Independent Q-Learning	#States	25	36	43
	#Actions	4	4	4
DEC-MDP	#States	625	1296	1849
	#Actions	4	4	4
MMDP	#States	625	1296	1849
	#Actions	16	16	16
2Observe	#States	25	36	43
	#Actions	5	5	5

Table 4.1: Comparison of the state-action space in which different RL approaches are learning for the gridworlds of Figure 4.12

state learners (DEC-MDP) learn in a state space represented by the joint locations of the agents, but select their actions independently, so they have 4 possible actions per joint state. The MMDP learner also uses the joint state space representation of the agents but also selects joint actions, resulting in 16 possible actions for every joint state (i.e. all possible combinations of the 4 individual actions). For 2Observe, the actual size of the joint state space is not so important because no explicit value is kept for every state. Hence, our algorithm is learning in the same state action space as the independent Q-learners, relying on some communication in situations where collisions might occur.

All experiments were run with a learning rate of 0.05 for the Q-learning algorithm and Q-values were initialised to zero. An ϵ-greedy action selection strategy was used, where ϵ was set to 0.1. The GLA have a learning rate of 0.01, use a Boltzmann action selection strategy and were initialised randomly. All experiments were run for 10.000 episodes, where an episode is the time needed for both agents to reach the goal, starting from their initial positions and all experiments were averaged over 10 runs. The number of steps shown in the results is the total number of steps required to complete an episode. The episodes were not bounded to allow agents to find the goal without time limit. If an agent reaches the goal, it receives a reward of $+20$. For the MMDP learner the reward of $+20$ was given when both agents reach their goal positions, but once an agent is in its goal, its actions no longer matters, since the goal state is an absorbing state. If an agent collides with another agent, both are penalised by -10. Bumping into a wall is also penalised by -1. For every other move, the reward was zero. For every collision, whether it was against a wall or against another agent, the agent is bounced back to its original position before the collision occurred.

In these experiments we assume that the range of the sensors is larger than the distance within which interaction with other agents can occur. Following this

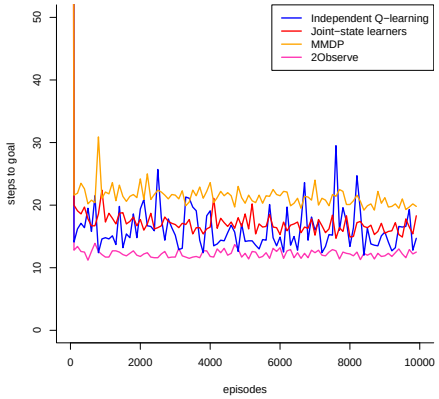
assumption, the GLA were rewarded individually according to the following rules:

- GLA coordinated if there was danger of collision or did not coordinate if there was no danger: $+1$
- GLA coordinated if there was no danger of collision or did not coordinate if there was danger: -1

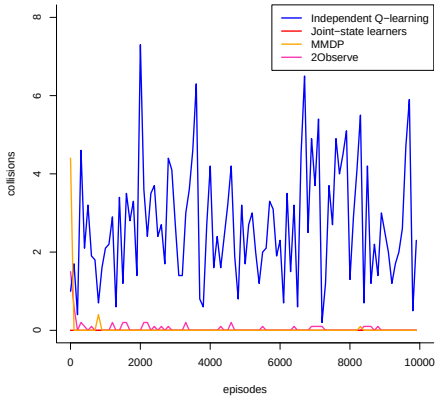
This reward scheme assumes that the environment is aware of the situations in which coordination must occur, so that it can reward the agents accordingly.

The left part of Figure 4.13 ((a),(c) and (e)) show the average number of steps both agents needed to reach their goal during learning in the different environments. The right part of Figure 4.13 ((b),(d) and (f)) show the number of times agents collided with each other per episode (i.e. from start to completion). The results for the TunnelToGoal environment are given at the top, The 2-robot in the middle, and ISR at the bottom. Both joint-state approaches have not learned the shortest path after 10^5 iterations, due to the limited exploration rate and the size of the state action space ($2,500$; $5,184$ and $7,396$ values to be learned for the different environments). In the TunnelToGoal and in the IRS the independent Q-learners both behave poorly because when following their individual shortest paths to reach the goal, they will certainly collide with each other. In the 2-robot environment the independent Q-learners manage to find a good collision-free solution, due to the fact that agents can avoid each other by going through different doorways since multiple shortest paths exist. In all environments, 2Observe manages to find a policy without collisions with other agents and without a large overhead in the number of steps needed to complete an episode.

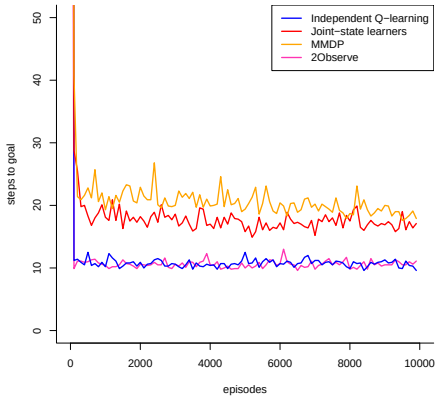
In Figure 4.14 we show the number of times the agent used their coordination mechanism to avoid a collision, from top to bottom, for TunnelToGoal, 2-robot game, ISR. There is a slight overhead in the number of times this action is used, because the agents are unable to distinguish when another agent is very close but moving away or moving towards it. For the TunnelToGoal environment, this means that the agents are coordinating the entire path in the tunnel, even though this is only needed at the entrance, since after that, they are just following each other towards the goal. In the next chapter we propose another approach for sparse interactions that mitigates this issue.



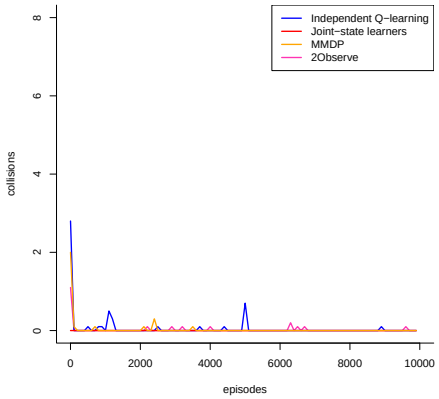
(a) **TunnelToGoal** - Steps to goal



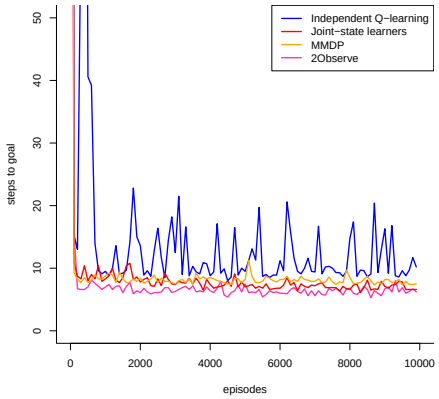
(b) **TunnelToGoal** - Collisions



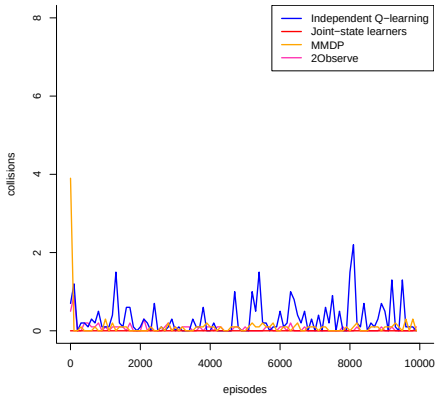
(c) **2-robot** - Steps to goal



(d) **2-robot** - Collisions

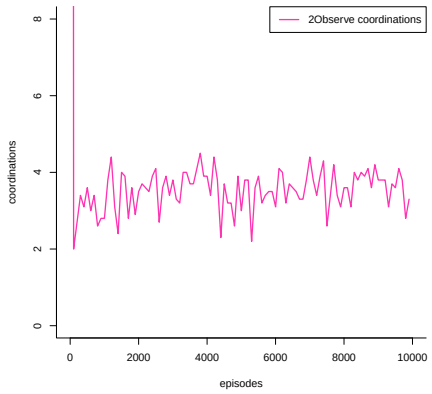


(e) **ISR** - Steps to goal

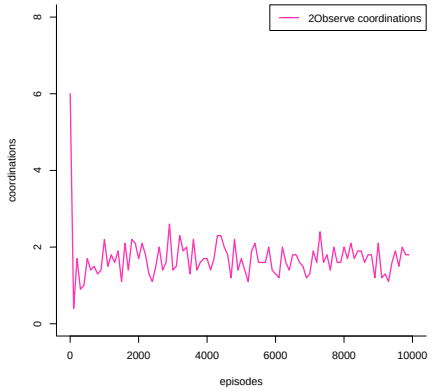


(f) **ISR** - Collisions

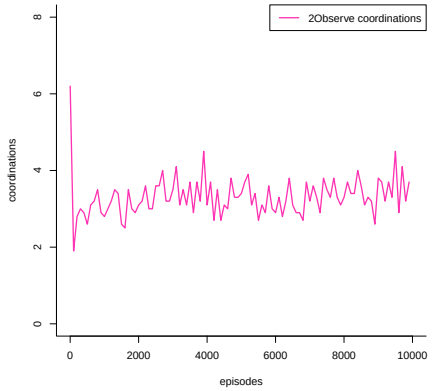
Figure 4.13: Left side: average number of steps, right side: average number of collisions (results averaged over 10 runs) for TunnelToGoal at the top, 2-robot game in the middle and ISR at the bottom.



(a) **TunnelToGoal**



(b) **2-robot**



(c) **ISR**

Figure 4.14: Average number of coordinations (results averaged over 10 runs) for (a) TunnelToGoal, (b) 2-robot game, (c) ISR.

4.6 Summary

The contributions presented in this chapter are fourfold. First we introduced a way to decompose multi-agent learning tasks in two layers. On the top layer it is decided which state information of other agents should be observed in the current state, in order to detect and solve coordination problems. At the bottom level there is on one hand a traditional single agent RL technique, if the agent can safely act upon its own local state information, or, on the other hand, a multi-agent approach, capable of dealing with the influences from other agents. In this chapter we are concerned with mobile robots as an application domain. In these domains, the influence of other agents can be characterised by their relative closeness to each other. This means that the top level of our framework is concerned with learning when *close* is *too close*.

Second, we introduced a formal model, called decentralised local interaction Markov decision process. This model is very similar to the DEC-SIMDP introduced in Section 3.2.3 but rather than containing a list of states that are part of an interaction area, a DEC-LIMDP contains a function which returns this list of states, given the current state of the agent. Hence, in mobile robot systems, this interaction function returns exactly when *close* is *too close*.

Third, we described generalized learning automata and applied these units to achieve state aggregation in multi-agent systems. These associative RL units are capable of online learning of a mapping from a given input to an output without the need to store a value for every possible state-action combination.

As such, these GLA provide the basis for the fourth contribution presented in this chapter: the 2Observe algorithm. This algorithm is a concrete implementation of the two layer framework. A GLA is installed at the top level of this framework to learn how close robots may be from each other, before the possibility exists that these robots will collide. At the bottom level a single agent Q-learner was used if the robots could not collide during the next time step. However, if both agents deemed that there was a risk of colliding in the next time step, a coordination mechanism was employed, during which the agents communicated their intentions in order to select the best and safest action. This risk is measured by the presence of another agent, independent of the history of its actions in that particular situation.

Note that 2Observe can be seen as a form of obstacle avoidance for robots. It is however more advanced than traditional algorithms because it can learn to avoid other robots, based on their specifications, without prior knowledge about them. If two other robots are in the environment, both operating at different speeds, the 2Observe algorithm will learn a different mapping for both of these robots without prior knowledge about which robot moves faster.

Although having strong theoretical convergence properties, the GLA used in the

2Observe algorithm need a separate reward signal from which they learn about the coordination requirements in the environment. If this reward signal is combined with the reward signal of the navigation task the agents are solving, the GLA are unable to learn a good coordination policy in a reasonable time. Due to the nature of certain problems this assumption is not always possible. As such, other algorithms that perform good, without these theoretical guarantees might be a better choice for certain problem domains. A wide range of techniques can be used in the context of the general two-layer framework. In the next chapter we introduce algorithms and present their results in which we use statistical tests on the reward signal at the top level to determine the influence of other agents in a given state.

Chapter 5

Solving immediate coordination problems

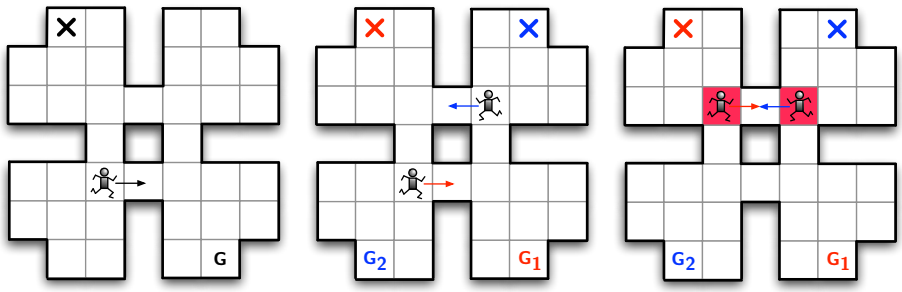
Mistakes, obviously, show us what needs improving. Without mistakes, how would we know what we had to work on?
– Peter McWilliams, 1949-2000 –

In this chapter we introduce *Coordinating Q-Learning* (CQ-Learning). The objective of this algorithm is to reduce the state space in which agents have to learn compared to the traditional multi-agent learning techniques described in Chapter 3. To do so, CQ-Learning learns the compact set of states in which agents are influencing each other. By only considering the joint state information in these states, CQ-Learning does not need to observe the state information of other agents, if this does not yield any additional information. I.e. only sparse interactions are considered. We begin by analysing the characteristics of these interaction areas before introducing two versions of CQ-Learning, both with similar performance results but with different initial assumptions. The first variant uses a single agent model of the reward function of the problem the agent is attempting to solve and uses this as a baseline to detect influences from the other agent. The second variant exploits the initial exploration strategy of the agents during their learning process. This initial exploration is a good measure for what happens if agents were to be learning alone in the environment. Hence, it is a good baseline against which interactions with other agents can be identified.

5.1 Analysis of sparse interactions

When multiple agents are learning a certain goal in an unknown environment, independently from each other, an agent might at a certain point experience the

influence from the other agents. The situations and more specifically the states where these influences are experienced are the interaction areas of the environment. In Figure 5.1 this concept is represented using the 2-robot gridworld environment introduced in the previous chapter. The goal is marked by the letter **G**, together with the index for respectively agent 1 and agent 2 and the initial position of the agents is indicated with a red **X** for agent 1 and a blue **X** for agent 2.



(a) Single agent, alone (b) Multi agent, far away from each other (c) Multi agent, (too) close to each other

Figure 5.1: Sparse interactions that occur between the agents. (a) An agent acting alone never interacts (b) multiple agents far from each other are not interacting (c) multiple agents close to each other do interact

Using full observation it is straightforward to predict that if both agents take the shortest path to the goal, the influence they will experience, will occur at the corridors in the center of the environment. In Section 3.2.1 we showed that if multiple agents learn to accomplish a certain goal, independently from each other, the number of interactions between them increases as learning progresses. Initially agents are behaving random, but as their Q-values start to converge, the agents begin to follow a certain policy. In certain environments this policy will result in an increase in the number of strategic interactions between the agents. In this dissertation we consider MARL tasks in which these interactions will occur in the same limited set of states. This chapter is concerned with learning this set of states, in order to use a focused multi-agent approach when selecting an action in these states. In Figure 5.1(a) we illustrate the base case, from which we will start. When an agent is acting alone, the environment it experiences is stationary and the rewards it receives from the environment are only due to its own actions. In 5.1(b) we show the situation in which two agents are present, but they are in locations far away from each other. Hence, during the next time step they are unable to hinder each other and, from the view point of one agent, this situation can be seen as the one depicted in Figure 5.1(a). I.e. it can act independently, since the other

agent will not influence it. In Figure 5.1(c) we depict the situation in which agents have already learned how to reach their goal and start to collide more often at a corridor in the center of the environment. Both agents are close to each other and will collide in the next timestep if they execute their preferred action (indicated by the arrow).

These interactions are reflected in the reward signal the agents receive. Collisions result in high penalties. Hence, by detecting these penalties it is possible to determine the required state information to avoid these significantly lower rewards. As for 2Observe, the techniques we present in this chapter are an instantiation of the two layer framework (Section 4.1). The 2Observe algorithm from the previous chapter uses an agent centric view of the environment and aims at approximating a function which reflects when agents are interacting. The purpose of the techniques in this chapter are to learn the absolute set of states in which the agents are interacting. As such, these techniques solve the underlying DEC-SIMDP the agents are facing. This framework is described in Section 3.2.3 and models the interaction states in which agents are influencing each other. We will approach this problem from two angles. In the first one we assume that agents have been learning alone in the environment for some time before being put together. As such, they have not only learned a policy for accomplishing the single agent version of the task at hand, but they have also learned a model of the reward function for the single agent problem. This information will be used to detect the influence of other agents. Second, we will explain an approach when both agents are learning together from the beginning. This approach is based on our findings of Section 3.2.1 in which we have shown that the number of strategic interactions between agents increases in a compact set of states as learning progresses.

5.2 Learning interaction states

In this section we introduce CQ-learning. This algorithm learns the set of states that belong to the interaction area of the underlying DEC-SIMDP. The algorithm identifies these states by detecting statistically significant changes in the reward signal and by testing which of the state information of other agents is causing these changes. In these interaction areas an agent should take other agents into account when choosing its preferred action. If it has identified such interaction state, CQ-Learning will select its actions based on the combination of its own local state information together with the relevant state information about the other agents participating in the interaction. In the context of gridworlds we can formulate these situations as follows: if two agents are adjacent to each other, but their respective preferred actions will not influence each other, the algorithm will not coordinate since there is no need to do so (Figure 5.1(b)). If they are however adjacent and their preferred action, will result in a collision (Figure 5.1(c)), the CQ-Learning

agents will select an action using *augmented state* information:

Definition 23. *We assume the state space S is factored as $S = S_0 \times S_1 \times \ldots S_n$, where S_0 contains the state information about the environment itself, and S_k contains the internal information about agent k with $k = 1, \ldots, n$ and n the number of agents present in the system.*
The ***augmented state*** *$\vec{s_k}$ of an agent k is a tuple $(s_0, s_k, [S^I])$, which contains the current state information about the environment s_0, the state information about the agent s_k and the state information S^I about all the agents participating in the interaction.*

A high level graphical representation of augmented state information is given in Figure 5.2. Agents begin with 9 local states. The combination of these states is the system state. After some time, Agent k, using CQ-learning detects a change in the immediate reward signals it experiences in state 4. The algorithm will observe the system state each time it encounters this local state in order to identify the extra relevant information required to avoid this change in the reward signal. Once the agent has learned on which other agents it is dependent in this local states, these states are augmented to include the state information of those other agents. In our example, CQ-Learning detected a change in the reward signal for state 4, and identified the relevant state information of agent l to be states 1, 2 and 3. $\langle 4-1\rangle$,$\langle 4-2\rangle$,$\langle 4-3\rangle$ are the resulting additional augmented states. If agent l is in any other state, when agent k is at state 4, agent k selects its actions, purely based on state 4.

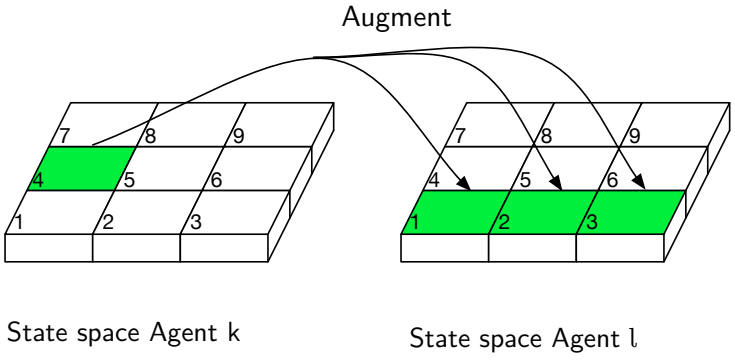


Figure 5.2: Graphical representation of CQ-learning. Independent single states are expanded to augmented states where necessary. In this example, the state information about another agent are added to the state information of agent k in state 4 after detecting changes in the immediate reward signal.

5.2.1 CQ-learning with initialised agents

So far we have introduced augmented states and explained the general idea about CQ-Learning. This algorithm is designed for environments in which the influence of other agents is reflected in the immediate reward signal. We will now describe in depth the three main parts of which this algorithm is composed: detecting conflict situations, selecting actions and updating the Q-values:

- **Detecting conflict situations**
  Agents must identify in which states the reward signal they experience is affected by another agent. The algorithm stores the last N rewards it experienced for a particular state-action pair in order to perform a statistical test on them to determine whether these rewards are as expected or if they are influenced by external factors. CQ-Learning needs a baseline for this test. In this section we assume that agents have learned a single agent model about the expected payoffs for selecting an action in a particular local state. In future sections we will relax this assumption. We will refer to this expectation as $E(R_k(s_k, a_k))$. Alternatively, this model can be provided to the agents. From this model, the agents can calculate the optimal single-agent policy for the scenario where there are no other agents in the environment. The agents follow this policy even when multiple agents are present and keep a history of the last N rewards they receive for every local state-action pair they encounter and store it in a sliding window $W_k(s_k, a_k)$ (for agent k). Since we have an expectation of rewards, and a sufficiently large sample of actual obtained rewards, the influence of other agents can be detected by performing a one-sample Student t-test to verify if the null hypothesis that the population of $W_k(s_k, a_k)$ is equal to the expected value $E(R_k(s_k, a_k))$ holds [Cochran & Snedecor (1989)]. A description of this test is given in Appendix B.
  We distinguish two possible outcomes:

  1. The Student t-test rejects the null hypothesis that the population $W_k(s_k, a_k)$ comes from a normal distribution with a mean equal to $E(R_k(s_k, a_k))$. In this case, CQ-learning marks this state. Each time it visits a marked state, the algorithm will sample the reward signal according to the joint state space. This is done pairwise, i.e. local state of the agent + local state information of another agent, for all other agents. These samples are stored in $W_k(s_k, a_k, s_l)$ where s_l contains the state information about another agent l, with $k \neq l$. The same statistical test is performed to identify for which values of s_l the null hypothesis is rejected. s_k is augmented to $\vec{s}_k$ which includes these values of s_l for which the

null hypothesis did not hold[1]. In the context of Figure 5.2, s_k is local state 4 of agent k, s_l are states 1,2 and 3 of agent l, resulting in augmented states $\vec{s}_k^1 = \langle s_k^4, s_l^1 \rangle$, $\vec{s}_k^2 = \langle s_k^4, s_l^2 \rangle$ and $\vec{s}_k^3 = \langle s_k^4, s_l^3 \rangle$. In our gridworld example, the augmented state contains the joint location of agents that will collide if they were to select the action with the highest single agent Q-value. Note that agents could also always sample rewards from the entire joint state space and only perform the second statistical test. This would however not be an improvement, since this would imply a multiplication of the complexity of the algorithm in terms of memory to store all these samples and in terms of computational requirements to perform the statistical tests. Moreover, in the majority of the joint state space, the null hypothesis will hold, in case of sparse interaction, which we assume in this work. CQ-learning first identifies where these interactions occur and then detects which other agents are part of the interaction.

2. The Student t-test fails to reject the null hypothesis that the immediate rewards the agent receives $W_k(s_k, a_k)$ come from a normal distribution with mean $E(R_k(s_k, a_k))$. If the distribution is unknown, alternative non-parametric tests can be used.
   From this Student t-test can be concluded that the agent is not experiencing any influence from other agents and can continue selecting actions independently, using only its own local state s_k.

This process is shown graphically in Figure 5.3

[1] During our experiments we observed that the sliding window, containing the samples of the important state information $W_k(s_k, a_k, s_l)$, was the first to contain N samples and hence also the first to be detected by the statistical test. This is due to the fact that agents follow their policies and will interact with each other more often in the same states that are causing the conflict.

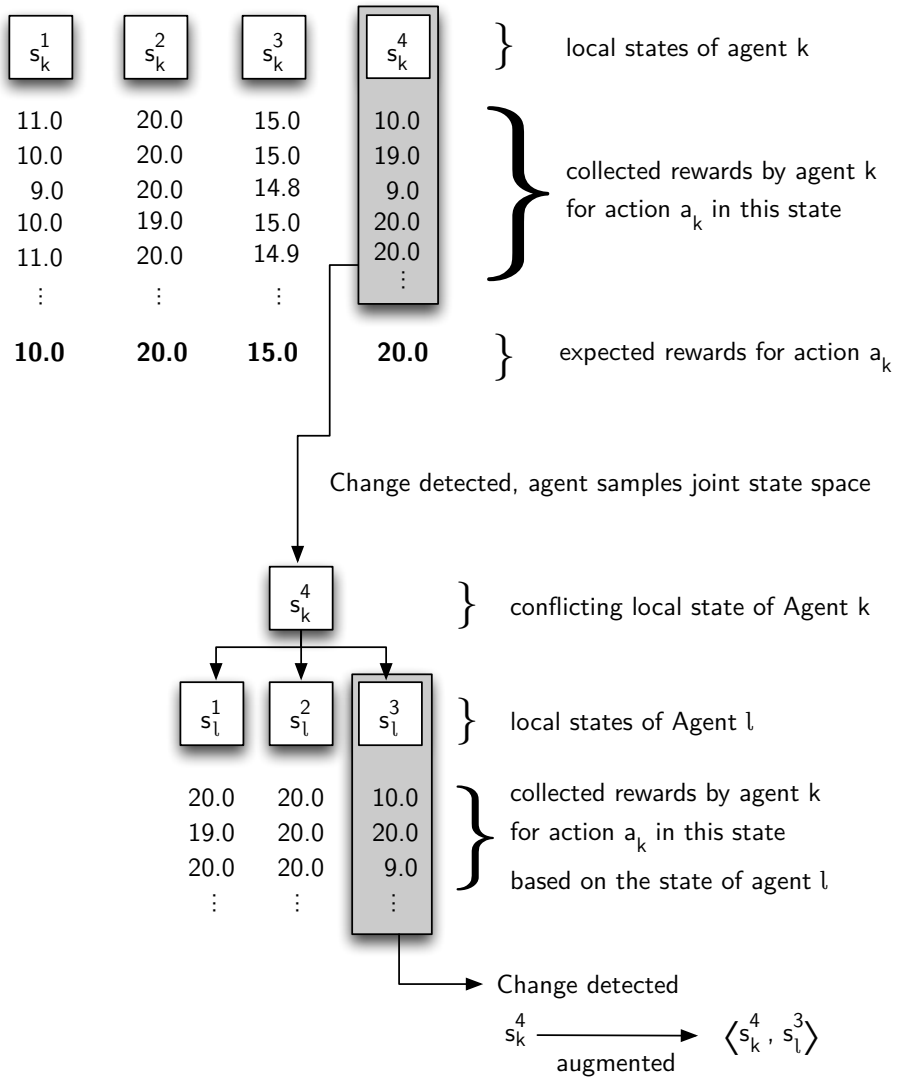


Figure 5.3: Evolution of the statistical tests in CQ-learning. A change is first detected by agent k in state s_k^4, after which it starts sampling the state information of agent l and detects s_l^3 to be the relevant state information causing this change.

- **Selecting actions**
  At every time step, the agents observe their local state s_k. In the base case, in which s_k is not part of an augmented state, an agent can safely select its action based only on s_k. On the other hand, if s_k is part of an augmented state $\vec{s}_k$, the agent will observe the other state information in $\vec{s}_k$. If the system state contains the information encapsulated by $\vec{s}_k$, i.e. another agent l is in

the same local state s_l as stored in $\vec{s}_k$, the agent will select an action based on the augmented state. In the context of Figure 5.3 this situation occurs when agent k is in state s_k^4 and agent l is in state s_l^3. If the system state does not contain the information from $\vec{s}_k$, it means that the other agents are in a different state that will not influence agent k. Hence, the agent will, as in the base case, select an action, solely based on its local state s_k.

- **Updating the Q-values**
  We distinguish two cases for updating the Q-values depending on whether the agent was in an augmented state or not:

  1. The system contained all the information from an augmented state $\vec{s}_k$ for agent k and the agent selected its action based on $\vec{s}_k$. In this situation the following update rule is used:

  $$Q_k^{\text{aug}}(\vec{s}_k, a_k) \leftarrow (1-\alpha_t)Q_k^{\text{aug}}(\vec{s}_k, a_k) + \alpha_t[R_k(s, a_k) + \gamma \max_{a'_k} Q_k(s'_k, a'_k)] \tag{5.1}$$

  where Q_k stands for the Q-table containing the local states, and Q_k^{aug} contains the Q-values for the augmented states. Note that this second Q-table is initially empty and is being created during the runtime of the algorithm, as interaction states are being learned. The Q-values of the local states of an agent are used to bootstrap the Q-values of the augmented states. This is done for two reasons. First, during the update process, the agent has not yet determined whether its next state is also an augmented state. Second, the Q-values of the local states represent the best possible situation, when the other agents will not influence the agent in the future. As such, these Q-values are an optimistic estimate of the future rewards the agent can expect[2].

  2. The agent selected its action based on the local state information s_k. These Q-values have already been learned in the single agent version of the task at hand and hence they do not have to be updated. However, the regular Q-learning update rule (as in Equation 2.10) can still be applied:

  $$Q_k(s_k, a_k) \leftarrow (1-\alpha_t)Q_k(s_k, a_k) + \alpha_t[R_k(s, a_k) + \gamma \max_{a'_k} Q_k(s'_k, a'_k)] \tag{5.2}$$

The entire pseudo-code for CQ-learning is given in Algorithm 7.

[2] Single agent Q-learning also bootstraps using the best possible future scenario by using the max operator in Equation 2.10.

Algorithm 7 CQ-Learning algorithm for agent k

Train Q_k independently first, initialise Q_k^{aug} to zero and W_k = empty;
Set $t = 0$
while true do
  observe local state $s_k(t)$
  if $s_k(t)$ is part of a augmented state $\vec{s}_k$ and the information of $\vec{s}_k$ is present in the system state $s(t)$ **then**
    Select $a_k(t)$ according to Q_k^{aug}
  else
    Select $a_k(t)$ according to Q_k
  end if
  observe $r_k = R_k(s(t), a(t))$, s'_k from $T(s(t), a(t))$
  Store $\langle s_k(t), a_k(t), r_k(t)\rangle$ in $W_k(s_k, a_k)$
  if Student t-test rejects $h_0 : W_k(s_k, a_k)$ comes from a distribution with mean $E(R_k(s_k, a_k))$ **then**
    Store $\langle s_k(t), a_k(t), s_l(t), r_k(t)\rangle$ in $W_k(s_k, a_k, s_l)$ for all other agents l
    for all extra state information s_i about another agent l present in s(t)) **do**
      if Student t-test rejects $h_0 : W_k(s_k, a_k, s_l)$ comes from a distribution with mean $E(R_k(s_k, a_k))$ **then**
        augment s_k with s_l to $\vec{s_k}$ and add it to Q_k^{aug}
      end if
    end for
  end if
  if $s_k(t)$ is not part of any $\vec{s}_k$ or the information of $\vec{s}_k$ is not in $s(t)$ **then**
    No need to update $Q_k(s_k)$.
  else
    Update $Q_k^{\text{aug}}(\vec{s}_k, a_k) \leftarrow (1 - \alpha_t)Q_k^{\text{aug}}(\vec{s}_k, a_k) + \alpha_t[r_k + \gamma \max_{a'_k} Q_k(s'_k, a'_k)]$
  end if
  $t = t + 1$
end while

5.2.2 Illustrative results

As explained, CQ-learning uses a single agent model of the reward function of the multi-agent problem it is trying to solve. This model can either be provided, or it can be learned by allowing the agents to learn independently for some time in the environment previous to acting together in it. Before introducing an extension to CQ-learning for which this single agent model is not required, we will show some initial results of CQ-learning for the environments used in the experiments of 2Observe in the previous chapter: TunnelToGoal, 2-robot game and ISR (see Figure 4.12). The arrows with full tails indicate actions the agents took using only their local state information. The arrows with dotted tails represent actions taken based on augmented state information. Coloured cells indicate the augmented state information for Agent 1 in red, and Agent 2 in blue.

