Vrije Universiteit Brussel

Faculty of Science and
Bio-Engineering Sciences
Department of Computer Science

Local Coordination and Adaptive Strategies in Robot Soccer

Master thesis submitted in partial fulfilment of the requirements for the degree of
Master of Science in Applied Sciences and Engineering: Computer Science

Tim Brys

Promotor: Prof. Dr. Ann Nowé
Advisors: Dr. Yann-Michaël De Hauwere
Dr. Peter Vrancx

2010-2011

Vrije Universiteit Brussel

Faculteit Wetenschappen en
Bio-Ingenieurswetenschappen
Vakgroep Computer Wetenschappen

Lokale Coordinatie en Adaptieve Strategieën in Robot Voetbal

Master thesis ingediend met het oog op het behalen van de graad van
Master in de Ingenieurswetenschappen: Computerwetenschappen

Tim Brys

Promotor: Prof. Dr. Ann Nowé
Begeleiders: Dr. Yann-Michaël De Hauwere
Dr. Peter Vrancx

2010-2011

Abstract

In cooperative multi-agent systems, the performance of a group of agents, or team, often depends much more on the combined strategy of the team, than on that of a specific individual. In such cases, coordination is necessary to optimize the combined strategy, and local coordination is a way for a team to coordinate, while keeping communication and computational complexity low. Local coordination is especially effective when solving problems that have an inherent structure, favouring coordination between certain agents. We evaluate our contribution to local coordination on distributed bandit problems and distributed constraint optimization problems, as well as in a robot soccer context. In robot soccer, agents that play in neighbouring positions in a team's formation are more likely to need to coordinate, since their respective actions will influence each other's far more than those of agents in far removed positions. We investigate how a robot soccer team can adapt its team strategy to a specific opponent, by allowing agents to choose between different variations of behaviours and making the team learn which combinations of these variations work best, building a new strategy using only local coordination.

Contents

List of Figures

Chapter 1

Introduction

In this first chapter, we introduce some concepts, such as agents, reinforcement learning, multi-agent systems and robot soccer, that are crucial to understand the work presented in this thesis, as well as the problems of local coordination and adaptive strategies investigated in this thesis.

1.1 Agent

An **agent** is defined as any entity that perceives its **environment** through **sensors** and acts upon it through **actuators**. A robot is an example of an agent. Most robots are designed to carry out a specific task, and are equipped with the necessary actuators and sensors to achieve this. A cleaning robot, for example, needs to clean the room, and to do that, it has wheels and a cleaning device as actuators, and infrared sensors or a camera for sensors. During operation, the robot uses these sensors to perceive the state of the environment and decides what action to perform, e.g. driving to a patch of filth, or cleaning up filth it is standing on.

The way an agent behaves, i.e. how it uses its actuators, can be described by a function mapping all its past perceptions to an action. This is called the **agent function**. This function is realised by the internal **agent program** and can be as simple as random action selection, disregarding all information, and as ingenious as to use all previous information and inherent knowledge to calculate the action to be executed. Figure 1.1 illustrates a simple reflex agent. Such an agent maps its current perceptions to an action based on a set of built-in rules.

As anything that perceives and acts is an agent, irrespective of the way it behaves, we need to introduce the concept of **rationality**, in order to distinguish between agents that perform 'good', and those that don't. A **rational agent** is one that always chooses the action that it believes will cause it to be most successful. Success is defined as an objective **performance measure**, in terms of the

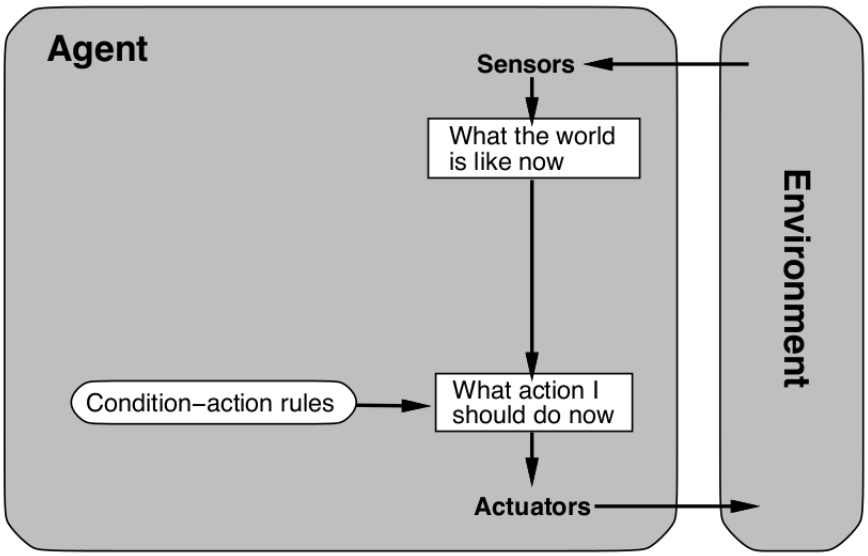


Figure 1.1: A simple reflex agent that maps its current perceptions to an action based on a set of built-in rules. [Russell and Norvig, 2003]

sequence of states the agent causes the environment to go through. The definition of a rational agent:

> For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has. [Russell and Norvig, 2003]

It is important to differentiate between a **rational agent** and a **perfect agent**. The perfect agent will maximize its *actual* performance, while a rational agent maximizes its *expected* performance. A perfect agent must be omniscient, but omniscience is impossible in reality.

To reason about the problem an agent faces, a formal description of the problem is needed. This is called the **task environment**. It is composed of the performance measure, the environment and the agent's sensors and actuators. A task environment can be characterized along these dimensions:

- **Fully observable vs. partially observable**

  An environment is fully observable if an agent's sensors can detect all the environment's relevant features. Otherwise, it is only partially observable. A feature is relevant if it influences the performance measure in some way.

- **Deterministic vs. stochastic**

From an agent's perspective, the environment is deterministic if its current state and the agent's action fully determine its next state.

- **Episodic vs. sequential**

  An environment is episodic if the agent's experience is divided into atomic episodes, which consist of perception and a single action. An action taken in one episode has no effect on subsequent episodes. In a sequential environment, actions can influence the environment, and change the effect of future actions.

- **Static vs. dynamic**

  An environment is dynamic if it can change while the agent is deciding an action to take.

- **Discrete vs. continuous**

  The discreteness/continuousness of an environment depends on that of its state, of the way time is handled and of the percepts and actions of the agent. An environment's state is continuous if there is an infinite possible number of different states. An environment is continuous if it changes smoothly with the flow of time, while discrete-time environments such as a chess environment change at (variable) discrete time steps. Percepts and actions are continuous if they can take on a range of continuous values.

- **Single agent vs. multi-agent**

  Only one agent operates in a single agent environment, while at least two agents exist in a multi-agent environment. The latter can be cooperative, where agents share the same goals, competitive, where agents have conflicting goals, individualistic, where agents have different, non-interfering goals, or a combination of these.

The **agent program**, implementing the mapping of percepts to actions, takes the current environment perceptions from the sensors as input and returns an action to the agent's actuators to be executed. Four basic kinds of programs exist, as described below and shown in Figure 1.2:

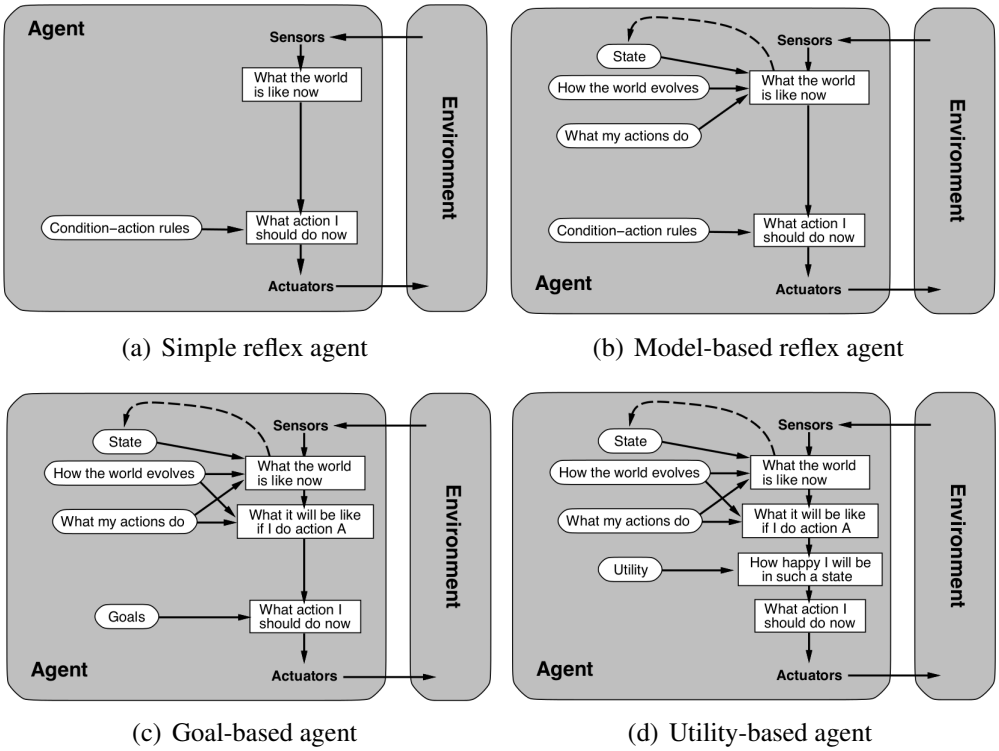
- **Simple reflex agents**. These agents select actions only based on the current percept. **Condition-action rules** map percepts to actions.

- **Model-based reflex agents**. Such agents keep track of the state of the world, based on their percept history. Using a **model** of the world, describing how it evolves, the agent can update its internal state of the world by combining the old state and current percepts. This helps resolve the

problem of partially observable environments, because the agent can keep track of (temporarily) unobservable aspects of the environment. Action selection is also calculated with condition-action rules, only now based on the internal state of the world.

- **Goal-based agents**. Goal-based agents incorporate their (long-term) **goals** into action selection. **Searching** and **planning** may be required to find what actions will lead to their goals.

- **Utility-based agents**. Utility-based agents use a **utility function** instead of goals to determine what actions to take. A utility function maps a world state to a real number, reflecting the extent to which the agent is 'happy' with that state of the world. Utility-based agents are superior to goal-based ones, in that they can handle multiple, possibly conflicting goals, by maximizing their utility function.



(a) Simple reflex agent

(b) Model-based reflex agent

(c) Goal-based agent

(d) Utility-based agent

Figure 1.2: Diagrams of the four kinds of agent programs identified. [Russell and Norvig, 2003]

Depending on the agent program, the agent relies more, or less, on predefined knowledge, i.e. knowledge not gained through perception, but defined by the designer of the agent. Agents that do not completely rely on prior knowledge must **learn** from their experiences in the environment to complement this knowledge and correct it if found to be incorrect.

The advantage that **learning agents** have over non-learning agents, is that the agent's creator is not required to equip the agent with the knowledge to be able to effectively handle every possible situation it might encounter. The creator can teach the learning agent, i.e. **supervised learning**, by providing examples of situations (environment states) and the appropriate action to take, allowing the agent to generalize to unknown situations, or the agent can learn independently, by observing and operating in its environment, which is **unsupervised learning**. Learning agents can thus operate in an unknown environment and become more competent than their initial knowledge alone would allow.

Figure 1.3 shows a model of a learning agent. It is necessary to distinguish between the **learning element** and the **performance element** in a learning agent. The learning element is responsible for making improvements, while the performance element handles action selection. The learning element receives feedback from the **critic** about the performance of the agent and modifies the performance element in order to improve the agent's performance. The last component, the **problem generator**, takes care of the need for information gathering or exploration. It suggests actions that will potentially lead to informative experiences, with which the learning element can effectively improve the agent's performance.

1.2 Reinforcement Learning

An agent can learn through interaction with its environment. Executing actions and observing the changes in the environment provides information about the consequences of actions. An external performance measure informs the agent about the quality of its behaviour and about how to achieve goals.

Consider animal conditioning. Animals can learn to associate certain actions with received reward or punishment. For example teaching your dog to shake hands (or paws) by taking its paw in your hand and rewarding it with food. After several such interactions, the dog will associate the shaking hands with food, and will choose to execute the shaking action when possible, in the expectation of positive reward. Associating desired actions with reward or punishment is called **reinforcement**.

Reinforcement learning (RL), is a computational approach to this type of learning. In RL, an agent must learn what actions to take in order to reach its goal(s). An important aspect is the numerical **reward** signal, which is the perfor-

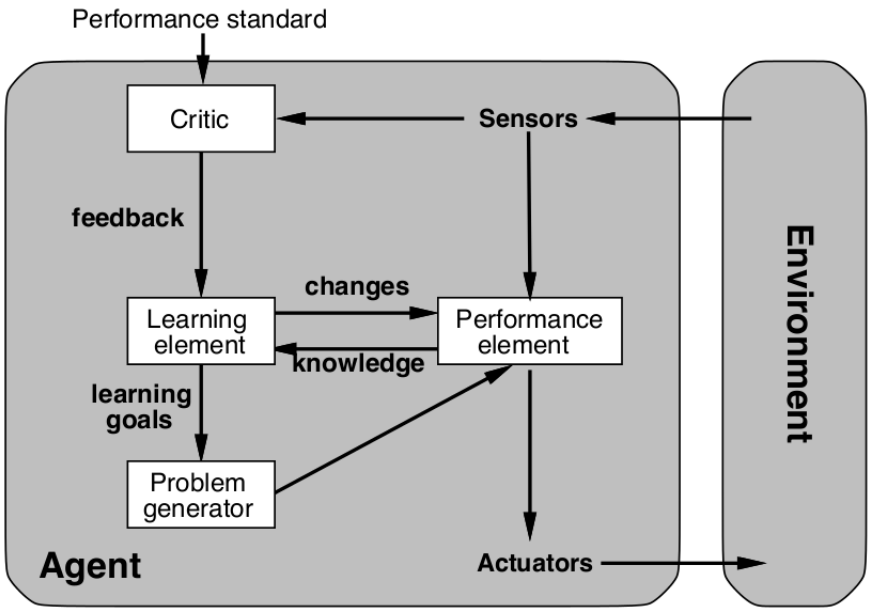


Figure 1.3: A general model of learning agents. [Russell and Norvig, 2003]

mance standard by which the agent's success is measured. Through **exploration** of the environment, the agent needs to learn how it can maximize the reward signal; to learn which actions yield the most reward. The effect of actions on the success of an agent can be obscured by delayed reward; reward that does not follow immediately after the execution of an action.

To obtain as much as possible reward, the agent must select actions it expects to perform well, i.e. **exploiting** its knowledge, yet to find such actions, it needs to select actions it has not executed before, i.e. **exploring**. Balancing between these two conflicting needs is necessary, because the agent will fail if either one is pursued exclusively. The agent must explore by trying various actions, but he must also progressively prefer those actions that are expected to perform well. Different solutions to this problem are described in Chapter 2.

An RL system includes three or four of these elements:

- A **policy** determines the behaviour of an agent. It defines which actions an agent takes, given a perceived state. Specifically, it maps environment states to probability distributions over actions. The policy is the performance element in a learning agent's architecture as described before, see Figure 1.3.

- A **reward function** defines the goals of an agent. It maps environment states and actions to a real number, reflecting the desirability of the agent being in that state. A RL agent's objective is to maximize the total reward received in the long term.

- A **value function** defines what is good in the long run. Given an environment state, the value function gives the amount of reward an agent can accumulate continuing from that state on. In contrast to the reward function, which gives the immediate desirability of a state, the value function gives the long-term desirability of a state. A large part of the RL problem is estimating the value function from the reward function.

- Possibly a **model of the environment**, which simulates the behaviour of the environment. Agents incorporating a model of the environment can plan ahead using the model, and can even simulate exploration on the model, reducing the need for real-world exploration. The success of this planning and learning depends on the accuracy of the model.

Chapter 2 gives a more detailed description of RL.

1.3 Multi-Agent Systems

Systems involving the operation of multiple autonomous agents, are called **Multi-Agent Systems** or MASs. MASs have many advantages over single agent systems, when solving the same problem. In MASs, agents can compute in parallel, there is often no single point of failure, the addition of more agents helps to better solve the problem and they are easier to maintain because of the modularity [Sycara, 1998]. Some problems even require multiple agents to be solved, e.g. network routing.

Agents in a MAS operate and interact, each pursuing its own goals. Different agent's goals can be the same, requiring them to cooperate in a team, achieving their common goals as good as possible, or they can have conflicting goals, in which case they compete and need to balance their own need to satisfy their goals and the need for the whole system to behave optimally. In any case, agents need to **coordinate** for the whole system to be effective.

Coordination involves sensing other agents, reasoning about them and acting accordingly. By communicating, agents can reach decisions together, or share knowledge that can help improving the performance of the group. A common problem with MASs is the exponential increase in complexity as the number of agents in the system increases. This in turn impacts coordination, as it quickly becomes impractical, or even impossible, to coordinate with all other agents, i.e. **global coordination**.

Local coordination, i.e. allowing an agent to coordinate with only a subset of the other agents, relieves this problem of complexity, although global performance suffers, unless special techniques are used to approach the performance of

global coordination. Such techniques, and local coordination in general, are discussed in Chapter 3, where we also explain and evaluate our contribution to local coordination.

1.4 RoboCup

RoboCup [Kitano et al., 1997] is an international project, developed to provide a test bed to evaluate research in the fields of Artificial Intelligence and robotics, specifically multi-agent Systems [Kalyanakrishnan et al., 2007,Devlin et al., 2010, Bowling et al., 2004], reinforcement learning [Stone et al., 2005,Kalyanakrishnan et al., 2007, Takahashi et al., 2005, Devlin et al., 2010], robot motion planning [Sherback et al., 2006,Johansson and Saffiotti, 2002], vision [Jamzad et al., 2002], etc.

RoboCup is a contraction of "Robot Soccer World Cup", which indicates the major domain RoboCup research is performed in, i.e. the game of soccer. RoboCup organises a world cup competition every year, where researchers from around the globe can enter their teams. This competition provides a platform for the researchers to evaluate their findings in a realistic setting. Besides the world cup, there are various other RoboCup Soccer competitions during the year, such as the RoboCup German Open, being the unofficial European Championship and the RoboCup IranOpen.

The ultimate goal of the RoboCup Soccer project is stated as follows [RoboCup.org, 2011]:

> By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup.

RoboCup Soccer is divided into different leagues, each with its specific challenges. For example, in the Humanoid League, "dynamic walking, running, and kicking the ball while maintaining balance, visual perception of the ball, other players, and the field, self-localization, and team play are among the many research issues investigated". On the other hand, the Small Size League "focuses on the problem of intelligent multi-robot/agent cooperation and control in a highly dynamic environment with a hybrid centralized/distributed system". A complete overview is given in Table 1.1.

The league of interest for the research presented in this thesis, is the Middle Size league. Until recently, the outcome of games in this league depended largely on the hardware capabilities of the robots. Only now are teams, such as TechUnited, beginning to investigate learning and team strategies.

League	Characteristics	Research Focus
Humanoid	Autonomous robots with a human-like body plan and human-like senses. Teen Size, Kid Size and Adult Size sub-leagues	Dynamic walking, running, and kicking the ball while maintaining balance, visual perception of the ball, other players, and the field, self-localization, and team play
Middle Size	Robots of no more than 50 cm diameter, teams of up to 6 robots, use wireless networking to communicate	Full autonomy and cooperation at plan and perception levels
Simulation	Independently moving software players (agents) on a virtual field inside a computer. 2D and 3D sub-leagues	Artificial intelligence and team strategy
Small Size	Robots of no more than 18 cm diameter, teams of 5 robots, centralized decision making	The problem of intelligent multi-robot/agent cooperation and control in a highly dynamic environment with a hybrid centralized/distributed system
Standard Platform	All teams use identical (i.e. standard) robots, robots are fully autonomous	The teams concentrate on software development only

Table 1.1: The characteristics of the five RocoCup Soccer leagues, data from [RoboCup.org, 2011]

Current and future research in various leagues includes Opponent Modelling [Akiyama and Shimora, 2010, Marian et al., 2010, Reis et al., 2010], Motion Planning [Bai et al., 2010], Online Game Analysis and Adaptive Strategies [Akiyama and Shimora, 2010, Marian et al., 2010], Communication Strategies [Reis et al., 2010] and Self Localization [Röfer et al., 2010, Acar et al., 2010].

Soccer, although the main Robocup competition domain, is not the only one. RoboCup Rescue is a competition that focuses on the management of disaster situations. These generally involve very large numbers of heterogeneous agents in a hostile environment.

The intention of the RoboCup Rescue project is to promote research

> and development in this socially significant domain at various levels involving multi-agent team work coordination, physical robotic agents for search and rescue, information infrastructures, personal digital assistants, a standard simulator and decision support systems, evaluation benchmarks for rescue strategies and robotic systems that are all integrated into a comprehensive systems in future. [RoboCup.org, 2011]

RoboCupJunior is another RoboCup initiative. It sponsors robotic events around the world and includes soccer and rescue, as well as robot dance competitions. Its main focus is education and getting children interested in robotics.

1.4.1 The RoboCup 2D Simulator

From the official soccer server manual [Chen et al., 2002]:

> The RoboCup simulator league is based on the RoboCup simulator called the soccer server [...], a physical soccer simulation system. All games are visualised by displaying the field of the simulator by the soccer monitor on a computer screen. The soccer server is written to support competition among multiple virtual soccer players in an uncertain multi-agent environment, with real-time demands as well as semi-structured conditions.
>
> One of the advantages of the soccer server is the abstraction made, which relieves the researchers from having to handle robot problems such as object recognition [...], communications, and hardware issues, e.g., how to make a robot move. The abstraction enables researchers to focus on higher level concepts such as co-operation and learning.

1.4.2 Complexity of Soccer

The domain of simulated RoboCup soccer is a good test bed for reinforcement learning techniques, because of its sheer complexity, mirroring the real world, making unsupervised learning hard. Some of its properties are a continuous state space, continuous action set and multiple agents.

