Coordination and Learning in Cooperative Multiagent Systems

Jelle R. Kok

Coordination and Learning in Cooperative Multiagent Systems

Academisch Proefschrift

ter verkrijging van de graad van doctor

aan de Universiteit van Amsterdam

op gezag van de Rector Magnificus

prof. mr. P. F. van der Heijden

ten overstaan van een door het college voor promoties ingestelde

commissie, in het openbaar te verdedigen in de Aula der Universiteit

op vrijdag 3 november 2006, te 10:00 uur

door

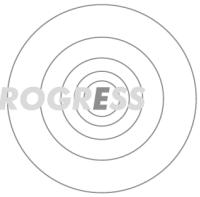
Jelle Rogier Kok

geboren te Amsterdam

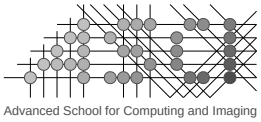
Promotiecommissie:

Promotor:	Prof. dr. ir. F. C. A. Groen
Copromotor:	Dr. N. Vlassis
Overige leden:	Prof. drs. M. Boasson
	Prof. dr. H. J. Kappen
	Prof. dr. ir. J. A. La Poutré
	Dr. M. van Someren
	Dr. P. Stone

Faculteit der Natuurwetenschappen, Wiskunde en Informatica



This research has been performed at the IAS group of the University of Amsterdam and has been supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs, and the Technology Foundation STW, project AES.5414.



This work has been carried out in the ASCI graduate school.
ASCI dissertation series number 133.

ISBN-10: 90-9021073-3
ISBN-13: 978-90-9021073-5

Contents

1

Introduction

This thesis focuses on *distributed cooperative decision making*. Simply stated, decision making is the process of selecting a specific action out of multiple alternatives. This process occurs continuously in daily life. Humans, for example, have to decide what clothes to wear, which route to take to work, what political party to vote for, etc. Decision-making problems are always centered around a decision maker, often referred to as an *agent*, which is defined as anything that is situated in an environment and acts, based on its observations of the environment and its prior knowledge about the domain, to accomplish a certain goal. This general definition does not only describe humans, but also robotic and software agents. As a running example for this chapter we consider the individual players in a soccer team as agents. Each agent has to choose an action based on the observed positions of the players on the field and the game plan decided upon before the match. In this example, the soccer players might be humans, but can also be robotic agents that perceive their environment with cameras and control the ball with simple kicking devices.

We are interested in the design of *intelligent* decision-making agents, which is one of the major goals of artificial intelligence (AI). Contrary to other approaches which measure success related to human performance [Rich and Knight, 1991], we adopt an ideal concept of intelligence, that is, we try to construct agents that are both *autonomous* and *rational* [Poole et al., 1998; Russell and Norvig, 2003]. Autonomous implies that the agent makes its decisions without the guidance of a user. Rational means that the agent selects those actions that are expected to achieve the best expected outcome based on the available information. A rational soccer player that is in control of the ball, for example, will always pass the ball to the best positioned teammate, even when he dislikes this teammate.

Agents are seldom stand-alone systems, but often coexist and interact with other agents. This thesis is involved with problems consisting of *multiple* decision makers that have to work together in order to accomplish their goal. This complicates the problem significantly. Not only can the decision of one agent influence the outcome of the other agents, but the total number of possible action combinations also grows exponentially with the increase of the number of agents. In this thesis, we investigate scalable, distributed solution techniques that can cope with these complications and still ensure that the individual agents select actions resulting in coordinated behavior.

1.1 Multiagent systems

Distributed artificial intelligence is a subfield of AI which is concerned with systems that consist of *multiple* independent entities that interact in a domain. Traditionally, this field has been broken into two sub-disciplines: *distributed problem solving* (DPS) and *multiagent systems* (MASs) [Sycara, 1998; Weiss, 1999; Stone and Veloso, 2000; Vlassis, 2003]. DPS focuses on developing solutions using the collective effort of multiple agents by combining their knowledge, information, and capabilities [Durfee, 2001]. It is often applied to solve large problems by decomposing them into smaller subtasks, each assigned to a different agent, and can therefore be regarded as a tightly-coupled top-down approach. A MAS, on the other hand, is a loosely-coupled bottom-up approach that aims to provide principles for the construction of complex systems consisting of multiple independent agents, and focuses on the coordination of the behaviors of the agents in such systems [Stone and Veloso, 2000]. In principle, the agents in a MAS can have different, even conflicting, goals. However, in this thesis we are interested in cooperative MASs in which the different agents form a team with the same goal. This is an important topic in AI since many large-scale applications, for example, robotics and the Internet, are formulated in terms of spatially or functionally distributed entities. Collaboration enables the different entities to work more efficiently and to complete activities they are not able to accomplish individually. More specifically, the use of a MAS has the following advantages [Sycara, 1998; Stone, 1998; Vlassis, 2003]:

- The existence of multiple agents can speed up the operation of a system because the agents can perform the computations in *parallel*. This is especially the case for domains in which the overall task can be decomposed into several independent subtasks that can be handled by separate agents.

- A MAS usually has a high degree of *robustness*. In single-agent systems a single failure can cause the entire system to crash. A MAS on the other hand degrades gracefully: if one or several agents fail, the system will still be operational because the remaining agents can take over the workload.

- The modularity of a MAS enables one to add new agents to the system when necessary, and therefore has a high *scalability*. Adding new functionality to a monolithic system is often much more difficult.

- A MAS can make observations and perform actions at multiple locations simultaneously, and therefore it can take advantage of *geographical distribution*.

- A MAS usually has a higher *performance-cost ratio* than single-agent systems. A single robot with all the necessary capabilities for accomplishing a task is often much more expensive than the use of multiple, cheaper, robots each having a subset of these capabilities.

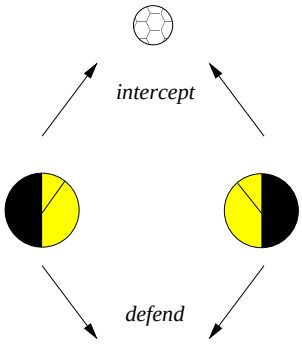


Figure 1.1: An example coordination problem in which each of the two soccer agents should select a different action.

A major challenge is to formalize these type of problems and construct solutions to coordinate the different behaviors of the agents. Application domains in which these types of coordination problems are addressed include network routing [Boyan and Littman, 1994; Dutta et al., 2005], sensor networks [Lesser et al., 2003; Modi et al., 2005], but also distributed spacecraft missions [Clement and Barrett, 2003], tactical aircraft simulation [Marc et al., 2004], and supply chain management [Chaib-draa and Müller, 2006].

1.2 Coordination

We are interested in fully cooperative multiagent systems in which all agents share a common goal. Each agent selects actions individually, but it is the resulting *joint action* that produces the outcome. A key aspect in such systems is therefore the problem of *coordination*: the process that ensures that the individual decisions of the agents result in (near-)optimal decisions for the group as a whole. Fig. 1.1 shows a simple example of a coordination problem in which each of the two soccer agents has to choose between two tasks: intercept the ball or move to a defensive position. Without knowing the choice of the other agent, the goal of the agents is to select an action that differs from the other agent. The coordination problem addresses the important question how the agents should select their individual actions.

Boutilier [1996] assumes the agents know the outcome of each possible joint action, and thus are able to determine the set of all optimal joint actions, also called equilibrium in this context. In the example in Fig. 1.1 this corresponds to the two joint actions in which the actions differ. Now, the coordination problem is simplified to the problem of agreeing on a single joint action from this set. Boutilier [1996] gives three different solution techniques to this problem: communication, social conventions, and learning. Communication allows each agent, in a predefined sequence, to inform the other agents of its action, restricting the choice of the other agents. Social conventions

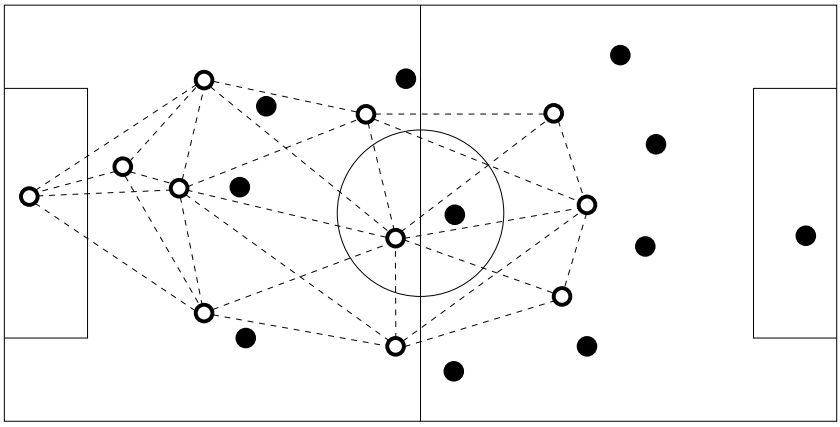


Figure 1.2: An example coordination graph which models the dependencies between the agents on a soccer field. Only nearby agents are connected.

are constraints on the action choices of the agents. Beforehand, the agents agree upon a priority ordering of agents and actions. When an action has to be selected, each agent derives which actions the agents with a higher priority will perform, and selects its own action accordingly. A common example of a social convention is the right-of-way rule in traffic in which an agent coming from the right has priority in crossing a crossroad. The agent coming from the right knows it has the highest priority and decides to drive through (action with highest priority). The other agent can derive the action of the first agent, and decides to stop. Finally, learning methods can be applied to learn the behavior of the agents through repeated interaction.