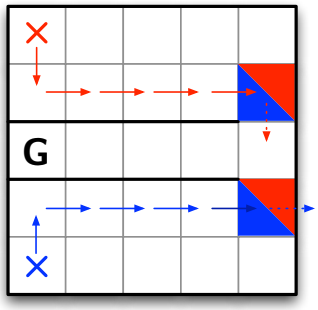


Figure 5.4: Solution found by CQ-learning for the TunnelToGoal environment. Agent 2 (with start position marked by the blue cross), performed a kind of `wait` action by moving towards the boundary of the grid, so Agent 1 (with start position marked by the red cross) could go first in the tunnel towards the goal.

In Figure 5.4 we show the relevant part of the path the agents follow from their start position to the goal. Agent 1 started at the top left of the grid (marked by the red cross), Agent 2 started at the bottom left (marked by the blue cross). CQ-learning has learned to augment the cells at the entrance of the tunnel with the opposing cell for both agents. The result of this can be seen in the actions the agents selected. At the entrance of the tunnel, Agent 1 selected to go down and continue following its shortest path. Agent 2 selected to go to the right, which was a zero-operation since the grid is bounded on that side. Hence, this action could be seen as a form of `wait` action. After this timestep in which both agents played an action based on augmented state information, they selected their actions using local state information and followed the shortest path to the goal. Note that the identification of these states is not necessarily symmetrical. It is possible that one

agent detects a state in which it collides often while the other agent has not marked this state as being one in which coordination with the other agent is required. As such, the first agent can solve the problem by augmenting its state and avoid the other agent before the latter ever detected the conflict.

This solution seems unfair for Agent 2, but this is due to the technique used at the multi-agent level, which is ϵ-greedy independent action selection, based on joint state information. Since both agents in an interaction state also have a common goal, i.e. avoid a collision, other approaches are possible in which agents alternate between who goes first. This lies however outside of the scope of this dissertation. We refer to [de Jong (2009)] for a recent overview of fairness in MAS.

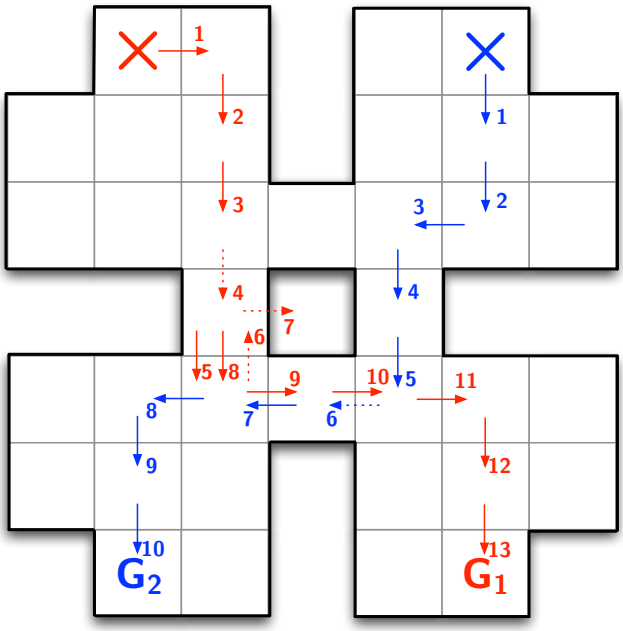


Figure 5.5: Solution found by CQ-learning for the 2-robot environment. The numbers next to the arrows indicate the timesteps. Agent 1 (indicated in red), performed some actions to allow Agent 2 (indicated in blue) to pass first.

The 2-robot game depicted in Figure 5.5 offers a more challenging environment. In this environment, the single agent solution might already provide a shortest path in which both agents would never collide. In this case, CQ-learning would behave identical to Q-learning with greedy action selection. In the figure we represent a situation in which the agents have learned a shortest path which leads to collisions when applied together. Agent 1 (the red arrows) performs some actions in the middle of the grid, to allow Agent 2 (the blue arrows) to pass first. The aug-

mented states for this environment are all centered around the passages between the different sections of the grid. This can also be seen from the locations in which the agents selected an action using their augmented states (the arrows with dotted tails). The indices next to the arrows indicate the timestep in which this action was performed. At timestep 7, Agent 2 is passing Agent 1, after which both go towards the goal, following the shortest path and selecting their actions using only local state information.

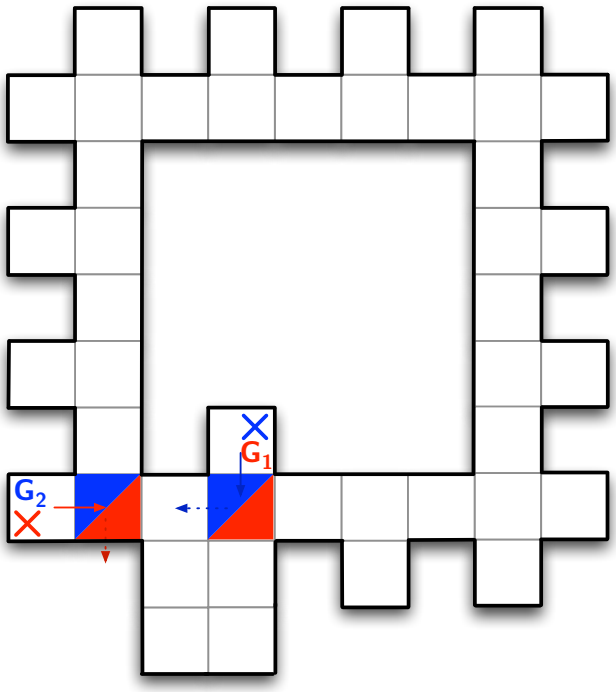


Figure 5.6: Solution found by CQ-learning for the ISR environment. Agent 1 performs a kind of `wait` action, by moving towards the boundary of the grid, so both agents can cross at the next timestep.

In the third environment, ISR, shown in Figure 5.6 we again show the relevant part of the path and the augmented states for both agents. Agent 1 (the red arrows) executed one action so it would remain in the same location to allow the other agent to come closer. During the next time step both agents then switched locations and followed their single agent policy to their respective goals. An alternative for Agent 1 would have been to move out of the way, but this would result in a longer path to reach the goal than to collect the penalty for moving towards the boundary of the grid.

5.2.3 CQ-learning with random initial policies

In some situations the requirement that agents have an expectation of the immediate payoffs for every state-action pair cannot be met. Either because this model is not available or because training the agents separately in the environment beforehand is not possible. In such situations another baseline is needed to detect the influence of other agents. In Section 3.2.1 we presented results for independent agents learning in the same environment. These results motivate the idea behind a baseline when a model is not available. Initially agents are acting randomly because the Q-values were uniformally initialised. Gradually, the agents converge to the optimal policy of their navigation task and they start to experience the systematic influence of each other in certain states. Hence, it is reasonable to conclude that the rewards collected in the initial stage of the learning process are a good estimate of the rewards the agent would receive if it would have been acting alone in the environment. This assumption can be used to detect conflict situations that might occur in a later stage of the learning process, when the agents' Q-values start to converge.

Initially, agents select their actions using only their own local state information. The agents maintain a list of rewards for every local state-action pair they visit. The first N rewards received for a certain action a_k in a local state s_k are stored forever in W_k^1. Every reward received for this particular pair (s_k, a_k) after the first N samples, is added to a sliding window W_k^2 of fixed size N, replacing the oldest sample in a first-in-first-out way. This concept is shown graphically in Figure 5.7.

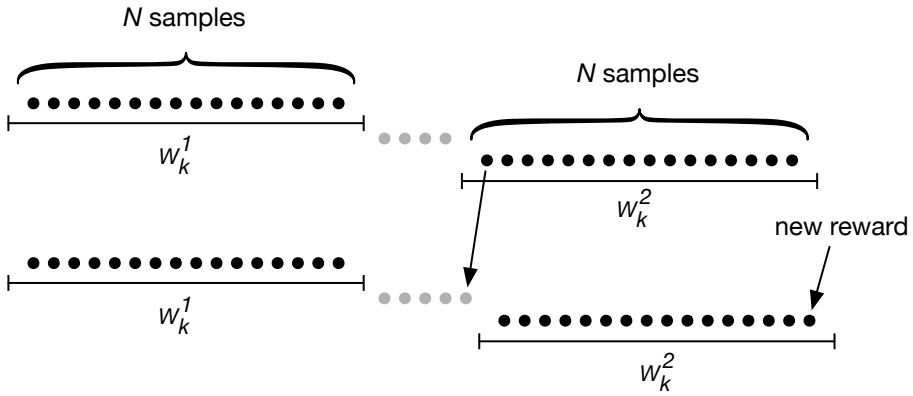


Figure 5.7: Sliding window principle of CQ-Learning. W_k^1 is fixed, W_k^2 contains the last N received rewards for a certain state-action pair (s_k, a_k).

This version of CQ-Learning also uses a Student t-test. However, now agents do not have a single expected mean of the payoffs for a certain state-action pair, but two histories of payoffs. The two-sample variant of this statistical test is used to de-

termine whether the means of the two normally distributed populations of samples are equal. By taking the difference between the populations and testing whether the mean is zero, we can determine if the samples from one population are significantly lower than the samples from the other population [Cochran & Snedecor (1989)]. At every timestep such an independent two sample Student t-test is performed for the current state-action pair of the agent to determine whether the null hypothesis holds. If the more recently received rewards (W_k^2) come from a distribution with the same or a higher mean than the rewards of W_k^1 the agent is observing significantly different (lower) rewards than in the initial learning phase and the algorithm will perform a one sample Student t-test to determine if the last received reward was smaller than the mean from W_k^2. If so, we can conclude that the agent's last action resulted in negative reward, due to a lack of coordination with other agents. As for the version in the previous section, the algorithm will augment the local state of the agent to include the state information of the conflicting agent, after which the agent will act using this augmented state.

The second adaptation that is made to the original CQ-learning algorithm is the introduction of a confidence value for the augmented states. Since the baseline the agents are using for comparison might be under heavy influence from the other agents, which have not yet converged, a confidence value is maintained for every augmented state. This value gives an indication of how often this augmented state is observed relative to how often the agent was acting in its local state. If this confidence value drops below a certain threshold, the augmented state is not used any more and the agent will again select its actions using only the local state information, which was part of this augmented state. This ensures that due to the exploration in the early phase of the learning process, the agents do not end up with a large augmented state space containing many states that are no longer encountered, once the Q-values converge. The pseudo-code for this modified CQ-learning algorithm is given in Algorithm 8.

Algorithm 8 CQ-Learning algorithm for agent k with random initial policies

Initialise Q_k, Q_k^{aug} to zero and $W_k^1 = W_k^2 =$ empty;
Set $t = 0$
while true do
  observe local state $s_k(t)$
  if $s_k(t)$ is part of $\vec{s}_k$ and the information of $\vec{s}_k$ is present in $s(t)$ **then**
    Select $a_k(t)$ according to Q_k^{aug}
  else
    Select $a_k(t)$ according to Q_k
  end if
  observe $r_k = R_k(s(t), a(t))$, s'_k from $T(s(t), a(t))$
  if $|W_k^1(s_k, a_k)| < N$ **then**
    Store $\langle s_k(t), a_k(t), r_k \rangle$ in $W_k^1(s_k, a_k)$
  else
    Store $\langle s_k(t), a_k(t), r_k \rangle$ in $W_k^2(s_k, a_k)$
    Delete oldest sample from $W_k^2(s_k, a_k)$ if $|W_k^2(s_k, a_k)| > N$
  end if
  if Student t-test rejects $h_0 : W_k^1(s_k, a_k) = W_k^2(s_k, a_k)$ **then**
    if Student t-test fails to reject $h_0 : r_k < W_k^2(s_k, a_k)$ **then**
      augment $s_k(t)$ with s_l to $\vec{s}_k$ for all state information s_l of agents l present in $s(t)$ and add them to Q_k^{aug}
    end if
  end if
  if s_k is not part of any $\vec{s}_k$ or the information of $\vec{s}_k$ is not present in $s(t)$ **then**
    Update $Q_k(s_k, a_k) \leftarrow (1-\alpha_t)Q_k(s_k, a_k) + \alpha_t[r_k + \gamma \max_{a'_k} Q_k(s'_k, a'_k)]$;
    lower the confidence value of all $\vec{s}_k$, where $s_k(t)$ is part of $\vec{s}_k$
  else
    Update $Q_k^{\text{aug}}(\vec{s}_k, a_k) \leftarrow (1 - \alpha_t)Q_k^{\text{aug}}(\vec{s}_k, a_k) + \alpha_t[r_k + \gamma \max_{a'_k} Q_k(s'_k, a'_k)]$
    increase the confidence value of $\vec{s}_k$
  end if
  $t = t + 1$
end while

5.3 Experimental results

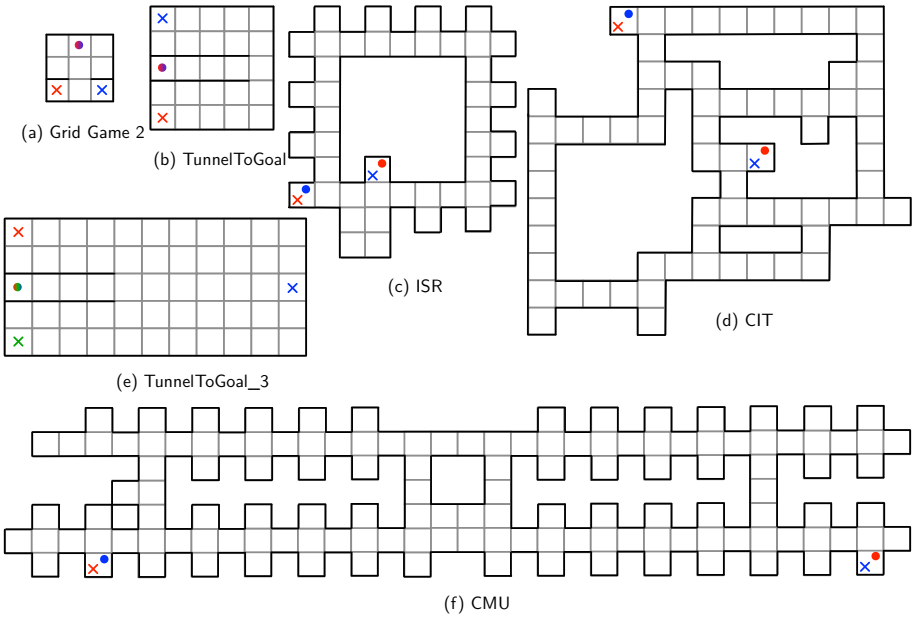


Figure 5.8: The different games used throughout this experiment.

The testbed for our algorithms is a set of two and three-agent gridworld games with varying difficulty in terms of size complexity and number of possible encounters between the agents. We compared our algorithms to independent Q-learners (Indep) that are learning without any information about the presence of other agents in the environment and are acting using local state information only. Joint-state learners (JS), which receive the state information of all the agents but chose their actions independently. Joint-state-action learners (JSA) which receive the complete system state as input and select a joint action. This is a so-called superagent. Finally, we also compare our approach with Learning of Coordination (LoC) (described in Section 3.3.1.2). For the JSA-learners, a reward is given when both agents reach their goal state, but once an agent enters his goal state, it remains in the goal. In other words, the goal states are absorbing states. As such, we could apply joint-state-action learners, despite the fact that this approach requires a pure cooperative MAS. With all other approaches a reward for reaching the goal state is given immediately to the agent entering its goal.

The environments we used are depicted in Figure 5.8. The layout of the environments (c), (d) and (f) was also used by [Melo & Veloso (2009)]. Unlike the research performed in that work, we allow agents to collide in all locations of the gridworld and not only in a small predetermined set of states. The starting positions of the agents are represented by an **X**. The respective goals for the agents are indicated

with dots in the same colour. For environments (c),(d) and (f) this is the starting position of the other agent. If the agents share the same goal state, as is the case in environments (a), (b) and (e), the dots are a linear blend of their respective colour codes.
All these environments induce some form of coordination problem where following the shortest path to the respective goals of the agents would result in collisions between them. All experiments were run for $20,000$ episodes (an episode is completed if all agents reach their respective goal state) using a learning rate of 0.1. Exploration was regulated using a fixed ϵ-greedy policy with $\epsilon = 0.1$. If the agents collide they remain in the location they were in before the collision and receive a penalty of -10 for colliding. In all other situations, transitions and rewards are deterministic. For CQ-learning with random initial policies (CQ_NI), the windows, containing the rewards W_k^1 and W_k^2, have a maximum size of 60. For CMU the ϵ parameter was set to 0.2 for CQ_NI and LoC and the size of the sliding window was reduced to 10 samples. This was necessary, because due to the nature of this environment. This environment has many corridors, where a lack of exploration would cause the agent to loop for a long time before finding the exit, if it collided several times at that exit. The number of samples for the sliding window was reduced for the same reason. The large environment caused agents to easily get stuck in a loop, before having learned to observe the state information of the other agent. By reducing the number of samples required for this test, agents detect the influence of the other agents earlier and augment their state information sooner.

Note that our approach is concerned with learning when interaction between agents is necessary and not with using the most optimal single agent learning approach for the task at hand. Several alternative approaches to standard Q-learning are possible since CQ-learning follows the two-layer framework introduced in the previous chapter. The results described in the remainder of this paragraph are the averages taken over 50 independent runs.

For LoC we implemented the active perception function as follows:

- return `TRUE` if another agent is less than 2 locations away (i.e. agents could collide in the current timestep);
- return `FALSE` otherwise.

In [Melo & Veloso (2009)] a list of states in which the agents had to coordinate was maintained. Since in the environments used in our experiments, collisions can occur in every location, this list would contain every possible combination of locations that are less than two cells apart from each other. Hence, we feel that our implementation of this active perception function is more realistic when dealing with mobile agents. Our function could easily be put in the context of sensors detecting other agents in a certain range, as was the focus of the research described in the previous chapter.

In Table 5.1 we show the results of the performance of all six algorithms in the different environments. The results are the averages over the last 100 episodes when agents have converged. Below the name of the environment we list the minimum number of steps one agent needs to reach its goal if it is not influenced by another agent. Next to the size of the state and action space, we show the number of times the agents collide on average during one episode as well as the number of steps needed for both agents to reach the goal. We see that all approaches manage to find a collision free path, except for the independent learners in the *Grid game 2* environment. This is due to the nature of this environment as explained in detail in Section 3.2.1. If both agents take the shortest (and only) path to the goal from their initial states, they are immediately penalised by a collision. Since they are not aware of each other they cannot condition their actions on the location of the other agents.

In all other environments this problem is much less due to the exploration strategy of the agents. If we take *TunnelToGoal* for instance, taking the shortest path to the goal would result in a collision at the entrance of the tunnel. However since the agents take a random action in 10% of the cases, they only rarely find themselves in that situation. Using a different exploration strategy, such as Boltzmann exploration with a decreasing temperature parameter results in similar behaviour as observed in *Grid game 2*, because the collisions would only start to occur often, when the temperature is already quite low. At this point, their Q-values change to avoid that action, and, as temperature lowers even more, the action that leads them to the goal will no longer be explored.

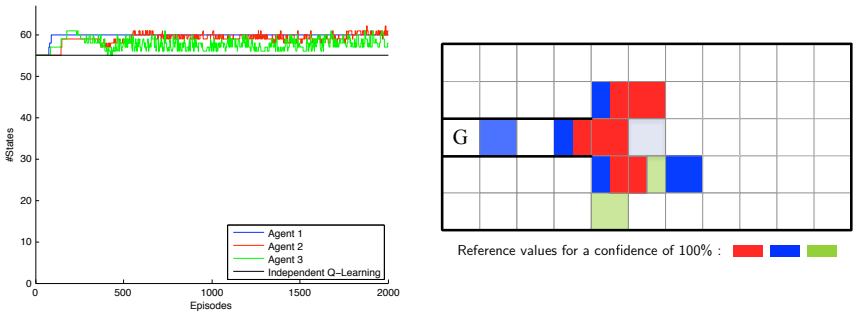
CQ-learning on the other hand uses exactly this situation to start observing the state of the other agents and to condition its actions on the local state information of the other agents. Throughout most experiments, CQ-learning manages to find a shorter path to the goal compared to the other algorithms, using a much smaller state space than both joint-state learners and without suffering from the caveats of ignoring the other agents. For the *TunnelToGoal* gridworld for instance, both agents learned to take each other into consideration at the entrances of the tunnel, resulting in a state space consisting of 26 states, which is only one more than independent learners and 599 states less than joint-state learners.

Env	Alg	#states	#actions	#coll	#steps
Grid_game_2	Indep	9	4	2.7	22.2 ± 17.9
	JS	81	4	0.1	4.0 ± 0.2
(min steps: 3)	JSA	81	16	0.0	4.7 ± 0.1
	LOC	9.9 ± 0.5	5	0.1	4.0 ± 0.4
	CQ	10 ± 0.0	4	0.0	$\mathbf{3.6 \pm 0.3}$
	CQ_NI	10.9 ± 2.0	4	0.1	4.0 ± 0.3
TunnelToGoal	Indep	25	4	0.9	30.1 ± 131.7
	JS	625	4	0.0	14.7 ± 9.2
(min steps: 10)	JSA	625	16	0.0	18.0 ± 2.3
	LOC	26 ± 0.6	5	0.2	12.3 ± 11.9
	CQ	26 ± 0.0	4	0.0	$\mathbf{10.6 \pm 0.3}$
	CQ_NI	26 ± 0.0	4	0.1	11.8 ± 0.6
ISR	Indep	43	4	0.4	9.3 ± 44.8
	JS	1849	4	0.1	5.7 ± 1.6
(min steps: 4)	JSA	1849	16	0.0	7.6 ± 1.4
	LOC	51.3 ± 82.3	5	0.2	6.7 ± 7.5
	CQ	49.0 ± 2.3	4	0.1	$\mathbf{5.1 \pm 0.7}$
	CQ_NI	49.9 ± 7.8	4	0.1	6.0 ± 1.9
CIT	Indep	69	4	0.0	19.3 ± 58.5
	JS	4761	4	0.1	17.3 ± 19.3
(min steps: 10)	JSA	4761	16	0.1	22.7 ± 12.9
	LOC	96.4 ± 61.7	5	0.2	18.9 ± 60.2
	CQ	74.2 ± 2.5	4	0.0	$\mathbf{10.7 \pm 0.3}$
	CQ_NI	70.9 ± 21.8	4	0.0	18.9 ± 59.9
TunnelToGoal_3	Indep	55	4	0.6	23.6 ± 84.16
(3 agents)	JS	166375	4	0.0	24.0 ± 31.6
(min steps: 10)	JSA	166375	64	0.2	34.5 ± 24.5
	LOC	61.6 ± 10.0	5	0.6	$\mathbf{13.7 \pm 87.7}$
	CQ	78.7 ± 31.17	4	0.1	14.52 ± 9.9
	CQ_NI	159.2 ± 1240.5	4	0.4	19.49 ± 213.9
CMU	Indep	133	4	0.0	43.8 ± 33.1
	JS	17689	4	0.0	55.5 ± 129.0
(min steps: 31)	JSA	17689	16	0.0	74.8 ± 24.1
	LOC	154.0 ± 53.28	5	0.1	46.0 ± 68.8
	CQ	133.0 ± 0.0	4	0.0	$\mathbf{31.0 \pm 0.0}$
	CQ_NI	135.52 ± 21.1	4	0.0	40.6 ± 14.3

Table 5.1: Results and state action space information for the different environments and algorithms. (Indep = Independent Q-Learners, JS = Joint-state learners, JSA = Joint-state-action learners, CQ = CQ-Learners.)

For LoC the size of the state space is the number of locations in the environment combined with the number of states in which the pseudo coordination action has the highest Q-value. The upper bound of the size of the state space LoC can observe, in our implementation of the active perception function, is the number of possible locations in the gridworld multiplied by 12. This is because the active perception function will return true for a maximum of 12 locations of another agent (i.e. the locations that can be reached within 2 steps).

Our variant on CQ-learning with random initial policies also performs well compared to other approaches. This technique also finds collision free policies, but needs a few steps more than with initial policies, but less than the other approaches. This is due to the simultaneous learning of the goal and avoiding the other agents. It is possible that agents have learned to avoid each other using only their independent state information, before enough samples have been collected for the statistical test to conclude that a richer state information would prove more beneficial. Moreover we also see that the state space of this algorithm is even slightly smaller than for CQ-learning. This is because of the confidence value we maintain for every augmented state. If these augmented states are not encountered a sufficiently number of times, they are reduced to simple states again. This process is shown in Figure 5.9(a) for the *TunnelToGoal_3* environment. We show the evolution of the size of the state space over time for the different agents.



(a) Size of the state space (b) Global states used

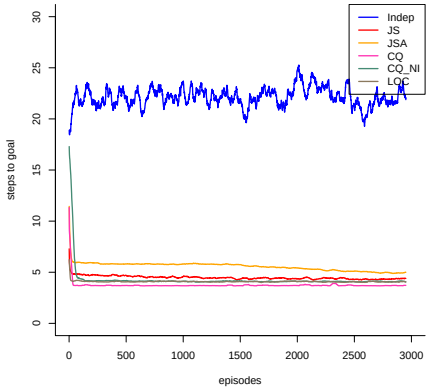
Figure 5.9: (a) Evolution of the size of the state space in which CQ-learning is learning in the *TunnelToGoal_3* environment and (b) graphical representation of states in which global state information is used.

We have indicated with the solid black line the size of the state space in which each independent Q-learning agent is learning. For joint-state and joint-state-action learners the line representing the size of the state space is constant at $55^3 = 166,375$ for this environment. To allow for better readability, this line is not shown in the

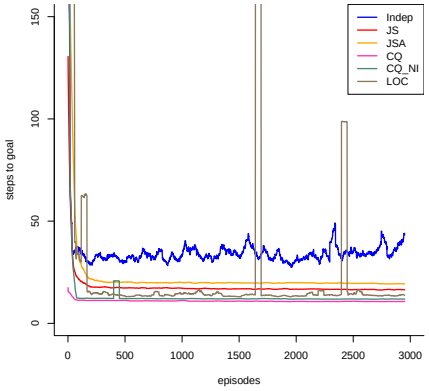
figure. The variation in the lines representing the state space of CQ-learning with random initial policies for the different agents is caused by the fixed exploration strategy which is used. This causes agents to deviate sometimes from their policy which results in additional states in which collisions might be detected. These states however are removed again pretty quickly thanks to the confidence value that is maintained for augmented states. Since these states are only occasionally visited and the other agents are only rarely at the same location as when the collision state was detected, so the confidence value of these states decreases rapidly. In Figure 5.9(b) we show in which locations the agents will observe other locations in order to avoid collisions. We used the same colours as in Figure 5.9(a) for the different agents. The alpha level of the colours represent the confidence value each agent has in that particular augmented state. The lighter the colour, the lower the confidence value for that augmented state. The agents have correctly learned to observe other agents around the entrance of the tunnel, where collisions are most likely if the agents follow their respective shortest paths to the goal. In all other locations, they act using only their local state information.

In Figure 5.10 we show a running average with a window of 50 episodes of the number of steps needed to complete the episode for the different environments. The first $3,000$ episodes are shown. In all environments we see the number of steps rapidly decreasing. CQ is right from the beginning lower than all the algorithms because they have a model of the reward function for the corresponding single agent task at their disposal and can quickly follow the optimal policy. As soon as this algorithm collected enough samples to conduct its statistical test, the number of steps needed to complete an episode using this algorithm lowers even more since agents also avoid collisions now.
CQ_NI has a similar behaviour as Indep in the initial phase of the learning process. In terms of behaviour both algorithms are identical at the beginning, until CQ_NI identified interaction states and augmented them. At this point, CQ_NI quickly learns a shorter path to the goal than Indep. The algorithm reaches a solution in which a comparable or slightly lower number of steps is required to complete an episode compared to JS or JSA. This can be seen in Table 5.1 in which we also list the number of steps needed if agents are alone in the environment, following the shortest path.
LoC initially needs to learn in the local state space, using 5 possible actions (4 navigation actions + `COORDINATE`). The `COORDINATE` action causes an update in the Q-table containing joint state values which is bootstrapped with the single state Q-values. Contrary to CQ_NI this causes the joint Q-values to be updated using local state Q-values which are not representing the correct state-action values yet. In CQ_NI this table already contains more useful information since these have been updated while the algorithm was collecting samples for the sliding windows. This

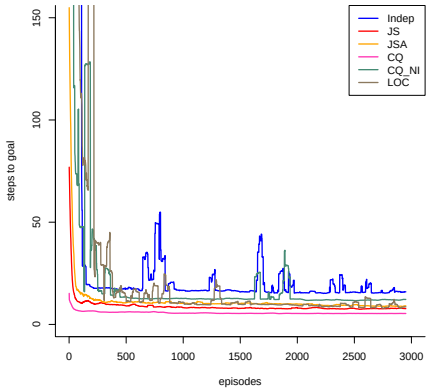
effect is visible in all but the smallest environments, in which learning goes so fast, that LoC already has good estimates very early in the learning process.
It is clear that the environment with the largest joint state space, i.e. *TunnelToGoal_3* is the most challenging for the joint-state and joint-state-action learners. They both need a long time before finding an acceptable solution in terms of the number of steps needed to complete an episode. Even after 20,000 episodes their policy is still significantly worse than the other algorithms (see Table 5.1).
In Figure 5.10(f) we show the evolution over the first 15,000 episodes for the CMU environment. We see that CQ_NI and Indep struggles with the layout of the environment. The agents have to select the same action for many timesteps to reach the other end of the environment, while they are unaware of the presence of other agents. Collisions with other agents result in long episodes before the goal is reached (if it is reached at all) if agents do not explore enough. CQ_NI is able to learn a solution faster and circumvent this issue since after some time, certain states get augmented and the agent can select its action using these augmented states.