A characterization of the simulated soccer task environment:

- **Partially observable**. An agent's sensors cannot detect all relevant information about the environment at any time. For example, most of the time, not all players are in view, and thus their positions and actions are uncertain. The same goes for the decision making of other agents. On top of that, the simulator injects random noise in all basic perceptions.

- **Stochastic**. The evolution of the environment cannot be fully determined by its current state and the actions of the agents. The simulator injects random noise in all movements.

  On top of that, from the perspective of a single agent, the environment is stochastic because its next state is not only influenced by its own actions.

- **Sequential**. An agent executes a sequence of actions during the game. Each action has an effect on the environment and previous decisions have the potential to influence the effect of subsequent decisions.

- **Dynamic**. The environment can change while an agent is still reasoning about its next action. Even when not a single agent acts, the environment can change because of the operation of the physical laws implemented.

- **Continuous**. Although time is handled discretely, the state of the environment and agent's actions are not discrete. The state of the environment is continuous because the positions and orientations of all agents and the ball can take on continuous values within a specific range. An agent's actions are also continuous since he can choose an infinite number of actions by e.g. varying the strength of kicking a ball, his turning angle or his dashing speed.

- **Multi-agent**. There are 22 agents operating simultaneously in a soccer match, divided into two adversarial teams of 11 players. These 11 players cooperate to defeat the competing team.

In this thesis we report on the application of RL techniques in this context.

1.4.3 Adaptive Strategies

In real world sports, we recently saw the advent of professional statistical analysis as a means to improve the team performance. This method was popularized by [Lewis, 2003] and is currently applied on the highest level of such sports as baseball, basketball and *soccer*. A real example of such analysis and adaptation, is the player Lukaku, who debuted during the 2009-2010 season for RSC Anderlecht. Because he is a very large and powerful player, he could easily outmuscle his defenders, getting much scoring opportunities. In the next season, all teams knew about the play-style of Lukaku, and they adapted their strategy to better counter him, making defenders mark him from further away, avoiding close contact where he has the advantage. This lead to a period where Lukaku scored less frequently, until he himself learned to counter the different defensive style.

In short, a team using one specific strategy might consistently win against one team, yet lose against another, simply because the chosen strategy is perfect to counter the former opponent, but fails against the latter. This problem has previously been addressed in robot soccer.

One way is to use opponent modelling techniques [Iglesias et al., 2009,Fathzadeh et al., 2006, Stone et al., 2000, Riley and Veloso, 2002]. Teams using such techniques try to build a model of the opponent's behaviours, which can later be used to predict opponent's actions and to react accordingly. This can be done either before the actual game or during the game, in the latter case often with the help of a coach agent, who observes and plans. Opponent modelling can effectively improve the performance of a team, even over the course of one game.

A different approach to opponent modelling is, instead of observing the opponent's behaviours, making a team observe its own effectiveness to adapt to the opponent. In [Bowling et al., 2004], *plays* are proposed, which are team plans, a description of a course of action, applicable in different situations. When in a specific situation, the team selects probabilistically among the possible plays, and tries to execute it. When the play ends, having reached one of its predefined end states, its success is evaluated and the selection probabilities are changed accordingly. This allows the team to find specific plays that are suited to counter a specific opponent, effectively adapting the team's strategy to the opponent.

Such an approach, i.e. adaptation through self-evaluation, is interesting from a reinforcement learning perspective, since we can simply treat the opponent as part of the environment and learn a strategy that gets the most reward out of the environment. This assumes that the opponent's strategy is static and does not in turn adapt to our behaviour.

In this thesis we present such a method for adaptation, where agents select and evaluate variations of behaviours against a specific opponent, in order to build a combined strategy, suited to counter the opponent.

1.5 Outline

In the next chapter, reinforcement learning is further elaborated. All basic concepts of RL are formally defined, and several solution methods are described. The chapter after that describes our contribution to local coordination in multi-agent settings, which will later on be applied to robot soccer. This chapter is followed by one describing the method proposed for adaptive strategies. The behaviour variations are described in detail, followed by the various RL methods employed to learn a team strategy. This method is evaluated in the fifth chapter, with several experiments, comparing the different RL methods. Lastly, there is a chapter discussing the research presented in this thesis and proposing future work.

Chapter 2

Reinforcement Learning

All work described in this thesis is based on reinforcement learning. To familiarize the reader with its concepts, and give the methodologies described in Chapters 3 and 4 a theoretical foundation, we define the basic concepts of reinforcement learning and describe the various solution methods used.

2.1 Problem Setting

The reinforcement learning (RL) problem is for an agent to learn through interaction, in order to achieve a goal. An agent interacts with its environment at discrete time steps, at each step t executing an action a_t, given the environment's current state s_t. The next time step, the agent perceives the environment's new state s_{t+1} and receives a numerical reward r_{t+1}. Figure 2.1 illustrates these agent-environment interactions over time.

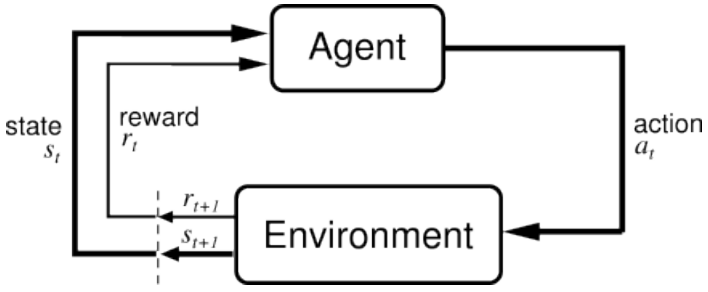


Figure 2.1: The agent-environment interaction in reinforcement learning [Sutton and Barto, 1998]

The agent keeps a policy π, which is a mapping of states to selection probabilities for each action. $\pi(s, a)$ is the probability of selecting action a in state s.

This policy may change over time, thus we denote the policy at a specific time step as π_t. The way a policy is changed based on experiences is specified by the specific RL method used. Since the reward signal is a formalization of the goal or purpose of the agent, he needs to change his policy in such a way to maximize the reward received over time, resulting in the best performance in function of the goals implemented by the reward signal. The reward signal can in effect also be a cost signal, in which case the agent needs to minimize the signal in order to perform optimally.

We need to formally define the way a reward signal is 'maximized'. Let the **expected return** R_t be a function of the reward sequence. If the operation of an agent in the environment reaches an end in a terminal state, with T denoting the final time step, we define the expected return R_t at time step t as the accumulation of expected rewards between t and T:

$$R_t = r_{t+1} + r_{t+2} + ... + r_T \tag{2.1}$$

Tasks in an environment where interaction ends in a terminal state and the environment is subsequently reset to a starting state, are called episodic. The agent-environment interaction is broken up into a sequence of episodes, in which an agent needs to maximize its reward. Maximizing the expected return as defined in Equation 2.1 equals performing optimal in such a context.

In many cases though, the agent does not operate within distinct episodes, but continuously. These are continuing tasks and require a different formulation for the expected return, since final time step T might equal ∞, and consequently the return might too. To resolve this problem, we need a concept that reduces the infinite return to a finite one. This concept is called **discounting**. With discounting, an agent tries to maximize the sum of the discounted future rewards [Sutton and Barto, 1998]:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.2}$$

γ is the discount rate, with $0 \leq \gamma \leq 1$. This discount rate defines the current importance of future rewards. Reward received k steps in the future is counted only γ^{k-1} times its actual worth. Equation 2.1 is a special case of this return formulation, where $\gamma = 1$. Setting $\gamma < 1$ renders the infinite sum of rewards finite, provided the reward sequence is bounded, i.e. not infinite. As γ approaches 1, the agent becomes more far-sighted, while setting $\gamma = 0$ makes the agent greedy, making him always select the action he expects to return the highest immediate reward. Most RL problems require the agent to do at least some long-term planning, since to achieve its goal, it may be necessary for the agent to go through some states that have a lower immediate reward.

2.1.1 N-Armed Bandits

An example of an episodic task, is the **n-armed bandit problem**. It features only one state, and after one action execution, the episode is over and reward is received. The problem is named after old slot machines, which had a big lever or arm on the side. With the n-armed bandit, instead of one lever, there are n levers or actions an agent can choose from. One episode encompasses one action execution and is called a play.

Every action or arm on the bandit has an associated reward distribution. If the agent knows the reward distributions of these actions, action selection is trivial and amounts to always selecting the actions with the highest mean reward. In the case where an agent does not know the distributions, he must estimate them, and act upon those estimates. Below, we describe a way to solve such a problem.

Action Value Estimation

Let us denote the actual value of action a, which is the mean of its reward distribution, as $Q^*(a)$, and the agent's estimate of $Q^*(a)$ at time t as $Q_t(a)$. A straightforward way to approximate $Q^*(a)$ is by averaging all the received reward for playing action a k times:

$$Q_t(a) = \frac{r_1 + r_2 + ... + r_k}{k} \tag{2.3}$$

As k goes to ∞, $Q_t(a)$ will converge to $Q^*(a)$. This is called the sample-average method for estimating the reward distribution of actions.

A problem that arises with this method is that it requires memory to store all received rewards. Each additional reward requires more memory and computing $Q_t(a)$ takes longer and longer. The solution to this problem is an incremental update formula for $Q_t(a)$, with which we only need to store $Q_t(a)$ in memory. Let $Q_t(a)$ be the average reward at play t, and r_{t+1} the newly received reward, then the average of $t + 1$ plays can be calculated as follows [Sutton and Barto,

1998]:

$$\begin{aligned} Q_{t+1} &= \frac{1}{t+1}\sum_{i=1}^{t+1} r_i \\ &= \frac{1}{t+1}\left(r_{t+1} + \sum_{i=1}^{t} r_i\right) \\ &= \frac{1}{t+1}\left(r_{t+1} + tQ_t + Q_t - Q_t\right) \\ &= \frac{1}{t+1}\left(r_{t+1} + (t+1)Q_t - Q_t\right) \\ &= Q_t + \frac{1}{t+1}\left[r_{t+1} - Q_t\right] \end{aligned} \tag{2.4}$$

This averaging method assumes the environment is stationary, i.e. the reward distribution for each action does not change. This may not be the case though, and a bandit can very well change over time. In that case, instead of averaging over all received reward, it makes more sense to make recent rewards outweigh rewards received a long time ago. These old rewards possibly represent a reward distribution that is no longer employed by the bandit. A commonly used way of doing this is using a constant step-size parameter. The update rule 2.4 is then modified to:

$$Q_{t+1} = Q_t + \alpha\left[r_{t+1} - Q_t\right], \tag{2.5}$$

with $0 < \alpha \leq 1$. This effectively gives larger weight to more recent rewards, keeping the estimates closer to the reward distributions at that time.

Action Selection

The problem an agent faces each play, is which action to choose. He can use the knowledge contained in the action value estimates to intelligently select an action. The most basic action selection method is greedy action selection, where the agent selects one of the actions with the highest associated reward estimate. This method will always maximize the expected reward, given the current estimates, but the agent will never sample actions he believes to be inferior, although they might actually be better. He needs to explore the actions space to find good estimates of all reward distributions.

A simple method that addresses this issue, is behaving greedily most of the time, but once in a while randomly selecting a non-greedy action. ϵ-greedy action selection is such a method. With probability $0 \leq \epsilon \leq 1$, a non-greedy action is chosen at random. Otherwise the greedy action is executed (if there are several

greedy actions, i.e. with the same expected return, one of these is randomly chosen). As the number of plays increases to infinity, the number of times each action is chosen also converges to infinity, ensuring the convergence of every $Q_t(a)$ to $Q^*(a)$, which in turn leads to the probability of choosing the optimal action converging to greater than $1 - \epsilon$.

An alternative action selection method that makes the agent explore, is **softmax action selection**. It is similar to ϵ-greedy action selection, but instead of choosing with a fixed probability the greedy action or completely random among the non-greedy actions, it chooses among all actions with a probability depending on their estimated value. The greedy action still has the highest probability of being chosen, but all actions will have selection probabilities that reflect their relative expected values.

A **Gibbs** or **Boltzmann** distribution is most commonly used for softmax action selection. Action a has a probability of being chosen in play t of:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}} \tag{2.6}$$

τ is the **temperature** parameter, $\tau \geq 0$, which controls the effect differences in action values have on relative probabilities. A high temperature will cause action selection to be almost completely random, while a temperature approaching zero makes the softmax selection greedy.

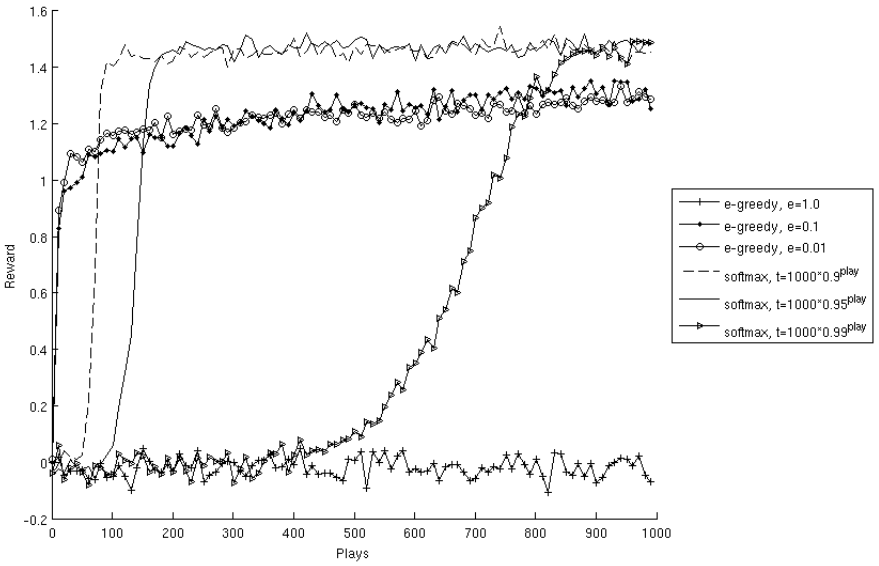


Figure 2.2: Comparison of ϵ-greedy and softmax action selection

Figure 2.2 shows a comparison of an agent solving a bandit problem using ϵ-greedy and softmax action selection with different parameters. The bandit problem has 10 actions and the mean rewards of every action are drawn from a normal distribution with mean 0 and standard deviation 1. As expected, ϵ-greedy with $\epsilon = 1$ behaves completely random. A lower ϵ initially yields better results than softmax, but is quickly outperformed. Softmax with $\tau = 1000 * 0.99^{play}$ converges very slowly because of the slow temperature decrease, making the agent explore longer.

Another simple way of making the agent explore, is by overestimating the initial action value estimates. The **optimistic initial values** will decrease as the agent tries actions, since the actual reward distributions are lower than the initial estimates. But, each time an estimate is decreased, another action becomes the greedy one, raising its probability of selection. This way, the agent will try each action several times during the initial plays, before its estimates approach the actual values. This simple method of optimistic initial values can even be successfully combined with greedy action selection, outperforming other action selection methods [Sutton and Barto, 1998], as shown in Figure 2.3.

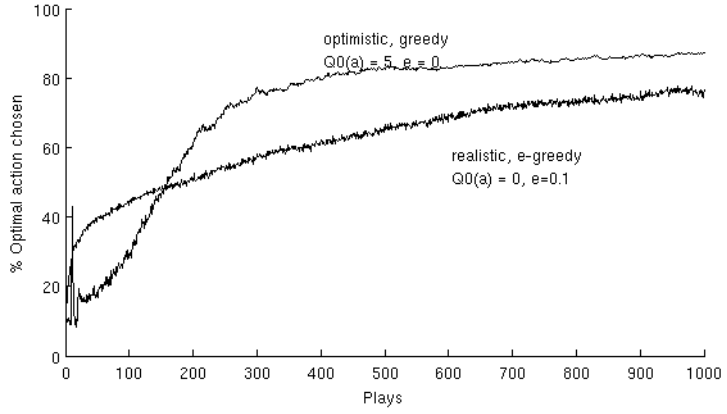


Figure 2.3: Greedy action selection with optimistic initial values compared to ϵ-greedy action selection

2.2 Markov Decision Problems

As part of the agent-environment interactions, an agent perceives at any time t the state of the environment. This state signal represents all the information available to the agent, from immediate sensations to models of the environment built from

the agent's history of perceptions. The new state of the environment at time $t+1$ after taking action at time t, can depend on everything between all the previous states and actions and only the last state and action. A state signal where the next state only depends on the previous state and the agent's action has the **Markov property**.

This property's one-step dynamics allow the prediction of the next environment state and its associated reward given the current state and action only. Even all subsequent states and expected rewards can be predicted, given the current state, action and policy. This makes state information with the Markov property ideal for decision making, as the best action policy in function of a Markov state is as good as the best policy in function of the complete state history.

From here on we will treat the RL state signal as being Markovian. Even if it actually only approximates the Markov property, it still helps to think of it as Markovian, since it still allows for predictions, albeit less correct ones.

2.2.1 Markov Decision Processes

An RL task whose state signal has the Markov property is called a **Markov Decision Process** (MDP), or a finite MDP if its state and action sets are finite. A finite MDP is defined by its states S, actions A and the dynamics of the environment, which are specified as state transition probabilities $\mathcal{P}$, and expected reward $\mathcal{R}$. This four-tuple $(\mathrm{S},\mathrm{A},\mathcal{P},\mathcal{R})$ defines an MDP.

Given a state s and action a, the probability of the environment transitioning to a specific state s' is [Sutton and Barto, 1998]:

$$\mathcal{P}^a_{ss'} = Pr\left\{s_{t+1} = s' | s_t = s, a_t = a\right\} \tag{2.7}$$

The expected reward, given the current state s and action a, and the next state s' is [Sutton and Barto, 1998]:

$$\mathcal{R}^a_{ss'} = E\left\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\right\} \tag{2.8}$$

2.2.2 Value Functions

The task an RL agent faces, is to find a policy π that maximizes long term reward as defined in Equation 2.2. In an MDP setting, we define the value of each state s under policy π, $V^\pi(s)$, as the expected return R_t starting from that state, following policy π [Sutton and Barto, 1998]:

$$V^\pi(s) = E_\pi\left\{R_t | s_t = s\right\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \tag{2.9}$$

The function $V^\pi(s)$ is called the **state-value** function for policy π.

Similarly, we define the **action-value** function $Q^\pi(s,a)$ for policy π, which is the expected return starting from state s, taking action a, and from then on following policy π [Sutton and Barto, 1998]:

$$Q^\pi(s,a) = E_\pi\left\{R_t | s_t = s, a_t = a\right\} = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\middle| s_t = s, a_t = a\right\} \tag{2.10}$$

These value functions $V^\pi(s)$ and $Q^\pi(s,a)$ can be estimated from experience. The agent follows its policy π for a certain time, while keeping track of the returns that followed after visiting a state (V), or executing a specific action in a state (Q). These averages will converge to the actual value functions, $V^\pi(s)$ and $Q^\pi(s,a)$ as the number of times the state is visited or the action executed in each state approaches infinity. Such estimation methods are called Monte Carlo methods.

A property of these value functions is that they can be rewritten to a recursive form [Sutton and Barto, 1998]:

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{R_t | s_t = s\right\} \\ &= E_\pi\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}\middle| s_t = s\right\} \\ &= \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}\middle| s_{t+1} = s'\right\}\right] \\ &= \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right\} \end{aligned} \tag{2.11}$$

This is the Bellman equation for V^π. It expresses the fact that the value of the current state depends on the values of all potential next states and their expected rewards. The contribution of each next state to the value is weighted by the probability of selecting an action that can lead to that state, according to π, and the probability of the environment actually transitioning to that state, according to $\mathcal{P}$.

Since an agent wants to maximize its long term reward when solving an RL problem, the agent needs to find a policy that achieves this objective. For finite MDPs, an optimal policy is defined as the policy whose expected return for each state is at least as high as that of any other policy. Formally, it is true for an optimal policy π^*, that $\forall\pi : \pi^* \geq \pi \Leftrightarrow \forall\pi, s \in S : V^{\pi^*}(s) \geq V^\pi(s)$.

There is always an optimal policy for a specific problem, but it is possible that there exist several optimal policies, yet they will all share the same optimal state-value function, $V^*(s) = \max_\pi V^\pi(s)$, $s \in S$. Optimal policies also share the same optimal action-value function, $Q^*(s,a) = \max_\pi Q^\pi(s,a)$, $s \in S, a \in A$.

2.3 Multi-Agent Reinforcement Learning

In the theory of Markov Decision Processes, the environment is assumed to be static. The introduction of other agents in the environment makes it uncertain and invalidates the assumption of a static environment. Methods for solving MDPs are no longer guaranteed to find the optimal policy, since an agent's decision influences the effect the other agents their decisions have on the environment, and vice versa. A different framework is necessary to reason about multi-agent systems.

2.3.1 Game theory

Game theory was developed to reason about decision making in multi-agent settings. Consider a simple game between two agents, consisting of a single move, taken simultaneously. The game defines a **pay-off matrix** that gives the reward the agents can receive for each combination of actions from both players. A classic example of such a game is the **prisoner's dilemma**:

> Two alleged burglars are caught near the scene of a burglary. If they confess to the crime, they will serve five years in prison for burglary. If both hold out, they each get one year of prison time for possession of stolen property. Each is offered a deal by the police: if they confess the other to be the leader of a burglars gang, they go free and the other goes for 10 years to prison. Their dilemma is whether to confess or refuse.

	A testifies	A refuses
B testifies	A=-5, B=-5	A=-10, B=0
B refuses	A=0, B=-10	A=-1, B=-1

Table 2.1: Payoff matrix for the prisoner's dilemma

Table 2.1 summarizes the pay-off for each prisoner for each combined action. When A thinks about which action to take (testifying or refusing), he knows the outcome of his action depends on the action of B as well. There are two scenarios for A: either B testifies, or B refuses. If B testifies, A gets 5 years for testifying as well, and 10 years for refusing to testify. Testifying is the best option in that scenario. If B refuses, A goes free if he testifies, and gets 1 year for also refusing to testify. Again, testifying is the best strategy (strategy is the game theory equivalent of a policy). Testifying is thus the dominant strategy in this game. B can reason

in the same way and employ the same strategy. Both burglars choosing to testify results in each getting 5 years.