The problem with the aforementioned approaches, however, is that they require the computation of all optimal joint actions. This becomes infeasible for problems with many agents because the number of action combinations grows exponentially with the number of agents. Fortunately, in many problems the action of one agent does not depend on the actions of all other agents, but only on a small subset of the agents. For example, in many real-world domains only nearby agents have to coordinate their actions, and agents which are positioned far away from each other can act independently. In such situations, we can model the dependencies between the agents using a coordination graph (CG) [Guestrin et al., 2002a]. In a CG, each node represents an agent, and an edge between agents indicates a local coordination dependency. Instead of specifying the outcome for every possible joint action combination, in this case only the outcome for each local dependency has to be specified. This allows for the decomposition of the global coordination problem into a combination of simpler problems. Fig. 1.2 shows an example of a CG that models the coordination dependencies between the agents on a soccer field. Only interconnected agents have to coordinate their actions. The goalkeeper, for example, only has to coordinate its action directly with the four defenders, and can ignore the actions of the other players.

1.3 Sequential decision making

The coordination problem addresses the problem of selecting a coordinated action for a specific situation, or state. *Sequential decision making* considers problems in which an agent, or a group of agents, has to perform a *sequence* of actions in order to reach a certain goal. In our soccer domain, for example, a large number of (coordinated) actions have to be performed before a goal is scored.

We first review sequential decision-making problems from the perspective of a single agent. In such settings the agent repeatedly interacts with its environment. This interaction is often described using a sense-think-act loop [Russell and Norvig, 2003]. The agent observes the environment using its sensors, based on this information it decides what action to choose, and finally executes the chosen action. The executed action on its turn influences the environment, and the loop repeats itself when the agent observes the changed environment. The effect of the agent's action on the environment might be uncertain, that is, the action does not always have the desired result. For example, when a player decides to pass the ball to a certain teammate, it might not always reach this teammate due to noise in the execution of the kick or noise in the movement of the ball.

Different mathematical models exist to specify sequential decision-making problems [Puterman, 1994; Bertsekas and Tsitsiklis, 1996]. Such models define, among others, the transition probabilities for reaching a new state when a specific action is performed in the current state. In order to be computationally tractable, many models assume that both the states and actions are discrete, but extensions exist to cope with continuous-valued state and action representations. Furthermore, such models specify a reward that the agent receives after performing an action in a specific situation. In our soccer domain, for example, the model could specify that an agent receives a positive reward when it scores a goal.

The goal of an agent is to optimize a performance measure based on the received rewards. Often, this corresponds to finding a policy, that is, a function that selects the best action for each possible situation. In our example, the agent should thus learn to select a sequence of actions that result in scoring a goal. This is a complicated problem because a specific decision can have long-term effects and its particular outcome often depends on the future actions that will be performed. However, different solution techniques exist to compute the optimal action for a specific situation. These methods can roughly be divided into two separate fields: decision-theoretic planning [Boutilier et al., 1999] and reinforcement learning [Sutton and Barto, 1998]. Planning methods have access to the complete model description, and thus to the probabilistic effect of each action. Therefore such methods are able to compute a course of actions by considering future situations before they are actually experienced. Reinforcement-learning techniques, on the other hand, do not have access to the model descriptions and have to learn based on observations and rewards received while interacting with the environment. An important consequence of the fact that the agent does not have access to the model is that the agent occasionally has to explore the environment by taking sub-optimal actions in order to experience all possible situations.

The existence of multiple agents in an environment has severe consequences on the characteristics and the complexity of the problem. Complications arise, among others, because the state and action representations grow exponentially with the number of agents, the action of an individual agent can have a different effect on the decisions of the other agents, the different agents have different (partial) views of the current world state, or the agents have to choose one out of multiple equilibria. Many different model extensions exist to model the existence of multiple agents in an environment [Boutilier, 1996; Bernstein et al., 2002; Pynadath and Tambe, 2002; Guestrin, 2003; Becker et al., 2003; Goldman and Zilberstein, 2004]. However, solution methods constructed for single-agent sequential decision-making problems are usually not directly applicable to these models. In this thesis, we investigate scalable, distributed solution techniques that can cope with the complications resulting from the existence of multiple agents.

1.4 Objective of the thesis

In this thesis, we focus on solution methods to coordinate the actions of a group of cooperative agents. Our main goal is to develop scalable, distributed solution techniques that are able to cope with the complications imposed by the existence of multiple agents in the environment, while still ensuring that the individual agents select actions that result in coordinated behavior. Our main approach is to only consider the actual dependencies that exist between the agents. These are modeled using the framework of coordination graphs, which is used extensively in the subsequent chapters of this thesis.

We consider two different types of problems in this thesis. The first type of problems are coordination methods that compute a coordinated joint action for a given situation when the dependencies and the corresponding outcomes are specified beforehand. We present a method to compute a coordinated joint action in a distributed manner for large groups of agents with many dependencies (Chapter 3), and show how the CG framework can be extended to coordinate the agents in dynamic and continuous domains (Chapter 6). Second, we consider methods to learn the coordinated behavior of the agents in sequential decision-making problems in which the agents do not have access to the model description. We will investigate solution methods in which the agents learn to coordinate their actions when the dependencies between the agents, but not the actual outcomes, are fixed and given beforehand (Chapter 4). Furthermore, we also describe a method in which the agents learn to coordinate their actions when the dependencies differ based on the current situation, and learn the coordination dependencies automatically (Chapter 5).

For every method, we present experimental results to illustrate how our methods can be applied to typical discrete AI problems, but also show how our coordination methods are successfully applied to large continuous domains as the RoboCup simulation domain [Chen et al., 2003].

1.5 Outlook

The structure of this thesis is as follows. In Chapter 2, we give an overview of different existing mathematical models for sequential decision making in stochastic domains for both single- and multiagent systems. We both describe the main characteristics and give the formal descriptions. Furthermore, we discuss several existing solution techniques to compute an optimal policy for the agents in the described models.

In Chapter 3, we focus on coordinating the behavior of a group of cooperative agents in a given situation. Specifically, we investigate problems in which the agents have to decide on a joint action, which results from their individual decisions, maximizing a given payoff function. This payoff function is represented using a coordination graph which models the relevant dependencies between the agents. To compute the joint action, we propose the *max-plus algorithm*, a payoff propagation algorithm that is the decision-making analogue of belief propagation in probabilistic graphical models. This results in a fast approximate alternative to existing exact algorithms.

In Chapter 4, we consider sequential decision-making problems in which the agents repeatedly interact with their environment and learn to optimize the long-term reward they receive from the system. We present a family of model-free multiagent reinforcement-learning techniques, called *sparse cooperative Q-learning*, which approximate the global action-value function based on the topology of a fixed coordination graph, and perform local updates using the contributions of the individual agents to the maximal global action value. In combination with the max-plus algorithm this approach scales linearly in the number of dependencies of the problem.

In Chapter 5, we also focus on learning the behavior of a group of agents in multiagent sequential decision-making problems, but now extend our sparse cooperative Q-learning approach to cope with dependencies between the agents that differ based on the current context. We first introduce a straightforward method, called *sparse tabular multiagent Q-learning*, in which, depending on the context, either all or none of the agents coordinate their actions. Then, we present our *context-specific sparse cooperative Q-learning* approach which models the coordination dependencies between subsets of agents using a context-specific coordination graph in which the dependencies can change based on the current situation. Furthermore, we describe our *utile coordination* approach that learns the coordination dependencies of a system automatically based on gathered statistics during the learning phase.

In Chapter 6, we present a method to ensure that multiple robots in a dynamic, continuous, domain select coordinated actions. Our approach is to assign roles to the agents based on the continuous state information and then coordinate the different roles using context-specific coordination graphs. We demonstrate that, with some additional common knowledge assumptions, an agent can predict the actions of the other agents, making communication superfluous. Furthermore, we describe how we successfully implemented the proposed method into our UvA Trilearn simulated robot soccer team which won the RoboCup-2003 World Championships in Padova, Italy.

Finally, in Chapter 7 we present some final conclusions and suggests some interesting directions for future research.

2

A Review of Markov Models

In this chapter, we describe several mathematical models for sequential decision making in stochastic domains for both single- and multiagent systems. It introduces many of the basic concepts, methods, and notations used in the subsequent chapters. For a more in-depth review, we refer the reader to Puterman [1994]; Bertsekas and Tsitsiklis [1996]; Sutton and Barto [1998]; Russell and Norvig [2003].