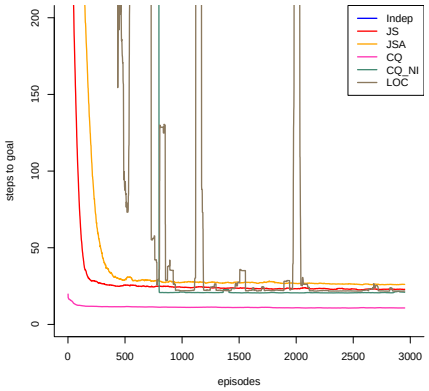
(a) Grid Game 2



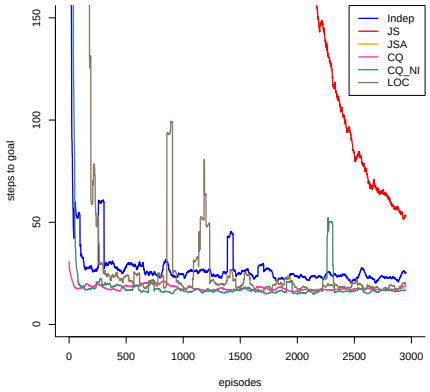
(b) TunnelToGoal



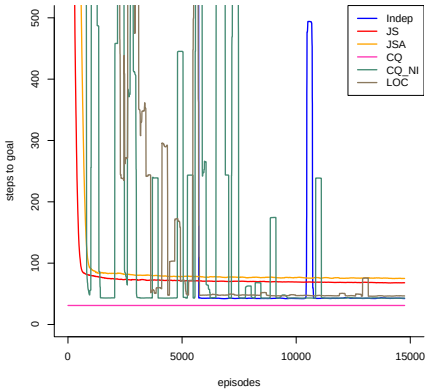
(c) ISR



(d) CIT



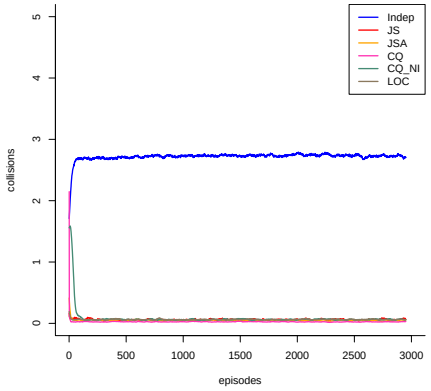
(e) TunnelToGoal_3



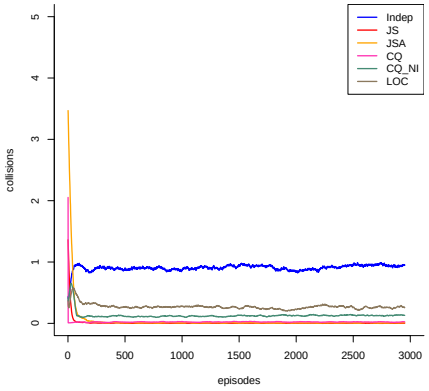
(f) CMU

Figure 5.10: Running average over 50 episodes of the number of steps all agents need to reach the goal in the different environments.

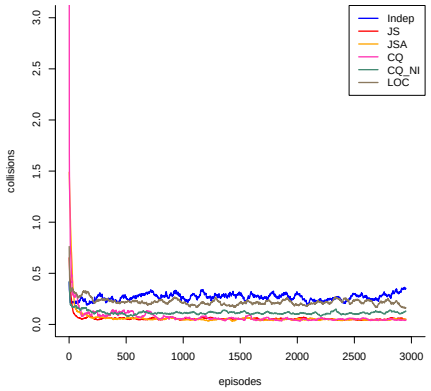
The evolution of the number of collisions between the agents is shown in Figure 5.11. Again, we only show the first 3,000 episodes.
Independent learners have trouble finding collision free policies except for environments (c), (d) and (f). In *ISR* agents learn to select non-interfering paths at the bottom left and as such still perform quite good, but in *CIT* and *CMU* however since the corridors are small, 1 agent takes a detour to the goal, whereas the other agent takes the shortest path. In *CIT* the shortest path consists of 10 steps, whereas the detour taken by the other agent requires 22 steps. In all other environments the agents collide in the same locations, but since they use an ϵ-greedy exploration strategy it is often the case that agents escape from a collision due to an exploratory action and as such do not observe the collision enough to learn to take a different path. The result is that they do not learn collision free policies.
Throughout all environments both CQ-learning variants learn collision free policies and learn to avoid collisions already early in the learning process. We see the same effect in the *TunnelToGoal_3* environment for JS and JSA as in the figure representing the number of steps to complete an episode. These algorithms need a much longer learning time before they can reduce the number of collisions.
In *CMU* the same behaviour as CQ_NI is observed. Because agents are still exploring a lot in this challenging environment, the number of collisions can suddenly increase if agents converge to a certain path towards the goal. Since it takes some time to collect enough samples for the statistical tests, we see several peaks in the number of collisions before the agents stabilise to a collision free policy.



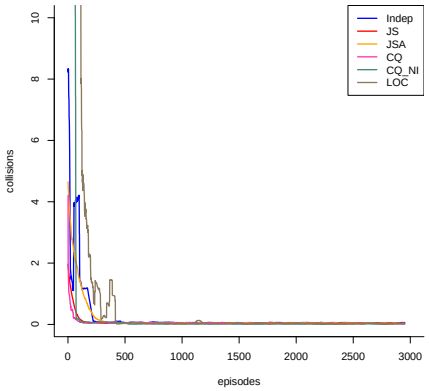
(a) Grid Game 2



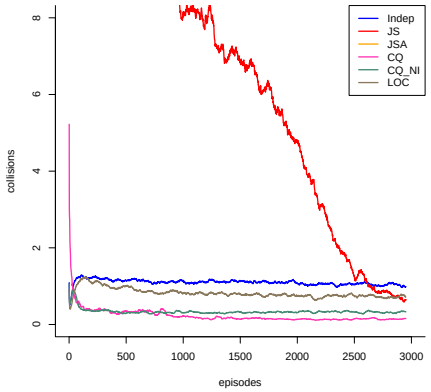
(b) TunnelToGoal



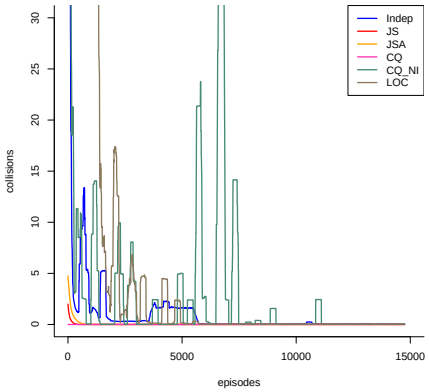
(c) ISR



(d) CIT



(e) TunnelToGoal_3



(f) CMU

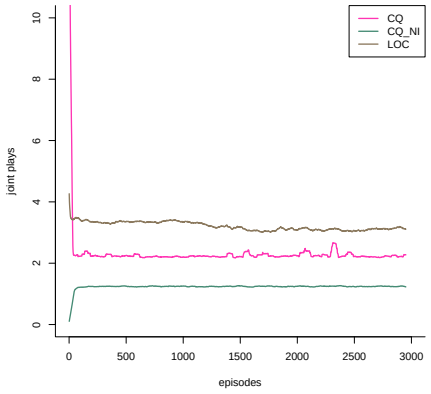
Figure 5.11: Running average over 50 episodes of the number of collisions that occur per episode.

Finally, in Figure 5.12 we show how often agents used more than just their local state space. For LoC the number of joint plays is the number of times the COORDINATE action was selected during an episode. For CQ and CQ_NI this is the number of times they used an augmented state to select an action. We do not show the other algorithms as for Indep this would be a constant line at 0 and for JS and JSA this information is the same as the number of steps per episode since these algorithms always use the complete system state to act.

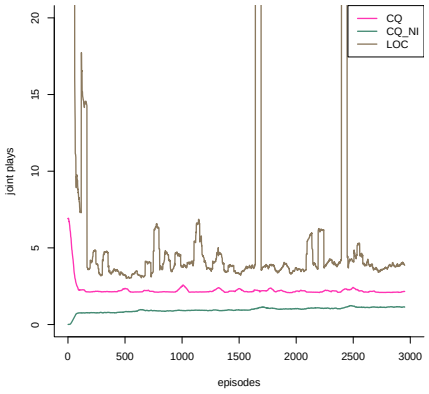
We see that LoC initially selects its COORDINATE action quite often, even more so in the *TunnelToGoal_3* environment. This is due to the fact that in the initial phases of the learning process, while agents are still searching for the goal, they can collide in every possible location and hence this COORDINATE action becomes attractive in every location. When agents start converging to their policy for reaching the goal, this action becomes less attractive due to the penalty associated with it, and the number of joint plays decreases.

Both CQ variants are more stable in the number of joint plays because selecting to use the state information of another agent does not depend on its exploration strategy, but on the samples it has collected. If a cost would have been attached to observing the state information of other agents, this means that this can be kept lower compared to LoC. Even though collision can happen in every location in the initial phase of the learning process, they must happen very regularly before the statistical test will conclude that the collisions happen significantly more often. With LoC this trade-off is less subtle since this algorithm uses a Q-value for the COORDINATE action. In combination with an ϵ-greedy strategy, this value is often selected as being the best, before the reward for reaching the goal is backpropagated enough to prefer one of the navigation actions.

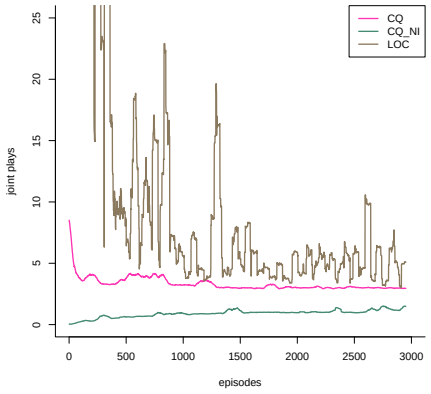
The cause of the peaks in the results of CQ_NI is the fact that the agents are simultaneously learning a policy and attempting to avoid other agents. If collisions suddenly occur more often, the state is augmented and the collisions decrease. This augmented state however influences the path the agents adopt and as such, the number of collisions might increase again since agents explore the best course of action in this augmented state. This continues until all relevant states are augmented and agents found a collision free policy. After some time, the augmented states that are not encountered anymore are reduced again and agents only use their local state information to act.



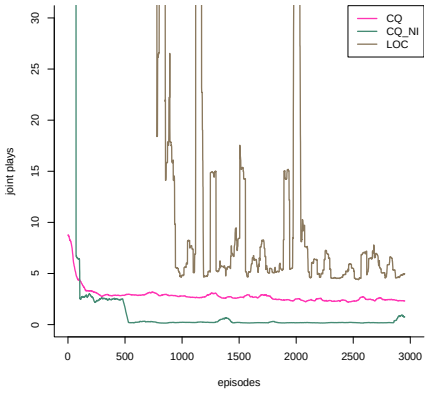
(a) Grid Game 2



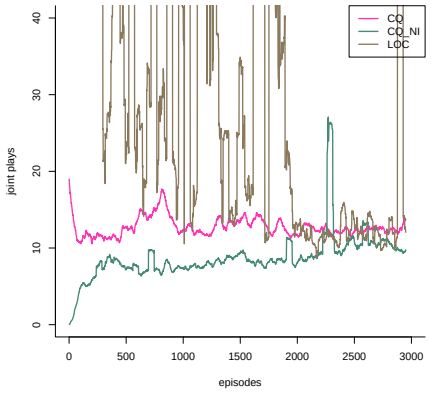
(b) TunnelToGoal



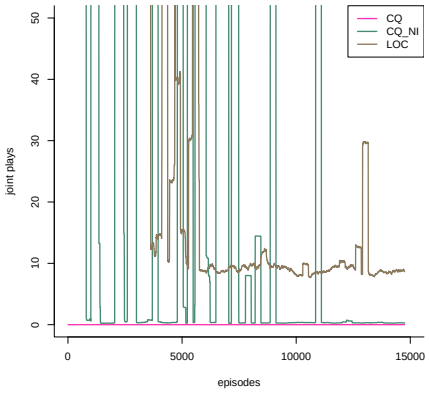
(c) ISR



(d) CIT



(e) TunnelToGoal_3



(f) CMU

Figure 5.12: Running average over 50 episodes of the number of times the agents choose to use state information about other agents per episode.

5.4 Notions on convergence

In this section we will analyse the behaviour of CQ-learning by using the *replicator dynamics* (RD). The RD is an approach, originating from evolutionary game theory (EGT), in which the dynamics of strategy reproduction in a population is investigated. EGT studies the propagation of strategies through a population of agents from a biological point of view. The spread of a strategy in a population is analysed by means of fitness and natural selection. At every timestep, two members of the population are selected randomly to play a game pairwise. The offspring an agent generates for the next generation is proportional to its performance in the game. Hence, strategies that have a high fitness will reproduce faster and spread throughout the population [Weibull (1997)]. An important principle is *evolutionary stability*. This means that when the entire population follows a strategy, and that the population is resistant to the invasion of another strategy. The kind of games we consider in this section are 2-player 2-action games which are represented by their payoff matrix. An example of such a matrix is given in Table 5.2.

		Column player	
		Action 1	Action 2
Row player	Action 1	1,1	1,2
	Action 2	2,1	0,0

Table 5.2: Payoff matrix of a normal form game

The first number in every cell in the matrix represents the payoff for the row player, whereas the second number represents the payoff for the column player. So if the row player selects Action 2 and the column player selects Action 1, the row player receives a payoff of 2 and the column player a payoff of 1. Given this basic introduction on (evolutionary) game theory, we will now present the basic notions on the replicator dynamics before showing how these can be used to analyse the behaviour of CQ-learning theoretically. For a more in depth overview of (evolutionary) game theory, we refer the reader to [Weibull (1997), Samuelson (1998), Osborne & Rubinstein (1999), Gintis (2000), Tuyls & Nowé (2005)].

5.4.1 Replicator dynamics

In this section, which is based on [Vrancx (2010)], we only consider the situation in which a population of agents play a pure strategy. We define the population of agents at timestep t as $n(t)$, with $n_i(t)$ agents playing a pure strategy $\sigma_i, i = 1, \ldots, r$ (for r pure strategies). The proportion of agents playing σ_i at timestep t is then $x_i(t) = n_i(t)/n(t)$ and the state of the population is described by the vector $\vec{x}(t) = (x_1(t), \ldots, x_r(t))$. Given an infinite population, this situation can be mapped on a single agent learning a mixed strategy for the game [Tuyls (2004)].

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

$$\bar{R}(\vec{x}) = \sum_i x_i R(\sigma_i, \vec{x}) \tag{5.3}$$

where R denotes the expected payoff in the game of strategy σ_i, when playing against a opponent, drawn randomly from population $\vec{x}$. Given that the amount of new individuals in the population 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:

$$\begin{aligned} x_i(t+1) &= \frac{N_i(t)R(\sigma_i, \vec{x}(t))}{\sum_{j=0}^{r} N_j(t)R(\sigma_j, \vec{x}(t))} \\ &= x_i(t)\frac{R(\sigma_i, \vec{x}(t))}{\bar{R}(\vec{x})} \end{aligned} \tag{5.4}$$

Which allows to write the evolution of σ_i in the population as:

$$x_i(t+1) - x_i(t) = x_i(t)\frac{R(\sigma_i, \vec{x}(t)) - \bar{R}(\vec{x})}{\bar{R}(\vec{x})} \tag{5.5}$$

The description above assumes that generations are not overlapping. Each generation lives for one period before being replaced by its offspring.

For the case where agents are continuously being replaced, the expected number of agents playing σ_i is given by:

$$n_i(t+\delta) = n_i(t) + \delta n_i(t)R(\sigma_i, \vec{x}(t)) \tag{5.6}$$

with δ the period of time in which a fraction of the population playing σ_i produces $R(\sigma_i, \vec{x}(t))$ offspring. This results in a population evolution described by:

$$x_i(t+\delta) - x_i(t) = x_i(t)\frac{\delta R(\sigma_i, \vec{x}(t)) - \delta\bar{R}(\vec{x}(t))}{1 + \delta\bar{R}(\vec{x}(t))} \tag{5.7}$$

Taking the limit $\delta \to 0$ and a proper rescaling of time, Equation 5.7 results in the continuous time replicator dynamic:

$$\frac{dx_i}{dt} = x_i(R(\sigma_i, \vec{x}(t)) - \bar{R}(\vec{x}(t))) \tag{5.8}$$

Assuming that the game rewards are given by the payoff matrix A, this equation can further be rewritten. The average payoff $\bar{R}$ for the entire population is the product $\vec{x}A\vec{x}$ and the expected payoff for strategy σ_i is the i-th component of the vector $A\vec{x}$ (denoted by $(A\vec{x})_i$). This results in the following equation:

$$\frac{dx_i}{dt} = x_i((A\vec{x})_i - \vec{x}A\vec{x})) \tag{5.9}$$

If multiple populations are playing the game, for instance in asymmetric games[3],

[3] In asymmetric games the different players of the game do not necessarily have the same action sets at their disposal.

we need two systems of differential equations: one for the population representing the row player (X), and one for the population representing the column player (Y). This results in the following replicator equations for the two populations:

$$\frac{dx_i}{dt} = [(A\vec{y})_i - \vec{x} \cdot A\vec{y}]x_i \tag{5.10}$$

$$\frac{dy_i}{dt} = [(B\vec{x})_i - \vec{y} \cdot B\vec{x}]y_i \tag{5.11}$$

From Equations 5.10 and 5.11 it is clear that the growth rate of the types in each population is also determined by the composition of the other population.

These replicator equations can be linked to evolutionary stable strategies by following theorem [Hofbauer et al. (1979)]:

Definition 24. *If a strategy $\vec{\sigma}$ is an evolutionary stable strategy, then the population state $\vec{x} = \vec{\sigma}$ is asymptotically stable under the replicator dynamic.*

This theorem means that for any path starting sufficiently close to state $\vec{x}$, this path will converge to state $\vec{x}$ under the RD. Hence, the evolutionary stable strategies are attractor points for the RD.

In 1997 Börgers and Sarin demonstrated the connection between evolutionary game theory and reinforcement learning [Börgers & Sarin (1997)], by studying the continuous time limit of an RL update scheme called Cross' Learning [Cross (1973)]. In 2004, Tuyls developed a version of the RD, allowing us to analyse the behaviour of Q-learning with a Boltzmann exploration strategy [Tuyls et al. (2003)]. Finally, Wunder et al. analysed the convergence of Q-learning with an ϵ-greedy exploration strategy in 2-player 2-action games in [Wunder et al. (2010)]. In the next section we use the replicator equations from [Tuyls et al. (2003)] to analyse the behaviour of CQ-learning. Suppose the payoff matrices are represented by A and B for the two players and τ is the temperature of the Boltzmann exploration function, these equations are:

$$\frac{dx_i}{dt} = \frac{\alpha}{\tau}((A\vec{y})_i - \vec{x} \cdot A\vec{y})x_i + x_i\alpha \sum_j x_j ln(\frac{x_j}{x_i}) \tag{5.12}$$

$$\frac{dy_i}{dt} = \frac{\alpha}{\tau}((B\vec{x})_i - \vec{y} \cdot B\vec{x})y_i + y_i\alpha \sum_j y_j ln(\frac{y_j}{y_i}) \tag{5.13}$$

5.4.2 Convergence of CQ-learning

5.4.2.1 One step coordination problems

In this section we analyse the behaviour of CQ-learning by means of the Grid game 2 environment as an example. The grid game, together with the numbered locations is shown in Figure 5.13(a).

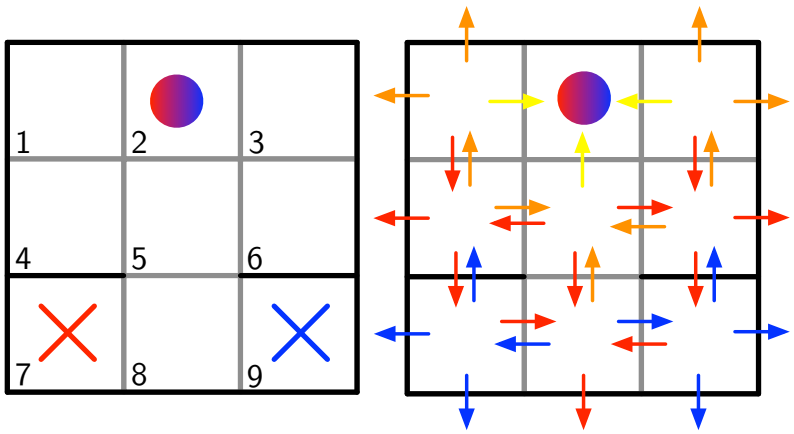


Figure 5.13: (a) Grid game 2 environment with numbered locations. (b) Representation of the Q-values of one agent acting alone for the different actions in the different states if only a reward of $+20$ is given for reaching the goal. (Yellow $= +20$, Orange $= +18$, Red $= +16.2$, Blue $= +14.58$)

We use the first variant of CQ-leaning, in which agents have converged to the optimal single agent policy. Three possible scenarios exist for the interaction state in which the agents are in locations $\langle 7, 9 \rangle$.

1. The red agent in location 7 has detected the conflict and is selecting an action based on augmented state information $\langle 7, 9 \rangle$, whereas the blue agent has not detected any conflict yet and selects an action using state information $\langle 9 \rangle$.
   In this situation the blue agent is still following its single agent policy, which is a stationary policy from the viewpoint of the red agent. Hence, the red agent will select an action that will cause it to remain in location 7, while the blue agent moves to location 8. In the next timesteps, the agents are not in a conflict situation and both of them follow their single agent policy until they reach the goal.

2. The blue agent in location 9 has detected the conflict and is selecting an action based on augmented state information $\langle 9, 7 \rangle$, whereas the red agent has not detected any conflict and selects an action using state information $\langle 7 \rangle$.
   The outcome of this situation is similar to the one described above, with the red agent selecting its preferred action and reaching the goal first.

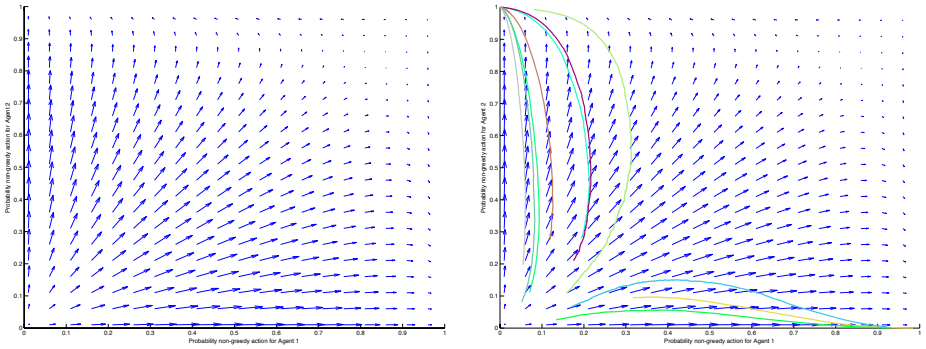
3. Both agents have detected the conflict situation and are selecting their action based on augmented state information. As such, they are playing a coordi-

nation game between selecting the action that would take them to location 8 (and colliding if the other agent does the same), or taking any of their other actions and remaining in their respective cells. Based on a reward of $+20$ for reaching the goal, a penalty of -10 for colliding and a discount factor of 0.9, the particular game is given by Table 5.3. These values are deduced from the single agent Q-values, since these are the ones used in the bootstrapping process of CQ-learning. These values are shown in Figure 5.13(b). A yellow arrow represents a value of $+20$, an orange arrow represents $+18$, a red arrow $+16.2$ and finally a blue arrow represents a value of $+14.58$. The greedy action in state $\langle 7 \rangle$ or state $\langle 9 \rangle$ results in a transition to state $\langle 8 \rangle$ if only one of the agents selects it, hence its value of $+16.2$. If both agents select it, the value in the table represents the discounted value of the state the agent is currently in minus the penalty for colliding. Note that the value of the state equals the Q-value of the best action for that state.

		Blue Agent	
		Non-greedy	Greedy
Red Agent	Non-greedy	14.58,14.58	14.58,16.2
	Greedy	16.2,14.58	4.58,4.58

Table 5.3: Payoff matrix of the coordination game in the Grid game 2 environment for the situation where both agents are selecting an action using augmented state information.

The behaviour of this system can be analysed using Equations 5.12 and 5.13 of the replicator dynamics, and the basins of attraction are shown in Figure 5.14(a). The settings were $\alpha = 0.00001$ and $\tau = 0.01$. On the right hand side, in Figure 5.14(b) we plotted sample paths of the Q-learning process with the same parameters. The probability of selecting the action `non-greedy` is on the x-axis for the red agent and on the y-axis for the blue agent. From this figure it is clear that the learning process approximates the paths of the differential equations to one of the stable points (i.e $\langle$`greedy`,`non-greedy`$\rangle$ or $\langle$`non-greedy`,`greedy`$\rangle$).



(a) (b)

Figure 5.14: The x-axis represents the probability of selection action `non-greedy` for the red agent. The y-axis represents this probability for the blue agent. (a) Vector field of the replicator dynamics for $\alpha = 0.00001$ and $\tau = 0.01$ and the payoff matrix given in Table 5.3. (b) Sample paths of Q-learning under the same settings.

From this we can conclude that CQ-learning will solve the coordination problem for all three scenarios, under the assumption that the payoff matrix for the coordination game resembles the matrix shown in Table 5.4, with $P < S < T$.

		Agent 2	
		Non-greedy	Greedy
Agent 1	Non-greedy	S,S	S,T
	Greedy	T,S	P,P

Table 5.4: Payoff matrix of the coordination game in CQ-learning for the situation where both agents are selecting an action using augmented state information.

If we take the value of the next state after an interaction as $V^*(C)$ (where C is state $\langle 8 \rangle$ in the Grid game 2 environment, so $V^*(C) = 18$ for this example), P_1 the penalty for the `non-greedy` action and P_2 the penalty for miscoordination, these values are as follows:

- $S = P_1 + \gamma^2 V^*(C)$
- $T = \gamma V^*(C)$
- $P = P_2 + \gamma^2 V^*(C)$

From these values it is clear that $P < S < T$ will hold as long as $P_2 < P_1$. In the example described above, $P_2 = -10$ and $P_1 = 0$.

5.4.2.2 Multiple step coordination problems

In some situations the agent transitions between multiple augmented states. This is the case in the solution depicted in Figure 5.5, where the actions of the red agent in timesteps 6 and 7 are both selected based on augmented state information. In this situation the agents repeatedly played the coordination game of Table 5.4. The values are obtained in the same way as for the Grid Game 2 environment. The game played at timestep 6 is given in Table 5.5 and at timestep 7 in Table 5.6.

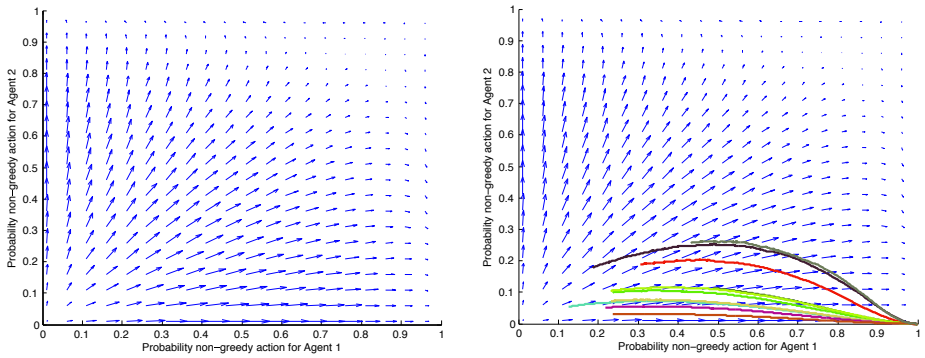
		Blue Agent	
		Non-greedy	Greedy
Red Agent	Non-greedy	11.81,11.81	11.81,13.13
	Greedy	13.13,11.81	1.81,1.81

Table 5.5: Payoff matrix of the coordination game in the 2-robot environment from Figure 5.5 at timestep 6.

		Blue Agent	
		Non-greedy	Greedy
Red Agent	Non-greedy	10.63,13.12	11.81,14.58
	Greedy	11.81,13.13	1.81,4.58

Table 5.6: Payoff matrix of the coordination game in the 2-robot environment from Figure 5.5 at timestep 7 if both agents augmented their state information.

The game in Table 5.5 follows the outline described in Table 5.4 and the agents will converge to one of the Nash equilibria. In Table 5.6 we show the game matrix for the situation where the blue agent played its `greedy` and the red agent played its `non-greedy` action in the previous timestep. The payoffs for both agents are not symmetrical anymore since the blue agent is closer to achieving its goal, hence also the higher values in its payoff matrix. The dynamics of this game are given in Figure 5.15(a). Because the payoffs of the blue agent are higher, the dynamics for playing its `greedy` action are stronger than those to play its `non-greedy` action. Moreover, it is logical to bias the agent having selected its `non-greedy` action at the previous timestep, to do this again. The sample paths of the behaviour of Q-learning agents are given in Figure 5.15(b). We clearly see that Agent 1 again selects its `non-greedy` action to solve the coordination problem.



(a) (b)

Figure 5.15: The x-axis represents the probability of selection action `non-greedy` for the agent having selected its `non-greedy` action in the previous timestep. The y-axis represents this probability for the other agent. (a) Vector field of the replicator dynamics for $\alpha = 0.00001$ and $\tau = 0.01$ and the payoff matrix given in Table 5.6. (b) Sample paths of Q-learning under the same settings.

The same reasoning can be made for even more consecutive interactions and the analysis of the dynamics performed in the same way. However, the scope of this dissertation is to learn using sparse interactions in which coordination is only needed in a limited set of states and not in the entire state space.

5.5 Discussion and related work

When trying to learn good policies in environments where multiple agents are present, many things must be taken into consideration when selecting an algorithm. For instance, is all the information about the other agents available or can it be obtained through communication, do the agents have common interests, how many other agents are present, etc. If all information is available, learning in the entire joint-state joint-action space will be beneficial, but at a high cost in terms of learning time. If this cost is too high, independent learners could be used, but the outcome of this approach is very uncertain.

For these reasons a new research track in multi-agent reinforcement learning has emerged that lies in between both of these approaches. Although sparse interactions is a new paradigm within MARL, the ideas behind it, are closely related to feature selection techniques which go back to the late 1990s. CQ-learning can be seen as a feature selection technique which is learning when state features, containing information about other agents, is relevant. For a thorough overview of feature selection we refer to [Guyon et al. (2006)]. These ideas were first transferred to

single agent RL. The G-tree algorithm is a decision tree learning algorithm that aims to generalise inputs in large domains [Chapman & Kaelbling (1990), Chapman & Kaelbling (1991)]. It does so, by incrementally building a tree-structured Q-table and splits leafs based on statistics of the reinforcement data rather than input-output pairs. With memory considerations in mind, this algorithm only maintains statistics in the leaf nodes. However, this implies that at each split, the algorithm must re-learn that part of the state space from scratch. Moreover, the G-algorithm is only capable of detecting significant distinctions in isolation. I.e. only one state feature may be responsible for a significant change in the reinforcement in order to be detected. These two limitations of the G-algorithm were the inspiration of an improved version, called the U-tree algorithm [McCallum (1995)].

These ideas from single agent RL found their way to multi-agent RL with the Utile Coordination algorithm [Kok et al. (2005)]. This algorithm is restricted to common interest problems and requires full observation about the state information of all agents, and the actions they performed. In certain domains, where this information is only available through explicit communication, this might be a very costly operation. CQ-learning relaxes this requirement by only requesting joint state information when there are strong indications that the presence of other agents causes significant influence for certain states.

Another approach using sparse interactions that does not explicitly rely on statistical tests is LoC [Melo & Veloso (2009)]. This algorithm, together with Utile Coordination, are explained in detail in Section 3.3.1. LoC allows for conflicting interest games, but imposes an additional requirement: an active perception mechanism must be present to detect the influence of another agent. A requirement not present in CQ-learning, since it uses its statistical tests on the joint state information to detect this influence. Moreover, LoC was only tested for domains where the influence of another agent was limited to a predefined number of states. If this influence was present in all states, as was the case in the experiments in this chapter, LoC could be seen as a simplified version of the 2Observe algorithm, introduced in Chapter 4. Both algorithms learn single and multi-agent policies in the same way, but 2Observe also simultaneously learns the active perception function, whereas LoC assumes this function is available. Also note that although the pseudocode in Algorithms 7 and 8 describe a setting in which a statistical test at every timestep, it is more realistic to perform this test every t timesteps, where t is chosen in such a way that the system still responds to coordination problems in a reactive way.