Agents A and B are stuck in a **Nash equilibrium**, which means that neither one of them has a benefit of changing its strategy, provided the other does not change. The dilemma is that there is a better outcome when each agent refuses to testify. In order to escape such a suboptimal equilibrium, agents need to **coordinate** to reach another, better equilibrium. The game previously described is an example of a single-stage game, i.e. the game is only played once. Games that are repeated multiple times, are called **multistage games**.

2.3.2 Markov Game

Extending game theory to MDP-like environments results in **Markov games** [Littman, 1994]. As an MDP is characterized by a four-tuple, a Markov game is characterized as a five-tuple $(\mathrm{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, k)$. Besides the addition of k, which denotes the number of agents in the game, the main difference between MDPs and Markov games is that in MDPs A is the action set of the single agent, while in Markov games, $\mathcal{A}$ is a collection of action sets, one for each agent: $\mathcal{A} = A_1, A_2, ..., A_k$. The state transition probability function $\mathcal{P}$ is a function of the state and all actions, $S \times A_1 \times ... \times A_k$, instead of only one action, $S \times A$. The same applies to the reward function $\mathcal{R}$. The set of states S stays the same.

As with MDPs, in Markov games, an agent needs to find a policy π that maximizes long term reward. For MDPs, different solution methods, such as Q-learning, exist. When applying these methods in the setting of a Markov game, ignoring the existence of other agents, we call the agent an **independent learner** (IL). In contrast, agents learning the values of their own actions in combination with those of other agents are called **joint action learners** (JAL). Although ILs have less knowledge available, and incorrectly assume the environment to be static, they have been shown to perform as well as JALs in specific settings [Claus and Boutilier, 1998]. Since ILs have been treated in previous sections, we will concentrate on JALs from here on.

2.3.3 Distributed N-armed Bandits

Recall the n-armed bandit problem. An agent has a set of actions to choose from, and after the execution of an action, the play is over and reward is received. The agent must learn over multiple plays which actions to take in order to maximize its reward. A **distributed n-armed bandit problem** is its extrapolation to a multi-

agent setting[1]. In such a problem, every agent must decide which action to execute and reward depends on the combination of all actions. When the reward for each agent is drawn from the same distribution, the game is **cooperative**. We will only focus on such cooperative or **coordination games**, because the problem domains in this thesis, such as robot soccer, are of this nature.

When applying the solution method described in section 2.1.1, estimating rewards only for the agent's own actions $a_0, a_1, ..., a_n$, the agent is an **independent learner** or IL [Claus and Boutilier, 1998]. A **joint action learner** or JAL on the other hand, will learn reward estimates for the combined actions of all agents involved: $< a_0, b_0, c_0, ... >, ..., < a_3, b_1, c_4, ... >, ... < a_n, b_n, c_n, ... >$. Through communication or observation, the agent can know the other's actions and update its estimates accordingly.

When selecting actions, JALs need to take more care than ILs. ILs only need to balance exploration and exploitation, depending on the estimates of their own actions, in order to maximize reward. JALs on the other hand need to maintain beliefs about the action selection of other agents, in order to achieve their own goals of exploration and exploitation, especially the latter.

A simple way of implementing a model of other agents, is to keep a probabilistic model of these agents' action selection, by using empirical distributions, i.e. counting the number of times C each action has been chosen by each agent. The probability of agent j selecting action a_j is:

$$Pr_{a_j} = \frac{C^j_{a_j}}{\sum_{b_j \in A_j} C^j_{b_j}} \tag{2.12}$$

Using their estimates for joint actions and their probabilistic models of other agents' action selection, agents can evaluate the expected value for selecting a specific action:

$$EV(a_i) = \sum_{a_{-i} \in A_{-i}} Q(a_{-i} \cup a_i) \prod_{j \neq i} Pr^i_{a_{-i}[j]} \tag{2.13}$$

The expected value EV for action a_i is the sum of all estimated rewards of joint actions including action a_i, multiplied by the probability of each joint action occurring. In this formula, a_{-i} is a set representing a joint action of all agents but the agent itself.

Instead of using the estimates Q, the agent can now use EV in its action selec-

[1]We use the term distributed bandit problem as in [Claus and Boutilier, 1998], although it can also be called a normal-form game.

tion. For example with softmax action selection using a Boltzmann distribution:

$$\frac{e^{EV_t(a)/\tau}}{\sum_b^n e^{EV_t(b)/\tau}} \tag{2.14}$$

Different ways of incorporating the beliefs about other agents exist. These include:

- *Optimistic Boltzmann (OB)*, where an agent always expects the others to select the optimal actions with respect to his own actions, using $MaxQ(a_i) = max_{a_{-i}} Q(a_{-i} \cup a_i)$ in Boltzmann selection as a value for a_i,
- *Weighted Optimistic Boltzmann (WOB)*, where the likelihood of each optimistic joint action's occurrence is taken into account, $MaxQ(a_i)Pr(A_i)$, where A_i is the optimal match for a_i
- *A Combined strategy*, which balances between OB and regular Boltzmann using EV, $CS(a_i) = \rho MaxQ(a_i) + (1 - \rho)EV(a_i)$, $0 \leq \rho \leq 1$, using $CS(a_i)$ in Boltzmann selection as a value for a_i.

Of these, the combined strategy has been shown to be the most flexible one, biasing exploration with empirical beliefs about other agents towards potentially better joint actions [Claus and Boutilier, 1998]. Optimistic Boltzmann on the other hand is reported to be no good in all but simple games.

Sliding Belief Windows

One problem with the empirical belief system, is that beliefs are hard to change. When agent B suddenly changes from using action b_0 for a long time to b_1, agent A will still have a firm belief about B selecting b_0. It may take a long time for agent A to overcome this belief and to start selecting actions appropriate for the actual situation. **Sliding belief windows** are a way of mitigating this problem. Only the last n observations are taken into account to calculate the action selection probabilities of other agents. This way, outdated data is disregarded in favour of more recent data allowing for faster adaptation to changes in the behaviour of other agents.

2.3.4 Experiments

Figure 2.4 shows a comparison of ILs and JALs on the prisoner's dilemma problem previously described. The JALs use the combined strategy with softmax action selection, without sliding belief windows. Both types of learners manage to

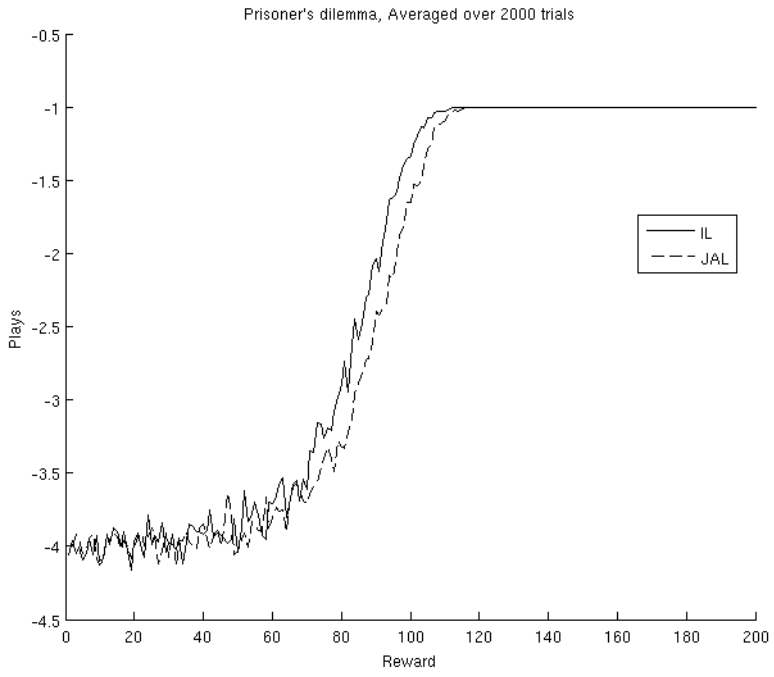


Figure 2.4: ILs and JALs learning a strategy for the prisoner's dilemma. Both types of learners find the optimal strategy.

find the optimal strategy, which is for both agents to refuse confessing, resulting in a cost of -1, one year in prison.

A more difficult problem, is the instance of the climbing game defined in Table 2.2, as found in [Claus and Boutilier, 1998]. Learners can quickly find the equilibrium $a_1 - b_1$, by starting from the strategy $a_2 - b_2$ and climbing up when first B discovers that changing to b_1 yields more reward, and then A discovering that changing to a_1 yields even more reward. When this equilibrium is reached, both agents need to change their action to reach a better equilibrium. Only one agent changing its strategy results in a very large penalty (-30).

	a_0	a_1	a_2
b_0	11	-30	0
b_1	-30	7	6
b_2	0	0	5

Table 2.2: Payoff matrix for the climbing game

The results of the experiment shown in Figure 2.5 clearly show that the ILs get

stuck in the suboptimal equilibrium, being unable to coordinate and change their strategy together to arrive at the optimal equilibrium. The JALs on the other hand are able to do just that.

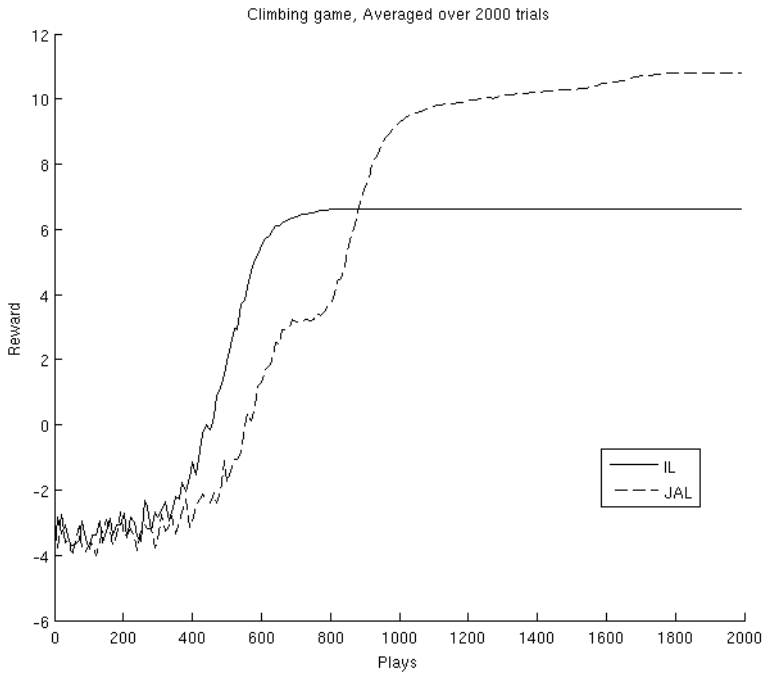


Figure 2.5: ILs and JALs learning a strategy for the climbing game. The ILs get stuck in the $a_1 - b_1$ equilibrium, while the JALs find the optimal equilibrium of $a_0 - b_0$.

2.4 Conclusion

In this chapter, we presented the reinforcement learning concepts and frameworks necessary to reason about decision making and learning in single and multi-agent settings.

Markov Decision Processes are RL tasks that satisfy the Markov property, which means that subsequent environment states depend only on the current state and actions taken by the single agent. MDPs and specifically the Markov property are important in RL because when dealing with tasks that satisfy the property, decisions can be made in function of the current state only.

Markov Games are the multi-agent extension of MDPs. They provide a framework for dealing with tasks that include many agents, which can be cooperative or competitive. RL agents can be independent learners, ignoring the influence of other agents on the effect of their own decisions, or joint action learners, coordinating with the other agents in order to increase performance.

Chapter 3

Local Coordination in Multi-Agent Settings

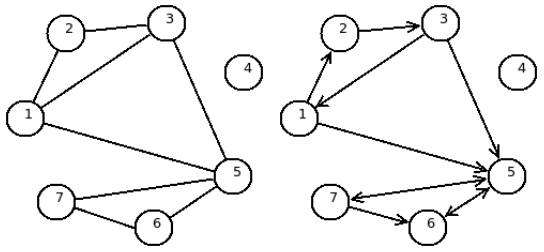
In the previous chapter, we showed how a group of agents can coordinate using reinforcement learning in order to better solve the problem they are facing. A significant problem in these multi-agent reinforcement learning settings is the explosion in complexity as the number of agents increases. Recall joint action learners, who learn reward estimates for every possible combination of actions among all agents. Using these estimates, the group of agents can evaluate which combinations of actions perform best, thus optimizing the team performance. The number of estimates they keep is a^n, with a the number of actions, and n the number of agents. As the number of agents increases, the number of joint actions increases exponentially and quickly becomes unmanageable, both memory-wise and computationally. Yet coordination is necessary because the reward for each agent depends on the actions of all agents.

In this chapter, we investigate local coordination as a way to address this complexity problem, and the impact local coordination, as opposed to global, has on solution quality. Problems that favour coordination between specific agents as well as problems that do not are evaluated. The findings described here will then be applied to the robot soccer problem, where fast decision making is necessary due to strong time constraints on computations.

3.1 Coordination Graphs

Coordination graphs [Guestrin et al., 2002] are a way to formalize the way agents coordinate when solving a problem. In a coordination graph or CG, vertices represent agents, and edges between two agents indicate coordination between these agents. Figure 3.1(a) is an example of a CG with 7 agents. In this graph, agent 1

coordinates with agents 2, 3 and 5; agent 4 does not coordinate and is in essence an independent learner; and agent 6 coordinates with agents 5 and 7. Figure 3.1(a) represents an undirected CG where both agents connected by an edge explicitly coordinate. Evidently, a CG can also be directed, as seen in Figure 3.1(b). In this graph, the same agents are connected as in Figure 3.1(a), but the edges are directed and the meaning of the CG thus differs. Agent 1 now coordinates with agents 2 and 5, and not 3; agent 4 is still an IL; and agent 6 coordinates with agent 5 only.



(a) Undirected Coordination Graph (b) Directed Coordination Graph

Figure 3.1: Example coordination graphs with 7 agents, the left one undirected, the right one directed.

[Guestrin et al., 2002] and [Kok and Vlassis, 2005] propose algorithms where agents, using a message passing scheme based on a CG, calculate a global joint action whose pay-off approximates that of the optimal global joint action. We will take a different approach and develop a generalization of ILs and JALs based on CGs, where agents only optimize their local joint actions.

3.2 Local Joint Action Learners

When viewing ILs and JALs in the context of CGs, we note that ILs can be represented with a fully disconnected graph and JALs with a fully connected or complete graph. This is illustrated by the two leftmost graphs in Figure 3.2. These are specific cases of what we coin as **local joint action learners** or LJALs. LJALs keep estimates for joint actions, like JALs, but instead of tracking all possible global joint actions, they only consider local joint actions, joint actions with the neighbours they are directly connected with in the CG. This reduces the complexity for a specific agent from a^n, a number actions and n number agents, to a^{e+1}, e the number of outgoing edges for the agent in the CG. A random example of

an LJAL CG is the rightmost graph in Figure 3.2. The action selection and reward estimation techniques for JALs as described in the Section 2.3.3 are readily applied to LJALs.

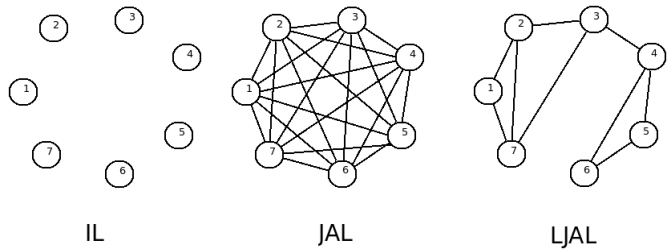


Figure 3.2: Coordination graphs for independent learners and joint action learners and an example graph for local joint action learners

The next sections are laid out as follows: in Section 3.3 we investigate the performance of LJALs on a typical distributed n-armed bandit problem. Section 3.4.2 deals with optimization problems with an inherent structure that could be exploited by LJALs. In Section 3.5 we propose and evaluate a method that allows LJALs to learn a CG optimized for a specific problem.

3.3 Distributed N-armed Bandit Problem

A first hypothesis we will test is whether LJALs in general perform at least as good as ILs and no better than JALs, in terms of solution quality, on a typical distributed bandit problem, as described in Section 2.3.3. Since LJALs possess more information than ILs and less than JALs, we propose that in fact this will be so. Moreover, because the complexity of LJAL joint actions can be no less than a, as in ILs, and no more than a^n, as in JALs, we propose that LJALs will computationally perform no better than ILs and no worse than JALs.

In short, we test whether ILs and JALs really are both ends of the performance spectrum that LJALs encompass.

The experiment that tests this hypothesis averages the results of ILs, LJALs and JALs over 10000 runs, each run consisting of 200 plays, each run on a new random bandit problem. Every play, 5 agents choose between 4 actions, and receive a reward for the global joint action. The reward for each joint action is generated before each run, drawn from a normal distribution with mean 0 and standard deviation 50. We compare ILs, JALs and two different LJALs, both with randomly generated, directed, CGs, one with a fixed outdegree of 2 for each agent (LJAL-2), and one with a fixed outdegree of 3 for each agent (LJAL-3)[1]. Example

[1]Coordination partners are randomly chosen.

graphs are illustrated in Figure 3.3. All learners employ softmax action selection with temperature function $\tau = 1000 * 0.94^{play}$. Figure 3.4 displays the results of this experiment and Table 3.1 details the speeds and solution qualities for the various learners, relative to those of the JALs.

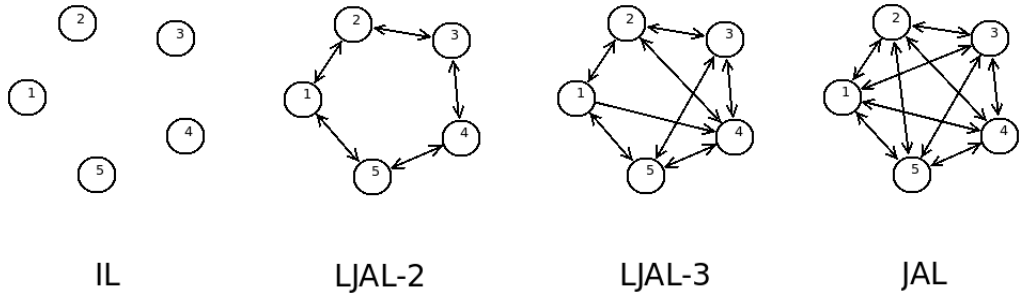


Figure 3.3: Coordination graphs for independent learners and joint action learners, and example coordination graphs for local joint action learners with outdegrees 2 and 3.

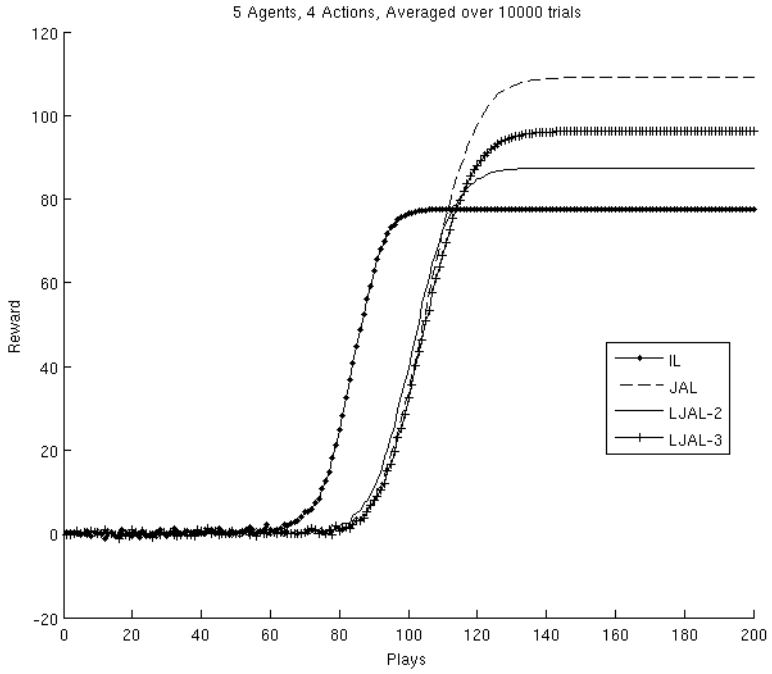


Figure 3.4: Comparison of independent learners, joint action learners and local joint action learners on distributed bandit problems.

Learner	Speed	Solution Quality
IL	x306	71.1%
LJAL-2	x13	80.2%
LJAL-3	x3.5	88.3%
JAL	x1	100%

Table 3.1: Comparison of speed and solution quality for independent learners, joint action learners and local joint action learners solving a typical distributed bandit problem.

As hypothesized, the complexer the CG, the better the solution quality, and the slower the computation. Notice that the sequence IL, LJAL-2, LJAL-3 and JAL corresponds with the sequence 0, 2, 3 and 4, the number of outgoing edges per agent in the CG. As shown in Figure 3.5, there is a strong correlation between the number of outgoing edges and the computation time. It fits an exponential curve.