2.1 Introduction

A woman orders a white wine in a restaurant. A truck driver turns left at a crossroad. A soccer player shoots at the goal. All these examples have in common that the performed action has been preceded by a decision, that is, the selection of one action out of multiple alternatives. Another common property is that the decision depends on the circumstance in which it was taken. For example, the woman's wine choice depends on the food she is ordering, the decision of the truck driver is based on the direction from which he approaches the crossroad, and the soccer player's decision to shoot depends on his distance to the goal and the position of the goalkeeper. In this chapter, we review several models for *decision making* in stochastic environments, and discuss different algorithms to compute the best action for a specific situation.

Decision-making problems are centered around the decision maker, or agent. An *agent* is anything that perceives its environment through sensors ands acts upon that environment through actuators [Russell and Norvig, 2003]. This general definition describes not only humans (with eyes as sensors and hands as actuators) and robotic agents (with cameras as sensors and robotics arms as actuators), but also software agents (with keyboard input as sensors and a graphical user interface as actuator).

In a *sequential decision-making problem* an agent repeatedly interacts with its environment and tries to optimize a performance measure based on rewards it receives. It is difficult to determine the best action in each situation because a specific decision can have a long-term effect and its particular outcome often depends on the future actions that will be performed. Referring back to our previous examples, it is impossible for the soccer player to reverse its decision to shoot after he has kicked the ball. Furthermore, even when turning left is part of the shortest route to the truck driver's destination, taking a different direction and following a longer, more familiar, route

might bring the truck driver quicker to his destination, when turning left brings him into an unfamiliar part of town.

We assume the sequential decision-making problems considered in this thesis all obey the *Markov property*. This implies that the current situation provides a complete description of the history, and previous information is irrelevant for making a decision. The woman's wine choice, for example, does not depend on what she drank earlier in the day, and the truck driver does not take into account the roads he crossed earlier for his current decision. Furthermore, we assume that the decision-making agent is both *autonomous* and *rational*. Autonomous means that the agent is capable of making decisions on its own, and thus without the guidance of a user. Rational means that the agent should select actions to maximize a given performance measure based on the available information received from its sensors and its knowledge about the problem.

In this thesis we mainly deal with sequential decision-making problems involving *multiple* agents, and address the question how the agents in a *group* can take the right decision in a certain situation. As we describe in more detail in Section 2.3, having multiple agents interact with the environment and, more importantly, with each other, has severe consequences on the characteristics and the complexity of the problem. We restrict our attention to cooperative multiagent systems in which the agents have to work together in order to achieve a common goal, that is, optimize the given performance measure. This differs from self-interested approaches [Shapley, 1953; Littman, 1994] in which each agent tries to maximize its own performance.

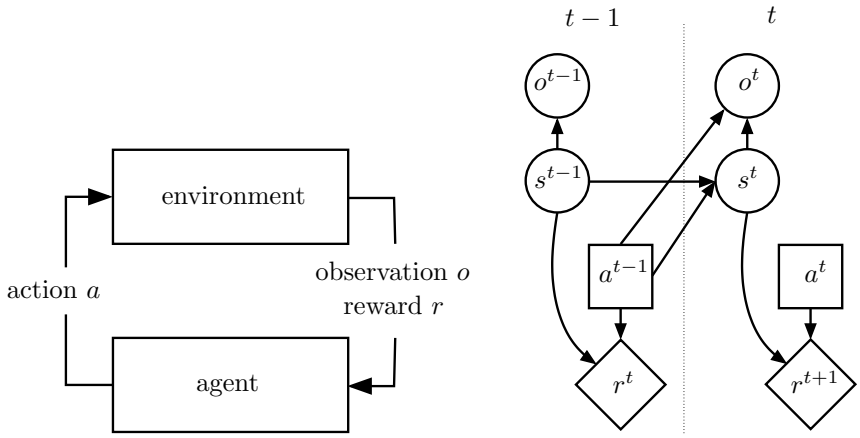
In this chapter we describe different models for sequential decision making and algorithms for solving them. First, we review single-agent models in Section 2.2 and then extend these to incorporate multiple agents in Section 2.3. We end with a discussion in Section 2.4. As a running example we consider a multiagent system consisting of a team of eleven soccer players. When considering single-agent models, we take the perspective from a single agent and ignore the other agents.

2.2 Single-agent models

This section describes a framework for sequential decision-making problems involving a single agent. First, we give some general characteristics of the model in Section 2.2.1, followed by a formal description in Section 2.2.2. Next, two instances of this formal model, a Markov decision process (MDP) and a partially observable MDP (POMDP), are described in more detail in Section 2.2.3. Finally, we describe some existing techniques to compute the best action for a specific situation in an MDP in Section 2.2.4.

2.2.1 Characteristics

As stated earlier, a sequential decision-making problem is a problem in which an agent repeatedly interacts with its environment in order to optimize a performance measure. Each interaction with the environment consists of several steps. First, an agent makes observations, for example, through its sensors, about the current *state*



(a) Agent loop. (b) Graphical model.

Figure 2.1: Two different representations of an agent interacting with its environment. (a) depicts the dependencies between the agent and its environment (b) shows the corresponding graphical model for two consecutive time steps.

of the environment. The state can be regarded as the collective information of the environment relevant for the decision at a particular instance. Based on the current state, the agent then determines an action and executes it. According to a stochastic transition model which is based on the current state and the selected action, the environment then changes to a new state. Furthermore, the environment provides the agent feedback to evaluate the new situation. This feedback is often represented as a scalar reward. After the agent has observed the new state it again selects an action, and the sequence repeats itself.

Fig. 2.1(a) depicts the general structure of an agent interacting with its environment. Note that from the perspective of the agent this structure follows a functional decomposition in three phases: sensing, thinking, and acting. This is therefore often referred to as the sense-think-act, or perception-cognition-act, loop.

The observation of an agent might not be identical to the exact true state. This can be the result of two different reasons. First, it can be the result of *noise* in the agent's sensors. For example, the same state might provide the agent different observations at different time steps due to inaccurate or malfunctioning sensors. A second cause is *perceptual aliasing*, the effect that different states result in the same observation. For example, a soccer player might receive the same perception when it observes one of the four corner posts. When this is the only information received by the agent, it is not possible to infer in which of the four corners the agent is positioned.

It is the objective of the agent to select actions that optimize a performance measure specified in terms of the received rewards. A formal model describing the model parameters of a sequential decision-making problem, including possible performance measures, is given in the following section.

2.2.2 Formal description

Many existing single-agent models for sequential decision making are derived from a general model and are distinguished by assumptions about the parameters of the general model. Next, we give an overview of the relevant model parameters for single-agent systems and discuss some related issues. Most of the theory is adapted from Puterman [1994]; Kaelbling et al. [1996]; Sutton and Barto [1998]. For simplicity, we focus on *discrete* environments which have a finite number of states and actions.

Parameters

A finite, discrete sequential decision-making problem can be specified using the following model parameters:

- A discrete time step $t = 0, 1, 2, 3, \ldots$.
- A finite set of environment states S. A state $s^t \in S$ describes the state of the world at time step t.
- A finite set of actions $\mathcal{A}$. The action selected at time step t is denoted by $a^t \in \mathcal{A}$.
- A finite set of observations Ω. An observation $o^t \in \Omega$ provides the agent with information about the current state s^t.
- A state transition function $T : S \times \mathcal{A} \times S \rightarrow [0, 1]$ which gives the transition probability $p(s^t|a^{t-1}, s^{t-1})$ that the system moves to state s^t when the action a^{t-1} is performed in state s^{t-1}. Since an action selected in a state possibly results in different outcomes, the environment is called *stochastic*, or *non-deterministic*. We do not consider *deterministic* environments in which every action has a single unique effect.
- An observation function $O : S \times \mathcal{A} \times \Omega \rightarrow [0, 1]$ which defines the probability $p(o^t|s^t, a^{t-1})$ the agent perceives observation o^t in state s^t when action a^{t-1} was performed in the previous time step. In some cases the previous state s^{t-1} is also incorporated in the observation function. The probability of receiving observation o^t is then defined as $p(o^t|s^t, a^{t-1}, s^{t-1})$. Unless explicitly stated, we ignore the previous state s^{t-1} in the observation function.
- A reward function $R : S \times \mathcal{A} \rightarrow \mathbb{R}$ which provides the agent with a reward $r^{t+1} = R(s^t, a^t)$ based on the action a^t taken in state s^t. The reward is assumed deterministic and is always the same for a specific state-action pair. We do not consider non-deterministic reward structures.

In the specifications of the transition function, the next state s^{t+1} only depends on the state s^t and the action a^t. In the more general case, the dynamics would be defined in terms of the complete history

$$p(s^{t+1}|a^t, s^t, a^{t-1}, s^{t-1}, \ldots, a^0, s^0), \tag{2.1}$$

for all possible values of the past events. With such a representation, however, it quickly becomes intractable to store the probabilities for every possible trajectory. A common assumption is therefore that the environment has the Markov property. This implies that the state of the world at time t provides a complete description of the history before time t. We can then ignore all information before time t and simplify (2.1) to $p(s^{t+1}|s^t, a^t)$. Fig. 2.1(b) shows a graphical model depicting the dependencies between the variables in two consecutive time steps.