Finally, CQ-learning also bears resemblance to Hierarchical RL (HRL) such as the Max-Q algorithm [Dietterich (1998)]. The learning problem could be decomposed in several subtasks, each of which requires different state information to accomplish that particular subtask. If this information is available, HRL approaches could be used. In [Dietterich (2000)] a proof is given regarding the assumptions to which

this state abstraction must comply in order to converge. In [Jong & Stone (2004)] a method is described to learn this state abstraction, by analysing the underlying MDP the agent is trying to solve. The focus of CQ-learning is also to learn which state information is relevant, but is learning in a multi-agent environment. It is augmenting its state representation to include all the relevant information in a state, rather than to remove all irrelevant information as is done in [Jong & Stone (2004)].

To summarise, CQ-learning uses more relaxed assumptions compared to other techniques that exploit the sparse interactions that exist between agents. The algorithm still assumes that state information about other agents is accessible, but attempts to explicitly learn when it is necessary to use this information. Compared to commonly accepted multi-agent approaches, CQ-learning also reaches collision free policies, but requires less steps to complete an episode and uses minimal extra information to do so. The cost however is that CQ-learning is computationally more expensive than these other approaches because of the statistical tests and the samples required for them. On the other hand, CQ-learning does not suffer from the exponential explosion in the state-action space in function of the number of agents and, as can be seen from the experiments with three agents, the benefits of CQ-learning become more explicit when more agents are present in the environment. Also note, that CQ-learning does not require all agents to be present in the environment from the beginning. Agents can be added and removed during the learning process, since CQ-learning treats all agents homogeneously and learns in which states to observe the other agents. If another agent is no longer present, the action can be selected using only local state information.

5.6 Summary

In this chapter we presented two variants of CQ-learning. This algorithm uses statistical tests on the immediate reward signal to take the midground between acting completely independent using only the local state space of an agent and acting in a complete joint-state space. The first variant uses a single agent model of the reward function of the problem task as a baseline to detect the influence of other agents. If the test concludes that a richer state representation is required, the model could be extended to include this additional information and plan in a sparse multi-agent model. Although not explored in this chapter, this single agent model of the reward function could also serve as a basis for Dyna approaches to increase the learning speed of the core task.

The second variant does not require this model, but uses a sliding window to store the immediate rewards and compares them against the first rewards seen for a particular state-action pair. This principle is inspired by the fact that the number of interactions between agents increases as learning progresses and as agents start

to converge to a policy. As such, these first rewards are a good estimate of the expected payoff if agents are acting alone in the environment. Since this baseline is heavily influenced by the behaviour of the agents in the early phases of the learning process a confidence value is maintained for the augmented states. This value allows the reduction to local states if experience shows that these augmented states are only rarely visited.

We have validated our approach in a set of gridworlds of various size and difficulty and showed the set of states in which the agents use global state information. These experiments allow us to conclude that agents using CQ-learning not only learn collision free policies in these gridworlds, but also learn shorter paths than other multi-agent approaches. CQ-learning was also applied to the Khepera robots, using an overhead camera to obtain the state information of other agents. The results obtained in this scenario were similar to the experiments described in this chapter.

CQ-learning is however not restricted to gridworlds, but can be applied to any domain, in which the influence of other agents is reflected in the immediate reward signal the agents experience. In many situations however, the influence of other agents, is only reflected several timesteps ahead in the future. When using gridworlds as an example this could for instance be when the order in which agents enter the goal is important. Problems characterised by these delayed influences are the scope of the research presented in the next chapter.

Chapter 6

Solving delayed coordination problems

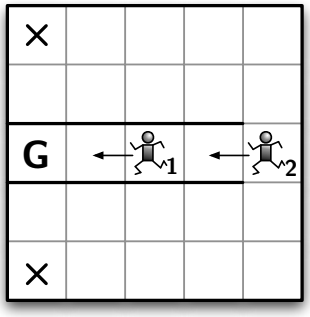
The future depends on what we do in the present.
– *Mohandas Karamchand Ghandi, 1969-* –

One of the main advantages of RL is the capability of dealing with a delayed reward signal. Using an appropriate backup diagram, rewards are backpropagated through the state space. This allows agents to learn to take the correct action that results in the highest future (discounted) reward, even if that action results in a suboptimal immediate reward in the current state. In a multi-agent environment, agents can use the same principles as in single agent RL, but have to apply them in a complete joint-state-joint-action space to guarantee optimality. Learning in such a state space can however be very slow as argued in previous chapters. In this chapter we present our approach for mitigating this problem. *Future Coordinating Q-learning* (FCQ-learning) detects strategic interactions between agents several timesteps before these interactions occur. FCQ-learning uses the same principles as CQ-learning (see Chapter 5) to detect the states in which interaction is required, but several timesteps before this is reflected in the reward signal. In these states, the algorithm will augment the state information to include information about other agents which is used to select actions. The techniques presented in this chapter are the first to explicitly deal with a delayed reward signal when learning using sparse interactions.

6.1 Delayed coordination problems

In single agent RL, the reward signal an agent receives for an action may be delayed. When multiple agents are acting together and influencing each other, the effect of

such interactions may only become apparent during the course of action. Let us consider a variant on the *TunnelToGoal* environment as an example, depicted in Figure 6.1. Agents have to reach the goal location in a predetermined order, i.e. Agent 1 must reach the goal location before Agent 2. This requirement is reflected in the reward signal the agents receive when they reach the goal. If Agent 1 is first, they both receive a reward of $+20$, if Agent 2 is first in the goal state, both agents only get a reward of $+10$. Independent learners are once again unable to detect the reason for this change in the reward signal since they are unaware of the other agent and as such cannot learn to reach the optimal policy. Agents observing the complete system state will be able to solve this problem, but as explained in the previous chapter, this imposes high requirements on the observability the agents have about the system or their communication abilities.



Reward for reaching the goal:
Agent 1 is first: Both receive $+20$
Agent 2 is first: Both receive $+10$

Figure 6.1: A variant of the TunnelToGoal environment in which the order with which the agents enter the goal influences the reward they observe.

Since the path to the goal is surrounded by walls, the agents must coordinate at the entrance of the goal, in order to enter the goal in the correct order. They will however only observe the fact that they had to coordinate when it is already too late, i.e. when they have reached the absorbing goal state. In a similar way, if we think about mobile robots navigating in an environment, it is possible that there are some bottleneck areas, such as small alleys where robots cannot cross each other.

6.2 Learning with delayed coordination problems

In this section we explain our approach of dealing with delayed coordination problems. So far, all research within the sparse interaction framework are only using immediate rewards as a way to detect the need for coordination. As we explained in the previous section, this view is too limited, since it is not acceptable to assume that this need for coordination is reflected immediately in the reward signal following the action. Using the full MG view of the system, such delayed reinforcement signals

are propagated through the joint state space and algorithms using this MG view can still learn optimal policies. It should however be clear by now, that this view of the system is not a realistic one. A DEC-SIMDP is more realistic since it models exactly the coordination dependencies that exist between agents in a limited set of states. Since it is not modeled how these dependencies can be resolved, the DEC-SIMDP is still applicable as a framework for delayed coordination problems. We will follow the same approach as in the previous chapter and will again introduce two variants of an algorithm, called *Future Coordinating Q-learning* (FCQ-learning). This algorithm is closely related to CQ-learning described in Chapter 5. Coordination states are again detected by means of statistical tests on the reward signal, after which the conflicting states are augmented to include the state information of the agents participating in the interaction.

6.2.1 FCQ-learning with initialised agents

The first variant of FCQ-learning assumes that agents have been learning for some time alone in the environment. As a result, their Q-values have converged to the true state-action values. These Q-values will be the baseline for the statistical tests that will determine in which states coordination is needed. The basic idea is that if agents experience a negative influence from each other, the Q-values for certain state-action pairs will decrease. Since the Q-values are used to bootstrap, this influence will gradually spread throughout the Q-table. We illustrate this effect in the environment depicted in Figure 6.2. The agent's initial position is marked with an **X**, its goal, with the letter **G**. One agent was learning alone in the environment and was given a reward of $+20$ for reaching the goal. Moving into a wall was penalised with -1. All other actions resulted in a payoff of 0. The agent was trained using a learning rate of 0.02 and acted completely random until its Q-values converged. This exploration strategy ensures that all state-action pairs are visited enough to allow the Q-values to converge to the true state-action values. After convergence, this Q-table was stored in Q^*.

After the learning process, the reward for reaching the goal was decreased to $+10$ and the agent selected its actions using an ϵ-greedy strategy with $\epsilon = 0.1$. In Figure 6.3 we show the evolution of the Q-values for the actions of the policy to which the agent converged. In the legend of this figure we show the index of the state (which corresponds to the indices in Figure 6.2) together with the index of the action ($1 =$ `NORTH`, $2 =$ `EAST`, $3 =$ `SOUTH`, $4 =$ `WEST`). The state at the top of the legend is the one closest to the goal, the one at the bottom is the initial position of the agent. We see that the Q-values quickly drop near the goal, followed by the Q-values for states further and further away from the goal until the start location of the agent.

To detect these changes statistically, FCQ-learning uses a Kolmogorov-Smirnov test (KS-test) for goodness of fit. This statistical test can determine the significance

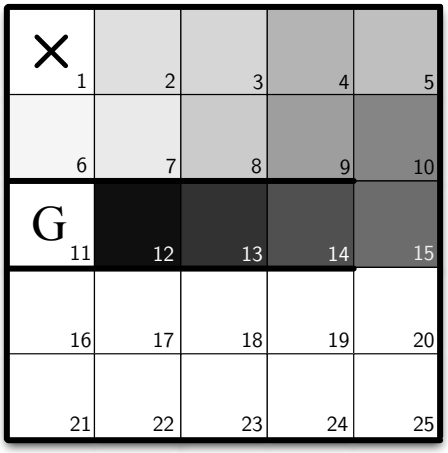


Figure 6.2: Evolution of the states in which a KS-test for goodness of fit detects a change in the Q-values. The darker the shade of the cell, the earlier the change is detected.

of the difference between a given population of samples and a specified distribution. A thorough explanation of this test is given in Appendix B.2. Since the agents have converged to the correct Q-values, the algorithm will compare the evolution of the Q-values when multiple agents are present to the values it learned when acting alone in the environment. To validate this idea we tested its concepts in the *TunnelToGoal* environments from the previous chapter.

To validate the use of a statistical test to detect these changes, a window of Q-values was maintained with the last N values of the Q-value of that particular state-action pair in the experiment described above. We will refer to this window as $W_k^Q(s_k, a_k)$. This window, contains the evolution of the Q-value of that state-action pair over the last N updates after we decreased the reward for reaching the goal. A KS-test for goodness of fit was used to compare the values of $W_k^Q(s_k, a_k)$, to the optimal Q-value $Q^*(s_k, a_k)$. The order in which significant changes in the Q-values are detected is shown in Figure 6.2. The darker the shade of the cell, the earlier the change was detected. The KS-test detected this change first in the Q-values of the cell adjacent to the goal state. Since the Q-values are still being updated, the KS-test continued detecting changes further away from the goal, towards the starting position of the agent. This experiment was done using a confidence level of 99.99% for the KS-test. Even with this confidence, the test correctly identifies the states in which the Q-values change due to the changed reward signal and does not identify additional changes due to small fluctuations in the Q-values.

These states narrow down the set of states we have to consider to detect in which state we actually have to coordinate. In these states, our Q-values are significantly

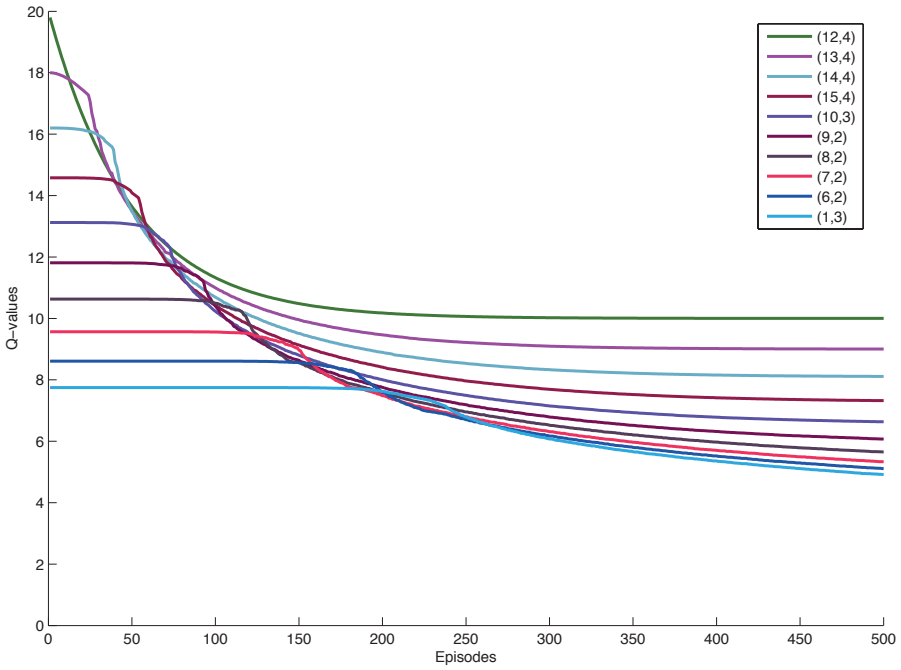


Figure 6.3: Evolution of the Q-values for the optimal policy after the reward signal for reaching the goal was altered from +20 to +10.

deteriorating and since the Q-values give an indication of the best possible future rewards an agent can expect from that state onward, it is in these states that FCQ-learning will sample the state information of other agents, together with the received rewards *until the episode ends*. This approach of collecting rewards until termination of an episode is known as *Monte Carlo* sampling. Again, this principle is similar to how CQ-learning samples, but in FCQ-learning the collected rewards until termination of the episode are stored instead of just the immediate rewards. These rewards are also grouped, based on the state information of the other agents. This is shown in Figure 6.4.

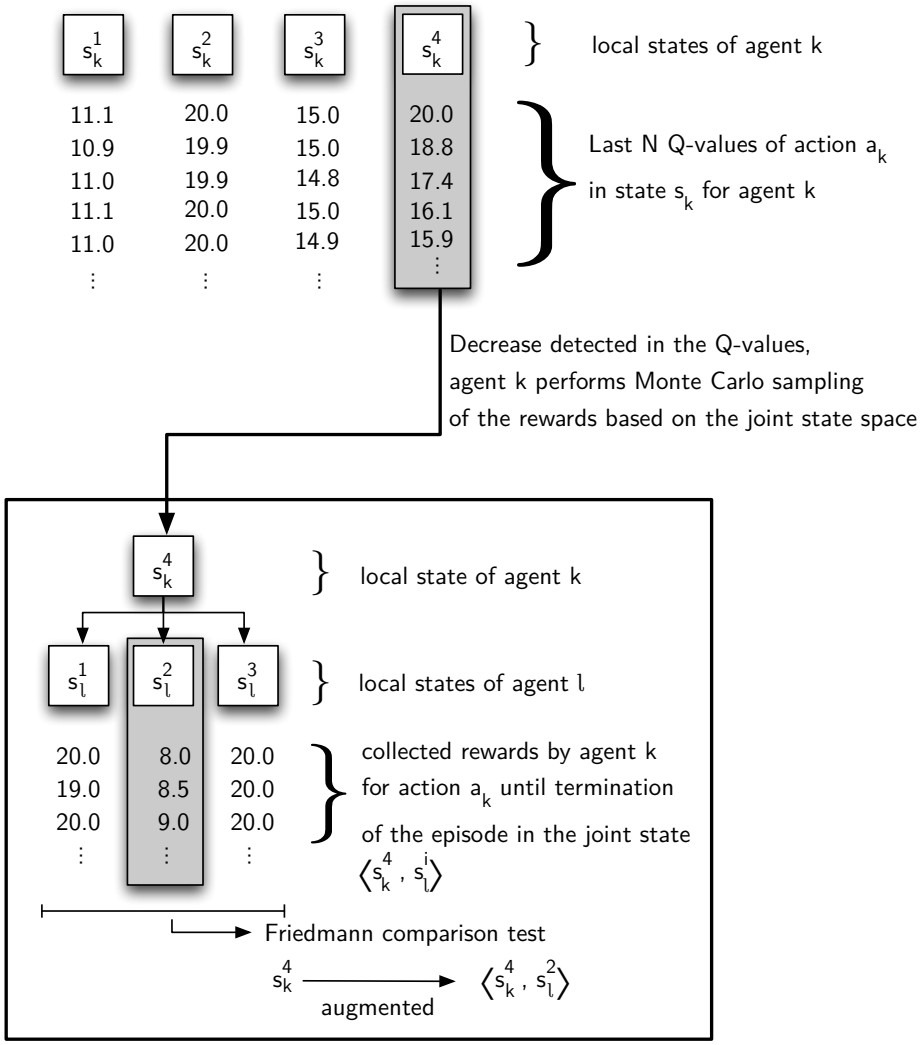


Figure 6.4: Detecting conflict states with FCQ-learning

In every local state in which a change in the Q-values was detected, the agent will observe the state information of the other agents when it is at that local state and collect the rewards until termination of the episode. When the algorithm has collected enough samples, it performs a Friedmann test. This non-parametric statistical test is used to compare observations repeated on the same subjects, or in this case, on the same local states. More information about this test can be found in Appendix B.3. Using a multiple comparison test on this statistical information, the algorithm can determine which state information of other agents is influencing these future rewards and hence augment the local state of the agent with the relevant information about other agents. It should be noted that these states will be augmented in a similar order as the changes in the Q-values are being detected. The

algorithm will however continue augmenting states, until it reaches a state in which the coordination problem can actually be solved. Similar to the second variant of CQ-learning, described in Section 5.2.3, a confidence value is maintained for every augmented state. This confidence value is increased each time an augmented state is observed, and decreased if the augmented states of the current local state are not observed. As such, states that are not observed anymore after the coordination problem was solved, are being reduced again to simple local states.

Action selection is done in the same way as for CQ-learning. Agents will check if their local state has previously been augmented. If this is the case, they will observe the system state in order to verify if the agents are currently in that augmented state. If so, they will select an action based on this augmented state and increase its confidence value. Otherwise, they will select an action using their local state information and decrease the confidence value for the augmented states.

Updating the Q-values is also done in the same way as CQ-learning. We refer the reader to Section 5.2.1 for the explanation and the update rules.
The pseudo code for FCQ-Learning is given in Algorithm 9

6.2.2 FCQ-learning with random initial Q-values

Having initialised agents beforehand which have learned the correct Q-values to complete the single agent task is an ideal situation, since agents can transfer the knowledge they learned in a single agent setting to a multi-agent setting, adapting only their policy when they have to. Since this is not always possible, we propose a simple variant of FCQ-learning. In the algorithm presented in Section 6.2.1, the initialised Q-values are being used for the KS-test which will detect in which states the agent should start sampling rewards. As such, this test prevents sampling rewards and state information about the other agents in those states where this is not necessary, since it allows an agent to only sample in those states that are being visited by the current policy and in which a change has been detected. If this limited set of states in which coordination problems should be explored cannot be obtained because it is impossible to train the agents independently first, it is possible to collect samples for every state-action pair at every timestep. This results in a lot more data to run statistical tests on, most of which will be irrelevant, but relaxes the assumption of having the optimal Q-values of the single agent problem beforehand. The changes in Algorithm 9 for this variant are to remove the lines regarding the KS-test on lines 11 to 14 and line 19 and to change the training of the agents on line 1. The resulting algorithm is given in Algorithm 10.

Algorithm 9 FCQ-Learning algorithm for agent k

Train Q_k independently first and store a copy in Q'_k, initialise Q_k^{aug} to zero, and list of sample states to $\{\}$;
Set $t = 0$
while true do
  observe local state $s_k(t)$
  if $s_k(t)$ is part of an augmented state $\vec{s}_k$ and the information of $\vec{s}_k$ is present in $s(t)$ **then**
    Select $a_k(t)$ according to Q_k^{aug} using $\vec{s}_k$
  else
    Select $a_k(t)$ according to Q_k using s_k
  end if
  observe $r_k = R_k(s(t), a(t))$, s'_k from $T(s(t), a(t))$
  if KS-test fails to reject the hypothesis that the Q-values of $Q'_k(s_k(t), a_k(t))$ are the same as $Q_k(s_k(t), a_k(t))$ **then**
    add state $s_k(t)$ to the list of sample states
  end if
  if $s_k(t)$ is a sample state **then**
    Store the state information of other agents, and collect the rewards until termination of the episode
    if enough samples have been collected **then**
      perform Friedmann test on the samples for the state information of the other agents. If the test indicates a significant difference, augment s_k to include state information of the other agents for which a change was detected.
    end if
  end if
  if $s_k(t)$ is part of an augmented state $\vec{s}_k$ and the information of $\vec{s}_k$ is present in $s(t)$ **then**
    Update $Q_k^{\text{aug}}(\vec{s}_k, a_k) \leftarrow (1-\alpha_t) Q_k^{\text{aug}}(\vec{s}_k) + \alpha_t [r_k + \gamma \max_{a'_k} Q_k(s'_k, a'_k)]$
    increment confidence value for $\vec{s}_k$
  else
    Update $Q_k(s_k) \leftarrow (1-\alpha_t) Q_k(s_k) + \alpha_t [r_k + \gamma \max_{a'_k} Q_k(s'_k, a'_k)]$.
    decrease confidence value for all $\vec{s}_k = \langle s_k, s_l \rangle$ for which s_l is not present in $s(t)$.
  end if
  $t = t + 1$
end while

Algorithm 10 FCQ-Learning algorithm with random initial Q-values for agent k

Initialise Q_k and Q_k^{aug} to zero, and list of sample states to $\{\}$;
Set $t = 0$
while true do
  observe local state $s_k(t)$
  if $s_k(t)$ is part of an augmented state $\vec{s}_k$ and the information of $\vec{s}_k$ is present in $s(t)$ **then**
    Select $a_k(t)$ according to Q_k^{aug} using $\vec{s}_k$
  else
    Select $a_k(t)$ according to Q_k using s_k
  end if
  observe $r_k = R_k(s(t), a(t))$, s'_k from $T(s(t), a(t))$
  Store the state information of other agents, and collect the rewards until termination of the episode
  if enough samples have been collected **then**
    perform Friedmann test on the samples for the state information of the other agents. If the test indicates a significant difference, augment s_k to include state information of the other agents for which a change was detected.
  end if
  if $s_k(t)$ is part of an augmented state $\vec{s}_k$ and the information of $\vec{s}_k$ is present in $s(t)$ **then**
    Update $Q_k^{\text{aug}}(\vec{s}_k, a_k) \leftarrow (1-\alpha_t)Q_k^{\text{aug}}(\vec{s}_k) + \alpha_t[r_k + \gamma \max_{a'_k} Q_k(s'_k, a'_k)]$
    increment confidence value for $\vec{s}_k$
  else
    Update $Q_k(s_k) \leftarrow (1-\alpha_t)Q_k(s_k) + \alpha_t[r_k + \gamma \max_{a'_k} Q_k(s'_k, a'_k)]$.
    decrease confidence value for all $\vec{s}_k = \langle s_k, s_l \rangle$ for which s_l is not present in $s(t)$.
  end if
  $t = t + 1$
end while

6.3 Experimental results

We use a set of two and three-agent gridworld games in which we introduced delayed coordination problems. These environments are shown in Figure 6.5. Agents cannot only collide with each other in every cell, but in environments (a), (b) and (c) the agents also have to enter the goal location in a specific order. In environment (d) it is clear that if agents adopt the shortest path to the goal, they collide in the middle of the corridor. The initial position of the agents is marked by an **X**, the goal is indicated with a bullet in the same colour as the initial position of that agent. For environments in which the agents share the same goal, the goal is indicated with a linear blend. Agents receive a reward of $+20$ if they reach the goal in the correct order, otherwise they only receive a reward of $+10$. Collisions between agents are penalised with -10, moving into a wall is penalised with -1. We use this illustrative setting, but this problem can easily be mapped on a production process where the different parts that constitute the finished product have to arrive in a certain order to the assembly unit.

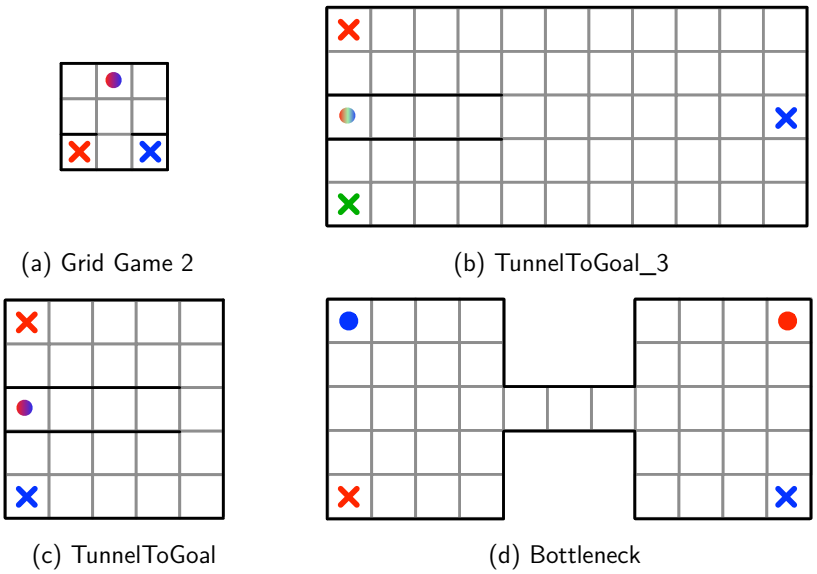
We compared both FCQ-variants to independent Q-learners (Indep) that learned without any information about the presence of other agents in the environment, joint-state learners (JS), which received the joint location of the agents as state information but chose their actions independently and with LoC (described in Section 3.2.3). For LoC we could not implement a form of virtual sensory input to detect when coordination was necessary for the active perception step as we did in the previous chapter. The reason for this is that a sensor cannot determine the need for interaction in the future. To circumvent this issue, we used a list of joint states in which coordination with the other agent would be better than to play independent[1]. For environment (d) for instance (*Bottleneck*), this list contained all the joint states in and around the tunnel at the middle, such that agents could still back out of the tunnel and let the other pass first. Note that FCQ-Learning is learning this list of states in which the active perception function returns true and this information should not be given beforehand.

All experiments were run for 20,000 episodes (an episode was completed when all agents were in the goal state) using a learning rate of 0.1 with a time limit of 500,000 steps per episode. Exploration was regulated using a fixed ϵ-greedy policy with $\epsilon = 0.1$. If agents collided they remained in their respective original locations and receive a penalty for colliding. On all other occasions, transitions and rewards were deterministic. The results described in the remainder of this paragraph are the running averages over 50 episodes taken over 50 independent runs. The size of the queue with the stored samples was 10.

We will begin by giving an overview of the final solutions found by the different

[1] As such this implementation could be seen as incorporating domain knowledge in the algorithm. If this knowledge however is not available, an active perception function that always returns true, might be a good option.

algorithms. Besides collision free, these solutions should yield the highest reward per episode and the least number of steps to complete an episode. The results are shown in Table 6.1. All values are averaged over the last 100 episodes after agents converged to a policy.



(a) Grid Game 2

(b) TunnelToGoal_3

(c) TunnelToGoal

(d) Bottleneck

Figure 6.5: Gridworld environments with future coordination problems. In environments (a), (b) and (c) agents have to enter the goal in a specific order. In environment (d) they have to coordinate before entering the corridor in the middle.

Environment	Algorithm	#states	#actions	#collisions	#steps	reward
Grid_game_2	Indep	9	4	2.4 ± 0.0	22.7 ± 30.4	-24.3 ± 35.6
	JS	81	4	0.1 ± 0.0	$\mathbf{6.3 \pm 0.3}$	$\mathbf{18.2 \pm 0.6}$
	LOC	9.0 ± 0.0	5	1.8 ± 0.0	10.3 ± 2.7	-6.8 ± 8.0
	FCQ	19.4 ± 4.4	4	0.1 ± 0.0	8.1 ± 13.9	17.6 ± 3.7
	FCQ_NI	21.7 ± 3.1	4	0.1 ± 0.0	7.1 ± 6.9	17.9 ± 0.7
TunnelToGoal	Indep	25	4	0.7 ± 0.0	37.9 ± 171.0	6.4 ± 3.6
	JS	625	4	0.0 ± 0.0	14.9 ± 8.5	$\mathbf{16.5 \pm 19.7}$
	LOC	29.7 ± 2.4	5	0.5 ± 0.0	20.0 ± 33.0	5.7 ± 15.7
	FCQ	71.3 ± 23.4	4	0.2 ± 0.0	$\mathbf{14.8 \pm 10.7}$	13.6 ± 12.8
	FCQ_NI	71.3 ± 28.0	4	0.2 ± 0.0	16.4 ± 31.3	11.6 ± 41.7
TunnelToGoal_3	Indep	55	4	0.7 ± 0.1	23.5 ± 67.8	0.2 ± 48.8
(3 agents)	JS	$166,375$	4	0.0 ± 0.0	24.9 ± 36.6	9.1 ± 28.6
	LOC	67.08 ± 10.4	5	0.6 ± 0.0	24.5 ± 38.2	-1.9 ± 827.3
	FCQ	148.0 ± 79.9	4	0.3 ± 0.0	$\mathbf{14.4 \pm 2.5}$	14.0 ± 3.1
	FCQ_NI	146.34 ± 76.3	4	0.3 ± 0.0	$\mathbf{14.4 \pm 3.7}$	$\mathbf{14.1 \pm 3.8}$
Bottleneck	Indep	43	4	n.a.	n.a.	n.a.
	JS	1849	4	0.0 ± 0.0	23.3 ± 30.8	13.1 ± 36.1
	LOC	54.0 ± 0.8	5	1.7 ± 0.6	$167.2 \pm 19,345.1$	$-157.5 \pm 10,327.0$
	FCQ	124.5 ± 32.8	4	0.1 ± 0.0	$\mathbf{17.3 \pm 1.3}$	$\mathbf{16.6 \pm 0.4}$
	FCQ_NI	135.0 ± 88.7	4	0.2 ± 0.0	19.2 ± 5.6	15.4 ± 2.3

Table 6.1: Size of the state space, number of collisions and number of steps for different approaches in the different games. (Indep = Independent Q-Learners, JS = Joint-state learners, FCQ = FCQ-Learners, with correctly initialised Q-values, FCQ_NI = FCQ-Learners without correctly initialised Q-values.)

In the smallest environments the agents always using the joint state space perform best. This is due to the fact that since agents actively have to coordinate and enter the goal in a particular order, always observing the other agents provides all the sufficient information. In small environments this is still manageable. In environments with larger state spaces, both FCQ variants reach policies that require a smaller number of steps to complete an episode than the other approaches. In the largest environment, *TunnelToGoal* with 3 agents, FCQ-learning outperforms all others in both number of steps to complete an episode and the average reward collected per episode. Independent learners simply don't have the required information to complete this task, whereas joint-state learners have too much information which causes the learning process to be very slow. Moreover, a lack of sufficient exploration still results in suboptimal policies after 20,000 learning episodes.

LoC is unable to reach acceptable results compared to the other approaches. Its active perception function is giving the correct states in which coordination should occur, but since this is not reflected in the immediate reward signal, the penalty for using this action is too big. An adaptation to use the sum of the rewards until termination of an episode could be beneficial, but as shown in [Melo & Veloso (2009)], there is an important relation between the immediate rewards and the penalty for miscoordination. Finding the right balance for the reward signal when this dependency between agents is reflected in the future rewards might prove to be very hard or even impossible, since this is not necessary uniform over the state space. FCQ-learning does not require such fine tuning of the reward signal for the specific problem task at hand and is as such more suitable for these future coordination issues.