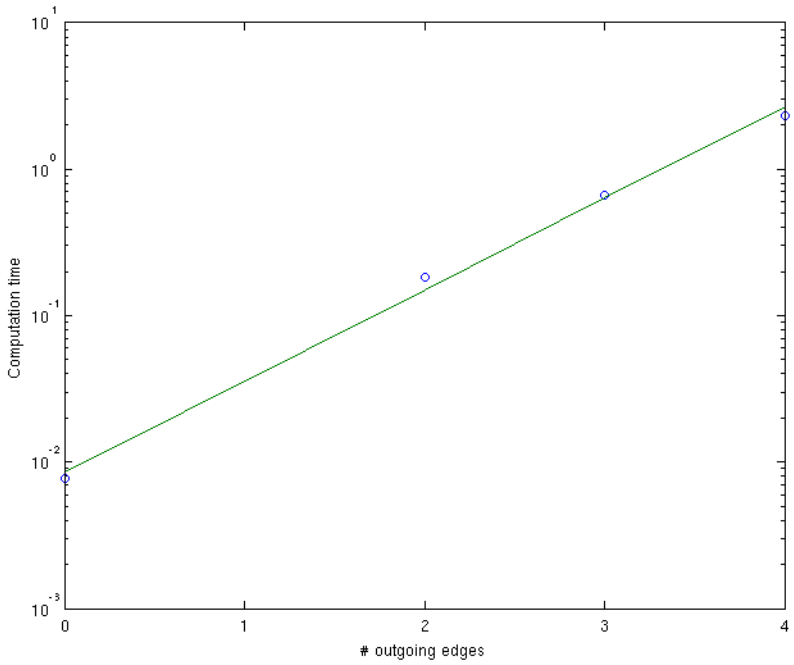
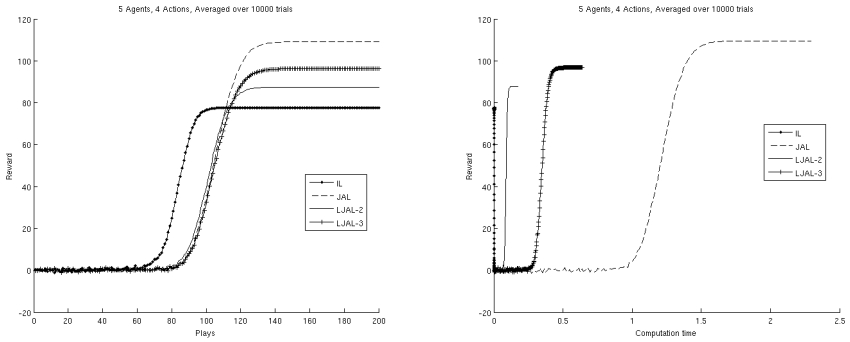


Figure 3.5: Correlation between number of outgoing edges per agent and the computation time, plotted on a logarithmic scale.

Compare Figures 3.6(a) and 3.6(b) for a confirmation of these results. In the left figure, JALs and both LJALs seem to converge to their respective solutions

at approximately the same time, but it is clear in the right figure that as the CG's density increases, computation takes much longer.



(a) Comparison over plays (b) Comparison over computation time

Figure 3.6: Comparison of independent learners, joint action learners and local joint action learners on distributed bandit problems. The left figure shows a comparison over a number of plays, as in Figure 3.4. The right figure shows the same over computation time.

3.4 Distributed Constraint Optimization Problem

In many real-world problems, the effects of an agent's actions are for a large part only influenced by the actions of a subset of all agents in the system. They often only need to coordinate with other physically close agents. In this section, we investigate how local joint action learners can achieve good solution quality, by exploiting the inherent structure of the problem at hand, while keeping computation times low.

We will introduce the Distributed Constraint Optimization Problem framework, or DCOP, as a test bed for LJALs.

3.4.1 Constraint Satisfaction and Optimization

Constraint satisfaction is the problem of assigning values to a series of variables, without violating any constraints defined over these variables. A typical constraint satisfaction problem, or CSP, is the graph colouring problem, where regions on a map need to be coloured, without adjacent regions having the same color. The variables in this problem are the regions to be coloured, the values are the possible colours and the constraints are defined over variables representing adjacent

regions, prohibiting the same value for both variables. A solution to a CSP must satisfy all constraints, or it is not considered a solution.

In constraint optimization, instead of having the satisfied-or-not constraints from CSP, constraints have an associated cost function. These are called soft constraints, as opposed to the hard constraints from CSP. A solution to a constraint optimization problem, or COP, has a total cost/reward, which is the sum of the costs of all constraints. Minimizing this cost, or maximizing this reward[2], yields better solutions to the COP.

Distributed constraint optimization is the distributed equivalent of COP. A group of agents must solve the COP in a distributed way, each agent controlling a subset of the variables in the problem.

Definition

Formally, a DCOP is a tuple ($\mathcal{A}$, $\mathcal{X}$, $\mathcal{D}$, $\mathcal{C}$, f).

- $\mathcal{A} = \{a_0, a_1, ..., a_l\}$, the set of agents.
- $\mathcal{X} = \{x_0, x_1, ..., x_n\}$, the set of variables.
- $\mathcal{D} = \{D_0, D_1, ..., D_n\}$, the set of domains. Variable x_i can be assigned values from finite domain D_i.
- $\mathcal{C} = \{c_0, c_1, ..., c_m\}$, the set of constraints. Constraint c_i is a function $D_a \times D_b \times ... \times D_k \rightarrow \Re$, projecting the domains of a number of variables onto a real number, being the reward.
- $f : \mathcal{X} \rightarrow \mathcal{A}$, a function mapping variables onto an agent.

The total cost, or reward of a variable assignment S is:

$$C(S) = \sum_{i=1}^{n} c_i \tag{3.1}$$

For simplicity, we assume only one variable per agent and only binary constraints. Unary constraints can easily be added[3] and higher arity constraints can be constructed using unary and binary constraints.

Illustration

Figure 3.7 illustrates a DCOP with three agents, each managing one variable, and two binary constraints, $R_{1,2}$ and $R_{2,3}$. These constraints have an associated reward

[2]Depending on the problem formulation.

[3]In our approach.

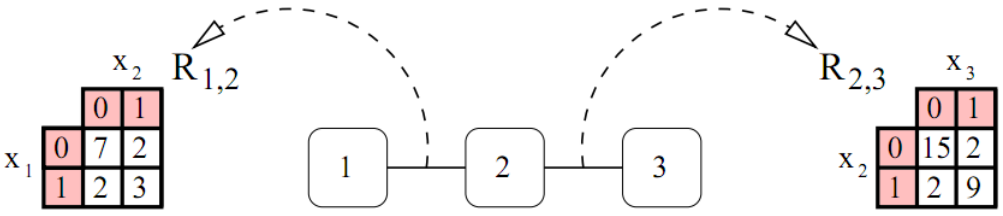


Figure 3.7: A three-agent distributed constraint optimization problem. Each agent must select a value for its variable and the global reward depends on the reward for each constraint, i.e. pairs of agents' selected values. [Taylor et al., 2011]

in function of the chosen values for the variables. Optimizing the reward for this problem means choosing values for the variables in such a way that the sum of the rewards for every constraint is as high as possible.

Suppose all agents choose the value 1 for their variables. The reward function of constraint $R_{1,2}$ yields reward 3 for input $(1, 1)$. Similarly, the reward for constraint $R_{2,3}$ is 9 for the given variable assignment. The total reward of the solution is 12. Looking at the reward functions, we can easily see that the optimal solution is the assignment of 0 to all variables, which yields a reward of 22. Yet no single agent can change its variable's value without getting a lower reward. Coordination is necessary to find a better solution. Several state of the art algorithms exist that are proven to find the optimal solution, although with an exponential communication or computational complexity [Modi et al., 2003, Petcu and Faltings, 2005, Chechetka and Sycara, 2006].

Local Coordination in DCOP

Since each constraint in a DCOP has its own reward function and the total reward for a solution is simply the sum of all rewards, some constraints can be of higher importance than other constraints, when they yield higher rewards. Due to this property, coordination between specific agents can become more important than between others for solving the problem.

In the next section, we will investigate the performance of LJALs on DCOPs where some constraints are more important than others. We will make the computer generate DCOPs, drawing the rewards of every constraint function from the same normal distribution. Since all rewards are drawn from the same distribution, we will impose a structure on the problem by means of weights, formalizing the importance of specific constraints with respect to the whole problem. We attach a weight $w_i \in [0, 1]$ to each constraint c_i, with which the reward is multiplied when calculating the total reward for the solution. A weight of 1 indicates the constraint

is of the highest importance, while 0 makes the constraint of no importance. The total reward for a variable assignment S becomes:

$$C(S) = \sum_{i=1}^{n} w_i c_i \tag{3.2}$$

Figure 3.8 illustrates a weighted DCOP, by means of a graph. The colours of constraints or edges indicate the importance of that constraint. The darker the constraint, the higher the weight. We will continue to use this visualization as a means to show the structure of a DCOP.

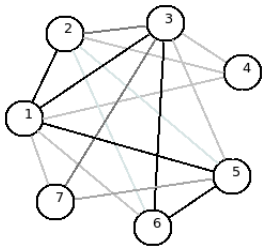


Figure 3.8: A distributed constraint optimization problem, with the constraint weights visualized.

3.4.2 Solving a DCOP using LJALs

As noted in [Taylor et al., 2011], a DCOP can be reformulated as a distributed n-armed bandit problem. Assign one variable to each agent and let him choose from the values in the domain corresponding to the variable, as he would select from an n-armed bandit. With such a formulation, we can apply our previously described learners to DCOPs. An advantage the reinforcement learning approach has in this context, is that it does not require knowledge of individual cost functions, as most DCOP algorithms do. A disadvantage, in some problems, is that each agent needs to have access to the global reward for the combined variable assignment.

Figure 3.9 visualizes the structure of the problem we will compare different LJALs on. The rewards for each constraint function are fixed before every run, drawn from a normal distribution with mean 0 and standard deviation 70. The black edges in the figure correspond to weights of 0.9, light-grey edges are weights of 0.1. What this graph formalizes, is that the constraints between agents 1, 2 and 3, and 5 and 6 are very important, while the contribution of all other constraints to the total is quite limited.

We will compare 5 different LJALs on this problem. The coordination graphs for these learners are illustrated in figure 3.10. One group of agents are ILs and one

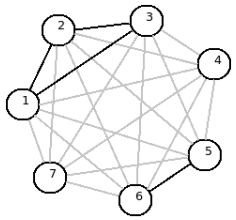


Figure 3.9: The distributed constraint optimization problem to be solved.

JALs. The learners with CG labelled LJAL-1 are LJALs with 2 random partners for every agent. CG LJAL-2 is a graph matching the structure of the problem. CG LJAL-3 is the same graph as LJAL-2, but with an extra edge. All these graphs are undirected, so both agents connected by an edge will explicitly coordinate.

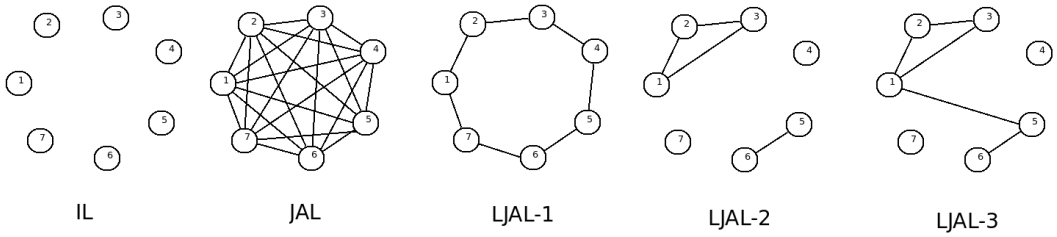


Figure 3.10: Different local joint action learners, visualized by their coordination graphs.

The results, averaged over 2000 runs, are shown in Figure 3.11 and Table 3.2.

Learner	Speed	Solution Quality
IL	x2677	86.8%
LJAL-1	x242	87.3%
LJAL-2	x457	92.0%
LJAL-3	x239	89.7%
JAL	x1	100%

Table 3.2: Comparison of speed and solution quality for independent learners, joint action learners and local joint action learners solving a distributed constraint optimization problem.

As in the previous experiment, JALs and ILs perform respectively best and worst in terms of solution quality. Interestingly, LJAL-2, the learners that use the CG that matches the structure of the problem, outperform LJAL-1, while coordinating less. With LJAL-1, each agent coordinates with two partners, while with

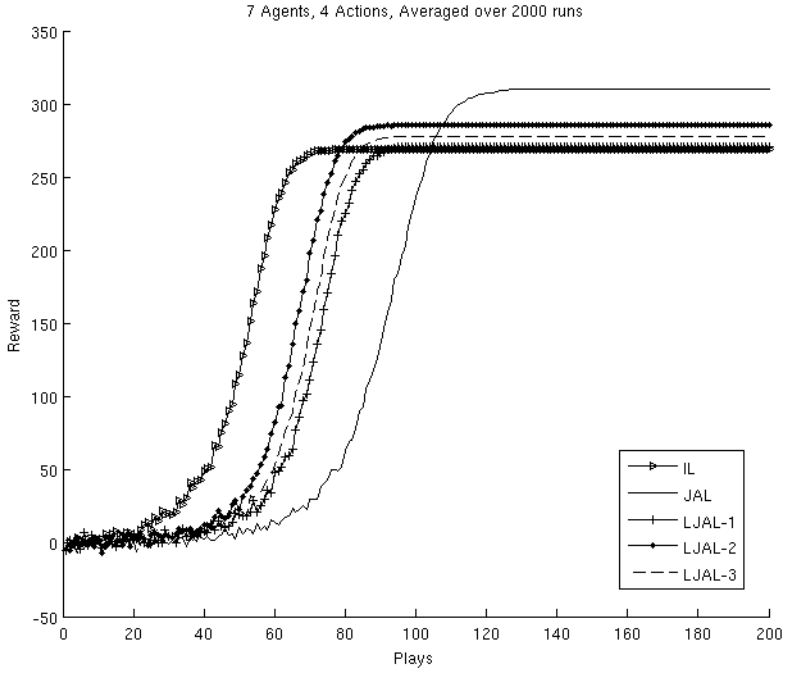


Figure 3.11: Comparison of independent learners, joint action learners and local joint action learners on a distributed constraint optimization problem.

LJAL-2, an agent coordinates with only 1.14 partners on average[4]. This means that using a specific CG can help LJALs solve a problem better, using less computational resources.

Another point of interest is the performance differences between LJAL-2 and LJAL-3. Although agents 1 and 5 in LJAL-3 possess more information than in LJAL-2, through increased coordination, LJAL-3 performs worse both in terms of solution quality and speed. This is because the coordination between agents 1 and 5 is not of great importance as determined by the problem, and apparently complicates coordination on constraints that are important.

Figure 3.12 and Table 3.3 show the results of an experiment set up to evaluate the effect an extra coordination edge has on solution quality. It compares LJAL-2 and LJAL-3, from the previous experiment, with LJAL-4, which uses the same CG as LJAL-2, augmented with a coordination edge between agents 4 and 7.

Since agents 4 and 7 are not involved in important constraints as defined by the problem, the addition of this edge actually improves performance slightly, because the agents will learn to optimize the marginally important constraint between 4

[4]Three agents with two partners, two with one and two without partners

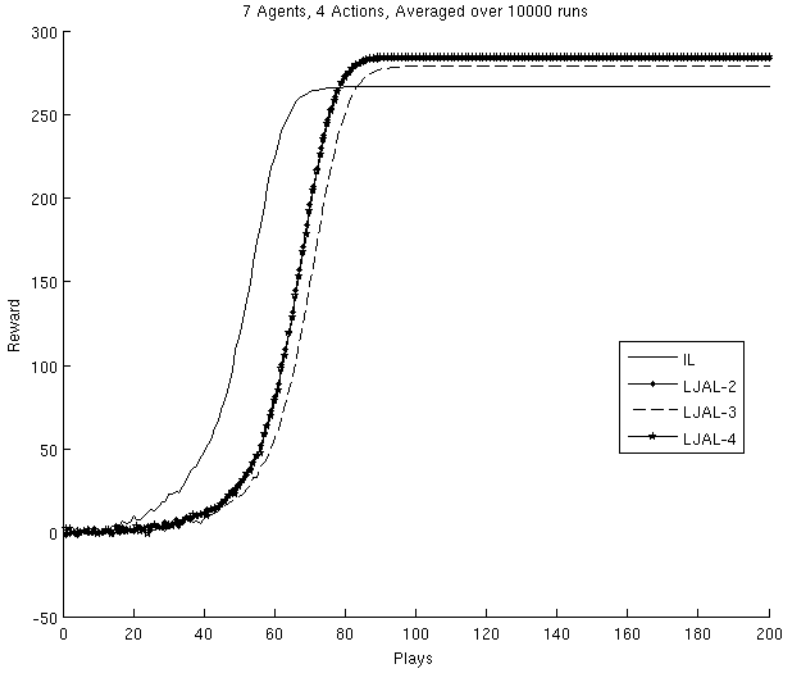


Figure 3.12: Evaluating the effect of extra coordination edges on solution quality.

Learner	Solution Quality
IL	100%
LJAL-2	106.5%
LJAL-3	104.7%
LJAL-4	106.8%

Table 3.3: Evaluating the effect of extra coordination edges on solution quality. Solution qualities are relative to that of independent learners.

and 7, without complicating the coordination necessary for important constraints. [Taylor et al., 2011] also conclude that increasing team work is not necessarily beneficial to solution quality. The occurrence of this phenomenon depends on the DCOP algorithm used; it has been observed in [Taylor et al., 2011] their DCEE algorithms, and now LJALs.

3.5 Optimizing Coordination Graphs

In the previous sections, we have shown that matching the coordination graph of local joint action learners to the inherent structure of a problem helps improve solution quality without having to deal with the total complexity of the problem. The next problem we can consider is finding this coordination graph. In some problems, such as the graph colouring problem, this graph may be obvious. In others, the structure of the problem may not be known beforehand and thus the programmer has no way of knowing what coordination graph to implement. In this section, we will investigate a way to allow the local joint action learners to optimize the coordination graph themselves.

3.5.1 Learning a CG

We view the learning of a CG as another distributed n-armed bandit problem. Each agent needs to choose which agents to coordinate with. An agent has as many actions as there are agents in the problem. For example, agent 2 choosing action 5, means a directed coordination edge in the CG from agent 2 to 5. Agent 2 choosing action 2, means agent 2 chooses not to coordinate, so no additional edge in the CG. The combined choices of the agents makes a CG where agents have at most one outgoing edge, i.e. one agent they coordinate with. This chosen graph is used to solve the problem. The reward for the solution found becomes the reward for the choices in the meta-bandit. This constitutes one play in the meta-bandit. Players learn reward estimates for their partner-choices, and next play choose again a partner using softmax action selection. This graph is again used to solve the problem, reward is received, and so on. This goes on until the CG has converged due to decreasing temperature in the meta-bandit.

We let the agents in the meta-bandit problem be ILs, since we want to keep the complexity as low as possible.

3.5.2 Learning on DCOPs

Our first experiment is the learning of a CG on the problem used in previous sections and illustrated in Figure 3.9. One meta-bandit run consists of 1200 plays. In each play, the chosen CG is evaluated in 100 runs of 200 plays. This evaluation is basically the same setup as the experiment illustrated in Figure 3.11. The average of the reward achieved over these 100 runs is the estimated reward for the chosen CG.

Figure 3.13 and Table 3.4 show the results of this experiment, averaged over 50 runs, each time a newly generated problem, although with the same inherent

structure. We compare ILs, JALs, LJALs with a CG matching the problem as LJAL-2 in Figure 3.11 and LJALs optimizing their CG.

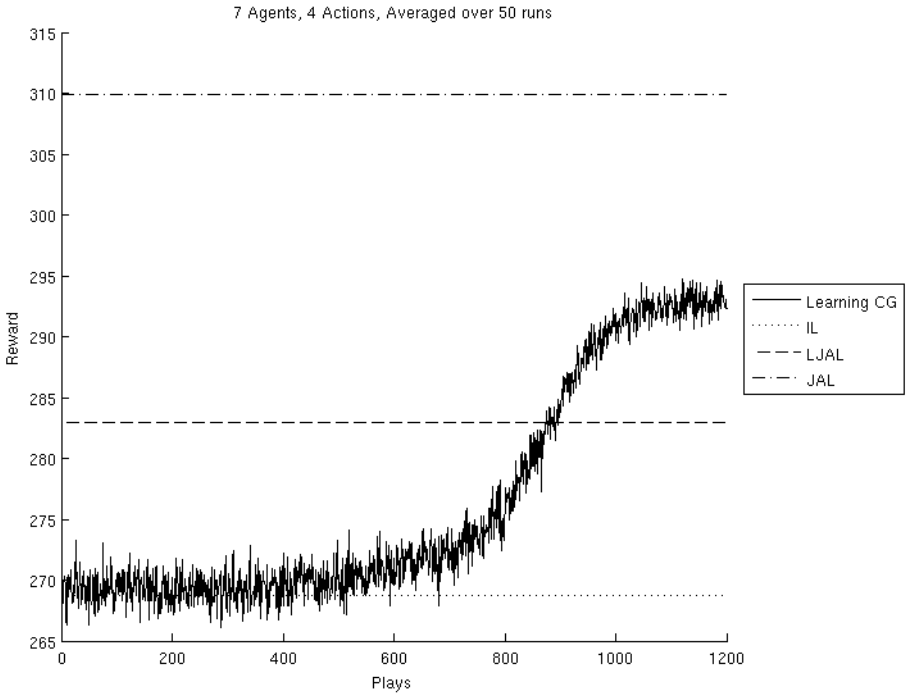


Figure 3.13: Comparing the solution qualities of independent learners, joint action learners, local joint action learners with the supposedly optimal coordination graph and local joint action learners who optimize their coordination graph.

Learner	Solution Quality
IL	86.7%
LJAL	91.3%
LJAL, optimizing CG	94.3%
JAL	100%

Table 3.4: Comparing the solution qualities of independent learners, joint action learners, local joint action learners with the supposedly optimal coordination graph and local joint action learners who optimize their coordination graph.

We can immediately conclude that it is possible to make LJALs optimize their CG, improving solution qualities found. Apparently, the undirected CG matching the problem is not the optimal CG for the problem, as the agents learning a

CG can find better solutions, even though they have a maximum complexity of one coordination partner, as opposed to two partners in the graph matching the problem.

The actual graphs learned by the meta-bandit give a greater insight into how coordination can work in LJALs. Figure 3.15 gives five examples of graphs optimized for the problem in Figure 3.14. A first trend to note is that agents 5 and 6 always learn to directly coordinate with each other, matching the problem definition. Of greater interest is what agents 1, 2 and 3 do, as they should also coordinate, according to the problem. We find that either agent 1 or 2 always coordinates with the other, and in one instance both choose to coordinate with each other. Only in one instance is agent 3 seen to directly coordinate with 2. In three other CGs, 3 coordinates indirectly with 1 and 2 through agents 4 and 7. These are very interesting results, since they show that coordination between two agents must not be explicit; indirect coordination works, as long as a path of coordination exists between both agents. The benefit of this indirect coordination is that additional constraints can be optimized, i.e. those on the path between agents that coordinate indirectly.

In Appendix A, the process of learning a coordination graph is visualized.

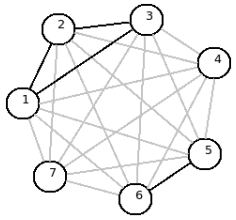


Figure 3.14: The distributed constraint optimization problem to be solved.