The environment is called *stationary*, or *static*, when the reward and transition probabilities are independent of the time step t. The action executed by an agent thus always has the same, possibly probabilistic, effect on the environment. An environment is called *non-stationary*, or *dynamic*, when these probabilities change over time. We focus on stationary transition functions. In our notation we therefore sometimes omit the time step t superscript when referring to a state s^t, and use the shorthand s' for the next state s^{t+1}.

Performance measures

The goal of the agent is to select actions that optimize a performance measure related to the received rewards. The most common measure is the *expected return* R^t, which is a specific function based on the reward sequence $r^t, r^{t+1}, r^{t+2}, \ldots$. We discuss two common definitions for the expected return. The first defines the return as the total, cumulative rewards collected from time step t onward:

$$R^t = r^t + r^{t+1} + r^{t+2} + r^{t+3} + \ldots + r^T = \sum_{k=t}^{T} r^k, \tag{2.2}$$

in which T is the final time step. This measure applies to *episodic* tasks in which the interaction with the environment is divided into episodes. After T steps, or when a terminal state is reached, the episode ends and the system resets to a starting state.

In *continuing* tasks there are no terminal states and the system continues indefinitely. Since in this case the sum in (2.2) is infinite, a standard approach is to discount future rewards and weigh rewards in the near future more. Formally, the expected *discounted return* is specified as

$$R^t = r^t + \gamma r^{t+1} + \gamma^2 r^{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r^{t+k}, \tag{2.3}$$

where γ, $0 \leq \gamma < 1$, is the *discount rate*.

For $\gamma < 1$, rewards received in the near future are favored more valuable than later received rewards. As γ decreases, this effect is more apparent; in the extreme case $\gamma = 0$ the agent only tries to maximize the immediate received reward. Unless stated otherwise, we consider discounted returns in the remainder of the thesis.

Policies

An agent selects actions based on its *policy*, or *strategy*, π. A deterministic policy is a function that maps the current state to a single action: $\pi : S \rightarrow \mathcal{A}$. A randomized, or stochastic, policy is a function that maps the current state to a probability distribution over all possible actions: $\pi : S \times \mathcal{A} \rightarrow [0, 1]$. The probability of taking action a^t in state s^t is denoted by $p(a^t|s^t)$.

The objective of an agent is to select an action at each time step that maximizes its performance measure. This corresponds to computing an optimal policy π^* which for every state returns the action that is optimal, given that π^* is also used to select actions in the future. In Section 2.2.4, we review several existing techniques for computing an optimal policy π^*.

Factorized representations

Until now, we used an explicit or *extensional* state representation in which a state is defined as an element from the finite set S. However, in many cases it is more convenient to describe a state in terms of a set of properties or features instead of enumerating all possible combinations explicitly [Boutilier et al., 1999; Guestrin et al., 2003]. In such an *intentional* representation the state is described via a set of m random variables $\mathbf{S} = \{S_1, \ldots, S_m\}$, where each $s_i \in S_i$ takes on a value from the domain S_i.[1] The current state $\mathbf{s} \in \mathbf{S}$ is defined as the cross-product of all individual features, or *factors*, s_i: $\mathbf{s} = s_1 \times s_2 \times \ldots \times s_m$. For example, the state of a soccer player on the field can be defined by three features: its x-position, y-position, and its orientation.

Such a representation is not only more descriptive, but also allows one to represent large problems compactly by using a *factorized representation* for the transition function. In this representation the new value of a state variable only depends on a subset of all state variables. A turn action performed by a soccer player, for example, only changes its orientation and is therefore independent of its x and y-position.

A factorized state transition function can be represented compactly by a *dynamic Bayesian network (DBN)* [Dean and Kanazawa, 1989; Jensen, 2001]. The *transition graph* of a DBN, for a given action a, is a two-layered directed acyclic graph in which the layers correspond to two consecutive time steps. The nodes in the first layer, $\mathbf{S} = \{S_1, \ldots, S_m\}$, represent the state variables related to a given time t, while the nodes in the second layer, $\mathbf{S}' = \{S'_1, \ldots, S'_m\}$, represent the state variables related to time $t + 1$. Directed arcs between the nodes indicate a probabilistic dependency between the corresponding features. $\texttt{Parents}(S'_i)$ denotes the variables on which node S'_i depends. For simplicity, we assume that all arcs are *diachronic*, that is, $\texttt{Parents}(S'_i) \subset \mathbf{S}$, and therefore no dependencies exist between nodes in the same time slice (*synchronic arcs*).

The value of a state variable $s'_i \in S'_i$ in time step $t+1$ is conditioned on the values of its immediate parents. Each node S_i is therefore associated with a *conditional probability distribution (CPD)* $P_a(S'_i|\texttt{Parents}(S'_i))$ which specifies the probability of

[1] A vector of two or more variables is emphasized using a bold notation.

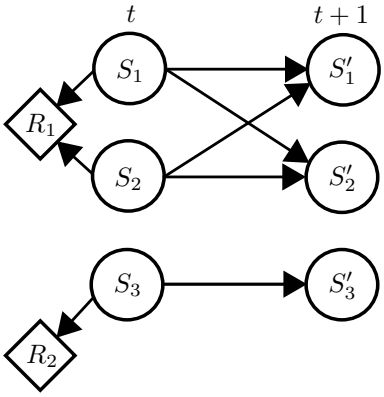


Figure 2.2: An example dynamic Bayesian network. The circles in the network represent state variables and the diamonds represent reward variables. The arcs denote directed dependencies between the connected variables.

its new value for every possible combination of its parents' values, given the action a. The global transition probability function T is defined by

$$p(\mathbf{s}'|\mathbf{s}, a) = \prod_{i=1}^{m} p_a(s'_i|\mathbf{s}[\texttt{Parents}(S'_i)]) \tag{2.4}$$

where $\mathbf{s}[\texttt{Parents}(S'_i)]$ represents the values of the variables in $\texttt{Parents}(S'_i)$ for the state $\mathbf{s}$. This representation can be very compact since in many problems only a small subset of all state variables influence the new value of a state. Furthermore, the (in)dependencies can also be exploited by planning and decision-making algorithms, resulting in additional computational savings [Jesse Hoey and Boutilier, 1999; Schuurmans and Patrascu, 2002; Guestrin et al., 2003]. Note that actions are not directly included in the DBN, and a separate model exists for every action.

The reward function is modeled as a linear combination of local reward functions, each depending on a subset of all state variables. When we define $R_i^a : S \rightarrow \mathbb{R}$ as the ith local reward function when performing action a, the global reward function R is defined as

$$R(\mathbf{s}, a) = \sum_{i=1}^{k(a)} R_i^a(\mathbf{s}[\texttt{Scope}[R_i^a]]), \tag{2.5}$$

where $k(a)$ is the number of local reward functions for action a and $\mathbf{s}[\texttt{Scope}[R_i^a]$ is the assignment of the variables in $\mathbf{s}$ that are associated with R_i^a.

Fig. 2.2 shows an example DBN with a factorized transition and reward function. In our soccer example, the depicted DBN might correspond to a movement command in which the state variables s_1 and s_2 specify the position (x and y-position) of the agent, and s_3 specifies the position of the ball. The new position of the agent depends both on the x and y-position of the previous state, while the new position of the ball

is independent of these values. The reward function is also decomposed into two independent functions; one is related to the position of the ball, while the other is related to the agent position. The reward related to the ball position is, for instance, only positive when the ball is inside the opponent goal and thus does not depend on the position of the agent.

2.2.3 Existing Models

Next, we describe two models that are derived from the general model described in Section 2.2.2.

Markov decision process (MDP)

A *Markov decision process (MDP)* [Puterman, 1994; Bertsekas, 2000] is a sequential decision-making problem in which the current state is fully observable to the agent, that is, the observation in time step t equals s^t. This model follows the general model from Section 2.2.2 with the additional assumption that the set of observations equals $\Omega = S$ and the only non-zero observation probability is $p(o^t = s^t | s^t, a^{t-1}) = 1$.

Because the agent observes the current state without uncertainty and an MDP obeys the Markov property, the agent can directly map its current observation $o^t = s^t$ to a new action $a^t = \pi(s^t)$. Such a policy is called a *reactive* or *memoryless* policy.

A *factored MDP* [Boutilier et al., 1999; Guestrin et al., 2003] is an MDP in which the transition and reward function are factorized as described in Section 2.2.2.