In Figure 6.6 are some sample solutions found by FCQ-learning for the different environments. Agent 1 is indicated in red, Agent 2 in blue and Agent 3, if present, in green. Arrows with full tails represent actions taken using only local state information. The arrows with dotted tails represent actions taken based on augmented state information. For environments (a), (b) and (c), Agent 1 (red) has to reach the goal before Agent 2 (blue) and Agent 2 in its turn had to enter the goal state before Agent 3 (green) if there are three agents present. In all environments FCQ-learning correctly coordinated. In Environment (b), we see that Agent 2 performed a small loop to let Agent 1 pass first. Similar, Agent 3 also 'delayed' for quite some time before going towards the entrance of the tunnel to reach the goal. Note that these policies are still using an ϵ-greedy strategy with $\epsilon = 0.1$, so the agents sometimes performed an exploratory action. This why Agent 1 (in red) did not follow the shortest path in environment (b). In environment (d) we can clearly see that Agent 2 backed out of the corridor again, so Agent 1 could pass first. In environments (a) and (c) Agent 2 performed the equivalent of a 'wait' action, by taking an action that would lead it towards a boundary in the grid. This is a better choice under the

current settings of the discount factor and the reward for reaching the goal than to move 1 cell away from the goal since this would make the final path to the goal 2 steps longer.

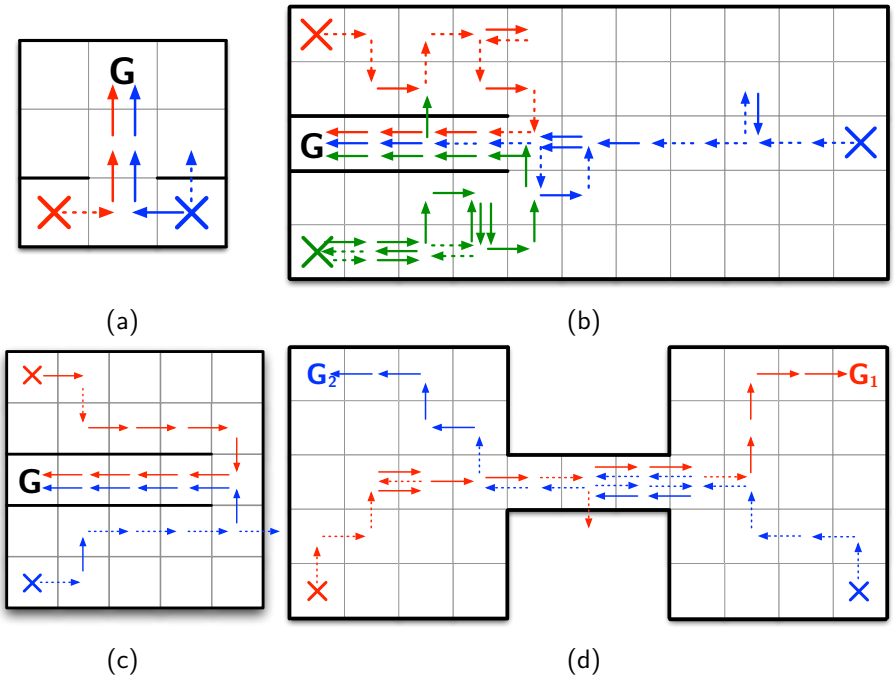
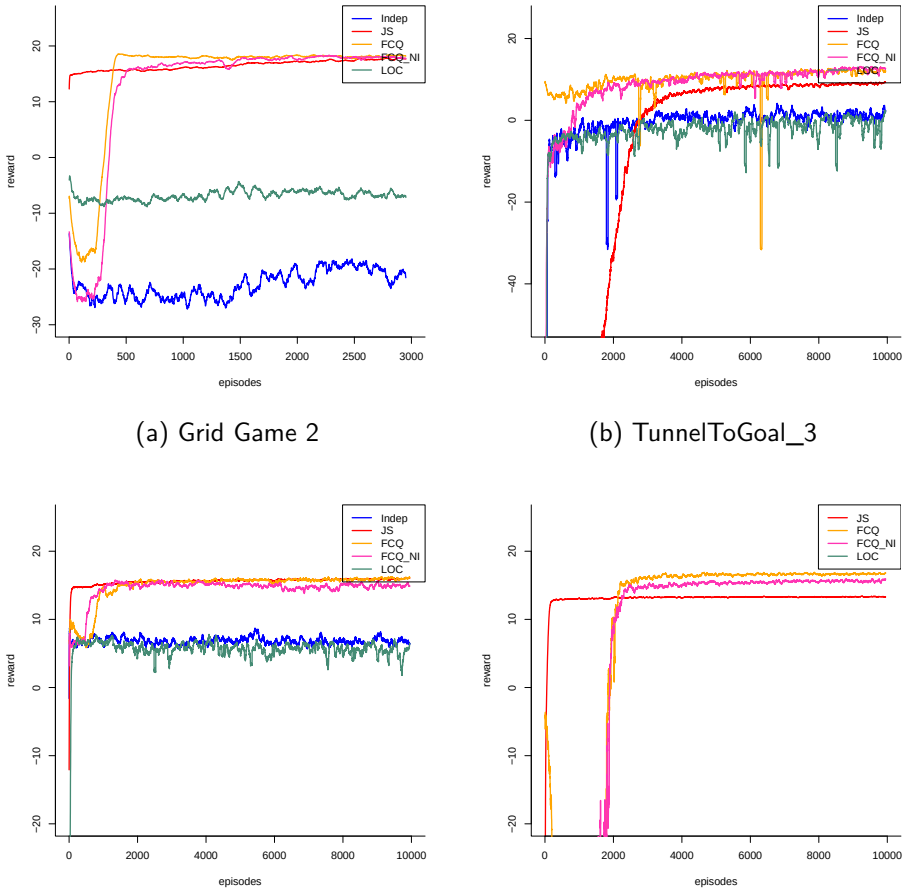


Figure 6.6: Sample solutions found by FCQ-learning for the different environments. Agent 1 is indicated in red, Agent 2 in blue and Agent 3 in green.

So far we have shown through these experiments that FCQ-learning manages to find good policies which are both collision free and in which agents have successfully solved the future interactions between them. Next, we are also concerned with the learning speed, as this is the issue most multi-agent approaches suffer from when using the complete joint-state joint-action space.

In Figure 6.7 we show the evolution of the rewards the agents collect per episode. Both independent learners and LoC have trouble correctly coordinating. They quickly settle for a suboptimal policy. JS improves its reward over time, but in the TunnelToGoal environment with three agents (Figure 6.7(b)), this approach needs over 2,000 learning episodes more than the FCQ-variants, to obtain a reward level that is still slightly less than FCQ. With FCQ we clearly see the sampling phase, during which a decrease in the reward is observed. This quickly increases again, as soon as interaction states have been augmented. In the Bottleneck environment (Figure 6.7(d)), FCQ-learning needs more time than joint state learners to reach

a stable policy, but this policy results in a higher average payoff than the policy of JS.



(a) Grid Game 2

(b) TunnelToGoal_3

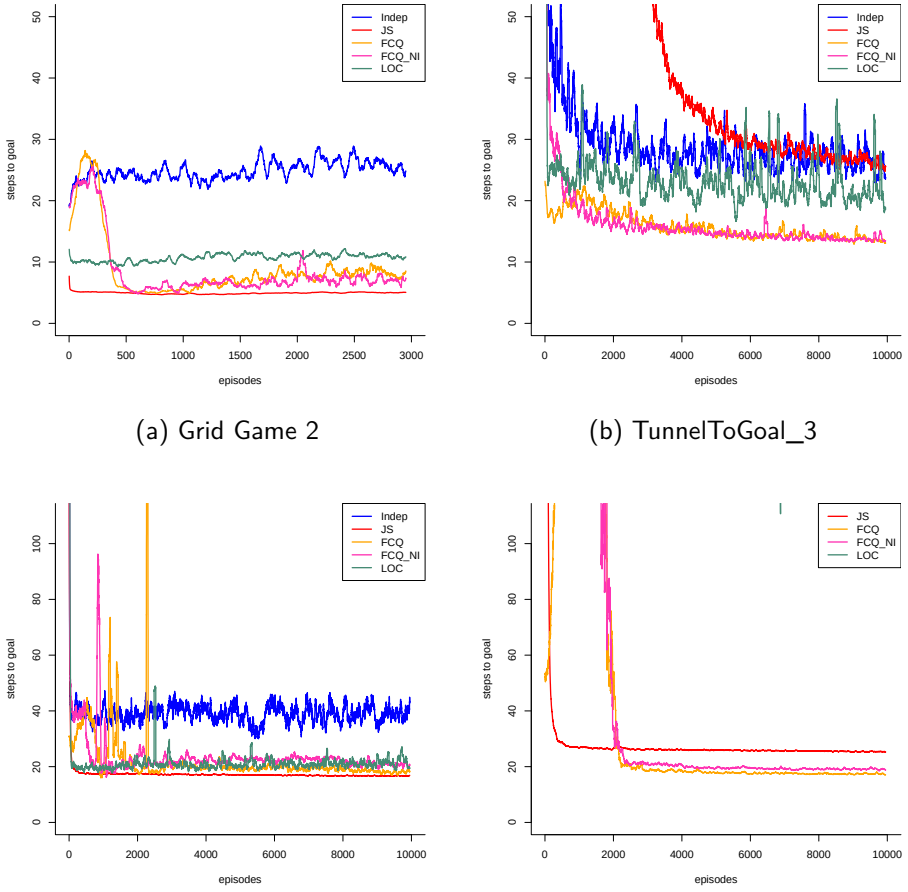
(c) TunnelToGoal

(d) Bottleneck

Figure 6.7: Reward collected per episode by the different algorithms for the (a) Grid game 2, (b) TunnelToGoal_3, (c) TunnelToGoal and (d) Bottleneck environments.

Figure 6.8 shows the number of steps needed to complete an episode during the learning process. In all environments we observe the same effect as in the figures for the collected reward per episode. Initially the number of steps of FCQ-learning is increasing. This is during the time frame in which it is collecting samples and identifying in which states it should observe the state information of other agents contained in the system state to select an action. As soon as the correct states are augmented, a sudden decrease in the number of steps to complete an episode can be seen. Again, JS needs a lot of time to reduce the number of steps required to complete an episode in the *TunnelToGoal_3* environment due to the size of the state space in which it is learning. FCQ-learning does not suffer from this issue since the size of the state space is not linked so closely to the number of agents in the system. In the *Bottleneck* environment (Figure 6.8(d)) the results for LoC are not visible, because after 10,000 learning episodes, this algorithm still did not manage to find a policy which required less than 100 timesteps to complete the task to reach the goal. Contrary to the independent learners however, it did manage to find a policy. Independent learners encountered high penalties in the corridor and as such this path was only rarely taken.

This is the same effect we saw in the *Grid Game 2* environment which was analysed in Section 3.2.1. In the Bottleneck environment the problem is even bigger, since agents have to take four consecutive actions to pass through the corridor. If the Q-values of these actions in these states are not the highest ones, the probability on this happening through consecutive exploratory actions is 0.0001.



(a) Grid Game 2

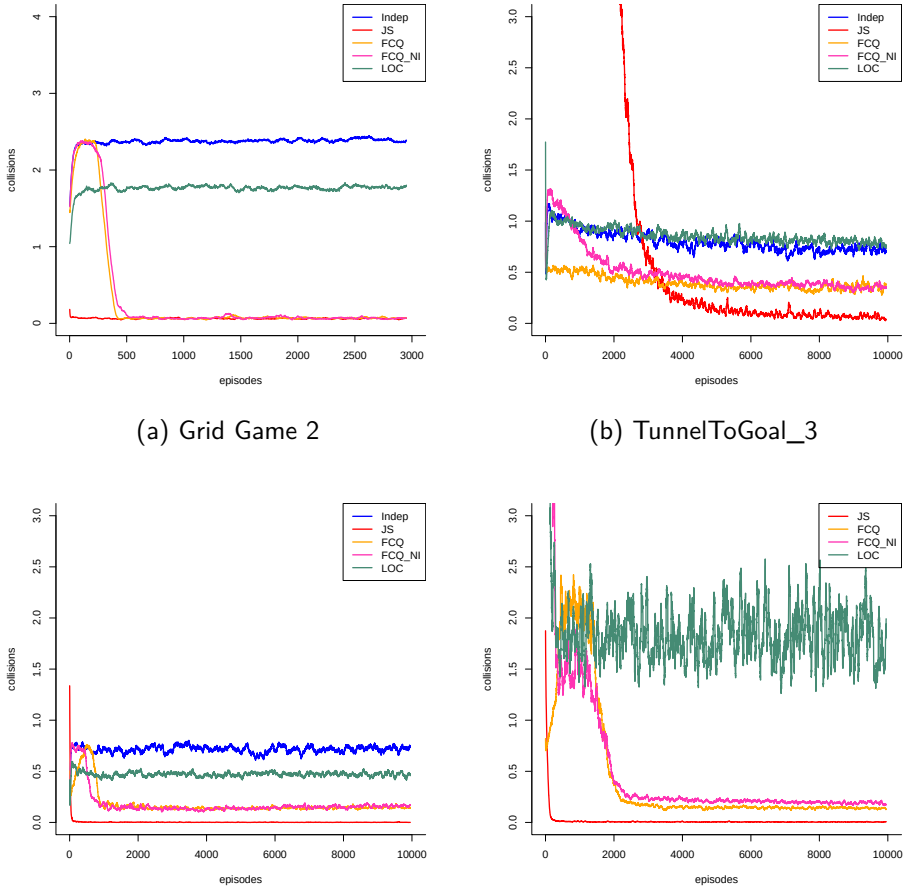
(b) TunnelToGoal_3

(c) TunnelToGoal

(d) Bottleneck

Figure 6.8: Number of steps needed to complete an episode by the different algorithms for the (a) Grid game 2, (b) TunnelToGoal_3, (c) TunnelToGoal and (d) Bottleneck environments.

Finally, we show the average number of collisions per episode in Figure 6.9. Again we see the effect of the sampling phase of both FCQ-learning variants. The number of collisions between the agents using this algorithm increases until the states in which coordination is required are augmented, after which this number drops to 0. Again, JS-learners need more episodes in the *TunnelToGoal_3* environment compared to both FCQ-learning algorithms and both independent learners as LoC are unable to learn collision free policies.



(a) Grid Game 2

(b) TunnelToGoal_3

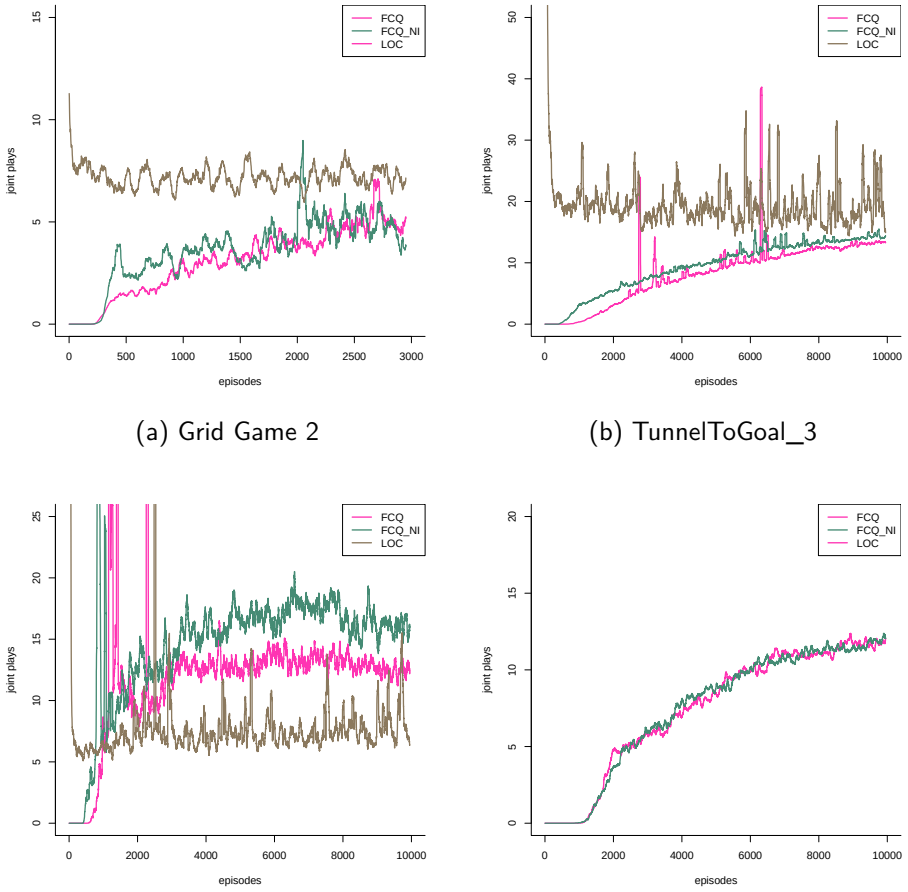
(c) TunnelToGoal

(d) Bottleneck

Figure 6.9: Number of collisions per episode for the different algorithms for the (a) Grid game 2, (b) TunnelToGoal_3, (c) TunnelToGoal and (d) Bottleneck environments.

Three approaches learn to augment their state information in the set of interaction states of the underlying DEC-SIMDP: LoC and both FCQ variants. In Figure 6.10 we show the number of times these algorithms selected an action using such augmented information per episode. For LoC this means the number of times the agents selected their `COORDINATE` action per episode. Initially the FCQ variants never do this, until enough samples of future rewards are collected. For the *Bottleneck* environment (Figure 6.10(d)) the line representing the results for LoC lies outside of the plotted area. This approach selects its `COORDINATE` action a lot and is still constantly selecting this action after $10,000$ learning episodes (approximately 400 times per episode). Both FCQ variants select an action based on augmented

state information about equally as much. FCQ with independently learned Q-values selects it a little less since it already knows the optimal single agent policy and thus agents do not collide as often as when they are still learning a policy to reach the goal state.



(a) Grid Game 2

(b) TunnelToGoal_3

(c) TunnelToGoal

(d) Bottleneck

Figure 6.10: Number of times the different algorithms used information from the system state to select an action per episode for the (a) Grid game 2, (b) TunnelToGoal_3, (c) TunnelToGoal and (d) Bottleneck environments.

6.4 Summary

In this chapter we presented an algorithm that learns in which states of the state space an agent needs to include knowledge or state information about other agents in order to avoid collisions that occur in the future. Situations in which such problems occur are for instance when multiple autonomous robots are required to go through a small corridor where they can only pass one at a time. By means of statistical tests on the obtained rewards and the local state information of other agents, FCQ-learning is capable of learning in which states it has to augment its state information in order to select actions using this augmented state information. We have shown two variants on this algorithm which perform similar in terms of the quality of the found solution, but have a different computational complexity in terms of processing power and memory usage, due to the number of samples collected and on which statistical tests have to be performed. The first one requires single agent Q-values of the problem task it is learning to detect good candidate states in which it might be able to solve interaction problems. In these candidate states the state information of other agents is observed in order to determine which of this state information is relevant to solve the future coordination problem. The second variant uses all states as candidate states. As such it does not require pre-learned Q-values, but it uses the system state more often and performs more statistical tests than the first variant.

FCQ-learning is the first algorithm capable of solving delayed coordination problems using only sparse interactions. Sparse interactions have already been shown to have many advantages in literature. When solving problems in which delayed coordination problems occur, sparse interactions also prove to be beneficial. The biggest improvement could be seen in our experiments using three agents. The learning process of agents who always use the joint state space was very slow compared to our approach based on sparse interactions. Not only in terms of learning speeds, but also in terms of quality of the final solution, FCQ-learning outperforms the other approaches in the larger environments. We can conclude that FCQ-learning goes beyond the state-of-the-art in multi-agent reinforcement learning and closes the gap between the sparse interaction framework and problem tasks characterised by delayed reward signals.

Chapter 7

Transfer learning and generalisation in multi-agent systems

If you have knowledge, let others light their candles in it.
– Margaret Fuller, 1810-1850 –

Reinforcement learning is a mature technique for solving complex problems. Significant advances have been developed to speed up the learning process through the incorporation of domain knowledge [Goetschalckx (2009)], the decomposition of problems in subtasks [Dietterich (2000), Hengst (2005)], the use of a generalised state space representation [Boyan & Moore (1995), Sutton (1996)] or learning higher level actions, composed of multiple one step actions, i.e. options [Sutton et al. (1999)]. All these approaches aim at improving the learning process but for every new task the RL algorithm has to start the learning process from the beginning. Recently the idea of transfer learning has been applied to reinforcement learning tasks. Transfer learning leverages the experience an agent acquires in a source task in order to improve its performance in a related target task. This chapter is concerned with applying transfer learning to the problem of coordination in multi-agent systems. The core idea is that if agents learn to coordinate in one environment, this knowledge can be generalised and used to speed up the coordination process in other related (multi-agent) tasks.

7.1 Transfer learning

In this section we provide a description of transfer learning for single agent reinforcement learning tasks. This summary is inspired by [Taylor & Stone (2009)]. The

concept of transferring knowledge is not a new one. As early as 1901, Thorndike and Woordworth experimented with how certain experiences learned using one function, affected the learning process of a different function [Thorndike & Woodworth (1901)]. The word function referred to various things such as spelling, multiplication, chess playing, reasoning, etc. From psychology, this area of research was applied to machine learning tasks, planning tasks and many others. Recently, this topic gained attention in the RL community by attempting to transfer knowledge between RL tasks. A fully autonomous RL agent capable of transferring knowledge needs to undertake three different steps according to [Taylor & Stone (2009)]:

1. Select an appropriate previously learned source task or tasks for the current target task at hand.
2. Learn the relation between the source and the target task.
3. Transfer the knowledge from the source to the target task.

At this time, no methods exist that are capable of solving all three steps. Most techniques use a single human-picked source task to transfer knowledge to the target task. Alternatives exist in which all tasks from a certain set are used to transfer, such as [Singh (1992)]. This work successfully transfers experience from elemental sequential tasks to composite sequential decision tasks. These composite task are the concatenation of multiple elemental tasks. More advanced approaches use a library of source tasks and use only the most relevant ones for transfer [Lazaric (2008)]. The work by Lazaric selects source tasks according to distance and alignment of $\langle s, a, r, s' \rangle$-tuples between source and target task. Another alternative is to modify a general source task to make it useful for the specific target task at hand [Sherstov & Stone (2005)]. The idea in this paper is to group states together in classes based on the probability of a given outcome for a certain action and then to transfer the set of optimal actions to the target task. As such, the target task can be learned using a reduced set of actions.

The relation between the source task and the target task defines what knowledge can be transferred. If both tasks share the same state variables and actions, it is possible to transfer the action-value function without any additional operations. If however there are differences in the way states are represented or how actions are labelled, a mapping between the source task and the target task is required. An example of a mapping that could be performed is changing the labels of the actions from the source task to the labels of the actions in the target task.

If transferring the action-value function is not possible, other information that could be shared between tasks go from a set of $\langle s(t), a(t), r(t+1), s(t+1) \rangle$-tuples at the simplest level to a complete model of the source task. A thorough overview of different mappings for the states and actions between source and target tasks is given by [Taylor (2008)].

Only two attempts at using transferred experience in multi-agent RL are cited in the overview paper of transfer learning by Taylor and Stone [Taylor & Stone (2009)]. These approaches, [Kuhlmann & Stone (2007)] and [Banerjee & Stone (2007)] both deal with extensive form games. No work has been reported on transfer learning in stochastic games (aka Markov games). These works both focused on the General Game Playing problem from [Pell (1993)]. The challenge in this problem is to create an agent that is capable of playing unseen games without human intervention. To do so, knowledge from past games needs to be transferred in order to improve its performance in the current game. Agents receive a description of the current game, so they can identify similarities to previous games. Although multiple agents are playing the same game, these approaches are not concerned with transferring coordination experience. An example of a task from [Kuhlmann & Stone (2007)] is a miniature checkers game played on a 5×5 board. In the source task the agent has 5 pieces at its disposal, whereas in the target task, only 4 pieces are present.
In the next sections we introduce a novel concept which applies transfer learning to the problem of multi-agent coordination in general Markov games.

7.2 Transfer learning using generalized learning automata

7.2.1 Overview of the approach

As explained in the previous section, transfer learning is a way of reusing knowledge learned in one task to speed up the learning process in a different task. This section continues on the work we explained in Chapter 4. In that chapter we introduced the 2Observe algorithm, capable of approximating the interaction function of the underlying DEC-LIMDP. This framework models a certain type of MAS with sparse interactions, where the interaction area can be calculated from the local state information of one agent. This is the case in mobile robots, where the interaction area is given by the surrounding locations of the agent. 2Observe uses a generalized learning automaton to approximate this interaction function. In the experimental section we used gridworlds as a testbed and provided the agents with the Manhattan distance between them as input for the GLA. We showed that agents were capable of learning a threshold for the safe distance at which agents could ignore each other. This GLA can be reused in different environments in which the underlying DEC-LIMDP uses the same interaction function and agents have the same basic capabilities as in the original environment. Such transfer is the simplest setting described in [Taylor & Stone (2009)] and apart from the fact that information is inherently multi-agent information, this can be seen as transferring knowledge between tasks. On the other hand, this GLA can also be reused for other agents using the same 2Observe algorithm. This approach is illustrated in Figure 7.1. The GLA

from one agent is transferred to another agent. As such, the target agent can reuse the experience from the source agent, informing it when coordination is needed.

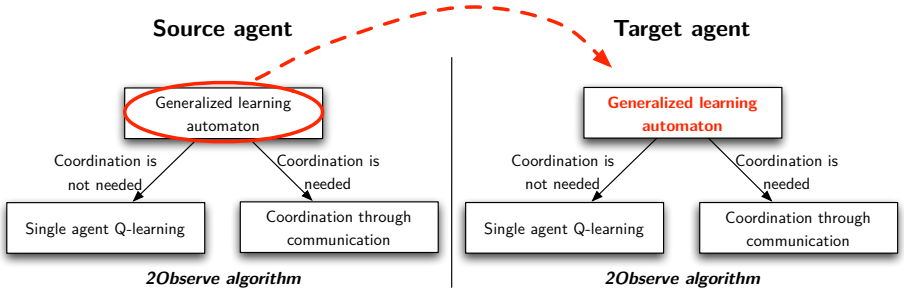


Figure 7.1: Transfer of GLA between agents using the 2Observe algorithm

Given that the input for the target agent has the same representation as the input given to the source agent, no task mappings are required to transfer this GLA. A GLA should not be seen as a black box, but rather as a computational entity. Hence, transferring this knowledge from the source agent to the target agent is done by transferring the internal state vector $\vec{u}$ from one agent to the other.

7.2.2 Experimental results

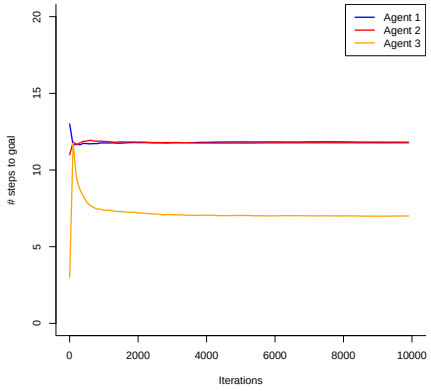
In this section we illustrate how transferring the knowledge about coordination from one agent to another improves the overall learning process of the task the agent is required to solve. We use the same environments as the ones used in Chapter 4, shown in Figure 4.12. We first let two agents learn in the environment using the 2Observe algorithm. After convergence, a third untrained agent is added to the environment. This third agent also uses the 2Observe algorithm, but receives at its top level the trained GLA of one of the other agents. At the second level the agent uses a single agent Q-learning algorithm, with the Q-values initialised to zero. The learning settings were identical to those of the other agents, i.e. an ϵ-greedy strategy with $\epsilon = 0.1$ and a learning rate of 0.05. Its goal was the same as the goal of one of the other agents, selected at random at the beginning of the learning process. Its initial position was selected at random at every episode. As such, unlike the other agents with a fixed initial location and fixed goal, this agent would not converge to a fixed path to reach the goal. Hence, it could appear near other agents at different locations in the grid, rather than only in those locations where the paths of the agents intersect. Since GLA use an agent-centric view, i.e. the distance between the agents, no mappings have to be performed to transfer the GLA to the third agent and this random initial location does not influence the decisions of the GLA.

In Figure 7.2 we show the results on each of the test environments. The top row shows the results for the *TunnelToGoal* environment, the middle row for the *2-robot* environment and the bottom row shows the results for *ISR*. On the left are the results for the number of steps required to complete an episode. On the right we show the number of times agents coordinated with each other and how often they collided. All results are averaged over 10 independent runs. Agent 3 is the target agent which used the GLA of one of the other agents. It should be noted that the agents only transfer experience related to coordination and not with regard to the navigation task.

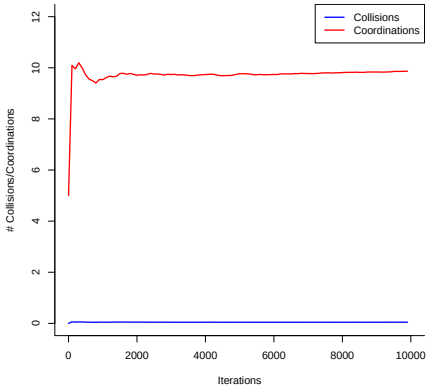
We barely see any influence with regard to the number of steps to complete an episode for agents 1 and 2. Since these agents have learned when to coordinate with another agent and have the correct Q-values, the presence of a third agent has no negative impact on their performance. The third agent converges to a shorter path in the *TunnelToGoal* and the *2-robot* environment and a longer path in the *ISR* environment compared to the other two agents. This is because this agent starts from a random position. In the first two environments this position is bound to be closer to the goal than the initial position of the other agents. In *ISR*, we see the opposite effect since the agent will most likely begin further away from its goal than the other agents are from their respective goals. Hence the longer number of steps to complete an episode we see in Figure 7.2(e). These figures clearly show that agents 1 and 2 do not suffer from adding a third untrained agent, indicating that the GLA still correctly returns whether coordination is needed.

On the right hand side we see that the number of collisions remains constantly at zero. This indicates that the transfer was successful and that not only the first two agents did not have to re-learn when coordination is required in this environment, but also that Agent 3 was coordinating with the other agents as if it had been trained together from the beginning. It can completely rely on its GLA and learn its navigation task without collisions. The number of coordinations per episode also quickly stabilises. Compared to the results of Chapter 4 in Figure 4.14, the agents coordinated more often. This is a consequence of the presence of the third agent. The number of agents has increased, but the size of the environment remained the same, so the density of agents increased in the environment.

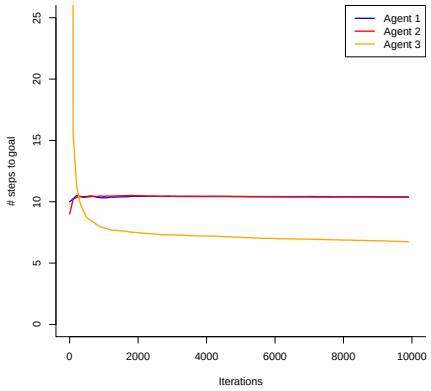
In this section we described a novel approach for transferring experience with regard to coordination between agents by means of GLA. This transfer could easily be accomplished because the GLA were trained using the 2Observe algorithm which uses agent centric state information about the need for coordination. In the next section we introduce an approach based on CQ-learning which uses absolute state information to represent the sparse interactions between agents. We will illustrate how, from these sparse interactions, a generalised representation can be used to transfer coordination experience.