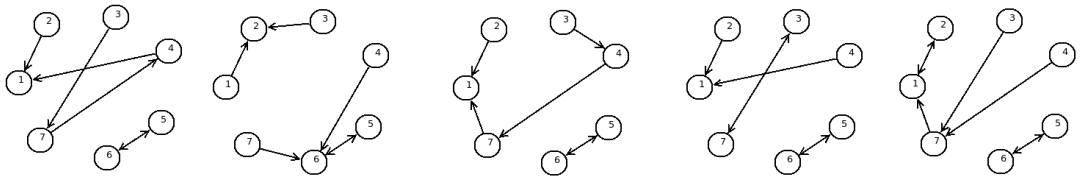


Figure 3.15: Coordination graphs optimized to solve a problem.

3.5.3 Learning on DCOPs with random structure

We have previously only focused on one specific problem, with only two very distinct categories of constraint importance, i.e. very important and not at all. We

will now also investigate problems with constraints of varying importance. One issue with such problems is that, even if the structure of the problem is known, it is not easy to decide which coordination is important and which not. Is it necessary to coordinate over the constraint with weight 0.6, and not over the one with weight 0.59? Obviously, learning the CG is a better approach than guessing or fine-tuning by hand.

The next experiment compares ILs, JALs, LJALs with fixed CG and LJALs learning a CG on DCOPs with a randomly generated weights graph. The non-learning LJALs have a coordination graph derived from the problem's weight graph; all constraints with weight 0.75 and higher are included in the CG. The results of this experiment are shown in Figure 3.16 and Table 3.5.

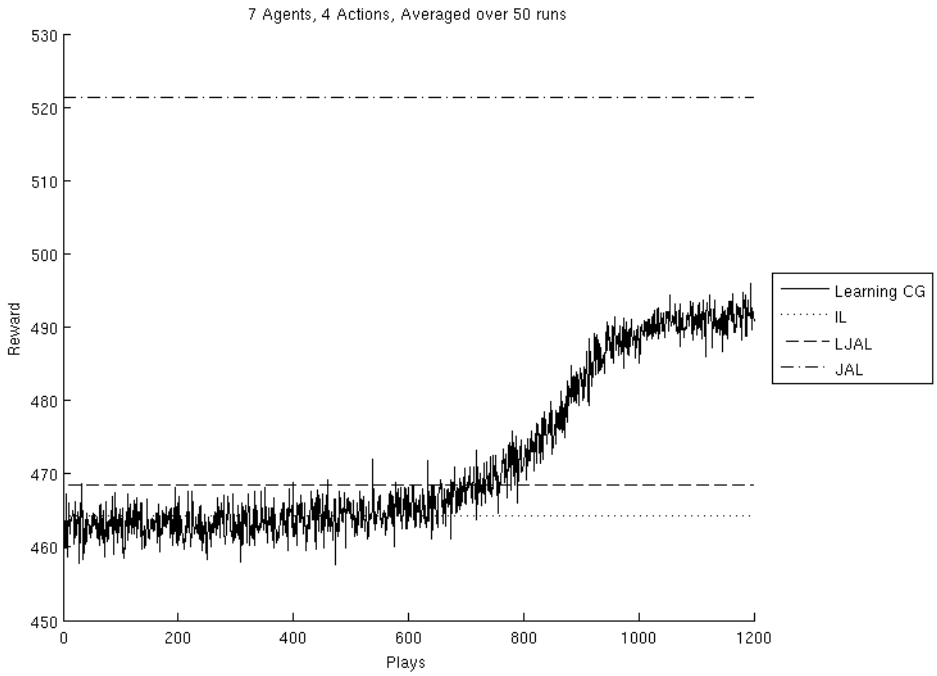


Figure 3.16: Comparing the solution qualities of independent learners, joint action learners, local joint action learners with fixed coordination graph and local joint action learners who optimize their coordination graph on distributed constraint optimization problems with a random weights graph.

Although the LJALs with fixed CG coordinate over a quarter of all the constraints, and the most important ones at that, they do not manage to improve much over the solutions found by ILs. These LJALs have a CG with on average $7 \times 6 \times 0.25 = 10.5$ directed coordination edges. Compare that to the maximum

Learner	Solution Quality
IL	89.0%
LJAL	89.8%
LJAL, optimizing CG	94.1%
JAL	100%

Table 3.5: Comparing the solution qualities of independent learners, joint action learners, local joint action learners with fixed coordination graph and local joint action learners who optimize their coordination graph on distributed constraint optimization problems with a random weights graph.

of 7 edges the LJALs that optimize their CG use. With less coordination, they again manage to improve much on the solution quality.

3.5.4 Learning on DCOPs without structure

We perform another experiment, similar to those before. In this setting, we do not impose a weights graph upon the randomly generated DCOPs, making the inherent structure much more subtle, and less exploitable. Figure 3.17 and Table 3.6 display the results of the experiment. Again, we find that the CG can be optimized to better solve the problem, although the solution quality achieved is slightly worse than in previous experiments (92% vs 94%). This shows that the agents still find a structure to exploit, but which is less pronounced.

Learner	Solution Quality
IL	87.8%
LJAL, optimizing CG	92.0%
JAL	100%

Table 3.6: Comparing the solution qualities of independent learners, joint action learners and local joint action learners who optimize their coordination graph on distributed constraint optimization problems without a very explicit structure.

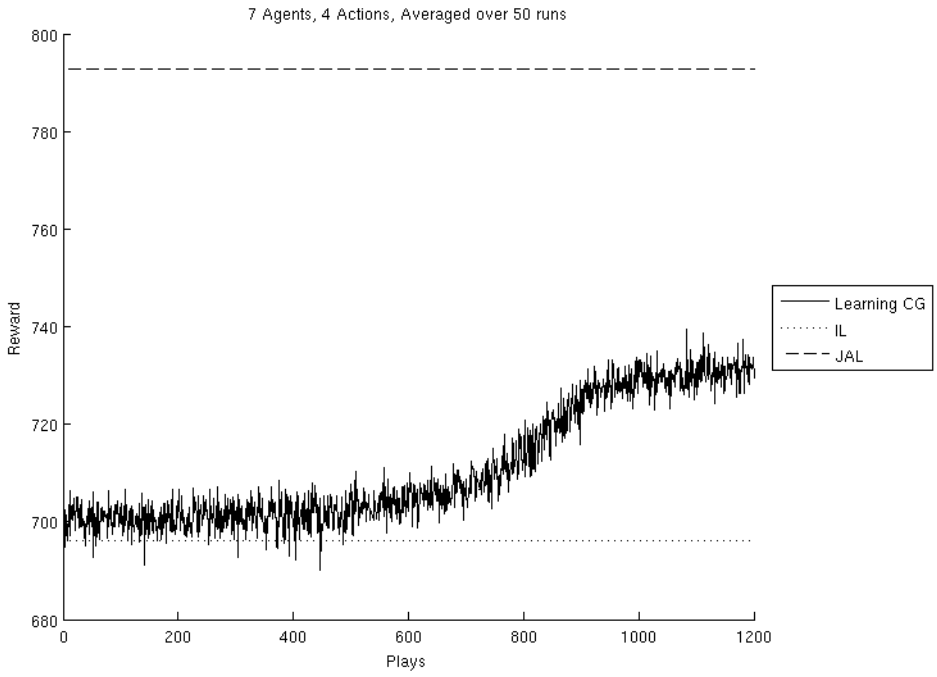


Figure 3.17: Comparing the solution qualities of independent learners, joint action learners local joint action learners who optimize their coordination graph on distributed constraint optimization problems without a very explicit structure.

3.6 Conclusion

In this chapter, we investigated local coordination in a multi-agent reinforcement learning setting as a way to reduce complexity. *Local joint action learners* were developed as a generalization of independent learners and joint action learners. Local joint action learners make use of a *coordination graph* that defines which agents need to coordinate when solving a problem. A coordination graph decomposes a global coordination problem into a set of simpler problems. The density of the graph determines the computational complexity for each agent, and also influences the solution quality found by the group of agents. Denser graphs yield better solutions, while requiring more computation. At least on regular problems.

Solutions to problems that have an inherent structure, making coordination between certain agents more important, can be optimized by local joint action learners that have a coordination graph adapted to the structure of the problem. Learners using such a graph can perform better than those using a random graph of higher density, both in terms of solution quality and computation time.

We have also shown that the coordination graph itself can be optimized to

better match the potentially unknown structure of the problem tackled. This optimization often leads to unexpected graphs, where important constraints in the problem are not mimicked in the coordination graph by a direct coordination link. Instead, this coordination is achieved through a *coordination path* via other agents.

Chapter 4

Adaptive Strategies in Robot Soccer

Our approach to adaptive strategies in a robot soccer context is detailed in this chapter. Although it is the RoboCup Midsize League our contribution is aimed at, we implemented the system and performed experiments in the RoboCup 2D Simulation League, because testing on real robots is time consuming, expensive and error-prone.

We first describe the simulator in Section 4.1 and the source code our implementation is based on in Section 4.2. These sections are followed by a description of soccer behaviours implemented (Section 4.4), and how these are used to adapt a team's strategy (Sections 4.5 and 4.6).

4.1 Simulation Setting

4.1.1 Description

The RoboCup Soccer Server [Chen et al., 2002] is the 2D simulation league's official simulator. It consists of three distinct parts, which are

- the server, where all simulation takes place,
- the client processes that connect to the server through UDP/IP sockets, each process managing one player or coach, and
- the monitor, used to visualize the simulation running on the server

To play a game, two teams of up to 11 clients must connect to the server. Each player is controlled by an individual process, which is only allowed to communicate with the server. The process controlling a player must send messages to the server to execute primitive actions, and receives in turn perceptions from the

server, both visual and aural. A notable complexity of the simulator is the asynchronous nature of this sensing and acting. Players are allowed to execute one action every 100ms, while they receive visual perceptions every 150ms, and aural perceptions whenever they are uttered. This contrasts with the sequence of operation often employed in AI, where an agent perceives, acts upon these sensations, perceives again, etc.

A typical game lasts 10 minutes, which is divided into 6000 discrete simulation steps of 100ms. During these 100ms, the clients have time to send a message detailing the action to be executed at the end of the step. When a client is too late or did not send a message, its agent will not act during that cycle. On the other hand, when multiple message are sent, the server chooses one message at random and executes the action described within. At the end of the simulation step, all actions are executed and the new world state is computed by the server.

A referee is included in the simulator. It enforces the RoboCup 2D simulation league soccer rules[1] and changes the play modes when necessary, e.g. the ball goes out of play.

The soccer server version used is 14.0.3.

4.1.2 Agent Observations

The visual perceptions an agent receives every 150ms report the objects currently seen by the agent. The information includes the type, distance and direction of the object, and the change in distance and direction compared to the previous perception. Players have a field of vision, which is an area extending from the player in the direction the player's head is looking, with an angle visible_angle, by default $90°$. The visual information about other players also includes identification, relating to their team and shirt number, although, the further away these players are in the field of vision, the less precise the identification information becomes. Figure 4.1 illustrates a possible situation. Player c will be identified by team and shirt number, d by team and possibly shirt, e possibly by team and not by shirt, and f will be unidentifiable. Another parameter, visible_distance, determines the size of an area around the player in which he perceives all objects, whether or not they lie within its field of vision, although the agent only perceives the type of object when he does not see it directly. This way, player a is also perceived by the agent, although he can not identify him.

Besides visual information, the agents also receive aural information, which simulates hearing sound. Players receive messages from the referee via this way; referee decisions are received immediately. These aural sensors can also be used for communication between players and coach and players. Such messages are

[1] Which should be detailed in [RoboCup.org, 2011], but are noted as TBA (06/2011)

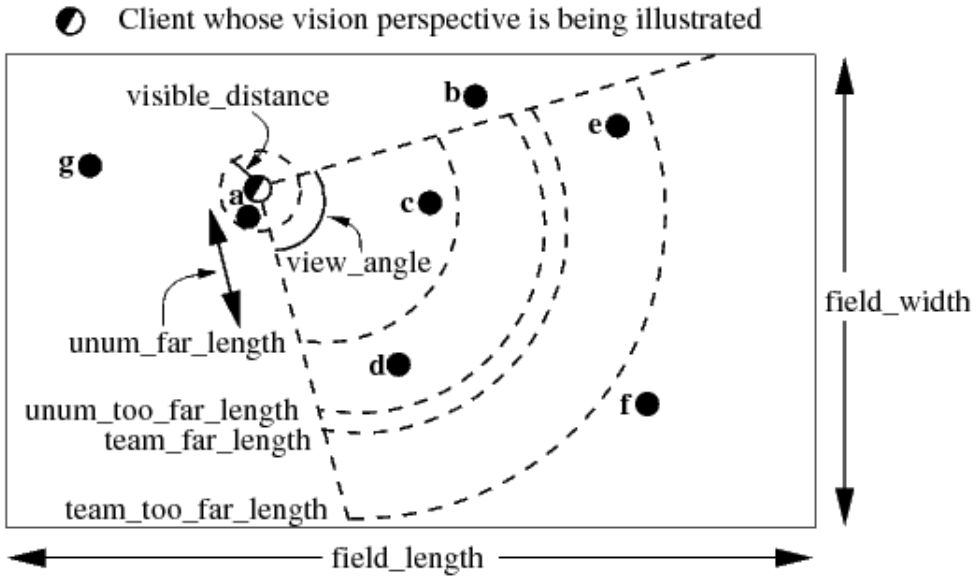


Figure 4.1: A player's visible range, determined by the view_angle (visible_angle). The further away players are, the lower the quality of the identification information. [Stone, 2000]

only received at the beginning of the next simulation step (100ms). When multiple player or coach messages are received, only the first one is actually heard and processed. This excludes messages from oneself. Communication between players is limited by the hearing distance, specified by the audio_cut_dist server parameter. All this makes communication difficult and agents are not guaranteed that their messages are heard by the intended receiver(s). Note that players also hear messages from the opposing team, although to remove the possibility of one team sabotaging all communication by constantly shouting, players can hear one message from a teammate and one from an opponent every cycle.

Lastly, agents receive information about themselves, such as stamina and speed. No positioning information is given and speed is only a magnitude. Players need to infer their own location and direction using visual perceptions.

4.1.3 Actions and Communication

Although the documentation for the simulator is sometimes very outdated and often incomplete, we try to give an overview of all possible actions available to a player. To execute an action, the client process needs to send a message containing the action and associated parameters to the server. This action is then executed at the end of the current simulation step. Every action lasts only one simulation

step, so the client process needs to keep sending messages every simulation step in order to keep the player active.

We describe the three most important actions in detail:

dash Players can dash in a specified direction, employing a specified amount of power, determining the acceleration during the current simulation step. The dashing's efficiency can be reduced when the player's stamina is low.

kick Kicking the ball with a specified force, in a specified direction, makes the ball deviate to the intended direction. The actual speed and direction are not set by the kicking action itself, but are calculated by the server by adding to the ball's current speed vector those of all kicks executed in that simulation step.

say To communicate with each other, players have the ability to 'say', or rather 'shout' one message each simulation step. This message is broadcast and audible to all players, teammates and opponents, within hearing range, although possibly not all these players will hear the message, because other players may be shouting as well, which interferes with the audibility of the message; players can only hear one message from each team per simulation step.

Table 4.1 lists all possible actions and gives an explanation where possible.

4.2 WrightEagleBase

4.2.1 Description

The team implementation with which the experiments described below have been run, is based on the WrightEagleBase code, version 2.1.0 [WrightEagle, 2011]. This is open source code developed in C++ at the Multi-Agent Systems Lab., School of Computer Science and Technology, University of Science and Technology of China. The code is basic code on which their team WrightEagle is based. This team has been participating in the annual RoboCup competition since 1999, and over the last six years, has won three times and placed second the other three. The main reason this code was chosen over other available sources, was its clear class structure and ease of extensibility.

4.2.2 Strategy

The WrightEagleBASE implementation provides the players with a set of behaviours that use the primitive actions provided by the simulator (Section 4.1.3) to achieve higher level goals. Theses behaviours are:

Action	Arguments	Explanation
attentionto	Side, Number	
catch	Direction	Only available to the goalkeepers. When catching the ball succeeds, the goalkeeper is attributed a free kick.
change_view	Width, Quality	Changes the frequency and quality of visual perceptions.
dash	Power[, Direction]	Accelerates the player in the direction he is facing, unless Direction is specified.
kick	Power, Direction	If the ball is within kickable distance from the player, the ball is kicked in the direction the player is facing.
move	X, Y	Moves the player to any location on the field. Is only allowed before a kick off.
pointto	Distance, Direction	
say	Message	Broadcasts a message to all players.
sense_body		Prompts the server to return physical information about the player (stamina, speed, etc.).
score		Prompts the server to return the current score of the game.
tackle	Direction[, Foul]	
turn	Moment	Changes the player's heading over a specified angle.
turn_neck	Angle	Changes the direction the player looks at, changing his field of vision.

Table 4.1: Actions available to RoboCup agents

- Intercepting the ball
- Shooting at goal
- Passing the ball
- Dribbling with the ball
- Positioning without the ball
- Blocking the opponent with the ball
- Marking an opponent without the ball

The policy used by the agents to select which behaviour to execute is a simple one step look ahead policy. Every simulation step, the expected effect of each behaviour is evaluated and assigned a real number. The behaviour resulting in the highest number is executed. For most of the behaviours, such as positioning, dribbling and passing, this evaluation amounts to rating the expected distance of the ball to the opponent's goal. As a result, agents will always pass the ball forward, since passing gets the ball faster close to the goal, until there are no players closer to the goal. That player will then start dribbling to the goal. A player will only shoot at the goal when he is within the penalty area, but in that situation, the shooting behaviour will overrule any other possible behaviour.

The positioning system provided in the code makes the agents stand in a circle around a homing point, which is the average position of every teammate, as shown in Figure 4.2. Note that this homing point moves when some players are not positioning themselves with this system, but are e.g. intercepting the ball, running with the ball, etc.

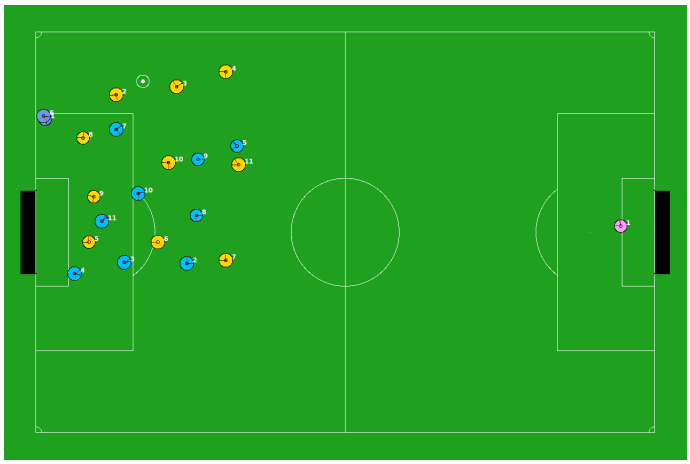


Figure 4.2: Two teams using the original WrightEagleBASE positioning system

Because the world is not fully observable for players, they communicate to improve their internal model of the world. Every time step, each agent shouts its own position and in turn hears the position of one of its teammates. Because there are 10 time steps in 1 second, and the probability of each teammate's message being heard is random, each agent has, on average, up-to-date information about all its teammates every second.

4.3 WrightEagleBASE Extensions

We made various major and minor improvements to the WrightEagleBASE code:

- A form of pass evaluation [Stone and Veloso, 2000], which is a method to estimate the likelihood of a pass arriving at the intended player. Being able to correctly predict the outcome of potential passes can help prevent the loss of possession resulting from bad passes. To this end, we calculate a value $e \in [0, 1]$, given the positions of the passer, receiver and opponents. e reflects the confidence an agent has of a pass straight to a specific player succeeding.

  Each opponent p its position is compared to an ellipse with as foci the passer and the intended receiver. If the opponent is located outside of this ellipse, $e(p)$ returns 1, meaning that the opponent is not expected to interfere with the pass. Positions within the ellipse are graded from 1 on the circumference to 0 on the line between the two foci, the result of Equation 4.2. The $e(p)$ values for all opponents are combined by taking their minimum. This is the value for the opponent most likely to interfere with the pass.

  Given this e-value and a threshold, the agent can decide whether or not a pass will succeed.

  The variables in Equation 4.2 are illustrated in Figure 4.3.

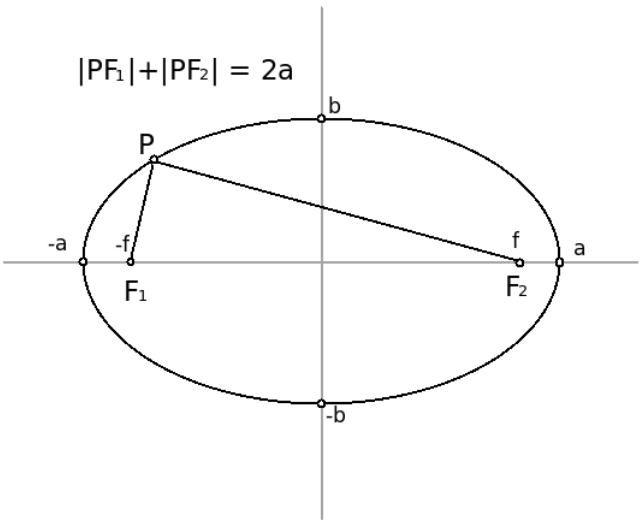


Figure 4.3: An ellipse

$$e = \min_{p} e(p) \tag{4.1}$$

$$e(p) = \begin{cases} 1 & if |F_1P| + |F_2P| > 2a \\ \frac{|F_1P|+|F_2P|-2f}{2a-2f} & if |F_1P| + |F_2P| \leq 2a \end{cases} \tag{4.2}$$

- A new, flexible positioning system, described in Section 4.4.1.
- A better goalkeeper, that uses the new positioning system to better defend the goal and that does not pick up passes from teammates, which is not allowed by the rules.
- A new type of pass, which is described in Section 4.4.2.
- Better evaluation for players choosing whether or not to shoot at goal, and a mechanism to evaluate where in the goal best to shoot, considering opponents and the goalkeeper.