Partially observable MDP (POMDP)

The general model in Section 2.2.2 is identical to a partially observable Markov decision process (POMDP) [Lovejoy, 1991; Kaelbling et al., 1998]. Contrary to fully observable domains, a received observation now only provides *partial* information about the current state. A memoryless policy might lead to suboptimal behavior in such settings since the received observations do not provide a full description of the current state, and thus do not provide a Markovian signal to the agent. Observations received in the past might provide additional information about the current state that can improve the decision of the agent. For example, when the soccer player saw the opponent goal to its left in its previous observation it should be able to derive in which corner it is currently positioned. In order to take into account the information of all previous observations, a common approach is to keep track of a belief state [Dynkin, 1965; Kaelbling et al., 1998]. A belief state is a probability distribution over all possible states and summarizes all information from the past. Using the known transition and observation model, the belief state is updated after each observation to compute the probability of the environment being in each state of the underlying MDP. The agent makes its decisions based on this belief state. This is a much more difficult problem than the fully observable case, since the agent has to find a mapping from a continuous space of probability distributions (over states) to actions, instead

of discrete states to actions. In the remainder of the thesis we only consider partially observable domains briefly. We mainly focus on solution techniques that are able to cope with the complexities related to the existence of multiple cooperative agents in an environment. We will not deal with the additional complexities resulting from the uncertainty in the observations, since these solution methods require completely different techniques than in the fully observable case.

2.2.4 Solution techniques

In this section we discuss several solution techniques to compute an optimal policy π^* for a given MDP. An optimal policy should for every possible situation return the action that maximizes the performance measure. It is well-known that a stationary and deterministic policy is sufficient for solving an MDP optimally [Howard, 1960; Puterman, 1994]. We distinguish between model-based and model-free techniques. Model-based techniques, also referred to as *planning* methods, require a complete description of the model, while model-free techniques, also referred to as reinforcement learning [Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998], only learn based on the received observations and rewards.

Value iteration

Dynamic programming (DP) refers to a collection of algorithms that compute an optimal policy given a complete description of the model as an MDP. *Value iteration* [Puterman, 1994] is a model-based dynamic programming algorithm which specifies the optimal policy in terms of a value function $V^\pi : S \rightarrow \mathbb{R}$. A value function for a policy π returns for every state an estimate of the expected discounted return when actions are selected according to the policy π:

$$V^\pi(s) = E\Big[\sum_{t=0}^{\infty} \gamma^t R(s^t, \pi(s^t)) | s^0 = s\Big], \tag{2.6}$$

where the expectation operator $E[\cdot]$ averages over stochastic transitions and $\gamma \in [0, 1]$ is the discount factor. The value function can also be defined recursively, and is then referred to as the Bellman equation [Bellman, 1957]:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^\pi(s'). \tag{2.7}$$

The expected return for a state s equals the sum of the immediate received reward and the expected discounted return from the next state. Since transitions are stochastic, the latter is the sum of the expected return for every state s' multiplied with the probability that s' will be reached after performing the action according to π.

The optimal policy can be derived from the optimal value function V^* that solves the so-called Bellman optimality equation,

$$V^*(s) = \max_{a \in \mathcal{A}} \Big[R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s')\Big], \tag{2.8}$$

and maximizes the expected return for each state s. Solving this set of nonlinear equations simultaneously is cumbersome for large problems [Puterman, 1994]. Instead, we can transform (2.8) into a recursive update, known as the Bellman backup:

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \Big[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s') \Big]. \tag{2.9}$$

For each iteration, the value function of every state s is updated one step further into the future based on the current estimate. The concept of updating an estimate based on the basis of other estimates is often referred to as bootstrapping. The value function is updated until the difference between two iterations, V_k and V_{k+1}, is less than a small threshold. For an arbitrary V_0, the sequence $\{V_k\}$ is known to converge to V^* [Puterman, 1994]. The optimal policy is then derived using:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \Big[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \Big]. \tag{2.10}$$

When using value iteration, each iteration involves a complete sweep over the state space. For larger problems, this might involve many irrelevant updates which do not change the value of a state. An alternative is therefore to update the states using *asynchronous* DP algorithms that backup the states in an arbitrary order, for example, based on the expected change in the value function. The algorithm will still converge to the optimal policy [Bertsekas and Tsitsiklis, 1989] as long as every state is eventually updated.

A different DP algorithm, which we will not review in detail, is policy iteration [Howard, 1960; Bertsekas and Tsitsiklis, 1996] that learns the policy directly. This algorithm starts with an initial random policy π, and iteratively updates the policy by first computing the associated value function V^π by solving the linear system (2.7), and then improving the policy based on V^π using an update step as in (2.10).

Q-learning

This thesis is mostly concerned with model-free methods in which the agent has no access to the transition and reward model. Specifically, we focus on Q-learning, which is a widely used reinforcement-learning technique [Watkins, 1989; Sutton and Barto, 1998]. The agent represents its policy π in terms of Q-functions, or action-value functions, which represent the expected future discounted reward for a state s when selecting a specific action a, and then following the specific policy π. The objective of the agent is to learn the optimal Q-function, $Q^*(s,a)$, which satisfies the Bellman optimality equation,

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q^*(s',a'). \tag{2.11}$$

Note that this equation is identical to (2.8) if we define $V(s) = \max_{a \in \mathcal{A}} Q(s,a)$.

When the transition probabilities $p(s'|s,a)$ are not available, it is not possible to apply an iteration step as (2.9) to update the Q-values. Instead, Q-learning starts with an initial estimate for each state-action pair. Each time an action a is taken in state s, reward $R(s,a)$ is received, and next state s' is observed, the corresponding Q-value $Q(s,a)$ is updated with a combination of its current value and the temporal-difference error. The latter is the difference between the current estimate $Q(s,a)$ and the expected discounted return based on the experienced sample $R(s,a) + \gamma \max_{a'} Q(s',a')$. The update is defined as

$$Q(s,a) := Q(s,a) + \alpha[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)], \tag{2.12}$$

where $\alpha \in [0,1]$ is an appropriate learning rate which controls the contribution of the new experience to the current estimate. The idea is that each experienced sample brings the current estimate $Q(s,a)$ closer to the optimal value $Q^*(s,a)$.

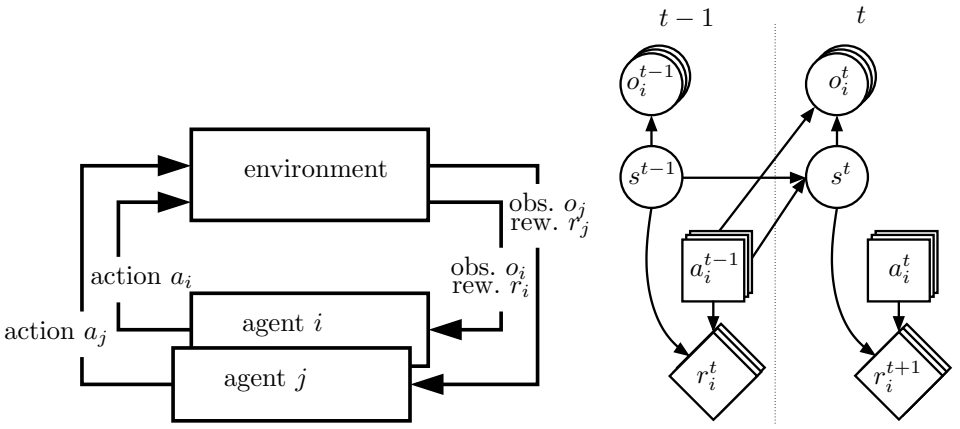
Contrary to value iteration, Q-learning performs updates based on *on-line* experiences, which depend on the action choices of the agent. In order to select an action at a particular time step the agent uses an *exploration strategy*. The most commonly used strategy is ϵ-greedy which selects the greedy action, $\arg\max_{a \in \mathcal{A}} Q(s,a)$, with high probability, and, occasionally, with a small probability ϵ, selects an action uniformly at random. This ensures that all actions, and their effects, are experienced. When every state-action pair is associated with a unique Q-value and every action is sampled infinitely often, for example, using the ϵ-greedy action selection method, iteratively applying (2.12) converges in the limit to the optimal $Q^*(s,a)$ values [Watkins and Dayan, 1992; Tsitsiklis, 1994]. Given Q^*, the optimal policy π^* is simply obtained by selecting the greedy action for every state s.

An alternative model-free approach to learn the optimal policy π^* is to first estimate the model based on the experienced state transitions, and then solve this model using standard DP techniques like value iteration. More common, however, is to combine the two approaches and perform updates based on real experienced samples and samples simulated from a, continuously changing, estimated model [Sutton and Barto, 1990; Barto et al., 1995].

2.3 Multiagent models

Contrary to a single-agent system in which only one agent interacts with the environment, a *multiagent system (MAS)* consists of multiple agents which are all executing actions and influence their surroundings [Sycara, 1998; Weiss, 1999; Vlassis, 2003]. Each agent receives observations and selects actions individually, but it is the resulting *joint action* which influences the environment and generates the reward for the agents. This has severe consequences on the characteristics and the complexity of the problem. Fig. 2.3(a) shows a graphical description, similar to Fig. 2.1(a), of different agents interacting with their environment.

We describe the general characteristics for sequential decision-making problems with multiple agents in Section 2.3.1. We mainly focus on cooperative MASs in



(a) Agent loop. (b) Graphical model.