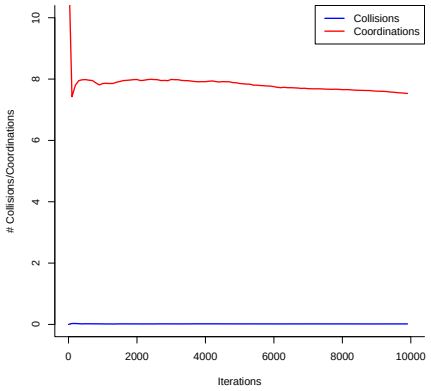
(a) **TunnelToGoal** - Steps to goal



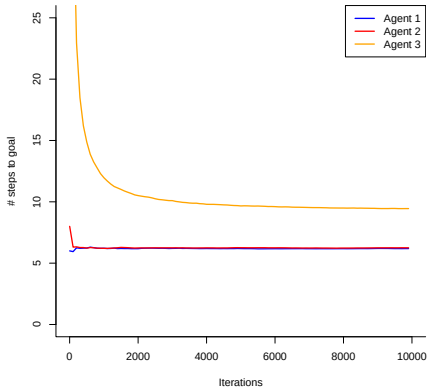
(b) **TunnelToGoal** - Coll/Coord



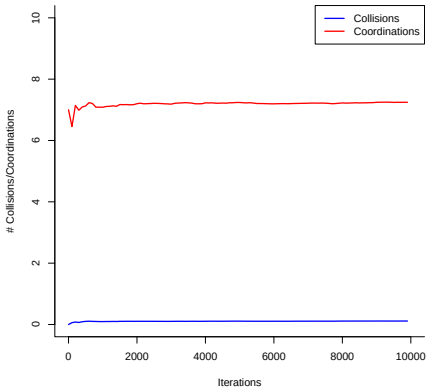
(c) **2-robot** - Steps to goal



(d) **2-robot** - Coll/Coord



(e) **ISR** - Steps to goal



(f) **ISR** - Coll/Coord

Figure 7.2: Results for transfer of coordination experience to additional agents

7.3 Transfer learning using CQ-learning

7.3.1 Overview of the approach

In this section we illustrate an approach to transfer coordination experience in multi-agent systems based on Coordinating Q-learning (CQ-learning), which we explained in Chapter 5. CQ-learning is a multi-agent extension to the basic Q-learning algorithm and aims at adapting the state space in which each agent is learning when using only local state information is not sufficient. As such it avoids the exponential increase of the state space that is often seen in multi-agent learning systems. It allows agents to learn using only their local state space whenever the agents are not influencing each other. The idea behind this algorithm is to start with a minimal state space, using only local states. CQ-learning will maintain statistics on the rewards an agent receives, together with the state information of other agents. If significant changes are detected, the local state information of the agents is augmented to include the state information of other agents. Using this richer state representation the agent is able to coordinate its actions based on the state information of other agents. The question behind the research we present in this section is whether it is possible to draw some conclusions from the states in which such coordination is required. In the navigation tasks we have considered so far it is easy to see that agents affect each other if they are too close to each other. In the gridworlds used in our previous experiments this was the case if agents were less than three locations apart from each other.

Our first goal is the same as in Chapter 4: learn this threshold. In this section, we will learn this based on the experience from a related, more simple task and transfer this information to a more complex task. Our source task is a simple 5×5 training grid shown in Figure 7.3. In this training grid, no goal was defined and agents acted randomly, learning the state information of interaction states. In Chapter 5 we demonstrate how CQ-learning identifies the absolute location of the other agents that is causing collisions in a gridworld environment. To allow for a better generalisation, this information is converted to an agent centric representation. By this we mean that samples are described using the distance between the agents horizontally (Δx) and vertically (Δy). Let us consider the situation where Agent 1 has learned to augment its state information in the location with coordinates (2,2), to observe if there is another agent present in the location with coordinates (1,3). This information was converted to (-1,1). Using such an agent-centric representations allows to map multiple danger states to the same Q-values. For example, the dangerous situations with the agents in grid locations (2,3) and (1,4) or with the agent in grid locations (3,4) and (2,5), are both mapped to Q-values for situation (-1,1). As such, agents can gather experience based on their mutual configuration, rather than on absolute locations. These agent centric Q-values for interaction states are

maintained in the Q^{aug}-table of CQ-learning.

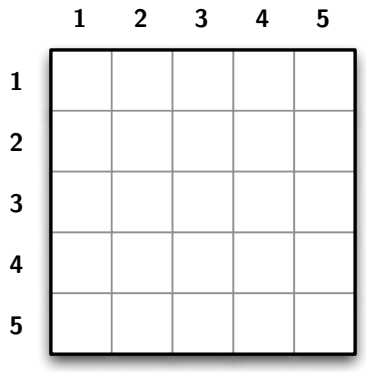


Figure 7.3: 5×5 training grid used as a source task.

With this relative information, a propositional rule learner is trained to learn simple rules about when interactions will take place. The rule system is based on Ripper [Cohen (1995)], which has also been used for rule transfer in single agent reinforcement learning [Taylor & Stone (2006)]. The classifier is trained with both safe and dangerous situations. Finally, after this classifier is trained, two pieces of information are transferred from the source task to the target task: the trained classifier and the Q-values (using agent-centric state information) of the danger states. In the target task, the agent can then rely on the classifier to decide whether coordination is required. If a danger state is encountered for the first time, this state is added as a separate state using the absolute state information of the agents and the Q-values are initialised with the transferred Q-values. These transferred Q-values already have a low value for the action that would result in a collision. If the current state is not dangerous, a single agent Q-learner is learning the navigation task. This approach is shown in Figure 7.4.

7.3.2 Experimental results

We now evaluate the transfer approach empirically, using a number of different settings. In each case the transfer agents were trained individually on the source task described above for $50,000$ time steps, before being transferred to their target problem.

Table 7.1 shows an example rule set, learned by this system on the source task. These rules show in an easy human-readable way, that agents should coordinate if they can move to the same location with a single action.

The target environments we used in these experiments are *TunnelToGoal*, *ISR* and *CIT*, which were also used in previous chapters. In this chapter, each agent

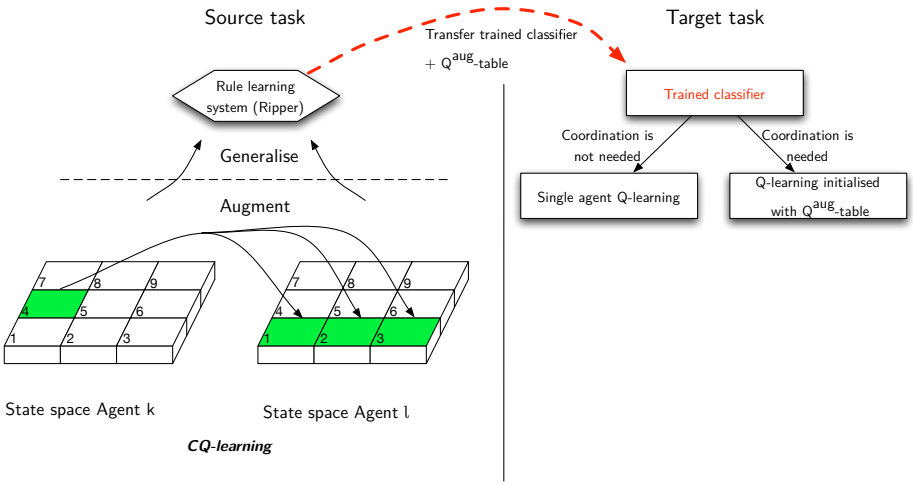


Figure 7.4: CQ-learning is identifying interaction states, which are used as input for a rule learning system. This trained rule classifier is then transferred to the target task, together with the Q-values for interaction states.

IF $\Delta x \leq 1$ **AND** $\Delta y \leq 1$ **AND** $\Delta x \geq -1$ **AND** $\Delta y \geq -1$	$\Rightarrow$ **DANGEROUS**
IF $\Delta x \leq 0$ **AND** $\Delta x \geq 0$ **AND** $\Delta y \leq 2$ **AND** $\Delta y \geq -2$	$\Rightarrow$ **DANGEROUS**
IF $\Delta y \leq 0$ **AND** $\Delta y \geq 0$ **AND** $\Delta x \geq -2$ **AND** $\Delta x \leq 2$	$\Rightarrow$ **DANGEROUS**
ELSE	$\Rightarrow$ **SAFE**

Table 7.1: Example classifier learned by Ripper after training on the source task.

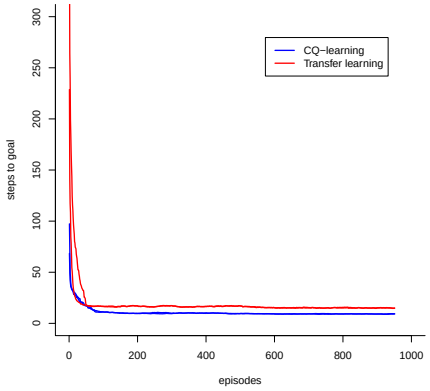
starts from a random location and must reach its respective goal location. When an agent reaches its goal, it stays there until all agents have finished and the episode ends. The transfer agents are compared with agents using standard CQ-learning. In Chapter 5 we already showed that CQ-learning outperforms both independent Q-learners and joint learners on these learning tasks. The transfer learners and CQ-learning are each allowed 2000 start-to-goal episodes on the target tasks. Both algorithms use identical Q-learning settings using a learning rate of 0.1, a discount factor of 0.9 and an ϵ-greedy action selection with a fixed $\epsilon = 0.2$.

Figure 7.5 shows the results on each of the target tasks. We evaluate the algorithms' performance based on the number of steps agents require to complete an episode and the total number of collisions between the agents per episode. All results are averaged over 25 independent experiments. When looking at the number of collisions during learning on the right hand side of the figure, the transfer agents

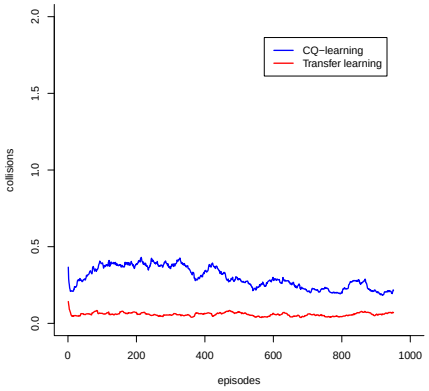
clearly perform better. They immediately start out with a lower number of collisions, and keep outperforming CQ-learners even in the long run.

When evaluating the learners with regard to the number of steps criterion shown on the left hand side of the figure, it should first be noted that the transfer agents do not transfer any knowledge with regards to the navigation task at hand, but only use transfer to avoid collisions. However, the transfer algorithm does show a better initial performance in terms of the number of steps to reach the goal needed in the *ISR* and *CIT* environments. In these larger environments the CQ-learners eventually match the transfer agents and asymptotically show a slightly better performance. This is caused by the transfer agents preferring a safe action (i.e. move away from the other agent) in states which are marked dangerous by the classifier, but which do no lead to collisions under the greedy policy. CQ-learning tends not to mark these states as dangerous and does not have this problem. In the smaller *TunnelToGoal* environment the CQ-learning agents show a better performance over the entire run. This is caused by the fact that in such a small environment, the classifier will almost always decide that coordination is required and select a safe action.

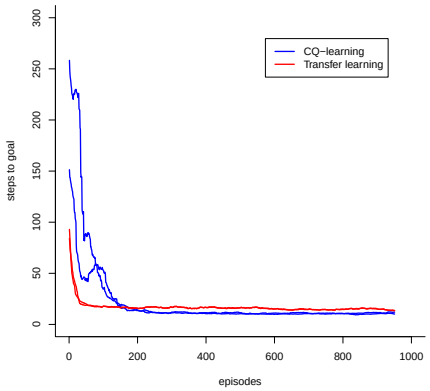
From these experiments we come to the same conclusion as in the previous section, where we transferred GLA between agents. Much of the complexities of learning in multi-agent systems, such as coordination with other agents and the size of the state space, can be avoided by using a layered approach. Learning when coordination should occur, is not necessarily linked to the core task of the system (in these cases navigating in a gridworld). So learning about coordination can be done in simpler settings under the assumption that the coordination requirements in both source task and destination task are the same or that they can be mapped on each other. This allows the agent to focus purely on the core task at hand and as such learn faster and only consider other agents, if there is a need for coordination. As shown in this section, these benefits become more apparent in larger target tasks.



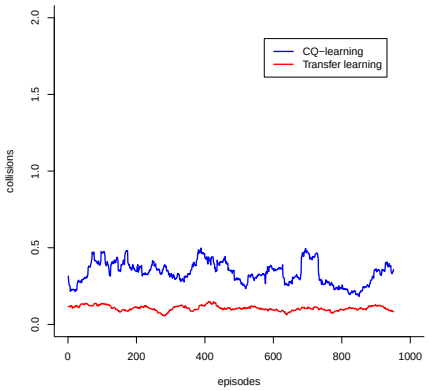
(a) **TunnelToGoal** - Steps to goal



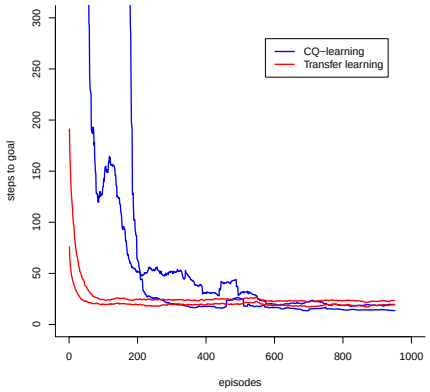
(b) **TunnelToGoal** - Collisions



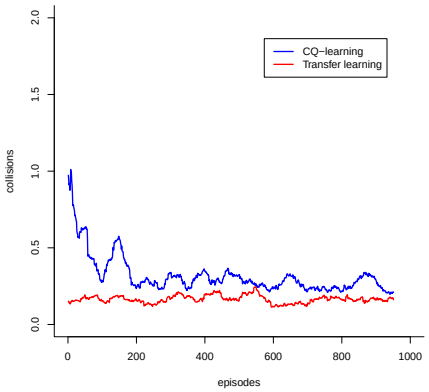
(c) **ISR** - Steps to goal



(d) **ISR** - Collisions



(e) **CIT** - Steps to goal



(f) **CIT** - Collisions

Figure 7.5: Results for transfer of coordination rules using CQ-learning from a simple training grid to more complex navigation tasks

7.4 Generalisation using CQ-learning

7.4.1 Overview of the approach

In the previous section we used an agent centric representation of the interaction states to train a rule learning system. In this section we introduce an alternative approach which also generalises this information regarding interaction states. Figure 7.6 shows the idea of our approach. The bottom part of the figure represents CQ-learning and was already shown in Figure 5.2.

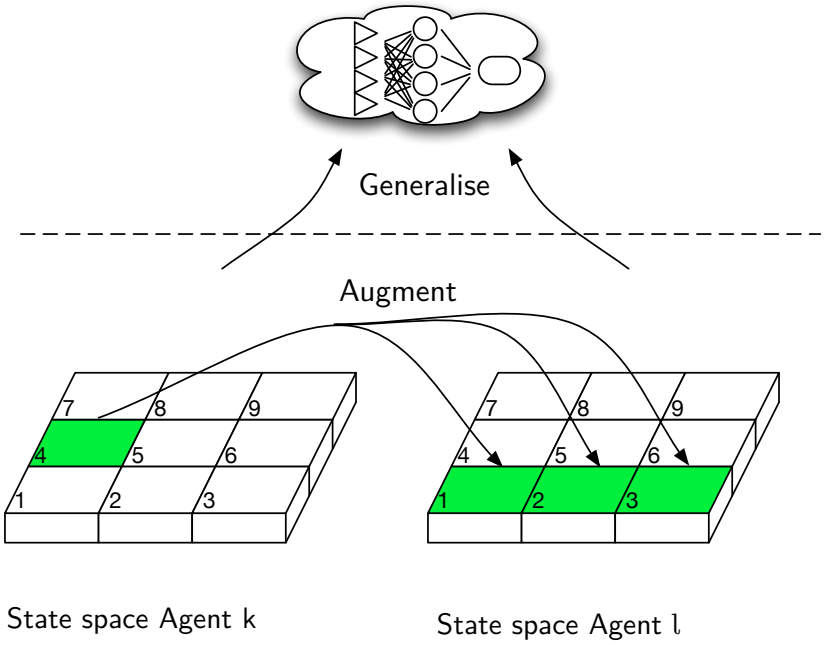


Figure 7.6: Extra layer on top of CQ-learning to learn a generalisation of when coordination is needed.

During the execution of CQ-learning, interaction states are identified and agents augment their local state information to include the state information of other agents. In each of these interaction states, the agent has learned the relevant state information when these interactions occur. The combination of an agents' local state, with the extra state information from this sparse interaction, can then be used to train a classifier in order to obtain a generalised concept of these sparse interactions. In the previous section this was done using the Ripper classifier. In this section we will use a feedforward neural network. Such networks are a robust approach of approximating target functions, by minimising the error over many attribute-value pairs.

As in the previous section, we refactor the information of augmented states to

an agent-centric representation. Samples are described using the horizontal (Δx) and vertical (Δy) distance between agents. Augmented states receive a value of 1 as target value, indicating that in these states coordination is required. All other combinations of agents' location that are not augmented because the agents are not influencing each other, receive a target value of 0. Finally, the samples with which the network are trained, also contain the actions of the agents. As such, it is possible to distinguish between situations in which agents are so close they could collide, but if their intended actions would lead them away from each other, this situation is still safe and receives an output value of 0. The neural network used for the generalisation of CQ-learning is shown in Figure 7.7.

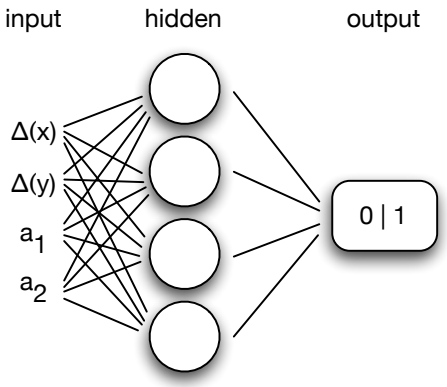
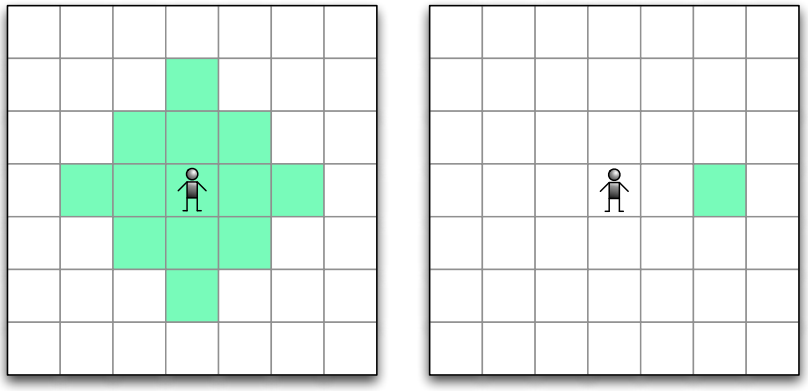


Figure 7.7: Neural network used for the generalisation after CQ-Learning. Different quantities of units in the hidden layer were used during the experiments.

Such a generalisation allows for very specific polling of the joint state space to verify if certain conditions are met. In our experiments we demonstrate that it allows the agent to select whether it should observe the presence of another agent in a certain location or not, independent of the actual location the agent is in. This is a similar result to what 2Observe is learning. In Figure 7.8 we show a side by side comparison of the generalisation both algorithms learn. On the left hand side we see the generalisation of 2Observe. This algorithm has learned to coordinate with agents that are in any of the surrounding locations that are less than 3 cells away from the current position of the agent. Note that this is the same generalisation as the rule learning system learned in the previous section. On the right hand side we show the generalisation we obtain using a neural network with the information of CQ-learning. The neural network gives a very specific location that should be observed when the agent wants to perform action `EAST`. In practice this neural network could be used as follows. Assume an agent wants to perform action `EAST`. It can query its neural network using various distances and possibilities for the location and action

of another agent. For the queries that result in a high output value, the agent can then activate a particular sensor, rotate its camera, or broadcast a request to ask for the location and intentions of an agent in the dangerous location. If this intention is not to perform the dangerous action, indicated by the neural network, no further coordination is required and the agent can select its actions independently, using its local state information only.



(a) Generalisation learned by 2Observe

(b) Generalisation learned with CQ-learning (for action EAST)

Figure 7.8: Comparison of generalisations learned by 2Observe and CQ-learning.

7.4.2 Experimental results

In our experiments we used some of the environments we used for CQ-learning in Chapter 5, shown in Figure 5.8. The experiments in this chapter are performed on *TunnelToGoal*, *2-robot game*, *ISR*, *CIT* and *CMU*. We allowed CQ-learning to learn in the environment, using the same settings as described in Section 5.3. After the learning process we used the collected samples to train the neural network. About an equal number of samples for dangerous situations as for safe situations are used. In our experiments we observed that this meant that many safe samples were discarded, since there are more safe situations in these environments than there are dangerous situations. Of all collected samples 80% were used to train the neural network. The remaining 20% were used for the validation of the network. In Table 7.2 we show the accuracy of the network on this validation set. Different quantities of hidden units with a logistic sigmoidal output activation function were used. Using at least 4 units in the hidden layer resulted in a correct identification of safe and dangerous situations in more than 96% of the cases for both agents.

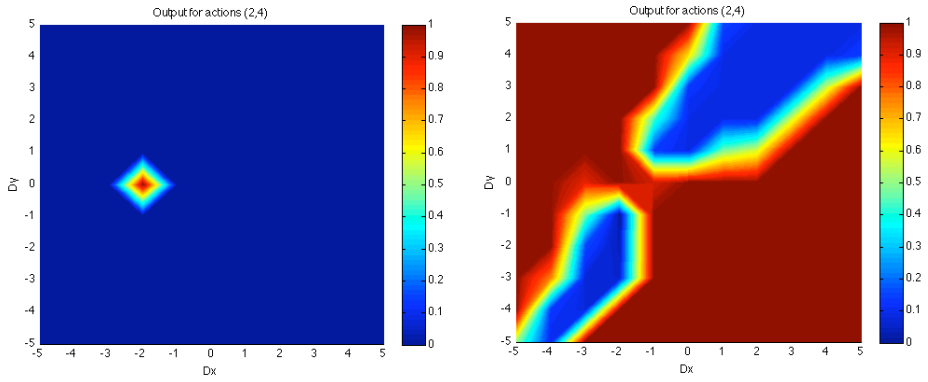
Environment	Agent	# hidden units in the neural network				
		1	2	4	6	8
TunnelToGoal	1	88.21%	93.70%	98.53%	98.54%	98.71%
	2	97.74%	98.52%	98.63%	98.64%	98.64%
2-robot game	1	98.35%	98.35%	98.35%	98.35%	98.35%
	2	99.62%	99.62%	99.62%	99.62%	99.62%
ISR	1	89.93%	92.54%	96.65%	97.31%	97.53%
	2	90.46%	95.24%	97.87%	98.43%	98.97%
CIT	1	81.97%	93.00%	97.00%	97.82%	98.04%
	2	86.88%	91.95%	97.62%	98.67%	98.84%
CMU	1	76.93%	96.53%	96.63%	97.11%	97.21%
	2	97.90%	98.67%	99.23%	99.29%	99.53%

Table 7.2: Accuracy of the neural network in the different games with different quantities of hidden units.

If avoiding miscoordination is critical in the particular situation at hand, one could choose to initialise the weights of the neural network, in such a way that initially all situations are considered to be dangerous. As more samples are used to train the neural network, the weights will be adapted to output a value of 0 (i.e. no coordination is required), for those situations where enough safe samples have been observed. Alternatively, the weights could also be initialised to see every sample as being safe, learning in a bottom-up way when coordination is needed. Sample results for these initialisations are shown in Figure 7.9. The contour plot shows the output of the neural network. The setting is as follows: two agents are two locations apart from each other and their selected actions are to go directly towards each other. Without coordination, performing this action would for sure result in a collision. A red value indicates a high need for coordination, whereas values closer to blue, represent samples in which agents do not need to observe each others' state information. On the left hand side, the weights were initialised to consider every sample as safe. A big red spot at $\langle \Delta x = -2, \Delta y = 0 \rangle$ indicates that coordination is required for the current situation. On the right hand side, the weights are initialised to consider every sample as dangerous. The network outputs high values for most situations but has already adapted its weights to output low values at the top right, where $\Delta x > 2$ and $\Delta y > 2$, and at the bottom left, where $-4 < \Delta x < -2$ and $\Delta y < -1$. The current situation $\langle \Delta x = -2, \Delta y = 0 \rangle$ also returns a high output value indicating a high need for coordination.

Depending on the particular problem at hand, one would prefer one approach over the other. Initialising with weights that return high output values is much safer to avoid collisions in unseen situations, but will result in more coordination actions

between the agents than if the network is initialised with weights to output a low value for unseen situations.



(a) (b)

Figure 7.9: Output of the neural network for actions EAST, WEST for respectively Agent 1 and Agent 2. A high output value (close to 1) means there is a high need for coordination for that situation, a low output value (close to 0) means there is no need for coordination. (a) the neural network was initialised with all safe samples and (b) the network was initialised with dangerous samples.

In Figure 7.10 we show the output of a neural network that was initialised with weights that return low values for all samples for the different actions an agent can take. We see that the network correctly identified the dangerous locations at a distance of 2 cells of the agent in the respective directions the action would take the agent. The actions of the other agents are the inverse, ensuring a collision would occur without coordination.

Similar to the generalisations learned by 2Observe and the rule learning system, the neural networks that are being trained by the samples from the CQ-learning algorithm are also useful in other environments, and for different agents. Depending on the original task, there might however be a bias in the neural network. Certain situations are prone to occur more often in certain environments. For instance, in the *TunnelToGoal* environment, the agents will observe more collisions when moving on the vertical axis towards each other, since this is a critical point in this environment.

If the generalisation is used within the same environment, for agents having the same start-goal positions this does not cause a problem. In these cases, the neural network could also be trained online, together with CQ-learning. Initially the agent could rely on exact state information to solve coordination issues, but as soon as the neural network reaches a certain performance level, it could start to select its

actions based on this, rather than using exact locations. This performance level could be heterogeneous over the state space. A possibility would be to start using the neural network over exact locations as soon as it outputs the same decision regarding coordination requirements as can be drawn from the augmented states. Such an approach would also be more robust for situations in which agents do not have fixed initial positions. If the generalisation is intended to be used in different environments or for different agents, it is best to train it in a simple training grid, as was done in the previous section.

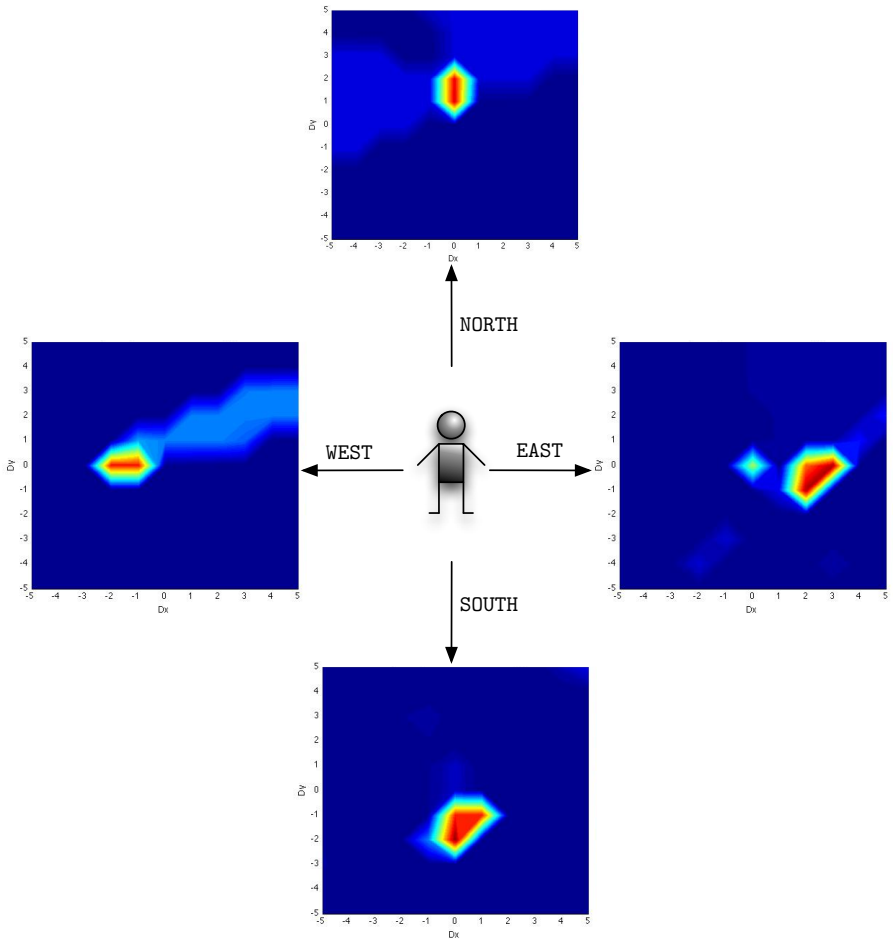


Figure 7.10: Output of the neural network for all actions that would result in two agents colliding in the next timestep.

7.5 Summary

In this chapter we have introduced three novel extensions to learning multi-agent coordination solutions. So far, such coordination problems have only been studied for one environment and were linked to specific states of that environment. We illustrated that with our 2Observe algorithm, which we introduced in Chapter 4, it is possible to transfer learned experience about coordination requirements between environments and between agents. This allows for faster learning in the target task, since agents are learning using only local state information and are using transferred knowledge for the coordination part of the task at hand.

A second approach, based on CQ-learning performed very similar. CQ-learning uses additional state information to augment local states and learn interaction states. Interaction states were refactored to agent centric representations after which a rule learning system was used to learn simple, human readable rules about the situations in which coordination between the agents was required. The idea behind this concept is that there is a relation between the interaction states which can be exploited. In the gridworld environments we have been using in our experiments, this relation is the distance between the locations of both agents. These rules were transferred from a simple training grid to more complex gridworlds. In our experiments we illustrated that this improved the learning speed of the agents in more complex target tasks.

Third, we introduced an alternative approach to this rule learning system. We again added an additional layer on top of CQ-learning to generalise the information about interaction states. This extra layer in its turn, uses this state information to learn a generalised representation of the coordination requirements that exist between the agents using a neural network. This neural network allows for very specific querying of the intentions of other agents. We tested this approach in gridworlds, but this idea is applicable to other environments as well. In routing problems the relationship between interaction states could be the sum of the number of packets in the queues of the agents. If this sum is larger than the throughput of the next node, agents must coordinate and possible choose an alternative routing scheme.

Chapter 8

Conclusion

All good things must come to an end

– Geoffrey Chaucer, 1343-1400 –

Learning in agent systems has been done since the early days of artificial intelligence. The purpose of such learning agents is the creation of intelligent systems capable of autonomous decision making without human interactions. Moreover these systems should also be capable of evolution and adaptation to changing (external) features or additional information that becomes available. A particular domain in AI that is concerned with creating such systems is reinforcement learning. RL is a powerful tool for learning in initially unknown, possible stochastic environments. It uses a numerical feedback signal from the environment to achieve optimal behaviour through repeated trial and error interactions with this environment. Several algorithms exist that converge to the optimal policy for single agent RL. The simplicity of the learning set-up, together with its broad applicability to a various range of problem domains which can be solved in a decentralised way, such as routing or automated guided vehicles, make RL extremely suitable for learning in multi-agent systems.

In MAS several additional challenges arise. One of the most important ones is the issue of coordination between the agents. Since one of the main advantages of a MAS is the ability to solve problems in a decentralised way it is desirable to avoid the requirement that each individual agent has access to all the information in the system. This would again result in a more monolithic system and, even worse, induce a large communication requirements in the system. Requirements that are usually undesirable or even impossible to guarantee. This issue can easily be solved if agents are aware of the coordination dependencies that exist in specific parts of

the state space. This concept has only recently gained attention in the multi-agent learning community. It is however not always realistic to assume that these coordination dependencies are known or that these can be calculated beforehand.