4.4 Behavior Variations

We defined variations for two behaviors, namely positioning and passing.

4.4.1 Positioning

The performance of a team during a game depends largely on the positioning of individual players and the combined formation of the team. Players often play on a certain location within a team formation, a location which is defined relative to the other players, relative to the field and often both, e.g. a left back defender, or a right wing attacker. Within that location, though, players can adopt a variety of play styles. Defenders can prefer to closely mark their opponent, or stay relatively clear to sweep up the ball if the opponent breaches the defensive line. Attackers can run free from their defender to be available for passing, or flirt with the offside position, relying on passes in the empty space behind the defence and speed to get in a scoring position.

The original positioning system of the WrightEagleBASE team is quite rigid, and does not directly take into account the location of opponents. Below is described a newly implemented system that can easily be changed to implement various play styles.

A play style can be defined as a set of objectives players want to optimize, and the degree of importance they attach to each objective. Such objectives can be

- minimizing the distance to opponents
- maximizing the distance to teammates
- not running offside
- being available to receive a pass

- having a clear line to the opponents goal
- staying in the assigned area within the agreed formation
- ...

These objectives and more were implemented and used in the experiments described further on.

The way objectives are represented is similar to potential functions [Latombe, 1991]. Each function $f_i(p)$ rates a position p with $r \in [0, 1]$, according to the objective it represents. Evaluating a position p with n functions $f_i(p)$ yields a result vector $\vec{f} = [f_1(p), f_2(p), .., f_n(p)]$. To combine all the results into a single evaluation $e(p) \in [0, 1]$, we need to calculate the dot product of the result vector with a weight vector $\vec{w} = [w_1, w_2, .., w_n]$.

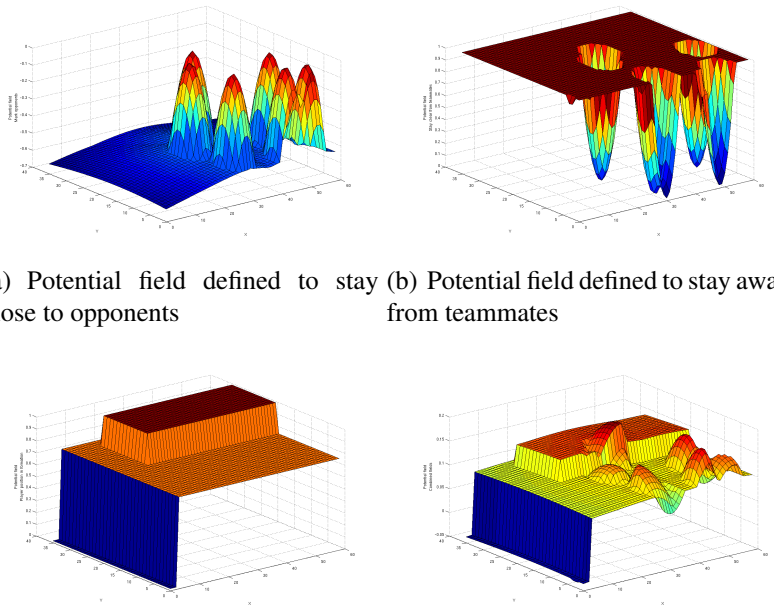
$$e(p) = \vec{f} \cdot \vec{w} = f_1(p)w_1 + f_2(p)w_2 + .. + f_n(p)w_n \tag{4.3}$$

This weight vector is used to formalize the relative importance of each function and its associated objective to the agent. Functions can be rendered attractive or repulsive by defining each weight as either positive or negative. The values in the weight vector can be either predefined, or learned by the agent through an optimization process. Since we want to predefine different static positioning systems, we have to predefine the weight vectors too.

Using this method, agents can evaluate the whole field and find the optimal position to go to, i.e. the location with the highest value. In practice, this is not efficient, because evaluating the whole field takes too long, making the agent miss simulation steps. A smaller search area, agent-centric, will suffice to find a good optimum. This search area can then be covered with a grid, of which each point should be evaluated. Finally, the agent simply needs to move in the direction of the best location.

Although the agent most likely will not be able to reach the optimal location in one time step, this is not a problem, since optima will in general move in a gradual way, and will thus be in the approximate same direction one step later. Also, note that too small search areas result in agents finding only less interesting local optima and thus performing suboptimal, e.g. marking the opponents goal keeper, when no other opponent is within the search area.

A variety of potentially very different play styles can be generated by means of the weight vector. In the experiments described further on, three play styles are added to the already existing one, making a total of four play styles. One new play style was designed with a defender in mind, making the agent mark an opponent, try to use the offside rule against the opponent and stay behind the ball. In a more offensive play style, the player tries to stay clear from defenders, be available to



(a) Potential field defined to stay close to opponents

(b) Potential field defined to stay away from teammates

(c) Potential field to guide the agent to its position within the team formation

(d) Combination of these three potential fields

Figure 4.4: Three examples of potential fields in an in-game situation and their combination. X and Y coordinates represent locations on the field, Z is their potential field evaluation.

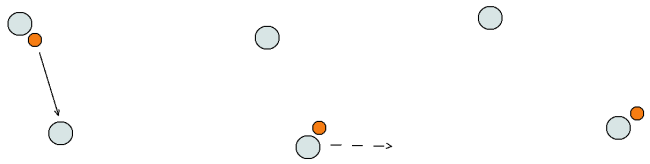
receive a pass and be in a good position to shoot at goal. The third play style defined combines the previous two and applies a defensive style when the team is not in possession of the ball and inversely an offensive style when the team is.

The team plays a formation of 4-3-3, and each agent has a preferred location within that formation, but is allowed to choose between these four play styles. This makes for a total of $4^{10} = 1048576$ number of possible combined positioning strategies for the team. The goalkeeper is excluded from this variation selection, since he has a very different role than the other players within the team.

4.4.2 Passing

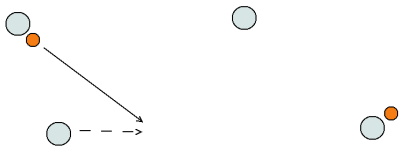
In two-dimensional soccer, the ball always stays on the ground. Therefore, one can distinguish two types of possible passes: a pass straight to a player, and a pass to a location some distance away from the intended player, a deep pass. The advantages of the latter pass are manifold. Such passes can be used to move the play forward more quickly, since the pass receiver does not have to wait for

the ball before moving in the desired direction. Opponents can be passed easily and their defensive line can be breached while still respecting the rule of offside, without having to dribble with the ball past defenders, incurring a higher risk of losing the ball.



(a) t=1 (b) t=2 (c) t=3

Figure 4.5: A regular pass: the receiver gets to the new position with the ball in three steps



(a) t=1 (b) t=2

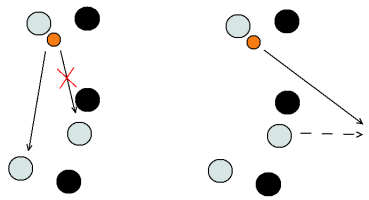
Figure 4.6: A deep pass: the receiver gets to the new position with the ball in two steps

Besides the regular pass system provided in the WrightEagleBASE code (augmented with our own rudimentary pass evaluation), we implemented a deep pass, specifically designed to breach the opponent's defense. This pass system only performs such a pass when on the opponent's side of the field.

Obviously this passing strategy incurs a risk of the ball being lost to the opponent. If the intended receiver is not positioned well or is not quick enough to react, an opponent's defender or even the goalkeeper might reach the ball first. There is a clear relation between the passing strategy of a player and the positioning strategy of the others, especially attackers.

4.5 Adaptive Strategies

In this thesis, we propose a new approach to make a cooperative team of agents adapt to an adversarial team and improve its performance against the opponent.



(a) Regular Pass (b) Deep Pass

Figure 4.7: Breaching the opponent's line of defence

The goal of this approach is to keep the original high level policy of an agent, adding a new layer where the adaptation takes place.

In our context, we take this high level policy to be the policy that the agent employs to decide between the behaviours described in Section 4.2.2 (positioning, passing, dribbling, etc.). For each of these behaviours, we define different variations, which have (slightly) different outcomes. Compare for example the defensive and offensive positioning variations described in the previous section. When the agent decides which high level behaviour he will execute, he then needs to select the specific variation that he actually will execute. This is visualized in Figure 4.8.

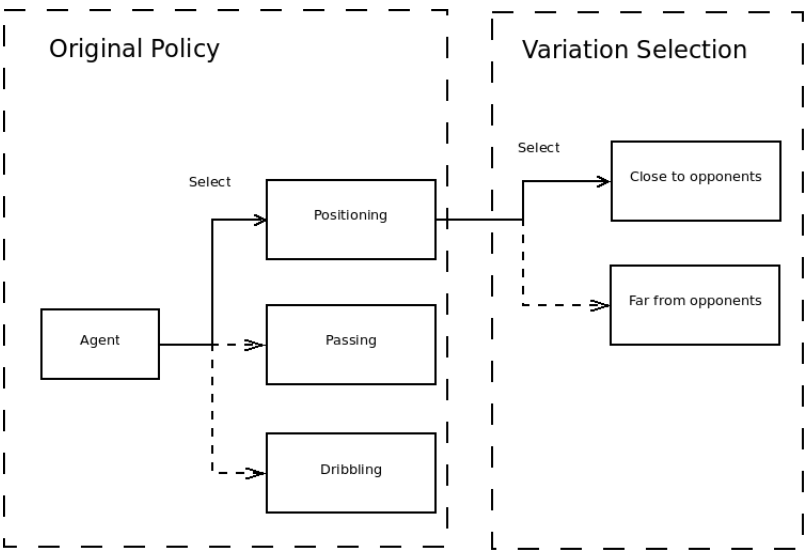


Figure 4.8: Adaptation layer added to existing policy

By using reinforcement learning techniques to evaluate these variations against a specific opponent, each agent can learn which variations to use in order to optimize the whole team's performance against the opponent. This way, the whole

team builds a strategy from the basic building blocks, which are each agent's behaviour variations. This strategy is optimized to counter the play style of the opponent being played against.

4.6 Learning Adaptive Strategies

In this section we propose a method for the selection of behaviour variations, as described before, that allows a team to optimize its strategy to a specific opponent.

4.6.1 Episodes

The performance of a team relies on the combined behaviours of all its players. If every agent asynchronously selects variations of its behaviours during the game, it is very difficult to determine the part each variation played in the successes or failures encountered, and even more so for the various combinations of behaviours that occurred.

To resolve this problem, we divide the game into so called episodes, which span from play interruption to play interruption, e.g. from a kick-off to a goal scored, or a throw-in to a foul. Before a new episode, each agent chooses for every behaviour a variation, which will stay fixed during the whole episode. Only at the end of an episode an agent will receive a reward. That way, it is clear what behaviour variations and combinations lead to this reward, although the extent in which each specific player and his selected variations contributed to the end result may be less clear. This is especially true in very short episodes, such as a corner immediately followed by a goal. In such episodes, only a handful players' strategies were important, while in longer episodes, all player's strategies may have had a more equal influence on the game.

4.6.2 Rewarding

An essential part of reinforcement learning, is the reward signal. In the context of episodic robot soccer as described above, we need to reward agents for their contribution to the end result of the episode. The better we can estimate specific agents and behaviours their contributions, the faster we can learn and the better the learned strategy will be.

Reward is received at the end of an episode. The reward signal is heterogeneous and differentiates between defenders, midfielders and attackers, encouraging each type of player to take up different roles. Rewarding attackers more for goals scored by our team, and punishing defenders more for goals scored by

the opponent, encourages them to learn respectively good attacking and defensive strategies. Midfielders are the connection between attack and defence, and should contribute to both, although through cooperation, they may diversify into defensive and offensive midfielders.

Only rewarding goals scored and goals received is quite limited. Episodes might very well end without a goal being scored[2]. This is especially a problem in games with well matched opponents with strong defensive strategies, where goals are few and far between. From a learning perspective, such episodes are lost time, since there is no reward at the end to give feedback on the selected behaviours.

Therefore, we add extra reward to allow some evaluation of episodes that end without a goal. This reward signal is implemented in a similar way as part of the reward function proposed in [Stone and Veloso, 1999], except with our addition of heterogeneity. Figure 4.9 shows the homogeneous reward signal as proposed by [Stone and Veloso, 1999], only with all rewards divided by 10. A small breakdown of the reward signal:

- When a goal is scored, players receive 10 or -10, for a goal for or against respectively
- When the ball goes out of play for a kick-in, depending on the location on the field and the team receiving the ball, the reward can range from -2.5 to 2.5
- Free kicks are treated the same as kick-ins
- Corners for are awarded 2.5 and corners against -2.5
- A goalkick for is awarded -1, negative because the opponent must have come dangerously close to our goal, and a goalkick against is awarded 1

Heterogeneity is achieved by scaling down the rewards depending on the agent's role and the game situation. For all but kick-ins and free kicks, the scaling is simple, i.e. reward from a situation on our side of the field is fully attributed to defenders, and divided by 10 for attackers, and inversely when the situation ended on the opponent's side of the field. Midfielders always receive the full reward. Table 4.2 shows the heterogeneous reward for goals scored.

For kick-ins and free kicks, the heterogeneity of the reward is slightly more complicated. Because these situations can take place anywhere on the field, we decide how large a part is distributed to different players depending on their role and the location on the field the situation takes place. The closer to the opponent's goal, the closer the scaling factor gets to 1 for attackers and $1/10$ for defenders.

[2]The RoboCup 2010 2D simulated soccer final counted 35 episodes, but only 3 goals.

Inversely, the closer to our own goal, the closer the scaling gets to $1/10$ for attackers and 1 for defenders. This scaling factor changes linearly along with the X position of the free kick or kick-in on the field.

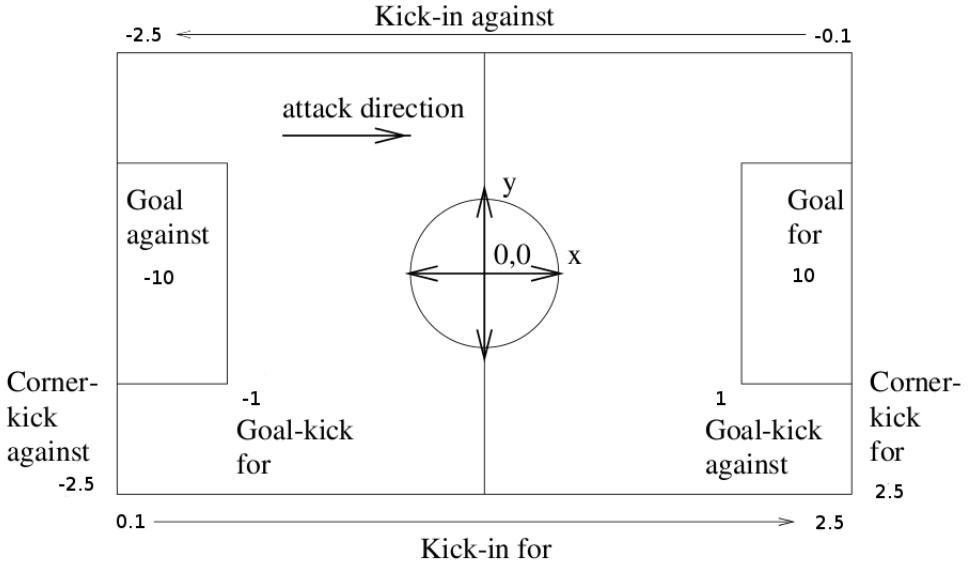


Figure 4.9: Reward system, adapted from [Stone and Veloso, 1999]

Role	We score	They score
Defenders	1	-10
Midfielders	10	-10
Attackers	10	-1

Table 4.2: Reward signal at the end of an episode

4.6.3 Bandit Problem

The variation selection at the beginning of an episode is a bandit problem. In fact, for every behaviour that has variations defined, there is a bandit problem. Every agent needs to select the variations that will cause the whole team to perform as good as possible against the current opponent.

In an initial stage, we employ the simple bandit solver presented in Section 2.1.1, using Boltzmann softmax action selection. The agents learn the reward distributions for every behaviour variation of their own, and progressively select the promising variations more. These agents are independent learners, since they

take no heed of the strategies of their teammates. The opponent is assumed to be static and part of the environment. This means that learning to perform better in the environment equals playing better against the opponent.

Note that in a bandit problem, plays are assumed to be independent. This assumption does not completely hold in our episode context, since the way an episode ends determines the way the next episode starts, allowing for various different situations. For example, an episode ends with the ball being kicked out of play by the opponent near their back line. The next episode starts with a kick-in close to the opponent's goal. Contrast this with an episode starting with the opponent taking a corner. The former situation is very favourable for our team, while the latter is favourable for the opponent.

Because we want to learn an optimal strategy that fits any situation, we do not take into account the possible differences between episodes.

4.7 Coordinating

Previous sections have described the building blocks of a learning method from a single agent's perspective, yet we have stated several times that the performance of a team in soccer largely depends on the combined strategy of the agents, and not on that of individual players. The play style of a single player can disrupt an otherwise well performing team and result in a loss[3].

The method for adaptation described in the previous section makes agents consider only their own behaviour variation selection; these are independent learners. Because the combined team effort determines the outcome of a game, we will now propose a way for players to coordinate their variation selection, in order to learn an optimal combined strategy, which we hypothesize will perform better than the strategies learned by independent learners.

In Section 4.4, we introduced variations for two behaviours, namely positioning and passing. With the coordination of the agent's positioning, it should be clear that the effectiveness of the whole team simply depends on the combination of the variations used by each respective agent. The coordination of passing is slightly different though, since the effect of a passing strategy does not depend on that of various agents, but on the passing strategy of the passer and the positioning strategy of potential receivers.

[3]For example, when every defender but one tries to put opponents offside, the one rather staying near his goal, opponent's attackers can easily hang around the goal and simply wait for a long pass to then easily score.

4.7.1 Distributed Bandit Problem

With coordination, the learning problem has now become a distributed bandit problem. In order to cooperate, agents need to learn the effect of joint actions, which are in our context combined strategies. For the positioning variations, a joint action is the set of positioning variations used by each agent. For passing, a joint action is the passing variation used by the agent itself, combined with the positioning variations of the other agents.

Reward estimates are learned for each joint action using the sample average method. Action selection is done using softmax action selection with the *combined strategy* outlined in Section 2.3.3. This strategy makes agents balance their beliefs about other agents their variation selection between

- them selecting the most optimal variations with respect to the current agent's choice, and
- them selecting variations with probabilities reflecting their variation selection in the past episodes.

We use *sliding belief windows*, which make the agents consider only a fixed number of previous episodes to calculate the probabilities of other agents selecting different variations. This in order to speed up coordination.

Because 2D simulated soccer is a fully distributed multi-agent system, and there is no memory-sharing, agents need to either communicate or observe each others actions to be able to coordinate. Observing which behaviour variations other players are using is in most cases very complex, and requires the agent to know the characteristics of other agent's behaviour variations. Because communication is available in the RoboCup Soccer Server, agents can share their variation selections for the current episode by shouting which ones they are currently employing.

Note that shouting the information once is not enough, since not all players might be in range, or other players might be shouting at the same time (which they are, to keep an up-to-date world model). Each player shouting at random intervals during the episode solves this problem. An episode generally lasts long enough for an agent to have a chance of hearing every other teammate's shouts and the knowledge is only needed at the end of the episode, because only then reward is distributed.

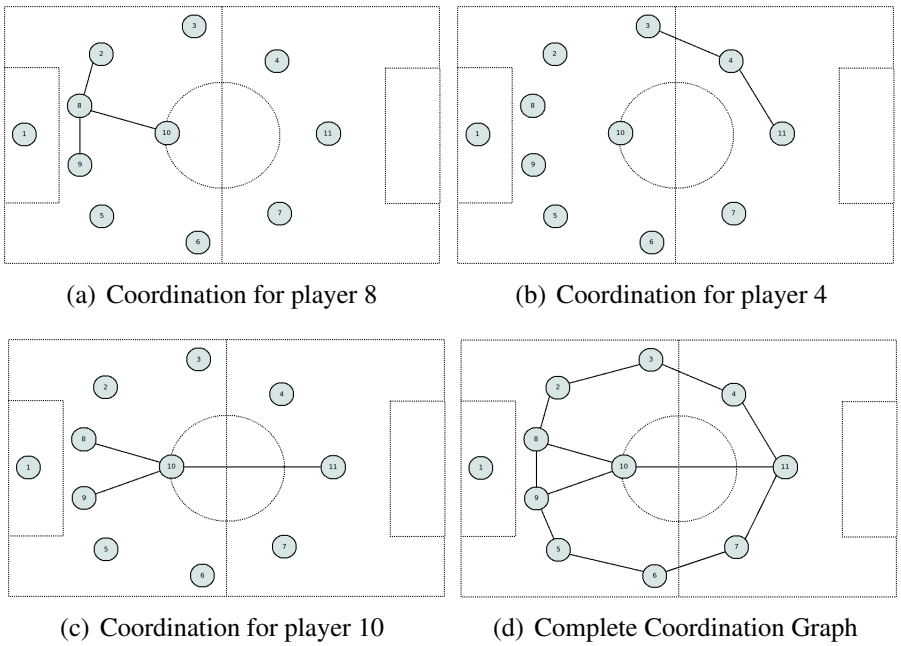
4.7.2 Reducing Complexity

A problem with reinforcement learning in a multi-agent setting, is that the complexity of the joint action space increases exponentially with the number of agents.

Each agent needs to learn a^n reward estimates, where a is the number of actions, and n the number of agents. In our case, for the positioning behaviour, this would be $4^{10} = 1048576$, and for the passing behaviour $2^{10} = 1024$, making for a total of $4^{10}2^{10} = 1073741824$ possible combinations. This results in a large search space that requires a lot of computational resources to traverse and learn an effective strategy.

In a soccer match, local coordination is more important than global coordination. The coordination of players in adjoining regions has a more direct impact on the game and can be defining for the whole team strategy. Two defenders next to each other, playing a coordinated strategy, can improve the defence of the team, while the immediate effect of coordination between a defender and an attacker is less clear, although certainly not without importance. We can use this idea of local coordination to reduce the complexity of the learning process, rendering coordination practical.