Figure 2.3: Two representations of multiple agents interacting with their environment. Fig. (a) depicts the dependencies between two agents and their environment. Fig. (b) shows the related graphical model for two consecutive time steps.

which the agents have to optimize a shared performance measure. This is followed by a general model description in Section 2.3.2. We give an overview of several existing multiagent models in Section 2.3.3, and describe existing model-based and model-free solution techniques in Section 2.3.4.

2.3.1 Characteristics

A MAS consists of multiple agents interacting with their environment. A resulting complication is the growth in both the action and the state space when a new agent is added to the system. Since the total number of joint actions is defined as the cross-product of the individual action sets, the action space scales exponentially with the number of agents. Usually, the same holds for the state space since every agent is at least related to one state feature. Next, we list several other fundamental characteristics of a MAS in more detail.

Dynamic environment

In most single-agent systems the environment is assumed to be static, which means that the transition and reward function do not depend on the time step t. However, when other agents are part of the environment the new state and received reward also depend on the actions selected by the other agents. As a consequence, the environment becomes dynamic from the perspective of a single agent. For example, when a player on a soccer field passes the ball to a teammate, the outcome depends on the behavior of the receiving agent. The receiving agent might anticipate the pass, but it is also

possible that it ignores it. The behavior of the other agent can change over time, resulting in a dynamic environment.

Dynamic environments are more difficult to handle than static environments since the same action can have different effects based on factors an agent is not able to influence. This might lead to oscillated behavior [Claus and Boutilier, 1998], and therefore requires solution techniques in which the agents actively synchronize and coordinate their behavior.

Homogeneous and heterogeneous agents

Agents in a MAS are either *homogeneous* or *heterogeneous*. Homogeneous agents are constructed in the same way and have identical capabilities. Examples are hardware robots manufactured by the same factory process or copies of software agents. On the other hand, heterogeneous agents have different designs and different capabilities. For example, two soccer robots are heterogeneous when they have different sensors to determine their position or are able to traverse the field with different velocities. Agents with the same underlying hardware/software structure but with different behaviors are often also called heterogeneous. We mainly consider homogeneous agents.

Control

The control of a MAS is decentralized. Each agent selects an action individually, but the system is affected by the joint action, that is, the combination of all selected actions. In order to reach a coordinated joint decision, each agent has to use the available knowledge about the other agents to reason about their possible action choices. Specific coordination mechanisms can be applied to ensure that the agents select their individual actions in agreement with those of the other agents [Boutilier, 1996; Vlassis, 2003]. Possible examples, which are discussed in more detail later, are to assume that specific knowledge is shared by all agents or to use communication.

Knowledge

Knowledge is the information an agent has about the world and the task it has to solve. An agent accumulates this information from different sources, and uses it to select its actions. First of all, the agent has specific internal knowledge, for example, it knows the actions it is able to perform or has information about the transition and reward functions. An agent can also have prior knowledge about the preferences of the other agents, as we describe in more detail in Section 2.3.4. Furthermore, an agent observes information about the current world state through its sensors, or receives communication messages from the other agents. The messages, for example, contain information about the current state or guide the action in selecting an action. We will treat both information sources later on in more detail.

In general, knowledge in a MAS is distributed, and every agent bases its decision on different information about the problem. Each agent, for example, has a different

interpretation of the current world state. As a consequence, an agent has to reason about the knowledge state of the other agents when choosing its own action, that is, each agent has to reason *interactively*. An important notion in this context is *common knowledge* [Osborne and Rubinstein, 1994], which is knowledge that is shared by the agents. More formally, common knowledge is information that is available to all members of a group, and every member knows that all members in the group know it. Furthermore, it is also known by all members that it is common knowledge. The latter is a recursive definition.

Observability

Observability is the degree to which agents, either individually or as a team, identify the current world state. Observability is an important aspect of a MAS since the decision of an agent is based on its current interpretation of the world state. Since the agents in a MAS are often spatially distributed, the observations are also distributed. This can lead to situations in which an agent is not able to observe the complete state and has to base its decision on incomplete information.

Pynadath and Tambe [2002] give the following four models for observability:

- *Individual observability*. Every agent observes the complete unique world state. This corresponds to full observability in single-agent systems.
- *Collective observability*. The combined observations of all agents uniquely identify the world state. Each agent observes a part of the state, and when the information of all agents is combined the complete unique world state is known.
- *Collective partial observability*. There are no assumptions on the observations. Each agent observes part of the full state information but there are no assumptions about the combined observations of the agents.
- *Non-observability*. The agents receive no feedback from the world.

Communication

Communication can help a team of agents to improve their performance. In the extreme case, the agents are able to communicate instantaneously to all agents for free and there are no limitations in the number of messages. Then, it is in principle possible to solve the system as one big single agent: one agent collects all the observations, solves the complete problem using single-agent learning techniques, and informs each agent which action it should take. Note that having access to noiseless, free, and instantaneous communication is much related to the collective observability discussed earlier. In both cases, each agent obtains perfect knowledge about the current situation, and therefore is able to model the complete problem by itself, and select the action corresponding to its own identity.

Normally, however, communication is restricted. For example, communication might not be available because of failing connections or spatial constraints. Furthermore, communication is often delayed. In order to model the drawbacks of communication, the sending of a message is sometimes associated with a cost, for example, in the form of a negative reward [Pynadath and Tambe, 2002].

Agents either *broadcast* a message to all agents at once, or directly send a message to a specific agent using *direct communication*. Irrespective of the used method, a communication message can be categorized in different *communicative acts*, or *speech acts* [Searle, 1969]. Communicative acts are communication primitives which have the characteristics of actions since they affect the knowledge of the receiving agent as standard actions affect the environment. Some of the most common communication acts are messages that inform an agent about the current world state, either as a direct observation or a summary of the previous observations of the sending agent, query for specific information, or direct another agent to perform a certain action, but many more exist.

2.3.2 Formal description

Next, we give a general model description for a MAS. Most model parameters extend the parameters for single-agent systems from Section 2.2.2. We also describe possible performance measures for such models, and alternative factorized representations.

Parameters

A MAS can be described using the following model parameters:

- A discrete time step $t = 0, 1, 2, 3, \ldots$.
- A group of n agents $A = \{A_1, A_2, \ldots, A_n\}$.
- A finite set of environment states S. A state $s^t \in S$ describes the state of the world at time step t.
- A finite set of actions $\mathcal{A}_i$ for every agent i. The action selected by agent i, A_i, at time step t is denoted by $a_i^t \in \mathcal{A}_i$. The joint action $\mathbf{a} \in \mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_n$ is the vector of all individual actions.
- A finite set of observations Ω_i for every agent i. An observation $o_i^t \in \Omega_i$ provides agent i with information about the current state s^t.
- A state transition function $T : S \times \mathcal{A} \times S \to [0, 1]$ which gives the transition probability $p(s^t|\mathbf{a}^{t-1}, s^{t-1})$ that the system moves to state s^t when the joint action $\mathbf{a}^{t-1}$ is performed in state s^{t-1}.
- An observation function $O : S \times \mathcal{A} \times \Omega_1 \times \ldots \times \Omega_n \to [0, 1]$ which defines the probability $p(o_1^t, \ldots, o_n^t|s^t, \mathbf{a}^{t-1})$ that the observations o_1^t, $\ldots$, o_n^t are observed

by the agents $1, \ldots, n$ in state s^t after joint action $\mathbf{a}^{t-1}$ is performed. When the previous state is also incorporated in the observation function, the probability is defined as $p(\mathbf{o}^t|, s^t, \mathbf{a}^{t-1}, s^{t-1})$. Unless otherwise stated, we use observation functions that depend on the current state and the performed action only.

- A reward function $R_i : S \times \mathcal{A} \rightarrow \mathbb{R}$ which provides agent i with a reward $r_i^{t+1} \in R_i(s^t, \mathbf{a}^t)$ based on the joint action $\mathbf{a}^t$ taken in state s^t. The global reward $R(s^t, \mathbf{a}^t) = \sum_{i=1}^{n} R_i(s^t, \mathbf{a}^t)$ is the sum of all individual rewards received by the n agents. Depending on the model, the agents either have access to their part of the reward, R_i, or to the global reward R.

These parameters are very similar to the ones in the single-agent case. However, difficulties arise due to the decentralized nature of the problem. Each agent receives observations and selects actions individually, but it is the resulting joint action that influences the environment and generates the reward. This is depicted in the graphical model in Fig. 2.3(b) which shows the dependencies between the variables in two consecutive time steps.

Performance measures

Just as in a single-agent model, different performance measures can be constructed based on the received rewards. Our focus is on cooperative systems in which the agents try to maximize the expected discounted reward, exactly as in (2.3). For a MAS, this equation depends on the sum of the local received rewards. Since they depend on the selected joint action, the agents have to coordinate their actions.

In this thesis, we assume each agent receives an individual reward. Many other multiagent models assume that the agents do not receive an individual reward, but rather that the same (global) reward is communicated to all agents [Boutilier, 1996; Bernstein et al., 2002; Pynadath and Tambe, 2002]. In this setting, a centralized system computes the reward based on the performed joint action and distributes this value among all agents. Using this representation, it is not possible to deduce the performance of an individual agent. For example, the positive reward resulting from the action of one agent can be canceled out by the action of another agent performing a suboptimal action. The combined reward might then be zero and does not provide any feedback to the agents. Furthermore, it is shown that using local rewards for solving distributed, cooperative, multiagent reinforcement-learning problems, reduces the number of examples necessary for learning [Bagnell and Ng, 2006].