In this dissertation we presented several novel approaches for mitigating this issue. In the problems we considered, agents could accomplish a certain subtask of the goal of the system, but in order to optimally satisfy this goal, coordination between agents was required. We presented several algorithms which allow agents to learn in what parts of the state space coordination dependencies are present if multiple agents are acting simultaneously. In the following section we will give an overview of the research contained in this dissertation.

8.1 Contributions

This section summarises the research explained throughout this dissertation concerned with reinforcement learning with sparse interaction in multi-agent systems. We started by presenting an overview of the field of RL. This background is needed for understanding the domain of MARL, which was introduced in Chapter 3. It is also in this chapter that we explained sparse interactions. This concept attempts to exploit the dependencies that exist between agents in certain parts of the state space. These dependencies are visible in the outcome of the transition and reward functions of the agents. During such sparse interactions these functions result in different outcomes if, other things being equal, the state information and actions of the other agents in the system change. Note that the algorithms described in this dissertation are aimed at systems with sparse interactions. Although the approaches scale up to multiple agents, if all agents are constantly interacting, other techniques are more suitable. We discussed the current state-of-the-art of approaches that exploit these sparse interactions and acknowledged its benefits in learning in MAS. What follows is an overview of the contributions we have made within this field that go beyond the state-of-the-art.

In Chapter 4 we considered a special case of sparse interactions. We described how the list of states in which coordination is needed can, in particular situations, be modeled as a function of the current state of the agent. We formalised this idea by introducing a DEC-LIMDP. This framework represents the interaction areas of an agent as a function of the current local state of the agent. This allows for a more compact representation of these interaction areas than the more general DEC-SIMDP, in which a list of the states of the interaction area is maintained. Moreover, a DEC-LIMDP is more suitable for representing MAS in which the interaction areas are spatially related in the joint state space. We introduced a new algorithm, called *2Observe* for learning in such systems. 2Observe uses a generalized learning automaton to approximate the interaction function. By querying the automaton

at every time step about the relevant state information for the current state, it is possible to allow agents to learn using only local state information if the GLA decides that no coordination is necessary. However, if the GLA decides that coordination is required, 2Observe uses a mechanism based on communication to select an action. We first demonstrated that GLA are an appealing technique for approximating these spatial relations in the joint state space. Second, we compared 2Observe to other commonly accepted multi-agent approaches and illustrated that it reached a better performance in various environments.

Chapter 5 introduced *CQ-learning*.This algorithm is a way of solving problems that can be modelled as a DEC-SIMDP. A CQ-learning agent, performs a Student t-statistical test to determine whether the immediate rewards it perceives are what it expects for that particular state-action pair. We introduced two variants on this algorithm, which use a different baseline for this statistical test. The first variant assumes that an agent has been learning for some time alone in the environment or that a model of the reward function is available to the agent. As such it can perform a one-group Student t-test to test for significant differences in the reward signal it receives when multiple agents are acting in the same environment. The second variant exploits the fact that agents are still exploring their actions in the early stage of the learning process. We showed that the number of interactions in this early stage of the learning process is low and increases as learning progresses. Therefore, the rewards an agent receives in the initial phase of learning are a good estimate of what it would perceive if acting alone in the environment. With this data a two-group Student t-test can be performed to detect significant changes between the expected rewards and the actual rewards an agent receives. These tests allow to detect in which states agents are interacting with other agents and initiates the second step of the algorithm. This is to detect which additional state information is relevant in order to learn to coordinate. This information is again obtained using statistical tests and as soon as the relevant state information of another agent has been identified, the local state information of the agent is extended to include this state information. We call this an augmented state. In our experiments we have shown that CQ-learning uses a reduced set of augmented states in which it learns to find conflict-free policies, while maintaining good learning times compared to approaches using always the entire system state (i.e. the information of all agents). CQ-learning was compared to Learning of Coordination (referred to as LoC in this dissertation) and was shown to outperform this approach. Moreover, CQ-learning does not require the presence of an active perception function available which informs agents about the need for coordination as LoC does. Finally, CQ-learning was only evaluated in settings with a deterministic transition and reward function, but scales easily to stochastic settings since both Q-learning as the statistical tests used do not require a deterministic environment.

In Chapter 6 we expanded this approach to detect future interaction problems.

One of the most appealing features in RL is the ability to deal with a delayed reward signal. In multi-agent RL, this is also possible but comes with a high cost. The reward signal must be propagated through the entire joint state-action space which is exponential in the number of agents. Our approach, called *FCQ-learning*, is capable of mitigating this issue by detecting and exploiting sparse interactions. In this algorithm we use a Kolmogorov-Smirnov test for goodness of fit to verify if two groups of samples come from the same distribution. These groups are a sequence of Q-values for a particular state action pair. Agents will perceive a change in their future rewards by detecting it in the changing Q-values. In states in which the Q-values have decreased, the agent will sample the future rewards based on the joint state space in order to identify the relevant state information that is causing this decrease. A multiple comparison test indicate which state information of another agent is relevant and, as for CQ-learning, the local state information of the agent is extended to an augmented state which contains this additional relevant information. Based on these augmented states, the agents can coordinate several timesteps before this need is explicitely reflected in the reward signal. Our experiments have shown that the benefits of FCQ-learning become more apparent if more than two agents are present in the environment. We again compared against commonly accepted multi-agent reinforcement learning approaches and against LoC, for which we used domain information to create the active perception function. FCQ-learning outperforms LoC since it takes future rewards into consideration, rather than only the immediate rewards, which do not sufficiently represent the coordination requirements. As with CQ-learning, FCQ-learning also scales to stochastic environments. FCQ-learning is the first algorithm in its kind to be able to solve future coordination problems without having to learn all the time in the joint state-action space.

Chapter 7 extended the work from chapters 4 and 5 by introducing a way to *transfer coordination experience* between environments and agents. Transfer learning has gained an increasing interest in the reinforcement learning community in recent years. It offers the ability to significantly improve learning speeds in new unseen tasks, based on previous experience. This makes it an appealing concept to use in a multi-agent reinforcement learning context. The work done in this area was restricted to a game-theoretic setting in which learned experience from past games was used to increase learning speed in unseen games. In our research we focused on transferring coordination experience between agents and environments. The first part of this chapter was based on the 2Observe algorithm. We have shown that if an additional agent starts acting in the same environment, the learned approximation of the interaction function can be transferred to this new agent. As such, this agent can purely focus on the core task it has to solve, and re-use the experience from other agents. In the second part of this chapter we elaborated on extensions to the CQ-learning algorithm. This algorithm is capable of identifying the relevant state information for coordination problems between agents. Using this state information

it is possible to train a classifier to learn a generalised representation of the interaction states of the agents. By training this classifier in a simple environment, before transferring it to more complex settings, agents could again purely focus on the core task they have to solve. Alternatively we used a neural network to generalise the state information from augmented states. This also allows for easy mapping of this coordination information to other agents/environments since it is no longer linked with concrete state information, but uses an agent centric representation. A neural network also allows for very specific querying of the system state to only explicitly coordinate with other agents in those states where this is required.

8.2 Additional remarks

Throughout this dissertation we used gridworlds as a running example for our experiments. This testbed is very popular within the RL community, since it allows for intuitive representations of the problem tasks. Moreover, in the context of this dissertation, it is easy to represent coordination requirements between agents as collisions in a gridworld, and dependencies between the policies of agents as the order in which agents have to enter the goal. The algorithms in this dissertation are however not restricted to such environments, but are applicable for a wider range of task problems in which agents only interact with each other in certain states. Consider for example a routing problem in which each agent controls a router. As long as the queue containing the packets in the system does not significantly grow, all the agents can select the best next hop and follow their normal routing scheme. If however, the latency of the system suddenly increases because of a bottleneck in the network layout, agents should observe each others state information, i.e. their queues, in order to follow an alternative routing scheme which improves the overall throughput of the system again [Littman & Boyan (1993)].

A second remark we would like to make regarding the research in this dissertation concerns the notion of collisions in a gridworld. These 'collisions' in a gridworld represent two or more agents transitioning to the same location. In the routing example explained above, a 'collision' represents a significant increase in the latency of a certain router. We would like to emphasize that even in the context of mobile robots, such collisions do not actually require two robots to bump into each other. Such mobile robots are equipped with sensors that will detect such collisions before they actually occur. The idea behind the algorithms in this dissertation is to avoid using this failsafe feature, rather than attempting to replace it.

8.3 Directions for future research

To conclude this dissertation, we list some future directions that could be taken to extend the presented research:

- *Planning algorithms for DEC-LIMDP:* in Chapter 4 we have introduced an algorithm, capable of solving a DEC-LIMDP using a RL approach. In some situations the underlying model of the environment is available, meaning an agent has the interaction function at its disposal, together with the transition and the reward functions. As such, planning in these environments becomes an appealing approach to exploit the particular structure of these environments and solve these problems. Algorithms for planning in a DEC-LIMDP can be based upon general planning algorithms for Markov games, since environments that can be modelled as a DEC-LIMDP, can also be modelled as a Markov game.

- *Alternative single-agent approach in CQ-learning:* In Chapter 5, we introduced CQ-learning, which uses normal single agent Q-learning to learn the core task of the problem at hand. Although having theoretical convergence guarantees, Q-learning is usually not the most suitable approach as it requires a long learning time and much exploration. It is straightforward to apply Dyna-Q (see Section 2.5.3) instead of basic Q-learning for the first variant of CQ-learning, because it uses a single agent model of the reward function. This model can be used to significantly speed up the learning process of the core task, and also lead to a faster identification of the interaction states.

- *Alternative multi-agent approach in CQ-learning:* to select actions in interaction states, CQ-learning uses a Q-learning approach based on joint state information. It uses the Q-values of the local states of the agent to bootstrap, since the agent is not aware at the time of the update, whether coordination will be required during the next timestep. This part of the algorithm leaves room for improvement. For instance using the principle of an elegibility trace to update the last N visited states (augmented or local), based on the remainder of the episode, could lead to more accurate Q-values. Moreover, using such updates, it could turn out that if agents have multiple optimal actions in a state, using this update rule, coordination problems can be solved, without having to use augmented state information.

- *Alternative interaction state detection in CQ-learning:* the statistical test used by CQ-learning which determines whether states are to be augmented is a Student t-test. This test assumes that the samples come from a normal distribution. As such this test is capable of dealing with stochastic noise, as long as the reward signal remains within the confidence interval of the normal distribution for that particular state-action pair in non-interaction states. In

[Vrancx et al. (2011)] this test was already replaced by a Friedmann test which has no requirements with regard to the distribution of which the samples originate.

- *Beyond pairwise interactions:* both CQ-learning and FCQ-learning are concerned with pairwise interactions that occur in the environment. In particular states, it is possible that a coordination with two or more other agents is required. A straightforward approach is to perform statistical tests, on all possible combinations of the state information of the other agents. This results in an exponential increase in the computational complexity of the algorithm, since it has to perform a statistical test on all combinations. However, the state space in which agents have to learn would still remain manageable, so an additional question here is whether agents are still reactive enough to perform actions in a timely fashion.

- *Learning options for non-interaction states:* the option framework is a way of speeding up the learning process, by allowing an agent to take a sequence of actions during a period of time [Sutton et al. (1999)]. This sequence of actions is called an option. The option framework can be combined with CQ-learning, which would allow agents to execute options, between interaction states, which would again speed up the learning process. On the other hand, the principles of CQ-learning can also be used in the option framework, to decompose an option into several suboptions, as interaction states are identified.

- *Higher level interactions:* both CQ-learning and FCQ-learning attempt to solve coordination issues at a primitive action level. These algorithms can however be seen in a broader way as a technique of detecting when the current policy fails due to the interference of other agents and in which situations this interference takes place. As such it can be put in the wider context, such as robocup, where a team of agents can evaluate its strategy and learn a set of pre-conditions about the other team to detect when their strategy fails. These algorithms would then be used on top of strategies, learning when to adapt a strategy to the particular strategy of the opponent.

- *Multi-agent transfer learning:* in Chapter 7 we introduced two approaches of transferring knowledge between agents and environments in general Markov games. These approaches were restricted to the transfer of coordination experience between agents. Transfer learning has however a much wider applicability and many principles of single agent-learning can be adapted to settings where multiple agents are present. This area of multi-agent research is yet to be explored. The combination of techniques from single agent RL, together with the capabilities of CQ-learning can significantly improve multi-agent learning in terms of both speed and quality of the found solution.

- *Combining 2Observe and CQ-learning:* 2Observe is capable of learning the relevant sensor range within which coordination with other agents is required. CQ-learning is capable of learning in which states coordination is required. Combined, it is possible to learn the relevant sensor range in different parts of the state space. This is especially interesting in environments where the dynamics of the transition function are not uniform over the entire state space.

In closing, although our experiments share many similarities with real world applications, they were still simulations. Future research should be motivated by a desire to apply these techniques and ideas to actual real world applications.

Appendix A

Gridworld environments

This appendix presents an overview of the gridworld environments used throughout this dissertation. The initial position of an agent is marked with a **X**, its goal with a dot in the same colour. If agents share the same goal location, the dot is composed of a linear blend of the colours of the agents.

Grid game 2

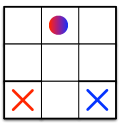


Figure A.1: Grid game 2
2 agents, 9 locations

TunnelToGoal

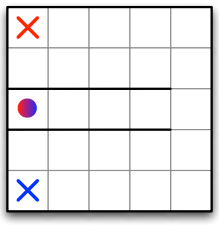


Figure A.2: TunnelToGoal
2 agents, 25 locations

2-Robot game

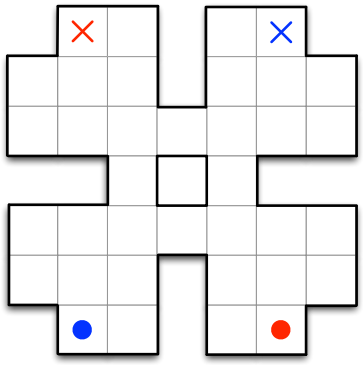


Figure A.3: 2-Robot game
2 agents, 36 locations

ISR

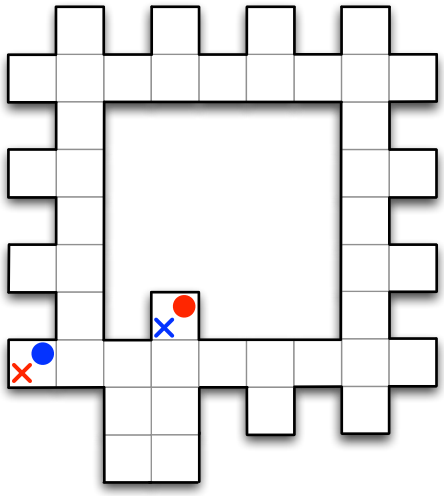


Figure A.4: ISR
2 agents, 43 locations

CIT

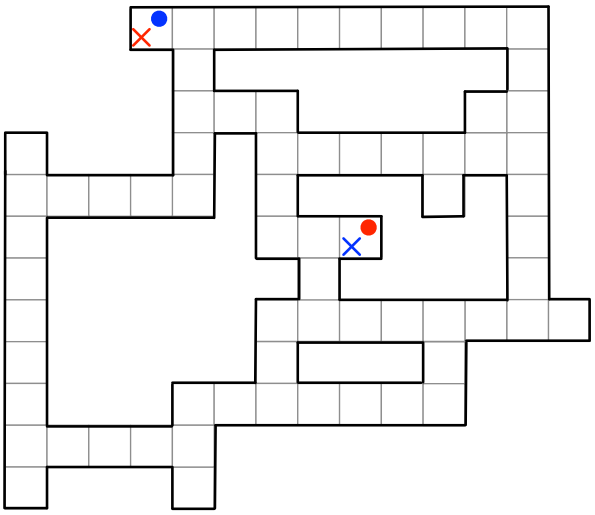


Figure A.5: CIT, 2 agents, 69 locations

TunnelToGoal_3

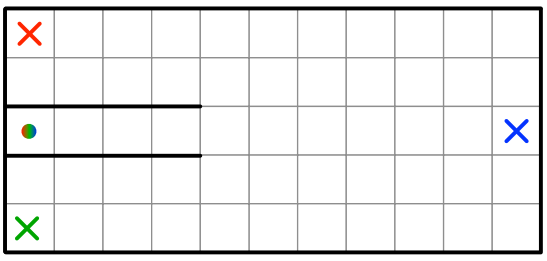


Figure A.6: TunnelToGoal_3
3 agents, 55 locations

CMU

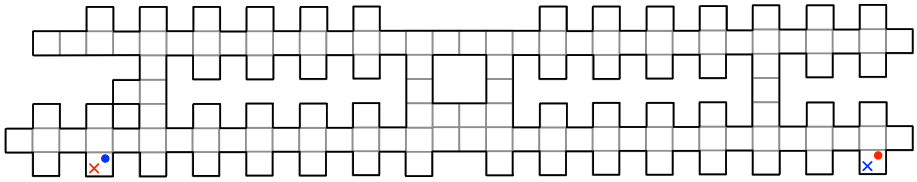


Figure A.7: CMU
2 agents, 133 locations

Bottleneck

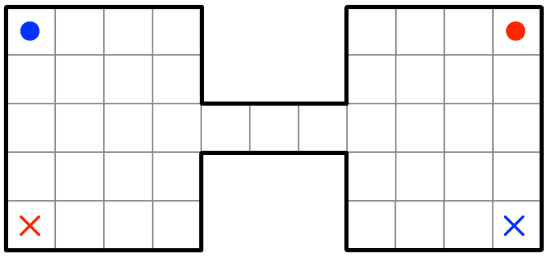


Figure A.8: Bottleneck
2 agents, 43 locations

Appendix B

Basic notions on statistical tests

B.1 Student t-test

The Student t-test was developed by William Sealy Gossett (*1876-†1937) in 1908 to test the quality of stout for Guinness®. Assume we have n independent samples $\{x_1, \ldots, x_n\}$ from a normally distributed population with mean μ and standard deviation σ. We would like to verify whether $\mu = \mu_0$, based on the samples. Both μ and σ are however unknown variables. μ can be estimated by using the mean $\bar{x}_n$ of the samples:

$$\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{B.1}$$

and the standard deviation S_n of the samples with:

$$S_n = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x}_n)^2 \tag{B.2}$$

With these estimates, we can calculate a confidence interval for a certain confidence level $1-\alpha$ around the mean $\bar{x}_n$.
This confidence interval for confidence level $1-\alpha$ is calculated according to:

$$\left[\bar{x}_n - \frac{S_n}{\sqrt{n}} t_{n-1,1-\alpha/2}, \bar{x}_n + \frac{S_n}{\sqrt{n}} t_{n-1,1-\alpha/2}\right] \tag{B.3}$$

where t_{n-1} represents a t-distribution with $n-1$ degrees of freedom.

The value t represents the deviation of the sample mean from the population mean measured in units of the mean's standard error $S_n\sqrt{n}$:

$$t = \frac{\bar{x}_n - \mu}{S_n/\sqrt{n}} \tag{B.4}$$

A t-distribution is derived from a normal distribution, but has a wider base. In Figure B.1 we show the t-distribution, for various degrees of freedom. Figure B.2

represents the corresponding cumulative probability distribution. Note that in the limit, where $n \to \infty$, the t-distribution overlaps with the normal distribution.

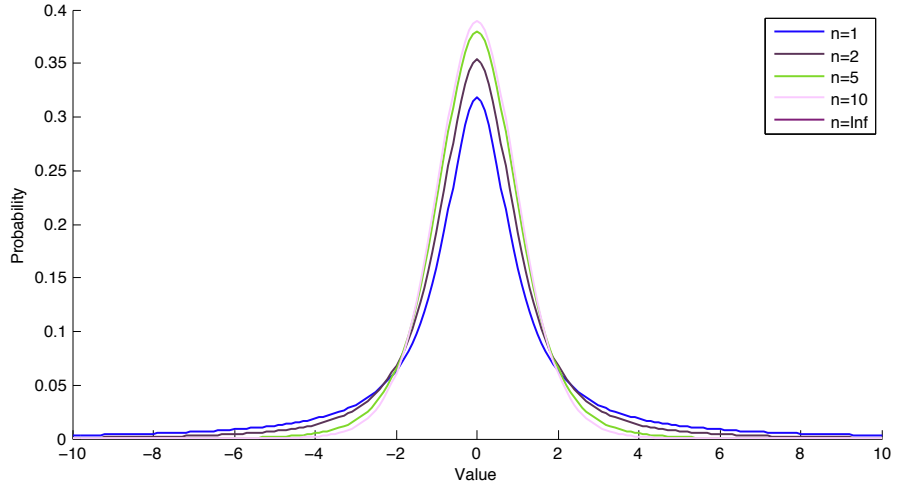


Figure B.1: T-distributions for various degrees of freedom.

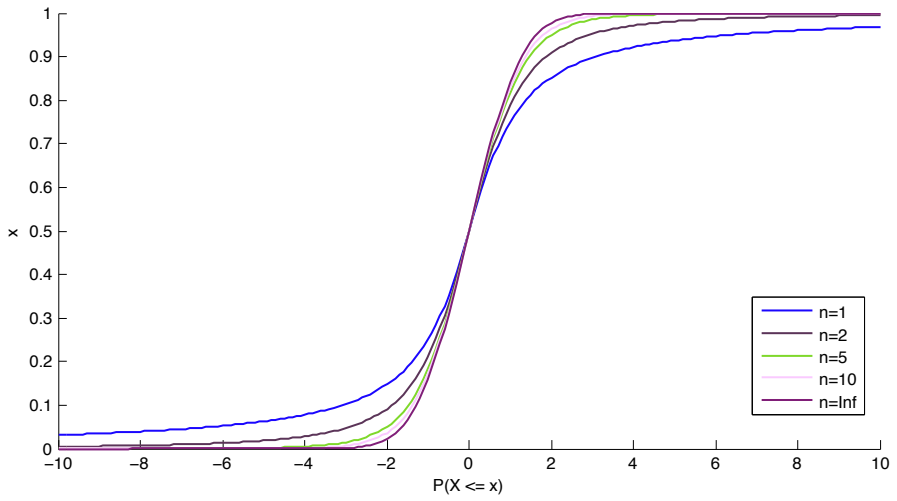


Figure B.2: Corresponding cumulative distribution functions of Figure B.1 for various degrees of freedom.

From Equation B.4 follows that the null hypothesis for the Student t-test is

based on following statistic:

$$T = \frac{\bar{x}_n - \mu_0}{S_n/\sqrt{n}} \quad \text{under } H_0 : \mu = \mu_0 \tag{B.5}$$

If rather than to verify if the sample mean come is equal to the population mean, we want to test if the means of two groups of sizes n and m are equal, this test is called a two-group t-test. The null hypothesis is that $\mu_1 = \mu_2$. The statistic T is now given by:

$$T = \frac{\bar{x}_n^1 - \bar{x}_m^2}{S_{nm}^{12}\sqrt{\frac{1}{n} + \frac{1}{m}}} \quad \text{under } H_0 : \mu_1 = \mu_2 \tag{B.6}$$

Where $\bar{x}_n^1$ and $\bar{x}_m^2$ represent the respective sample means of both groups and S_{nm}^{12} stands for the weighted average of the standard deviation of the samples in both groups. It can be calculated as follows:

$$S_{nm}^{12} = \frac{(n-1)S_n^1 + (m-1)S_m^2}{n+m-2} \tag{B.7}$$

The confidence interval for the two-sided two-group t-test is given by:

$$\left[-t_{n-1,1-\alpha/2}, t_{n-1,1-\alpha/2}\right] \tag{B.8}$$

α	Percentiles of the t-distribution (two sided)							
	0.20	**0.10**	**0.05**	**0.02**	**0.01**	**0.005**	**0.002**	**0.001**
1	3.078	6.314	12.706	31.820	63.657	127.321	318.309	636.619
2	1.886	2.920	4.303	6.965	9.925	14.089	22.327	31.599
3	1.638	2.353	3.182	4.541	5.841	7.453	10.215	12.924
4	1.533	2.132	2.776	3.747	4.604	5.598	7.173	8.610
5	1.476	2.015	2.571	3.365	4.032	4.773	5.893	6.869
6	1.440	1.943	2.447	3.143	3.707	4.317	5.208	5.959
7	1.415	1.895	2.365	2.998	3.499	4.029	4.785	5.408
8	1.397	1.860	2.306	2.897	3.355	3.833	4.501	5.041
9	1.383	1.833	2.262	2.821	3.250	3.690	4.297	4.781
10	1.372	1.812	2.228	2.764	3.169	3.581	4.144	4.587
11	1.363	1.796	2.201	2.718	3.106	3.497	4.025	4.437
12	1.356	1.782	2.179	2.681	3.055	3.428	3.930	4.318
13	1.350	1.771	2.160	2.650	3.012	3.372	3.852	4.221
14	1.345	1.761	2.145	2.625	2.977	3.326	3.787	4.140
15	1.341	1.753	2.131	2.602	2.947	3.286	3.733	4.073
16	1.337	1.746	2.120	2.584	2.921	3.252	3.686	4.015
17	1.333	1.740	2.110	2.567	2.898	3.222	3.646	3.965
18	1.330	1.734	2.101	2.552	2.878	3.197	3.610	3.922
19	1.328	1.729	2.093	2.539	2.861	3.174	3.579	3.883
20	1.325	1.725	2.086	2.528	2.845	3.153	3.552	3.850
21	1.323	1.721	2.080	2.518	2.831	3.135	3.527	3.819
22	1.321	1.717	2.074	2.508	2.819	3.119	3.505	3.792
23	1.319	1.714	2.069	2.500	2.807	3.104	3.485	3.768
24	1.318	1.711	2.064	2.492	2.797	3.090	3.467	3.745
25	1.316	1.708	2.060	2.485	2.787	3.078	3.450	3.725
30	1.310	1.697	2.042	2.457	2.750	3.030	3.385	3.646
40	1.303	1.684	2.021	2.423	2.704	2.971	3.307	3.551
50	1.299	1.676	2.009	2.403	2.678	2.937	3.261	3.496
60	1.296	1.671	2.000	2.390	2.660	2.915	3.232	3.460
70	1.294	1.667	1.994	2.381	2.648	2.899	3.211	3.435
80	1.292	1.664	1.990	2.374	2.639	2.887	3.195	3.416
90	1.291	1.662	1.987	2.369	2.632	2.878	3.183	3.402
100	1.290	1.660	1.984	2.364	2.626	2.871	3.174	3.391
150	1.287	1.655	1.976	2.351	2.609	2.849	3.145	3.357
200	1.286	1.652	1.972	2.345	2.601	2.839	3.131	3.340
∞	1.282	1.645	1.960	2.326	2.576	2.807	3.090	3.291

Table B.1: Percentiles of the two sided t-distribution

B.2 Kolmogorov-Smirnov test

The Kolmogorov-Smirnov test (KS-test) is a non-parametric test that is used to compare a sample probability distribution against a reference distribution (one-sample KS-test) or to compare two samples against each other (two-sample KS-test). For the former, the null hypothesis is that the sample is drawn from the reference distribution. For the latter the null hypothesis is that both samples are drawn from the same distribution. Since it is a non-parametric test, there are no additional restrictions on the distribution of the sample(s).

The test statistic is the maximum distance between both empirical distribution functions (two sample test) or between the empirical distribution and the cumulative distribution function against which we test (one sample test). The empirical distribution function for n independent and identically distributed random variables X_i is defined as:

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{X_i<x} \tag{B.9}$$

where $I_{X_i<x}$ is the indicator function:

$$I_{X_i<x} = \begin{cases} 1 & \text{if } X_i < x \\ 0 & \text{otherwise} \end{cases} \tag{B.10}$$

The formula for the statistic of the one-sample test is:

$$D_n = \max_x(|F_n(x) - F(x)|) \tag{B.11}$$

and for the two-sample test:

$$D_{n,m} = \max_x(|F_n^1(x) - F_m^2(x)|) \tag{B.12}$$

For the one sample test, the null hypothesis is rejected at level α if

$$\sqrt{n}D_n > K_\alpha \tag{B.13}$$

and for the two sample test if

$$\sqrt{\frac{nm}{n+m}}D_{n,m} > K_\alpha \tag{B.14}$$

where K_α is the critical value of the Kolmogorov distribution at level α. The critical values for some common confidence levels are given in Table B.2

An example of the empirical cumulative distribution for a one sided test is given in Figure B.3 and for a two sided test both empirical cumulative distributions are shown in Figure B.4.

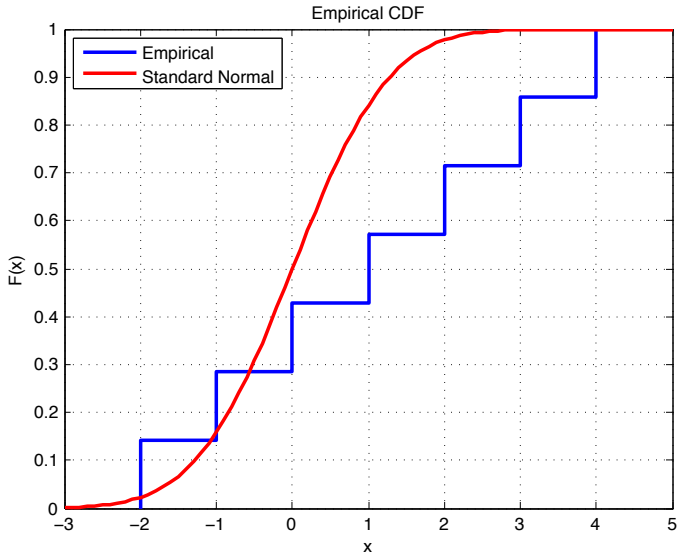


Figure B.3: Empirical cumulative distribution with the standard normal distribution.