To model these local coordination problems, we make use of coordination graphs as described in Chapter 3. In such a graph, vertices represent agents, and edges represent a need for coordination between two agents.



(a) Coordination for player 8 (b) Coordination for player 4

(c) Coordination for player 10 (d) Complete Coordination Graph

Figure 4.10: The coordination graph used in the robot soccer context.

When using a coordination graph, agents only need to communicate with their immediate neighbours in the graph to coordinate. These agents are called local

joint action learners, since they only consider the joint actions with their neighbours in the coordination graph. As we have shown in Chapter 3, matching the coordination graph with the structure of the problem allows the learners to achieve better performance, while keeping computational complexity low. Since learning the graph in the robot soccer context requires too much time, we define the graph ourselves, through reasoning, using knowledge of the game of soccer.

The graph in Figure 4.10(d) shows the graph used in the experiments. This graph is constructed this way for the following reasons:

- Defenders 2, 5, 8 and 9 are connected to their neighbouring defenders, to ensure a coordinated defence
- Attackers 4, 7 and 11 are similarly connected to have a coordinated offensive strategy
- Flank midfielders 3 and 6 are connected with the defender and attacker playing on the same flank, since the strategy of a midfielder heavily influences both defence and offence and we divide the midfield into three axes: two flanks and one central axis
- Central midfielder 10, is connected with the central defenders and attacker for the same reasons
- The goalkeeper is not connected to any other players, since he has a significantly different role to fulfil within the team, which does not depend on the strategies of field players[4].

[4] Although one could argue that in standard situations like an opponent's corner, the positioning of the goalkeeper and the defenders must be coordinated

4.8 Conclusion

The goal of all algorithms and methods described above, is to help a team of agents learn to improve their performance against a specific adversarial team. Agents have different variations defined for their positioning and passing behaviours, and need to choose between these variations in order to improve the team's performance.

Independent learners try to learn independently which variations work best against the opponent, using a reinforcement learning technique to solve n-armed bandit problems.

Local joint action learners coordinate explicitly to learn which combination of variations within the team work best against the opponent. The n-armed bandit solver is therefore adapted to learn the best strategy with respect to the agent's teammates. Agents need to communicate their selections in order to coordinate.

Chapter 5

Experimental Results on Robot Soccer

In this section, we describe the results of several experiments, comparing independent learners and local joint action learners in the robot soccer setting.

5.1 The Opponent

Any team competing in the RoboCup competition must make the binaries of their implementation public. These teams would be ideal opponents to test the adaptive strategies. Building a team able to compete with these RoboCup contenders was beyond the scope of this thesis, due to time constraints. The basic code provided by the WrightEagle team is not able to keep level with any but the worst performing teams that participated in the RoboCup 2010 edition.

We built a new team to test our learning strategy against, a team which is very similar to the learning team. It uses a fixed subset of the behaviour variations from which the learning team can choose. Specifically, the defenders use the positioning system labelled 'defensive', the midfielders use the 'midfield' one and attackers use the 'offensive' one. All agents use the regular passing variation, so no deep passes are executed.

The advantage of playing against this team is that they use the same behaviours, except for the extra variations in the learning team, meaning that both teams make the same mistakes under similar circumstances.

5.2 Experiments

In the previous chapter, we described four positioning variations and two passing variations. Our first experiment consists of making independent learners develop

a strategy by learning which of these variations to choose. We compare the results of three teams playing against the opponent over a series of 350 matches, using a running average of 100 games. The first team consists of the independent learners, the second team's players use the same fixed variations as the opponent and the third team's players select randomly among the variations. Figure 5.2 shows the results of this experiment, averaged over 10 runs[1]. The Boltzmann temperature parameter is decreased over time and follows an inverse exponential function, see Figure 5.1. Initial action selection is completely random, i.e. exploration, and near the end completely greedy, i.e. exploration. This means the agents learn a pure strategy.

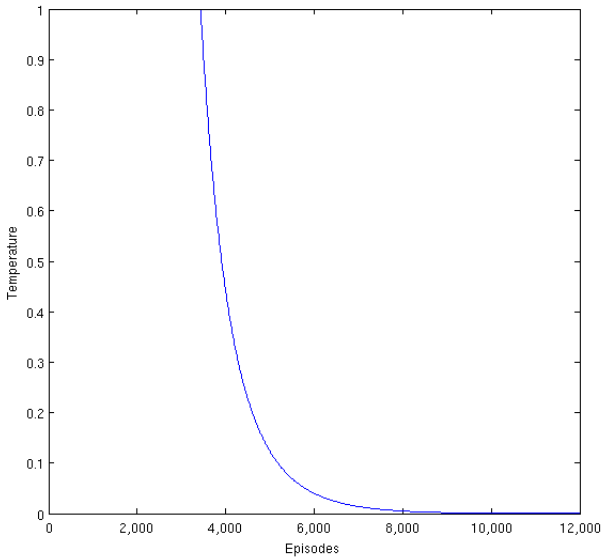


Figure 5.1: Boltzmann temperature decrease over episodes

As expected, the team playing the same fixed variations as the opponent, achieves an average of 50% wins. A more interesting fact is that the addition of a choice between variations already increases the performance of the team to a win ratio of about 57%. Note that this is not simply due to the addition of choice itself, but more specifically to the addition of one positioning variation and one passing variation that are not employed in the opponent team and seem to confer an advantage.

The learning team manages to raise their win ratio on average to about 72% after 350 matches. It is important to note that this is an average over different

[1]Note that all our soccer experiments have a very small sample size, since a soccer game requires a relatively long time, and a lot of computation power.

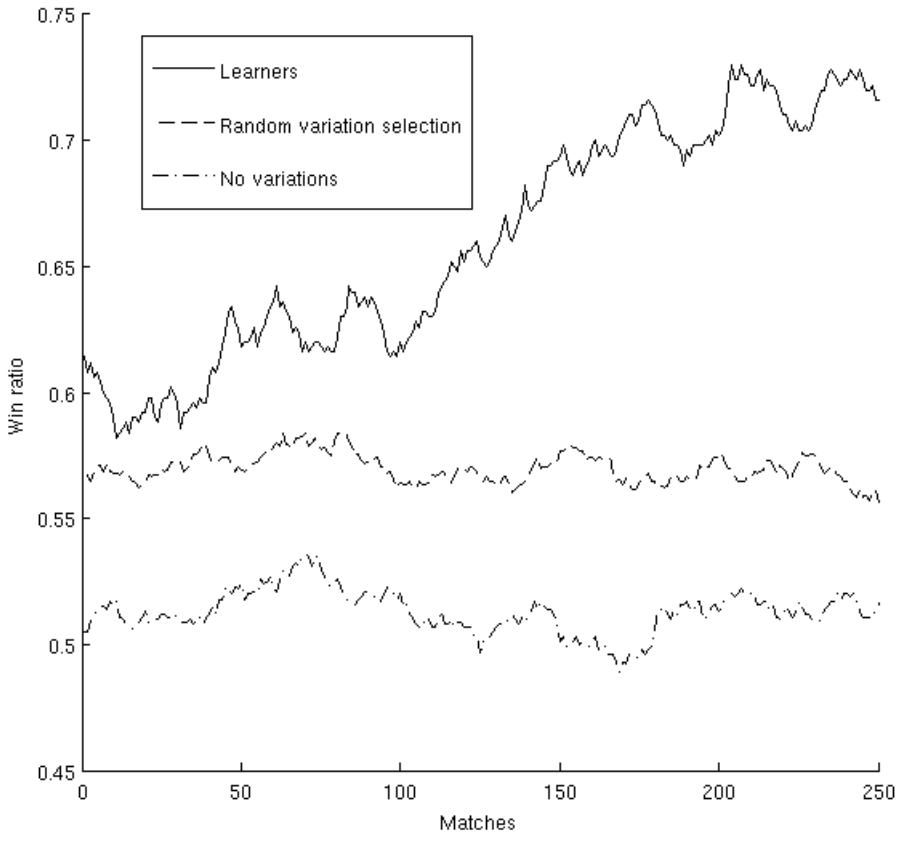


Figure 5.2: Independent learners, choosing between four positioning variations and two passing variations. Running average of 100 games.

runs. Single runs' learned strategy can vary very much in performance. Figure 5.3 shows the results of independent learners as in Figure 5.2, together with the performance graphs of the 2 best performing runs and the 2 worst performing ones. Some teams find strategies that result in a win ration of 90% and more, while others barely perform better than a random variation selection strategy.

The reason for this large difference, is the size of the joint action space. As noted earlier, with four positioning variations, two passing variations and ten players selecting variations, there are $4^{10}2^{10} = 1073741824$ strategies possible. Over 350 matches, the teams have on average played 12600 episodes. In the best case, they have tried as many different strategies. This is barely 0.001% of the total number of possible strategies. In other words, the agents cannot have explored the action space sufficiently to build reward estimates for their variations that ap-

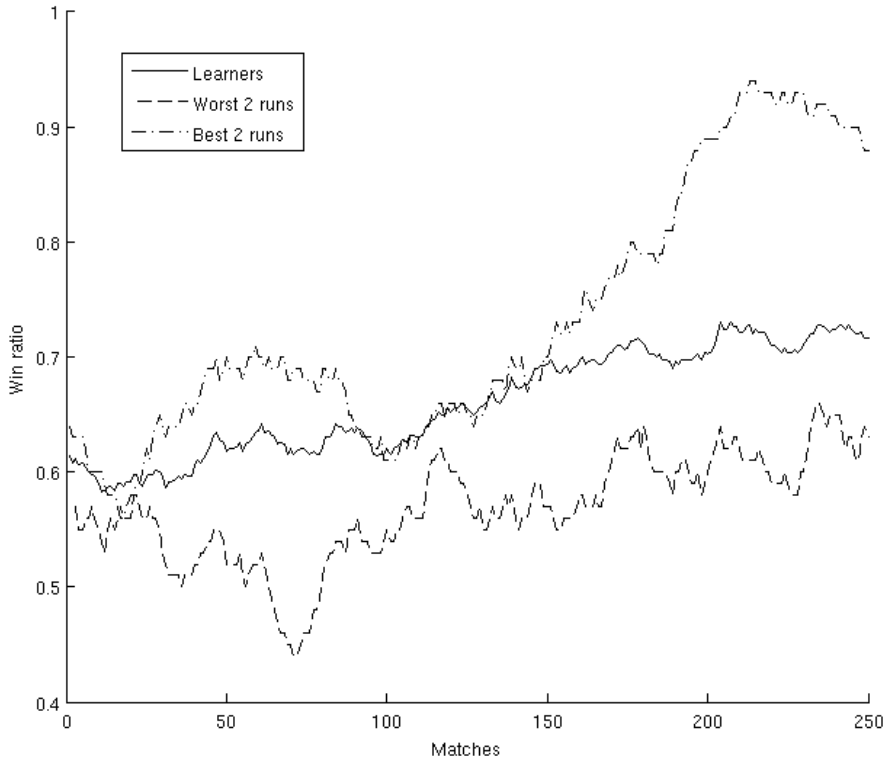


Figure 5.3: Independent learners, choosing between four positioning variations and two passing variations. Different runs result in strategies with widely varying performance. Running average of 100 games.

proach the actual values. The strategy they learn depends on the part of the action space seen. This is very clear in Figure 5.3, which shows that the runs that by accident perform well in the exploratory phase, also perform well in the exploitative phase. The agents in these runs have simply seen a part of the action space that includes some good strategies. The inverse is also true.

To decrease the size of the joint action space, we can either reduce the number of agents, or reduce the number of variations each agent can choose from. Since the RoboCup simulation league is designed for 11 players per team, and reducing the number of players increases the amount of running each player must do, making them too tired quickly, we opt to reduce the number of behaviour variations in the following experiments. We remove the through-pass, leaving only one pass variation. For the positioning behaviour, we give each agent the choice between only 2 variations: the original positioning system of WrightEagleBASE, and the

positioning variation defined for a specific role. This means that defenders choose between the original system and the new 'defensive' system, midfielders between the original and the new 'midfield' system, and attackers between the original and the new 'offensive' system. The size of the joint action space is now only $2^{10} = 1024$.

Figure 5.4 displays the results of a comparison between independent learners and local joint action learners, averaged over 18 runs, using the smaller set of variations.

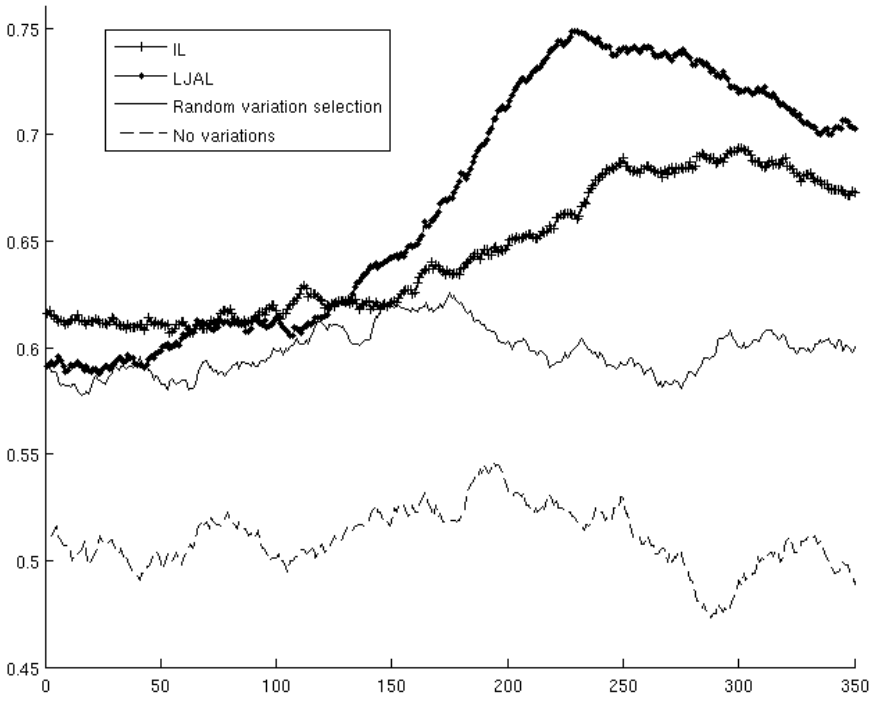


Figure 5.4: Independent learners and local joint action learners using two positioning variations compared. Running average of 100 games.

We can see again that the learners manage to improve their performance over that of the team with random variation selection. In table 5.1, we examine the statistical significance of the differences between the independent learners, local joint action learners and random variation selection. Both learners' results are significantly different than those achieved through random variation selection. Although apparently quite different on the figure, the difference between independent learners and local joint action learners is not significant. This insignificance can be

attributed to a small sample size of only 18, combined with the very noisy nature of this soccer setting, as examined in Section 5.2.1. The fact that the results are not (much) better than those achieved on a complexer action space, is probably due to a more limited choice for each agent in this configuration.

Comparison	t-test	Significant
Random - IL	$p = 0.0133 < 0.05$	Yes
Random - LJAL	$p = 0.0023 < 0.05$	Yes
IL - LJAL	$p = 0.1540 > 0.05$	No

Table 5.1: Statistical test on the significance of the differences between results obtained by different teams. Comparing ILs, LJALs and random variation selection.

Figure 5.5 shows that the results of different runs still vary a lot.

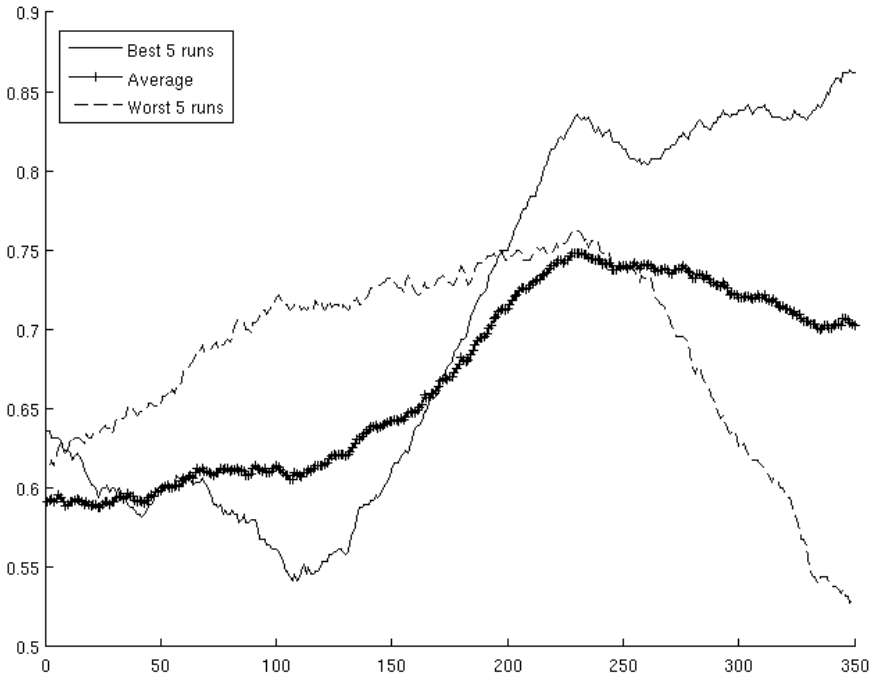


Figure 5.5: Local joint action learners using two positioning variations. Different runs result in strategies with widely varying performance. Running average of 100 games.

5.2.1 Noisy Soccer

In this section we investigate whether the large differences in performance found between similar runs in previous experiments is due to (extreme) noise in soccer.

Figure 5.6 illustrates how much variation exists in a single run; one team of local joint action learners optimizing their strategy. The first phase of the run as always is exploratory. As the softmax temperature decreases, the team converges to a strategy, Strategy 1 in the figure. This convergence happens around game 300. After some 50 games, performance suddenly drops, until around game 450, when one agent changes its positioning variation, resulting in Strategy 2. From then on the strategy does not change, yet two more significant peaks and valleys in win ratio are recorded. The win ratio running average of 100 games varies wildly between 55% and 92%, whilst playing the same strategy. The team is exactly the same at game 500 as at game 900, as in between.

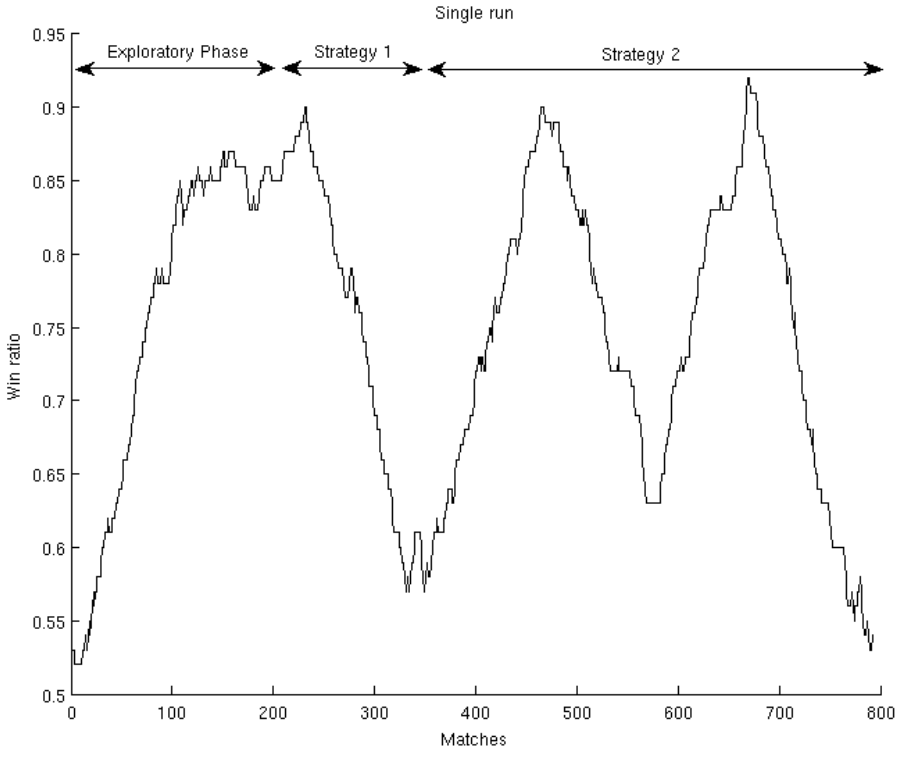


Figure 5.6: An illustration of the variability of a team's performance. Running average of 100 games.

When we evaluate Strategy 2 in the following experiment, we see that even averaged over 10 runs, there is a lot of variability in the performance with a fixed

strategy; over 20% difference between the maximum and minimum. These results are shown in Figure 5.7. Averaged over 4000 games, this strategy achieves a win ratio of $\sim 78\%$. In light of these findings, it is surprising, and a credit to RL's ability of dealing with stochasticity, that we were even able to learn strategies that improve the team's performance.

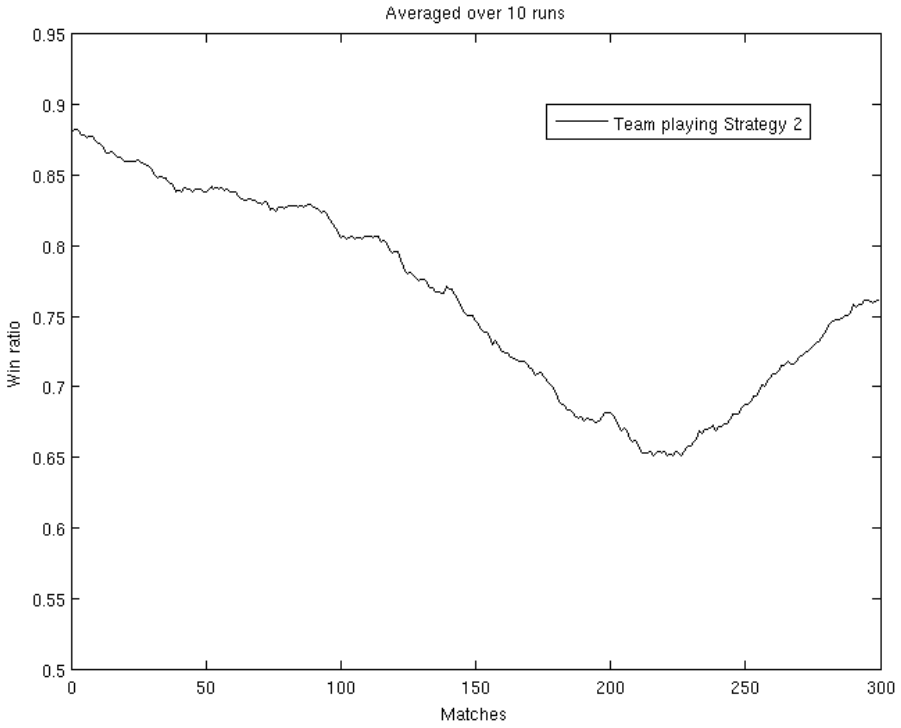


Figure 5.7: An illustration of the variability of a team's performance, playing a fixed strategy. Running average of 100 games.