Policies

In a MAS, each agent selects an action based on its individual policy π_i which, just as in the single-agent case, is a mapping from the current state to an action: $\pi_i : S \rightarrow \mathcal{A}_i$. The joint policy $\pi = (\pi_1, \ldots, \pi_n)$ is the vector of all individual policies.

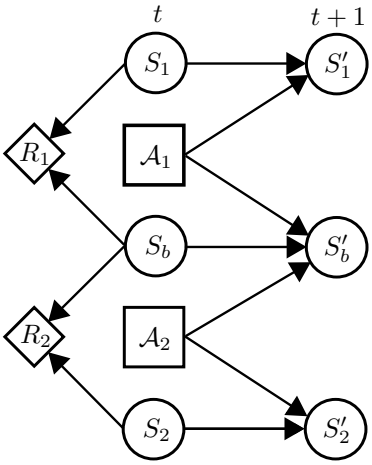


Figure 2.4: Example dynamic decision network for two consecutive time steps. States are represented by circles, actions by squares, and rewards by diamonds. Arcs define the directed dependencies.

Factorized representation

In many situations an agent has to coordinate its actions with a few other agents only, and acts independently with respect to the rest of the agents. On a soccer field, for example, the goalkeeper and the forward do not have to coordinate their actions. These dependencies can be used to factorize the transition and reward function in the same manner as the single-agent systems in Section 2.2.2. In the multiagent case, a state variable usually represents a feature of a specific agent, for example, its position, and the factorized transition function defines the probabilities for the, often independent, agent transitions.

Because the number of joint actions grows exponentially with the number of agents, it is not possible to store a separate network for each possible action. However, we can use a *dynamic decision network (DDN)* [Dean and Kanazawa, 1989; Guestrin, 2003], which is an extension of a DBN that also incorporates the actions of the agents in the network. In a DDN, the parents of a node S'_i consist of both state and action variables, that is, $\texttt{Parents}(S'_i) \subseteq \{\mathcal{A}, \mathbf{S}\}$, and the global transition probability distribution T is defined by

$$p(\mathbf{s}'|\mathbf{s}, \mathbf{a}) = \prod_{i=1}^{m} p(s'_i|\mathbf{s}[\texttt{Parents}(S'_i)], \mathbf{a}[\texttt{Parents}(S'_i)]), \tag{2.13}$$

where m is the number of defined state variables, $\mathbf{s}[\texttt{Parents}(S'_i)]$ represents the values of the variables in $\texttt{Parents}(S'_i)$ for state $\mathbf{s}$, and $\mathbf{a}[\texttt{Parents}(S'_i)]$ represents the values of the action variables. Note that the number of agents n does not have to be the same as the number of state variables m. Although it is possible that each agent is associated

with one single state variable, in general, this is not the case, and it depends on the specific problem under study which variables are related to which agents. Because each agent only depends on a subset of all state variables, a consequence of this representation is that an agent only has to observe the state variables on which it depends in order to determine its action.

Fig. 2.4 shows an example DDN. In this figure, S_1 and S_2, for instance, both represent the position of an agent on a soccer field, and S_b represents the position of the ball. From the dependencies in the figure, we can derive that the new position of an agent only depends on its previous position and its individual action, while the new ball position is specified in terms of its previous position and the actions of the two agents. The latter models the situation in which one of the agents kicks the ball.

The global reward is factorized and defined as the sum of all individual rewards. The individual reward $R_i(\mathbf{s}, \mathbf{a}) = R_i(\mathbf{s}[\texttt{Parents}(S'_i)], \mathbf{a}[\texttt{Parents}(S'_i)])$ obtained by an agent i depends on a subset of all state and action variables. In the Fig. 2.4 example, an agent receives a local reward depending on its own position and that of the ball.

2.3.3 Existing Models

Next, we describe several multiagent models which are derived from the general model described in Section 2.3.2. Just as with the single-agent models, the distinction between the different models are categorized based on assumptions about the model parameters. We first describe several models in which the agents have full observability, and thereafter models in which the agents only receive partial information about the current state.

Multiagent MDP (MMDP)

A straightforward extension of an MDP to multiple agents is a *multiagent Markov decision process (MMDP)* [Boutilier, 1996]. This model follows the general model from Section 2.3.2 with two additional assumptions. First, each agent receives the same global reward. Secondly, the system has individual observability, that is, each agent observes the complete state s^t in time step t. More formally, the set of observations equals $\Omega_i = S$ for every agent i, and the only non-zero observation probability is $p(o_i^t = s^t | s^t, \mathbf{a}^{t-1}) = 1$.

Collaborative multiagent MDP

A *collaborative multiagent MDP (collaborative MMDP)* [Guestrin, 2003] exploits the fact that in a MAS many agents act independently, and uses the known dependencies between the agents to create a factorized representation of the transition and reward function as described in Section 2.3.2. A consequence of this representation is that each agent only has to observe the state variables on which it depends. The goalkeeper, for example, can ignore the positions of the players which are positioned on the other side of the playing field. This model can be regarded as a factorized version of a

	intercept	defend
intercept	$0, 0$	$1, 1$
defend	$1, 1$	$0, 0$

Figure 2.5: An example strategic game.

MMDP with the additional difference that each agent receives an individual reward instead of a shared global reward. The goal of the agents is again to maximize a performance measure based on the global rewards. However, an important difference is that in a collaborative MMDP the agents only receive individual rewards, and they are thus not able to observe this value directly.

Stochastic games

The described model in Section 2.3.2 also bears resemblance to a *stochastic game* [Shapley, 1953]. This model is an extension of a *strategic game*, also called a *matrix game* or game in *normal form*, to multiple states [Osborne and Rubinstein, 1994]. A strategic game is a game in which all involved agents have to select an action a_i, and the resulting joint action $\mathbf{a}$, also called *outcome* in this context, provides each player i an individual payoff $R_i(\mathbf{a})$. Although the payoff structure is common knowledge, the agents do not know which actions the other agents will play and therefore have to reason about the other agent's strategies. There are different types of payoff structures. In zero-sum, or strictly competitive, games, one agent receives the opposite payoff of the other agent, and the sum thus equals zero. In identical payoff games, the reward received by all agents is identical. Finally, in general-sum games the individual payoffs can be arbitrary.

Fig. 2.5 depicts a graphical representation of such a strategic game between two agents using a payoff matrix. The rows and columns correspond to the possible actions of respectively the first and second agent, while the entries contain the returned payoff for the corresponding joint action. In this example, each agent chooses between two actions, either intercept the ball or move to a defending position, without knowing the choice of the other agent. The agents have to coordinate their actions in order to maximize their payoff. Both agents receive a payoff of 1 when the two selected actions of the agents differ, and a payoff of 0 when the actions are the same.

Stochastic games extend strategic games to multiple states by associating each state of the problem with a strategic game. In a state, each agent selects an individual action and the resulting joint action not only provides the agents with a payoff but also causes a transition to a next state in which a different strategic game applies.

Although there are many similarities between stochastic games and the general model described in Section 2.3.2, there are also some differences. First of all, a stochastic game assumes full observability of the current state and complete knowledge of the reward function. Furthermore, the agents in a stochastic game try to maximize their individual reward. For a completely observable identical payoff stochastic game

(IPSG) [Peshkin et al., 2000], in essence identical to an MMDP, this goal coincides with the objective of cooperative systems to optimize the sum of the received payoffs. For zero-sum and general-sum games, however, the goals differ, and in these cases different solution techniques are required. We briefly address such techniques in Section 2.3.4.

Decentralized POMDP (DEC-POMDP)

The multiagent extension of the single-agent POMDP model with collective partial observability is a *decentralized POMDP (DEC-POMDP)* [Bernstein et al., 2002]. This model is almost identical to the general model given in Section 2.3.2. It assumes that the observations of the agents are uncertain and given by the probability $p(\mathbf{o}^t|, s^t, \mathbf{a}^{t-1}, s^{t-1})$ which depends on the previous state, current state, and previous performed joint action. Furthermore, it assumes each agent receives a shared reward. This framework is similar to a multiagent team decision process (MTDP) [Pynadath and Tambe, 2002]. The only difference is that the latter ignores the previous state in the observation function and assumes a factorized state representation. When the agents receive individual rewards and are self-interested, that is, the agents try to maximize their own received rewards, the framework is referred to as a partially observable stochastic game (POSG) [Hansen et al., 2004].

The complexity of solving a DEC-POMDP falls in the complexity class NEXP-complete [Bernstein et al., 2002], and therefore several simplified models exist that make additional assumptions about the parameters of the model. A *factored DEC-POMDP* [Becker et al., 2003] factors the global state into different features, $S = S_1 \times \ldots \times S_m$. A common approach is to assume that each S_i corresponds to the local state of an agent, and make assumptions about the relation between the different features. In some cases, the model incorporates additional external state features S_o that all agents observe but are not able to influence [Becker et al., 2004]. The current time step, for example, is a commonly used external feature.