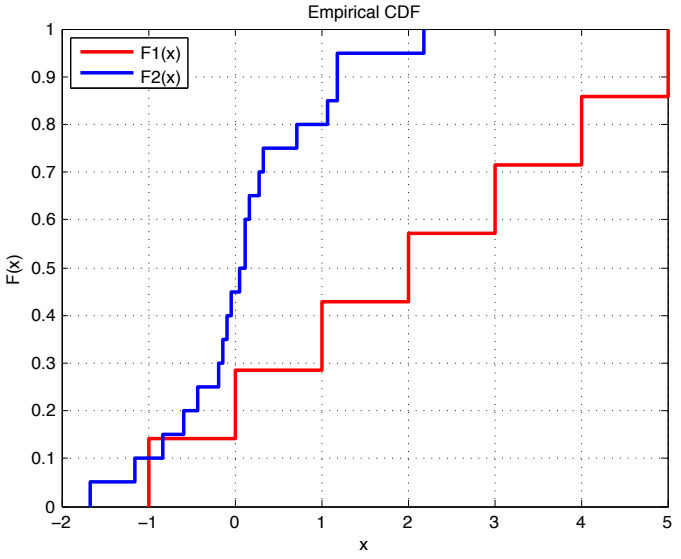


Figure B.4: Two empirical cumulative distributions.

n	D (one sided)				
	0.1	0.05	0.025	0.01	0.005
1	0.9000	0.9500	0.9750	0.9900	0.9950
2	0.6838	0.7764	0.8419	0.9000	0.9293
3	0.5648	0.6360	0.7076	0.7846	0.8290
4	0.4927	0.5652	0.6239	0.6889	0.7342
5	0.4470	0.5094	0.5633	0.6272	0.6685
6	0.4104	0.4680	0.5193	0.5774	0.6166
7	0.3815	0.4361	0.4834	0.5384	0.5758
8	0.3583	0.4096	0.4543	0.5065	0.5418
9	0.3391	0.3875	0.4300	0.4796	0.5133
10	0.3226	0.3687	0.4092	0.4566	0.4889
11	0.3083	0.3524	0.3912	0.4367	0.4677
12	0.2958	0.3382	0.3754	0.4192	0.4490
13	0.2847	0.3255	0.3614	0.4036	0.4325
14	0.2748	0.3142	0.3489	0.3897	0.4176
15	0.2659	0.3040	0.3376	0.3771	0.4042
16	0.2578	0.2947	0.3273	0.3657	0.3920
17	0.2504	0.2863	0.3180	0.3553	0.3809
18	0.2436	0.2785	0.3094	0.3457	0.3706
19	0.2373	0.2714	0.3014	0.3369	0.3612
20	0.2316	0.2647	0.2941	0.3287	0.3524
21	0.2262	0.2586	0.2872	0.3210	0.3443
22	0.2212	0.2528	0.2809	0.3139	0.3367
23	0.2165	0.2475	0.2749	0.3073	0.3295
24	0.2120	0.2424	0.2693	0.3010	0.3229
25	0.2079	0.2377	0.2640	0.2952	0.3166
26	0.2040	0.2332	0.2591	0.2896	0.3106
27	0.2003	0.2290	0.2544	0.2844	0.3050
28	0.1968	0.2250	0.2499	0.2794	0.2997
29	0.1935	0.2212	0.2457	0.2747	0.2947
30	0.1903	0.2176	0.2417	0.2702	0.2899
40	0.1655	0.1891	0.2101	0.2349	0.2521
$> \mathbf{40}$	$1.07/\sqrt{n}$	$1.22/\sqrt{n}$	$1.36/\sqrt{n}$	$1.52/\sqrt{n}$	$1.63/\sqrt{n}$

Table B.2: Critical values for the one-sided Kolmogorov-Smirnov test

B.3 Friedman test

The Friedman test is, like the KS-test, a non-parametric statistical test. It was developed by the U.S. economist Milton Friedman to detect differences in treatments across multiple test attempts.

If the data is for the form $\{x_{ij}\}_{n\times k}$, with each of the n rows representing a block and each of the k columns a treatment. A single observation is the intersection of each block and treatment. From this data, the rank can be calculated within each block:

$$\bar{r}_{\cdot j} = \frac{1}{n}\sum_{i=1}^{n} r_{ij}\bar{r} = \frac{1}{nk}\sum_{i=1}^{n}\sum_{j=1}^{k} r_{ij} \tag{B.15}$$

The test statistic is calculated as follows:

$$Q = \frac{SS_t}{SS_e} \tag{B.16}$$

where SS_t and SS_e can be calculated from the rank as follows:

$$SS_t = n\sum_{j=1}^{k}(\bar{r}_{\cdot j} - \bar{r})^2, SS_e = \frac{1}{n(k-1)}\sum_{i=1}^{n}\sum_{j=1}^{k}(r_{ij} - \bar{r})^2 \tag{B.17}$$

Abbreviations

AI	Artificial Intelligence
CG	Coordination Graph
CPD	Conditional Probability Distribution
CQ-learning	Coordinating Q-learning
DEC-MDP	Decentralised Markov Decision Process
DEC-MG	Decentralised Markov Game
DEC-LIMDP	Decentralised Local Interactions Markov Decision Process
DEC-SIMDP	Decentralised Sparse Interactions Markov Decision Process
DEC-SIMG	Decentralised Sparse Interactions Markov Game
DBN	Dynamic Bayesian Network
DP	Dynamic Programming
EGT	Evolutionary Game Theory
FCQ-learning	Future Coordinating Q-learning
FMDP	Factored Markov Decision Process
GT	Game Theory
GLA	Generalized Learning Automata
IDMG	Interaction Driven Markov Game
KS	Kolmogorov-Smirnov

LA	Learning Automaton
LoC	Learning of Coordination
MARL	Multi-agent Reinforcement Learning
MAS	Multi-agent Systems
MDP	Markov Decision Process
MMDP	Multi-Agent Markov Decision Process
MG	Markov Game
PE	Policy Evaluation
PI	Policy Improvement
POMDP	Partially Observable Markov Decision Process
RD	Replicator Dynamics
RL	Reinforcement Learning
SCQ	Sparse Cooperative Q-learning
TI-DEC-MDP	Transition Independent Decentralised Markov Decision Process
UC	Utile Coordination

Notation

general	
τ	Temperature parameter
$\alpha(t)$	learning rate at timestep t
γ	discount factor
Single agent reinforcement learning	
$S = s^1, \ldots, s^N$	Set of states
$A(s)$	action set in state s.
$T(s, a, s')$	probability of going from state s to state s' after performing action a.
$R(s, a, s')$	expected reward for performing action a in state s and transitioning to state s'.
π_t	The policy at timestep t
$\pi_t(s, a)$	Probability of taking action a in state s at timestep t
$V^\pi(s)$	The value of state s under policy π
$Q^\pi(s, a)$	The value of action a in state s under policy π
$V^*(s)$	The value of state s under an optimal policy
$Q^*(s, a)$	The value of action a in state s under an optimal policy
Multi agent reinforcement learning	
n	number of agents
$S = s^1, \ldots, s^N$	Set of system states.
$S_k = s_k^1, \ldots, s_k^N$	Set of local states of agent k.

Multi agent reinforcement learning (cont.)	
s^i	Each system state consists about the state information of the environment s_0, combined with the local state information of all agents $s_1, \ldots, s_n$ with every $s_j \in S_j$
$A_k(s)$	action set of agent k in state s.
$\vec{a}$	joint action $(a_1, \ldots, a_n)$ specifying the action of all agents
$R(s, \vec{a}, s')$	expected reward for all agents for performing joint action a in state s and transitioning to state s'.
$R_k(s, \vec{a}, s')$	expected reward for agent k for performing joint action a in state s and transitioning to state s'.

Note: superscripts refer to the index of actions/states/..., whereas subscripts refer to the index of the agent. For example s_k^i represents the i^{th} local state of agent k.

Bibliography

[Ahamed et al. (2002)] Ahamed, T., Nagendra Rao, P. & Sastry, P. (2002). A reinforcement learning approach to automatic generation control. *Electric Power Systems Research* **63**(1), 9–26.

[Aras et al. (2004)] Aras, R., Dutech, A. & Charpillet, F. (2004). Cooperation through communication in decentralized markov games. In: *Proceedings of the International Conference on Advances in Intelligent Systems - Theory and Applications*. Luxembourg-Kirchberg, Luxembourg: IEEE Computer Society.

[Astrom (1965)] Astrom, K. (1965). Optimal control of markov decision processes with incomplete state information. *Journal of Mathematical Analysis and Applications* **10**, 174–205.

[Auer et al. (2002)] Auer, P., Cesa-Bianchi, N. & Fischer, P. (2002). Finite time analysis of the multiarmed bandit problem. *Machine Learning* **47**(2-3), 235–256.

[Banerjee & Stone (2007)] Banerjee, B. & Stone, P. (2007). General game learning using knowledge transfer. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*.

[Barto et al. (1989)] Barto, A., Sutton, R. & Watkins, C. (1989). Learning and sequential decision making. In: *Learning and Computational Neuroschience*. MIT Press, pp. 539–602.

[Becker et al. (2004)] Becker, R., Zilberstein, S. & Goldman, C. (2004). Solving transition independent decentralized markov decision processes. *Journal of Artificial Intelligence Research* **22**, 423–455.

[Becker et al. (2003)] Becker, R., Zilberstein, S., Lesser, V. & Goldman, C. (2003). Transition-independent decentralized markov decision processes. In: *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*. New York, NY, USA: ACM.

[Bellman (1957)] Bellman, R. (1957). *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press.

[Bernstein et al. (2002)] Bernstein, D., Givan, R., Immerman, N. & Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Mathematics Operartion Research* **27**, 819–840.

[Bertsekas (1976)] Bertsekas, D. (1976). *Dynamic Programming and Stochastic Control*. Orlando, FL, USA: Academic Press, Inc.

[Bertsekas (1995)a] Bertsekas, D. (1995a). *Dynamic Programming and Optimal Control (Volume One)*. Nashua, NH, USA: Athena Scientific.

[Bertsekas (1995)b] Bertsekas, D. (1995b). *Dynamic Programming and Optimal Control (Volume Two)*. Nashua, NH, USA: Athena Scientific.

[Bertsekas & Tsitsiklis (1996)] Bertsekas, D. & Tsitsiklis, J. (1996). *Dynamic Programming and Optimal Control (Volume One)*. Nashua, NH, USA: Athena Scientific.

[Bond & Gasser (1988)] Bond, A. & Gasser, L. (1988). A survey of distributed artificial intelligence. *Readings in Distributed Artificial Intelligence* .

[Börgers & Sarin (1997)] Börgers, T. & Sarin, R. (1997). Learning through reinforcement and replicator dynamics. *Journal of Economic Theory* **77**, 1–14.

[Boutilier (1996)] Boutilier, C. (1996). Planning, learning and coordination in multiagent decision processes. In: *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*. Renesse, Holland.

[Boutilier et al. (1995)] Boutilier, C., Dearden, R. & Goldszmidt, M. (1995). Exploiting structure in policy construction. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2 (IJCAI)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[Boyan & Moore (1995)] Boyan, J. & Moore, A. (1995). Generalization in reinforcement learning: Safely approximating the value function. In: *Proceedings of the 7th International Conference on Advances in Neural Information Processing Systems (NIPS)*. MIT Press.

[Bush & Mosteller (1955)] Bush, R. & Mosteller, F. (1955). *Stochastic Models for Learning*. New York, NY, USA: John Wiley & Sons.

[Busoniu (2008)] Busoniu, L. (2008). *Reinforcement Learning in Continuous Stand and Action Spaces*. Ph.D. thesis, Technische Universiteit Delft.

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

[Busoniu et al. (2010)] Busoniu, L., Babuska, R., De Schutter, B. & Ernst, D. (eds.) (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, 1 ed.

[Cassandra (1998)] Cassandra, A. (1998). *Exact and Approximate Algorithms for POMDP*. Ph.D. thesis, Brown University.

[Chapman & Kaelbling (1990)] Chapman, D. & Kaelbling, L. (1990). Learning from delayed reinforcement in a complex domain. Tech. Rep. TR-90-11, Teleos Research.

[Chapman & Kaelbling (1991)] Chapman, D. & Kaelbling, L. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*.

[Claus & Boutilier (1998)] Claus, C. & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In: *Proceedings of the 15th National Conference on Artificial Intelligence*. AAAI Press.

[Cochran & Snedecor (1989)] Cochran, W. & Snedecor, G. (1989). *Statistical Methods*. Ames, IA, USA: Iowa State University Press.

[Cohen (1995)] Cohen, W. (1995). Fast effective rule induction. In: *Proceedings of the 12th International Conference on Machine Learning (ICML)*. Morgan Kaufmann.

[Cross (1973)] Cross, J. (1973). A stochastic learning model of economic behavior. *The Quarterly Journal of Economics* , 239–266.

[Dawkins (1986)] Dawkins, R. (1986). *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe without Design*. W.W. Norton & Company.

[De Hauwere et al. (2008)a] De Hauwere, Y.-M., Vrancx, P. & Nowé, A. (2008a). Multi-agent state space aggregation using generalized learning automata. In: *Benelearn, The Annual Belgian-Dutch Machine Learning Conference*. Spa, Belgium.

[De Hauwere et al. (2008)b] De Hauwere, Y.-M., Vrancx, P. & Nowé, A. (2008b). Using generalized learning automata for state space aggregation in mas. In: *the 12th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems* (Goebel, R., Siekmann, J. & Wahlster, W., eds.). Zagreb, Croatia: Springer Berlin / Heidelberg. URL `http://www.springerlink.com/content/gru865110318m5n6/`.

[De Hauwere et al. (2009)a] De Hauwere, Y.-M., Vrancx, P. & Nowé, A. (2009a). Learning what to observe in multi-agent systems. In: *the 21st Benelux Conference on Artificial Intelligence*. Eindhoven, The Netherlands.

[De Hauwere et al. (2009)b] De Hauwere, Y.-M., Vrancx, P. & Nowé, A. (2009b). Multi-layer learning and knowledge transfer in mas. In: *the 7th European Workshop on Multi-Agent Systems*. Ayia Napa, Cyprus.

[De Hauwere et al. (2010)a] De Hauwere, Y.-M., Vrancx, P. & Nowé, A. (2010a). Generalized learning automata for multi-agent reinforcement learning. *Journal of AI Communications: Special issue* **23**, 311–324.

[De Hauwere et al. (2010)b] De Hauwere, Y.-M., Vrancx, P. & Nowé, A. (2010b). Learning multi-agent state space representations. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Toronto, Canada.

[De Hauwere et al. (2010)c] De Hauwere, Y.-M., Vrancx, P. & Nowé, A. (2010c). *Multi-agent systems and large state spaces*, chap. to appear. Multi-agent system technology for Internet and Enterprise Systems. Springer.

[De Hauwere et al. (2011)a] De Hauwere, Y.-M., Vrancx, P. & Nowé, A. (2011a). Adaptive state representations for multi-agent reinforcement learning. In: *Proceedings of the 3th International Conference on Agents and Artificial Intelligence*. Rome, Italy.

[De Hauwere et al. (2011)b] De Hauwere, Y.-M., Vrancx, P. & Nowé, A. (2011b). Detecting and solving future multi-agent interactions. In: *Proceedings of the AAMAS Workshop on Adaptive and Learning Agents*. Taipei, Taiwan.

[De Hauwere et al. (2011)c] De Hauwere, Y.-M., Vrancx, P. & Nowé, A. (2011c). Solving delayed coordination problems in mas (extended abstract). In: *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Taipei, Taiwan.

[de Jong (2009)] de Jong, S. (2009). *Fairness in Multi-Agent Systems*. Ph.D. thesis, Universiteit Maastricht.

[Degris et al. (2006)] Degris, T., Sigaud, O. & Wuillemin, P.-H. (2006). Learning the structure of factored markov decision processes in reinforcement learning problems. In: *Proceedings of the 23rd International Conference on Machine learning (ICML)*. New York, NY, USA.

[Dietterich (1998)] Dietterich, T. (1998). Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research* **13**, 227–303.

[Dietterich (2000)] Dietterich, T. (2000). State abstraction in maxq hierarchical reinforcement learning. In: *Advances in Neural Information Processing Systems 12*. MIT Press.

[Dorigo & Stützle (2004)] Dorigo, M. & Stützle, T. (2004). *Ant Colony Optimization*. Concord, MA, USA: Bradford Company.

[Finney et al. (2002)] Finney, S., Wakker, P., Kaelbling, L. & Oates, T. (2002). The thing that we tried didn't work very well: Deictic representation in reinforcement learning. In: *Proceedings of the 2002 Conference on Uncertainty in Artificial Intelligence (UAI)*.

[Ghavamzadeh & Mahadevan (2004)] Ghavamzadeh, M. & Mahadevan, S. (2004). Learning to communicate and act using hierarchical reinforcement learning. In: *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Washington, DC, USA: IEEE Computer Society.

[Gillette (1957)] Gillette, D. (1957). Stochastic games with zero stop probabilities. *Annals of Mathematics Studies* **39**, 178–187.

[Gintis (2000)] Gintis, H. (2000). *Game theory evolving*. Princeton University Press.

[Goetschalckx (2009)] Goetschalckx, R. (2009). *The Use of Domain Knowledge in Reinforcement Learning*. Ph.D. thesis, Katholieke Universiteit Leuven.

[Goldman & Zilberstein (2004)] Goldman, C. & Zilberstein, S. (2004). Decentralized control of cooperative systems: categorization and complexity analysis. *Journal of Artificial Intelligence Research* **22**, 143–174.

[Greenwald & Hall (2003)] Greenwald, A. & Hall, K. (2003). Correlated-q learning. In: *AAAI Spring Symposium*. AAAI Press.

[Grzes (2010)] Grzes, M. (2010). *Improving Exploration in Reinforcement Learning through Domain Knowledge and Parameter Analysis*. Ph.D. thesis, University of York.

[Grzes & Kudenko (2009)] Grzes, M. & Kudenko, D. (2009). Theoretical and empirical analysis of reward shaping in reinforcement learning. In: *Proceedings of the 2009 International Conference on Machine Learning and Applications (ICMLA)*. Washington, DC, USA: IEEE Computer Society.

[Guestrin et al. (2004)] Guestrin, C., Hauskrecht, M. & Kveton, B. (2004). Solving factored mdps with continuous and discrete variables. In: *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI)*. Arlington, VA, USA: AUAI Press.

[Guestrin et al. (2002)a] Guestrin, C., Lagoudakis, M. & Parr, R. (2002a). Coordinated reinforcement learning. In: *Proceedings of the 19th International Conference on Machine Learning (ICML)*.

[Guestrin et al. (2002)b] Guestrin, C., Venkataraman, S. & Koller, D. (2002b). Context-specific multiagent coordination and planning with factored mdps. In: *18th national conference on Artificial intelligence*. Menlo Park, CA, USA: American Association for Artificial Intelligence.

[Guttmann (2009)] Guttmann, C. (2009). Towards a taxonomy of decision making problems in multi-agent systems. In: *Multiagent System Technologies* (Braubach, L., van der Hoek, W., Petta, P. & Pokahr, A., eds.), vol. 5774 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, pp. 195–201.

[Guyon et al. (2006)] Guyon, I., Gunn, S., Nikravesh, M. & Zadeh, L. (eds.) (2006). *Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing)*. Springer, 1 ed.

[Hengst (2005)] Hengst, B. (2005). *Discovering hierarchy in reinforcement learning*. Ph.D. thesis, University of New South Wales.

[Hofbauer et al. (1979)] Hofbauer, J., Schuster, P. & Sigmund, K. (1979). A note on evolutionary stable strategies and game dynamics. *Journal of Theoretical Biology* **81**(3), 609–612.

[Hu & Wellman (1998)] Hu, J. & Wellman, M. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In: *Proceedings of the 15th International Conference on Machine Learning (ICML)*. Morgan Kaufmann.

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

[Jong & Stone (2004)] Jong, N. & Stone, P. (2004). Towards learning to ignore irrelevant state variables. In: *Proceedings of the AAAI-2004 Workshop on Learning and Planning in Markov Processes — Advances and Challenges*.

[K-Team (2009)] K-Team (2009). Khepera III by K-Team. `http://www.k-team.com/mobile-robotics-products/khepera-iii`.

[Kaelbling et al. (1996)] Kaelbling, L., Littman, M. & Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* **4**, 237–285.

[Kaminka (2004)] Kaminka, G. (2004). Multi-agent systems. In: *Encyclopedia of Human-Computer Interaction*. Berkshire Publishing.

[Kok et al. (2005)] Kok, J., 't Hoen, P., Bakker, B. & Vlassis, N. (2005). Utile coordination: Learning interdependencies among cooperative agents. In: *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*.

[Kok & Vlassis (2004)a] Kok, J. & Vlassis, N. (2004a). Sparse cooperative Q-learning. In: *Proceedings of the 21st International Conference on Machine learning (ICML)*. ACM New York, NY, USA.

[Kok & Vlassis (2004)b] Kok, J. & Vlassis, N. (2004b). Sparse tabular multiagent q-learning. In: *Proceedings of the 13th Benelux Conference on Machine Learning (Benelearn)*.

[Kok & Vlassis (2006)] Kok, J. & Vlassis, N. (2006). Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research* **7**, 1789–1828.

[Koller (1999)] Koller, D. (1999). Computing factored value functions for policies in structured mdps. In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann.

[Könönen (2003)a] Könönen, V. (2003a). Asymmetric multiagent reinforcement learning. In: *IEEE/WIC International Conference on Intelligent Agent Technology (IAT)*.

[Könönen (2003)b] Könönen, V. (2003b). Gradient based method for symmetric and asymmetric multiagent reinforcement learning. In: *Intelligent Data Engineering and Automated Learning*, vol. 2690 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, pp. 68–75.

[Kuhlmann & Stone (2007)] Kuhlmann, G. & Stone, P. (2007). Graph-based domain mapping for transfer learning in general games. In: *Proceedings of the 18th European Conference on Machine Learning (ECML)*. Springer.

[Lazaric (2008)] Lazaric, A. (2008). *Knowledge Transfer in Reinforcement Learning*. Ph.D. thesis, Politecnico di Milano.

[Littman (1994)] Littman, M. (1994). Markov games as a framework for multi-agent reinforcement learning. In: *Proceedings of the 11th International Conference on Machine Learning (ICML)*. Morgan Kaufmann.

[Littman (2001)a] Littman, M. (2001a). Friend-or-foe q-learning in general-sum games. In: *Proceedings of the 18th International Conference on Machine Learning (ICML)*. Morgan Kaufmann.

[Littman (2001)b] Littman, M. (2001b). Value-function reinforcement learning in markov games. *Cognitive Systems Research* **2**(1), 55–66.

[Littman & Boyan (1993)] Littman, M. & Boyan, J. (1993). A distributed reinforcement learning scheme for network routing. In: *Proceedings of the 1993 International Workshop on Applications of Neural Networks to Telecommunications*. Erlbaum.

[Littman et al. (1998)] Littman, M., Cassandra, A. & Kaelbling, L. (1998). Learning policies for partially observable environments: scaling up. In: *Readings in agents* (Huhns, M. & Singh, M., eds.). San Francisco, CA, USA: Morgan Kaufmann, pp. 495–503.

[Mariano & Morales (2001)] Mariano, C. & Morales, E. (2001). Dql: a new updating strategy for reinforcement learning based on q-learning. In: *Machine Learning: ECML 2001* (De Raedt, L. & Flach, P., eds.), vol. 2167 of *Lecture Notes in Computer Science*. Springer, pp. 324–335.

[McCallum (1995)] McCallum, A. (1995). *Reinforcement Learning with Selective Perception and hidden State*. Ph.D. thesis, University of Rochester.

[Melo & Veloso (2009)] Melo, F. & Veloso, M. (2009). Learning of coordination: Exploiting sparse interactions in multiagent systems. In: *Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems.

[Melo & Veloso (2010)a] Melo, F. & Veloso, M. (2010a). Approximate planning for decentralized mdps with sparse interactions. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems.

[Melo & Veloso (2010)b] Melo, F. & Veloso, M. (2010b). Local multiagent coordination in decentralised mdps with sparse interactions. Tech. Rep. CMU-CS-10-133, School of Computer Science, Carnegie Mellon University.

[Mitchell (1997)] Mitchell, T. (1997). *Machine Learning*. Singapore, China: McGraw-Hill.

[Moore & Atkeson (1993)] Moore, A. & Atkeson, C. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* **13**, 103–130.

[Narendra & Thathachar (1989)] Narendra, K. & Thathachar, M. (1989). *Learning automata: An introduction*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

[Nowé et al. (2011)] Nowé, A., Vrancx, P. & De Hauwere, Y.-M. (2011). Game theory and multi-agent reinforcement learning. In: *Reinforcement Learning: State of the Art* (Wiering, M. & van Otterlo, M., eds.). Springer Verslag.

[Osborne & Rubinstein (1999)] Osborne, M. & Rubinstein, A. (1999). *A course in game theory*. MIT press.

[Pell (1993)] Pell, B. D. (1993). *Strategy generation and evaluation for meta-game playing*. Ph.D. thesis, University of Cambridge.

[Peshkin et al. (2000)] Peshkin, L., Kim, K., Meuleau, N. & Kaelbling, L. (2000). Learning to cooperate via policy search. In: *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[Phansalkar et al. (1990)] Phansalkar, V., Sastry, P. & Thathachar, M. (1990). An optimization approach to the analysis of generalized learning automata algorithms. In: *Proceedings of the 16th Annual Convention and Exhibition of the IEEE In India (ACE)*. IEEE.

[Phansalkar & Thathachar (1995)] Phansalkar, V. & Thathachar, M. (1995). Local and global optimization algorithms for generalized learning automata. *Neural Computation* **7**(5), 950–973.

[Puterman (1994)] Puterman, M. (1994). *Markov Decision Processes, Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons.

[Russel & Norvig (2010)] Russel, S. & Norvig, P. (eds.) (2010). *Artificial Intelligence: A modern approach*. Pearson Education, 3 ed.

[Samuelson (1998)] Samuelson, L. (1998). *Evolutionary games and equilibrium selection*. The MIT Press.

[Sen et al. (1994)] Sen, S., Sekaran, M. & Hale, J. (1994). Learning to coordinate without sharing information. In: *Proceedings of the 12th national*

conference on Artificial intelligence (AAAI). Menlo Park, CA, USA: American Association for Artificial Intelligence.

[Sherstov & Stone (2005)] Sherstov, A. & Stone, P. (2005). Improving action selection in MDP's via knowledge transfer. In: *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*.

[Shoham et al. (2003)] Shoham, Y., Powers, R. & Grenager, T. (2003). Multi-agent reinforcement learning: A critical survey. Tech. rep., Stanford, CA, USA.

[Singh (1992)] Singh, S. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning* **8**, 323–339.

[Sobel (1971)] Sobel, M. (1971). Noncooperative stochastic games. *The Annals of Mathematical Statistics* **42**(6), 1930–1935.

[Spaan & Melo (2008)] Spaan, M. & Melo, F. (2008). Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In: *Proceedings of the 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems.

[Steenhaut et al. (1997)] Steenhaut, K., Nowé, A., Fakir, M. & Dirkx, E. (1997). Towards a hardware implementation of reinforcement learning for call admission control in networks for integrated services. In: *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications 3*. Lawrence Erlbaum.

[Sutton (1990)] Sutton, R. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: *Proceedings of the 7th International Conference on Machine Learning (ICML)*. Morgan Kaufmann.

[Sutton (1991)] Sutton, R. (1991). Planning by incremental dynamic programming. In: *Proceedings of the 8th International Conference on Machine Learning (ICML)*. Morgan Kaufmann.

[Sutton (1996)] Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: *Proceedings of the 8th International Conference on Advances in Neural Information Processing Systems (NIPS)*. MIT Press.

[Sutton & Barto (1998)] Sutton, R. & Barto, A. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press.

[Sutton et al. (1999)] SUTTON, R., PRECUP, D. & SINGH, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* **112**, 181–211.

[Sycara (1998)] SYCARA, K. (1998). Multiagent systems. *AI Magazine* **19**(2), 79–92.

[Taylor (2008)] TAYLOR, M. (2008). *Autonomous Inter-Task Transfer in Reinforcement Learning Domains*. Ph.D. thesis, University of Texas.

[Taylor & Stone (2006)] TAYLOR, M. & STONE, P. (2006). Cross-domain transfer for reinforcement learning. In: *Proceedings of the 24th International Conference on Machine learning (ICML)*. New York, NY, USA.

[Taylor & Stone (2009)] TAYLOR, M. & STONE, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* **10**(1), 1633–1685.

[Thathachar & Sastry (2004)] THATHACHAR, M. & SASTRY, P. (2004). *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers.

[Thorndike (1911)] THORNDIKE, E. (1911). *Animal intelligence; experimental studies*. New York, NY, USA: Macmillan.

[Thorndike & Woodworth (1901)] THORNDIKE, E. & WOODWORTH, R. (1901). The influence of improvement in one mental function upon the efficiency of other functions: Functions involving attention, observation and discrimination. *Psychological Review* **8**(6), 553–564.

[Thrun (1992)a] THRUN, S. (1992a). Efficient exploration in reinforcement learning. Tech. Rep. CMU-CS-92-102, Computer Science Department, Pittsburgh, PA, USA.

[Thrun (1992)b] THRUN, S. (1992b). The role of exploration in learning control. In: *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches* (WHITE, D. & SOFGE, D., eds.). New York, NY, USA: Van Nostrand Reinhold.

[Tsetlin (1962)] TSETLIN, M. (1962). On the behaviour of finite autmata in ranom media. *Automation and remote control* **22**, 1210–1219.

[Tsitsiklis (1994)] TSITSIKLIS, J. (1994). Asynchronous stochastic approximation and Q-learning. *Journal of Machine Learning* **16**(3), 185–202.

[Tuyls (2004)] TUYLS, K. (2004). *Learning in Multi-Agent Systems: An Evolutionary Game Theoretic Approach*. Ph.D. thesis, Vrije Universiteit Brussel.

[Tuyls & Nowé (2005)] Tuyls, K. & Nowé, A. (2005). Evolutionary game theory and multi-agent reinforcement learning. *Knowledge Engineering Review* **20**, 63–90.

[Tuyls et al. (2003)] Tuyls, K., Verbeeck, K. & Lenaerts, T. (2003). A selection-mutation model for q-learning in multi-agent systems. In: *Proceedings of the 2nd International Joint Conference on Autonomous agents and Multi-agent Systems (AAMAS)*. ACM New York, NY, USA.

[Verbeeck (2004)] Verbeeck, K. (2004). *Coordinated Exploration in Multi-Agent Reinforcement Learning*. Ph.D. thesis, Vrije Universiteit Brussel.

[Vlassis (2009)] Vlassis, N. (2009). *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence*. Morgan Claypool.

[Vrancx (2010)] Vrancx, P. (2010). *Decentralised Reinforcement Learning in Markov Games*. Ph.D. thesis, Vrije Universiteit Brussel.

[Vrancx et al. (2011)] Vrancx, P., De Hauwere, Y.-M. & Nowé, A. (2011). Transfer learning for multi-agent coordination. In: *Proceedings of the 3th International Conference on Agents and Artificial Intelligence*. Rome, Italy.

[Vrancx et al. (2007)] Vrancx, P., Verbeeck, K. & Nowé, A. (2007). Optimal convergence in multi-agent mdps. In: *Knowledge-Based Intelligent Information and Engineering Systems* (Apolloni, B., Howlett, R. & Jain, L., eds.), vol. 4694 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, pp. 107–114.

[Vrancx et al. (2008)] Vrancx, P., Verbeeck, K. & Nowé, A. (2008). Decentralized learning in markov games. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* **38**(4), 976–981.

[Watkins (1989)] Watkins, C. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, University of Cambridge.

[Watkins & Dayan (1992)] Watkins, C. & Dayan, P. (1992). Q-learning. *Journal of Machine Learning* **8**(3), 279–292.

[Weibull (1997)] Weibull, J. (1997). *Evolutionary game theory*. The MIT press.

[Weiss (1999)] Weiss, G. (ed.) (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. Cambridge, MA, USA: MIT Press.

[Wheeler Jr & Narendra (1986)] Wheeler Jr, R. & Narendra, K. (1986). Decentralized learning in finite markov chains. *IEEE Transactions on Automatic Control* **31**(6), 519–526.

[Wiering (1999)] WIERING, M. (1999). *Explorations in Efficient Reinforcement Learning*. Ph.D. thesis, Universiteit van Amsterdam.

[Williams (1992)] WILLIAMS, R. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Journal of Machine Learning* **8**(3), 229–256.

[Witten (1977)] WITTEN, I. (1977). An adaptive optimal controller for discrete-time markov environments. *Information and Control* **34**(4), 286–295.

[Wooldridge (1999)] WOOLDRIDGE, M. (1999). Intelligent agents. In: *Multiagent Systems: A modern approach to Distributed Artificial Intelligence* (WEISS, G., ed.). MIT Press, pp. 27–77.

[Wooldridge (2002)] WOOLDRIDGE, M. (2002). *An Introduction to Multi-Agent Systems*. John Wiley & Sons, Ltd.

[Wooldridge & Jennings (1995)] WOOLDRIDGE, M. & JENNINGS, N. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review* **10**, 115–152.

[Wunder et al. (2010)] WUNDER, M., LITTMAN, M. & BABES, M. (2010). Classes of multiagent q-learning dynamics with epsilon-greedy exploration. In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*.

Index

S

T

U

V

Author Index