5.3 Conclusion

In this chapter, we showed experimentally that our approach to adaptive strategies works, improving performance significantly against a specific opponent. We could not conclude on the difference between independent learners and local joint action learners, although local joint action learners seem to perform better and converge faster; the difference in performance was not statistically significant. The noisy nature of robot soccer and its high load on computational resources prevented a better analysis.

Chapter 6

Discussion

In this chapter, we summarize and discuss the work presented in this thesis. Our contributions and the experiments testing the hypotheses put forth are repeated and further elaborated. Further more, we propose ideas for future research, extending the research direction taken in this thesis.

6.1 Discussion

The work in this thesis is focused on multi-agent systems. These are systems in which a group of agents operates, working to solve a problem, for example traffic in a road system, where each driver strives to get at his destination as quickly as possible. In order to work optimally as a group, agents need to take into account the actions of other agents, and act accordingly. This is called coordination. In the traffic example, if each driver chooses the path that would lead to its earliest arrival at destination in an ideal situation, there will quickly be high traffic congestion on the main roads in the system, resulting in most agents arriving very late. If on the other hand drivers coordinate, they can prevent congestion by taking different routes, balancing each agent's individual need to arrive quickly, and the need for all agents in the system to do so.

Coordination between agents can be global, i.e. every agent coordinates with every other agent. Such coordination can yield a much higher performance in terms of success, since every agent has all information necessary to operate in function of the whole system, as opposed to agents that do not coordinate. The large downside of global coordination is the explosion in complexity as the number of agents increases. When taking a reinforcement learning approach to coordination in multi-agent systems, agents consider all possible combinations of actions between the agents they coordinate with. It is this amount of combinations that quickly becomes unmanageable, both memory-wise and computationally. In

real-time systems, where agents have only a relatively small amount of time for decision making, this is an important issue.

To address this complexity problem, we can employ local coordination, as opposed to global. In such a system, agents only coordinate with a subset of all the agents. This coordination is formalized using a coordination graph, where vertices represent agents, and edges between agents represent coordination. Optimizing the behaviour of all agents becomes more difficult, because most agents will not possess all information. The variable elimination [Guestrin et al., 2002] and max-plus [Kok and Vlassis, 2005] algorithms are ways to approximate the globally best behaviour, by calculating conditional strategies for optimal action selection, using a message passing scheme and a coordination graph.

The approach to local coordination we propose also involves coordination graphs, although it does not require information to travel around the graph, only between directly coordinating agents, nor does it require bi-directional coordination, i.e. it is possible for one agent to adapt its strategy to that of another, while the other is unaware of the former agent, or chooses to ignore him. On the other hand, there are no guarantees for optimality of this approach. The way we propose for agents to coordinate, is for each agent to use reinforcement learning to learn which combinations of actions among the agents he coordinates with, is optimal. Locally optimal that is. By making each agent optimize its local sub-problem, we attempt to approach the performance achieved through global optimization. We call such learners local joint action learners (LJAL), as opposed to independent learners (IL), who do not explicitly coordinate, and joint action learners (JAL), who coordinate globally. These last two are effectively special cases of LJALs, with respectively a fully disconnected and fully connected coordination graph.

Experiments conducted in Section 3.3 showed that, on completely random problems, the amount of coordination strongly correlates both with solution quality and computation time, coordination graphs of higher density leading to better solutions, but longer computation times.

Many real-world problems are not completely random though; they often support an inherent structure that favours coordination between certain agents, where the effect of a single agent's action does not depend on that of all other agents, but on a small subset of them. This structure can be exploited by local coordinators, when the coordination graph is specifically modelled for the problem, as shown in Section 3.4.2. Correctly modelling this graph is very important, since we also showed that the addition of even one coordination edge too many can lead to a drop in solution quality.

Since the definition of the coordination graph is so important to the success of the group of agents, we developed a way to let the agents optimize this graph, instead of requiring the developer of the system to define it. Each agent is allowed to choose one coordination partner, or none, constructing a coordination

graph with at most as many directed edges as there are agents. This graph is then evaluated on the problem, and with this evaluation in mind, the agents again choose a coordination partner. We showed in Section 3.5.2 that over time, this graph can be optimized to better solve a problem. Even though this is quite a sparse graph, we showed that it can model problems with an inherent structure of much higher density, through what we call implicit coordination. Agents are seen to develop directed paths between agents, allowing all agents on the path to coordinate, although not every pair of agents is directly connected, nor is there explicit propagation of information. These are interesting results, since they show that coordination and high performance are possible with low computational and communication complexity.

In the second part of this thesis, we applied our findings in the domain of coordination to the problem of adaptive strategies in robot soccer.

In real soccer, teams have a general strategy, which is supposed to be their best strategy against any opponent. When they play an actual match though, they often adapt this strategy to fit the opponent, in the hopes of playing better against them. These are often slight changes that are expected to counter specific 'features' of the other team, such as making a specific defender mark his opponent in a different way, because that opponent is known to have a specific style of play.

These teams change their strategies based on advice of scouting staff and statistical analyses of the opponent, i.e. these changes are externally imposed upon the team. In our approach to adaptive strategies in simulated soccer, we make the team change its strategy through self-evaluation, while actually playing against the opponent, or at least a model of the opponent.

For different high-level behaviours, we defined several variations. For example a positioning variation where a player prefers to stay close to an opponent at all times, or one where the agent rather stays relatively clear from them. Then we divided a match into distinct episodes, which span from play-interruption to play-interruption, e.g. from kick-off to a goal scored. Before such an episode, each agent must choose which variation of each behaviour he will employ during this episode, if necessary. During the episode, the game unfolds, and at the end, the success of the chosen variations is evaluated. When a goal is scored, the combination of chosen variations is rewarded, while a goal received penalizes these variations. Over the course of several episodes, agents must learn which combinations of strategy work best against the opponent, thus optimizing the team strategy.

Since robot soccer is a dynamic, real-time setting, agents have relatively little computation time to decide which actions to take. Because learning to optimize the team strategy through global coordination takes computationally too long, resulting in agents reacting slowly, we applied our local coordination findings to this problem. Because of time constraints, we were not able to learn a coordination

graph optimized for this problem, but had to define one ourselves, although using sound reasoning, see Section 4.7.2. We showed in Section 5.2, that the learning agents manage to adapt their strategy to an opponent much like themselves, reaching a win ratio of over 70% on average, though some teams reached 90% and more. We also showed that there is a high variability in results obtained when playing soccer games, even with a fixed strategy, accounting for the variability in results of 'optimized' strategies.

6.2 Future Work

Both in local coordination as in adaptive strategies, there are a lot of research avenues available for future research. We list some that continue the work exposed in this thesis:

- The meta-bandit learning a coordination graph is composed of ILs in this thesis. Of course, they could be any kind of LJAL. We should investigate what kind of coordination graphs are best for this meta-bandit, which itself is learning a coordination graph. Maybe the meta-bandit can even use the same coordination graph it is learning, improving its own ability to learn a coordination graph online.

- Until now we limited the learning of a coordination graph to one outgoing edge per agent. Of course, this can be generalized to a maximum of n edges per agent. We need to investigate whether adding new meta-levels on top of the current graph learning bandit may work to allow the agents to learn denser coordination graphs. Other optimization methods, such as genetic algorithms can also be evaluated and compared to the reinforcement learning of the graph.

- In this thesis, we focused on repeated games or single state decentralized Markov decision processes (DEC-MDP), i.e. solving the same problem over and over, improving the solution quality over time. A lot of problems in reinforcement learning are sequential decision problems though, so testing our local coordination contribution on these multi-state problems is a logical continuation of our research. The influence that different states have on the performance of a coordination graph can be evaluated, and a technique to adapt the graph to different states may need to be developed. We can also investigate how our approach relates to DEC-MDPs with sparse interactions (DEC-SIMDP). Varying the density of the coordination graph according to the perceived need for coordination may be one way to go.

- We should compare our LJALs with other algorithms aiming to reduce complexity using local coordination on these criteria: communication and computational complexity, or scalability, flexibility (ability to adapt to the problem's structure) and most importantly, solution quality. The state of the art algorithms are max-plus [Kok and Vlassis, 2005] and utile coordination [Kok et al., 2005].

- Our reward scheme for episodes in robot soccer is not perfect, since it equally rewards different agents, even different variations, whether they had much influence on the result of the episode or not. Techniques that better estimate the influence every variation had in an episode should significantly improve the learning of an optimal strategy, speeding up learning by removing a lot of noise. Reward shaping can be used to add extra reward for objectives that lead to a goal, e.g. moving the ball closer to the opponent's goal. Eligibility traces is another reinforcement learning technique that can be used to estimate which variations contributed most to the end result. As a behaviour variation is used, its trace is set to 1, which then decays as time moves on, converging to zero at infinity. When reward is received, this reward is multiplied by the eligibility trace, resulting in more reward for more recently used variations.

- As noted earlier, we defined the coordination graph ourselves for the robot soccer experiments, because optimizing it as described in this thesis was impossible due to time constraints. In the future, we should investigate how we can optimize this coordination graph in simpler settings, making learning practical, and then transferring it to the full soccer game context.

6.3 Conclusion

The first major contribution discussed in this dissertation concerns local coordination in multi-agent systems. We developed a way for agents to coordinate locally, as opposed to globally, reducing the high complexity associated with multi-agent systems, resulting in much faster computations for each agent. Through local coordination, the agents can exploit the inherent structure most problems have, approaching the solution quality achieved through global coordination, while keeping complexity very low. We also show how agents can learn a coordination graph that is optimized to solve a problem with a specific structure.

The second contribution presented in this thesis is a way to approach adaptive strategies in robot soccer, applying our findings in local coordination to this complex problem. We have shown that it is possible for a team of soccer agents to build a team strategy that is optimized to counter a specific opponent, using behaviour variations as building blocks.

Appendix A

Learning a Coordination Graph

In this chapter we visualize the optimization of a coordination graph over time, as described in Section 3.5. The structure of the problem is shown in Figure A.1, and the optimization in Figure A.2

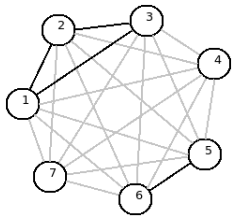
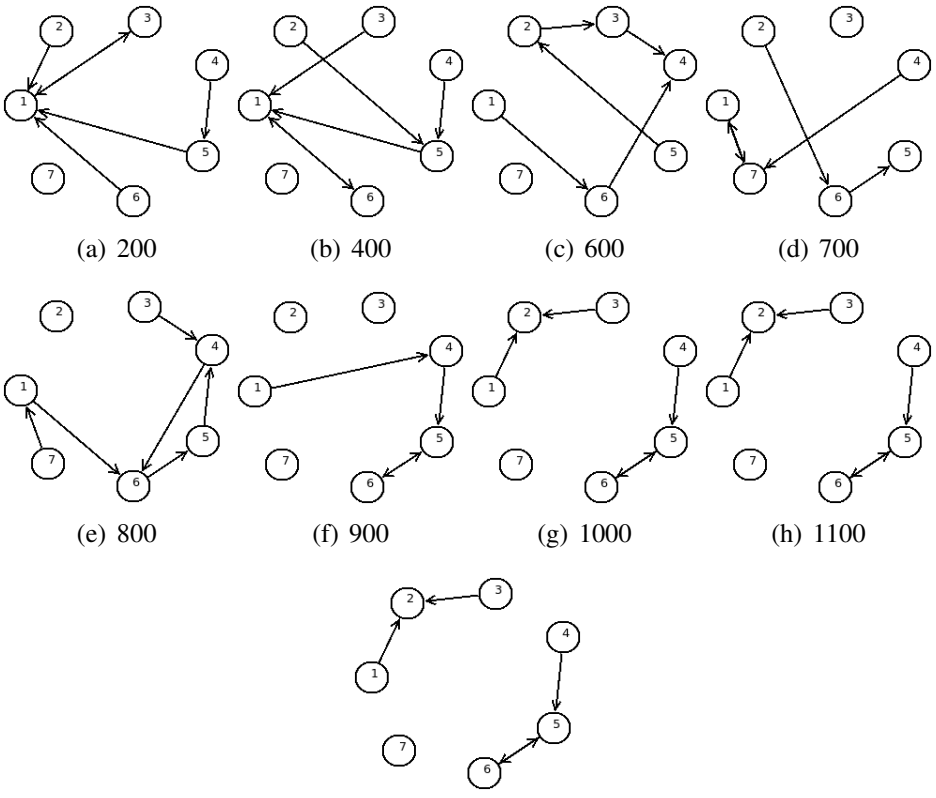


Figure A.1: The distributed constraint optimization problem.



(a) 200 (b) 400 (c) 600 (d) 700

(e) 800 (f) 900 (g) 1000 (h) 1100

(i) 1200

Figure A.2: Optimization of a coordination graph over time. Captions indicate the timestamp of the snapshot.

Bibliography

[Acar et al., 2010] Acar, B., Asta, S., Avcuoglu, S., Sarabi, M. H., Itauma, I., Kavasoglu, Z., Oztokmak, C., Sozmen, O., Topaloglu, N., and Sariel-Talay, S. (2010). beestanbul robocup 3d simulation league team description paper 2010. Technical report, Articial Intelligence and Robotics Laboratory Istanbul Technical University Computer Engineering Department Istanbul, TURKEY.

[Akiyama and Shimora, 2010] Akiyama, H. and Shimora, H. (2010). Helios2010 team description. Technical report, Information Technology Research Institute, AIST, Japan.

[Bai et al., 2010] Bai, A., Wang, J., Lu, G., Wang, Y., Zhang, H., Zhu, Y., Shi, K., and Chen, X. (2010). Wrighteagle 2d soccer simulation team description 2010. Technical report, Multi-Agent Systems Lab., School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui Province, China.

[Bowling et al., 2004] Bowling, M., Browning, B., and Veloso, M. (2004). Plays as effective multiagent plans enabling opponent-adaptive play selection. In *ICAPS-04 Proceedings*, pages 376–383.

[Chechetka and Sycara, 2006] Chechetka, A. and Sycara, K. (2006). No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, AAMAS '06, pages 1427–1429, New York, NY, USA. ACM.

[Chen et al., 2002] Chen, M., Foroughi, E., Heintz, F., Huang, Z., Kapetanakis, S., Kostiadis, K., Kummeneje, J., Noda, I., Obst, O., Riley, P., Steffens, T., Wang, Y., and Yin, X. (2002). *User Manual Robocup Soccer Server for Soccer Server version 7.07 and later*.

[Claus and Boutilier, 1998] Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of National Conference on Artificial Intelligence (AAAI-98*, pages 746–752.

[Devlin et al., 2010] Devlin, S., Grzes, M., and Kudenko, D. (2010). Multi-agent reinforcement learning with reward shaping for keepaway takers. In *Proceedings of AAMAS 2010 Workshop on Adaptive and Learning Agents (ALA 2010), Toronto, Canada*.

[Fathzadeh et al., 2006] Fathzadeh, R., Mokhtari, V., Mousakhani, M., and Mahmoudi, F. (2006). Mining opponent behavior: A champion of robocup coach competition. In *Robotics Symposium, 2006. LARS '06. IEEE 3rd Latin American*.

[Guestrin et al., 2002] Guestrin, C., Lagoudakis, M., and Parr, R. (2002). Coordinated reinforcement learning. In *In Proceedings of the ICML-2002 The Nineteenth International Conference on Machine Learning*, pages 227–234.

[Iglesias et al., 2009] Iglesias, J., Fernández, J., Villena, I., Ledezma, A., and Sanchis, A. (2009). The winning advantage: Using opponent models in robot soccer. In Corchado, E. and Yin, H., editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2009*, volume 5788 of *Lecture Notes in Computer Science*, pages 485–493. Springer Berlin / Heidelberg.

[Jamzad et al., 2002] Jamzad, M., Sadjad, B., Mirrokni, V., Kazemi, M., Chitsaz, H., Heydarnoori, A., Hajiaghai, M., and Chiniforooshan, E. (2002). A fast vision system for middle size robots in robocup. In Birk, A., Coradeschi, S., and Tadokoro, S., editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Computer Science*, pages 159–203. Springer Berlin / Heidelberg.

[Johansson and Saffiotti, 2002] Johansson, S. J. and Saffiotti, A. (2002). Using the electric field approach in the robocup domain. In *RoboCup 2001: Robot Soccer World Cup V. Volume 2377 of Lecture Notes in Artificial Intelligence*, pages 399–404. Springer.

[Kalyanakrishnan et al., 2007] Kalyanakrishnan, S., Liu, Y., and Stone, P. (2007). Half field offense in robocup soccer: A multiagent reinforcement learning case study. In Lakemeyer, G., Sklar, E., Sorrenti, D., and Takahashi, T., editors, *RoboCup 2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Computer Science*, pages 72–85. Springer Berlin / Heidelberg. 10.1007/978-3-540-74024-7_7.

[Kitano et al., 1997] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., and Matsubara, H. (1997). Robocup: A challenge problem for AI. *AI Magazine*, 18(1):73–85.

[Kok et al., 2005] Kok, J. R., Jan, P., Bram, H., and Vlassis, B. N. (2005). Utile coordination: Learning interdependencies among cooperative agents. In *In Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG'05)*, pages 29–36.

[Kok and Vlassis, 2005] Kok, J. R. and Vlassis, N. (2005). Using the max-plus algorithm for multiagent decision making in coordination graphs. In *RoboCup-2005: Robot Soccer World Cup IX*.

[Latombe, 1991] Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA.

[Lewis, 2003] Lewis, M. (2003). *Moneyball : the art of winning an unfair game*. W.W. Norton.

[Littman, 1994] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann.

[Marian et al., 2010] Marian, S., Luca, D., Sarac, B., and Cotarlea, O. (2010). Oxsy 2010 team description. Technical report, OXygen-SYstems laboratory, Str. Constantin Noica, Bl.5, Sc.C, Ap.36, C.P. 550169, Sibiu, ROMANIA.

[Modi et al., 2003] Modi, P. J., Shen, W.-M., Tambe, M., and Yokoo, M. (2003). An asynchronous complete method for distributed constraint optimization. In *In AAMAS*, pages 161–168.

[Petcu and Faltings, 2005] Petcu, A. and Faltings, B. (2005). A scalable method for multiagent constraint optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 266–271, Edinburgh, Scotland.

[Reis et al., 2010] Reis, L. P., Lau, N., and Mota, L. (2010). Fc portugal 2d simulation: Team description paper. Technical report, DEI/FEUP – Informatics Engineering Department, Faculty of Engineering, Univ. of Porto, Portugal.

[Riley and Veloso, 2002] Riley, P. and Veloso, M. (2002). Planning for distributed execution through use of probabilistic opponent models. pages 72–81. *Best Paper Award*.

[RoboCup.org, 2011] RoboCup.org (2011). Robocup.org. http://www.robocup.org/.

[Röfer et al., 2010] Röfer, T., Laue, T., Graf, C., Kastner, T., Fabisch, A., and Thedieck, C. (2010). B-human team description for robocup 2010. Technical report, Deutsches Forschungszentrum für Künstliche Intelligenz, Sichere Kognitive Systeme, Bremen, Germany.

[Russell and Norvig, 2003] Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

[Sherback et al., 2006] Sherback, M., Purwin, O., and D'Andrea, R. (2006). Real-time motion planning and control in the 2005 cornell robocup system. In Kozlowski, K., editor, *Robot Motion and Control*, volume 335 of *Lecture Notes in Control and Information Sciences*, pages 245–263. Springer Berlin / Heidelberg.

[Stone, 2000] Stone, P. (2000). *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, Cambridge, MA, USA.

[Stone et al., 2000] Stone, P., Riley, P., and Veloso, M. (2000). Defining and using ideal teammate and opponent models. In *Proceedings of the Twelfth Annual Conference on Innovative Applications of Artificial Intelligence*.

[Stone et al., 2005] Stone, P., Sutton, R. S., and Kuhlmann, G. (2005). Reinforcement learning for RoboCup-soccer Keepaway. *Adaptive Behavior*, 13(3):165–188.

[Stone and Veloso, 1999] Stone, P. and Veloso, M. (1999). Team-partitioned, opaque-transition reinforcement learning. In *Proceedings of the third annual conference on Autonomous Agents*, AGENTS '99, pages 206–212, New York, NY, USA. ACM.

[Stone and Veloso, 2000] Stone, P. and Veloso, M. (2000). Layered learning. In de Mántaras, R. L. and Plaza, E., editors, *Machine Learning: ECML 2000 (Proceedings of the Eleventh European Conference on Machine Learning)*, pages 369–381. Springer Verlag, Barcelona,Catalonia,Spain.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press.

[Sycara, 1998] Sycara, K. P. (1998). Multiagent systems. *AI Magazine*, 19:79–92.

[Takahashi et al., 2005] Takahashi, Y., Edazawa, K., Noma, K., and Asada, M. (2005). Simultaneous learning to acquire competitive behaviors in multi-agent

system based on a modular learning system. *Proceedings of the Meeting of Special Interest Group on AI Challenges*.

[Taylor et al., 2011] Taylor, M. E., Jain, M., Tandon, P., Yokoo, M., and Tambe, M. (2011). Distributed on-line multi-agent optimization under uncertainty: Balancing exploration and exploitation. *Advances in Complex Systems (ACS)*, 14(03):471–528.

[WrightEagle, 2011] WrightEagle (2011). WrightEagle. http://wrighteagle.org/en/robocup/index.php.