Becker et al. [2003]; Goldman and Zilberstein [2004] describe several additional extensions. In a (factored) *transition-independent* DEC-POMDP the transitions of the different agents are independent, and the action of an agent never influences the state of another agent. More formally, for every agent i holds $p(s'_i|\mathbf{a}, \mathbf{s}, \mathbf{s}'_{-i}) = p(s'_i|a_i, s_i)$ where $\mathbf{s}'_{-i}$ represent the state variables of all agents except agent i. Furthermore, in an *observation-independent* DEC-POMDP the observation function is decomposed in individual observation functions $O_i : S_i \times \Omega_i \rightarrow [0, 1]$ such that the observation an agent receives only relates to its own local state: $p(o'_i|\mathbf{s}', \mathbf{a}, \mathbf{s}, \mathbf{o}'_{-i}) = p(o'_i|s'_i, a_i, s_i)$ and $O = O_1 \times \ldots \times O_n$. In this case, the agents are not able to observe each other. In a *fully-observable DEC-POMDP*, which is identical to an MMDP, each agent can uniquely determine the global state $\mathbf{s}'$ from its local observation o'_i, that is, $p(o' = \mathbf{s}'|\mathbf{s}', \mathbf{a}, \mathbf{s})$ is the only non-zero probability. Finally, in a *jointly fully-observable DEC-POMDP* collective observability is assumed, that is, the combined observations of all agents uniquely determine the current state. The latter is also referred to as a *decentralized MDP (DEC-MDP)* [Bernstein et al., 2000].

2.3.4 Solution techniques

In this section, we discuss several existing solution techniques to compute a policy π for the agents in a MAS. We concentrate on the collaborative MMDP model since it is used as a basis in the subsequent chapters. Our main focus is on decomposing a problem with multiple agents into smaller subproblems that can be solved locally and this model uses a factorized representation of the reward and transition function, and provides the agents with individual rewards. This allows us to fully decompose the problem. Many other models provide each agent the same global reward which makes it impossible to determine the local contribution of an agent.

We first describe model-based solution techniques, and show different approaches to solve the resulting *coordination problem*. After that, we concentrate on existing distributed model-free reinforcement-learning techniques, which will be used more extensively in the remainder of the thesis.

Coordination problem

In principle, it is possible to regard a MAS as one big single agent, and apply standard MDP solution techniques to compute the optimal policy for the agents. Boutilier [1996], for example, assumes each agent has access to the complete transition and reward functions, and is able to compute the optimal joint action for every state using the single-agent solution techniques described in Section 2.2.4. However, since actions are taken individually by the agents, an important problem is to ensure, without using communication, that the selected joint action results in a good joint policy. In general, there are a number of distinct optimal policies for an MDP and even when each agent selects a *potentially individually optimal (PIO)* action, that is, an individual action which is part of an optimal joint action, the resulting joint action does not have to be optimal. This problem of selecting individual actions that correspond to an optimal joint action is referred to as the *coordination problem*.

In order to place the coordination problem in a broader context, we analyze it from a game-theoretic point of view. The coordination problem can be modeled as a strategic game (see Section 2.3.2) in which all agents share the same payoff function. For this stateless problem, each agent independently has to select an action from its action set. Each agent i receives a payoff $R_i(\mathbf{a})$ based on the resulting joint action $\mathbf{a}$. The goal of the agents is to select, via their individual decisions, the most profitable joint action. Fig. 2.5 shows an example coordination problem: all actions of the two agents are PIO actions, but only two combinations result in an optimal joint action.

A fundamental solution concept in a strategic game is a Nash equilibrium [Nash, 1950; Osborne and Rubinstein, 1994]. It defines a joint action $\mathbf{a}^* \in \mathcal{A}$ with the property that for every agent i holds $R_i(a_i^*, \mathbf{a}^*_{-i}) \geq R_i(a_i, \mathbf{a}^*_{-i})$ for all actions $a_i \in \mathcal{A}_i$, where $\mathbf{a}_{-i}$ is the joint action for all agents excluding agent i. Such an equilibrium joint action is a steady state from which no agent can profitably deviate given the actions of the other agents. For example, the strategic game in Fig. 2.5 has two Nash equilibria corresponding to the situations where both agents select a different action.

Another fundamental concept is Pareto optimality. An action $\mathbf{a}^*$ is Pareto optimal if there is no other joint action $\mathbf{a}$ for which $R_i(\mathbf{a}) \geq R_i(\mathbf{a}^*)$ for all agents i and $R_j(\mathbf{a}) > R_j(\mathbf{a}^*)$ for at least one agent j. That is, there is no other joint action that makes every agent at least as well off and at least one agent strictly better off. There are many examples of strategic games where a Pareto optimal solution is not a Nash equilibrium and vice versa (for example, the prisoner's dilemma [Osborne and Rubinstein, 1994]). However, in identical payoff strategic games games such as the one in Fig. 2.5 each Pareto optimal solution is also a Nash equilibrium by definition.

Formally, the coordination problem can be seen as the problem of selecting one single Pareto optimal Nash equilibrium in a strategic game [Vlassis, 2003]. Boutilier [1996] describes three different methods to ensure that the agents select a coordination joint action: communication, social conventions, and learning. Communication allows each agent to inform the other agents of its action, restricting the choice of the other agents to a simplified strategic game. If in the example of Fig. 2.5 the first agent notifies the other agent that it will intercept the ball, the strategic game is simplified to the first row which contains only one equilibrium. Secondly, *social conventions* are constraints on the action choices of the agents. The agents beforehand agree upon a priority ordering of agents and actions that is common knowledge among the agents. When an action has to be selected, each agent derives which actions the agents with a higher priority will perform, and selects its own action accordingly. A social convention is a general, domain-independent method which always results in an optimal joint action, and moreover, it can be implemented off-line: during execution the agents do not have to explicitly coordinate their actions by communication. Assuming the ordering '$1 \succ 2$' (meaning that agent 1 has priority over agent 2) and 'intercept $\succ$ defend' in our example, the second agent is able to derive from the social conventions that the first agent will intercept the ball and therefore chooses the defend action. Finally, learning methods can be applied to learn the behavior of the agents through repeated interaction. Each agent, for example, can keep track of the distribution of actions played by the other agents, and chooses its individual action according to these statistics [Boutilier, 1996; Bowling and Veloso, 2002; Wang and Sandholm, 2003].

The described model-based approaches assume that all equilibria can be found, for example, by applying single-agent reinforcement-learning techniques on the complete state-action space, and coordination is the result of each individual agent selecting its individual action based on the same equilibrium. However, the number of joint actions grows exponentially with the number of agents, making it infeasible to determine all equilibria in the case of many agents. This calls for approximation methods that first reduce the size of the state-action space before solving the coordination problem.

Another complication arises when the agents do not have access to the complete model description, and it is not even possible to compute all equilibria beforehand. In these cases, model-free techniques have to be applied which automatically learn to converge to an equilibrium. Next, we discuss some model-free reinforcement-learning techniques to learn the behavior of multiple agents. We first describe an approach which considers the complete state-action space, and then we describe techniques which reduce the full state-action space by exploiting the structure of the problem.

MDP learners

As stated earlier, a collaborative MMDP can be regarded as one large single agent in which each joint action is represented as a single action. It is then possible to apply a model-free reinforcement-learning technique like Q-learning, described in Section 2.2.4, to learn optimal Q-values for the joint actions using standard single-agent Q-learning, that is, by iteratively applying (2.12). When a specific Q-value is stored for every state and joint action combination, this approach eventually results in an optimal joint policy.

In this *MDP learners* approach either a central controller models the complete MDP and communicates to each agent its individual action, or each agent models the complete MDP separately and selects the individual action that corresponds to its own identity. In the latter case, the agents do not need to communicate but they have to be able to observe the executed joint action and the received individual rewards. However, the distributed nature of the problem requires the agents to explore at the same time. This problem can be solved by using the same random number generator (and the same seed) for all agents [Vlassis, 2003].

Although this approach leads to the optimal solution, it is infeasible for problems with many agents for several reasons. In the first place, it is intractable to model the complete joint action space, which is exponential in the number of agents. For example, a problem with 7 agents, each able to perform 6 actions, results in almost $280,000$ Q-values per state. Secondly, the agents might not have access to the needed information for the update because they are not able to observe the state, actions and reward of all other agents. Finally, it takes many time steps to explore all joint actions for every state. This results in slow convergence to the optimal joint action.

Independent Q-learning

The MDP learners approach assumes that all agents depend on each other agent in every state, that is, in every situation all actions in the joint action influence the payoff received by an individual agent. However, in many situations an agent acts independently with respect to the other agents, and the payoff an agent receives only depends on a subset of all actions. At the other extreme, we can therefore assume that all agents act independently, and have each agent ignore the actions and rewards of the other agents [Claus and Boutilier, 1998]. In this *independent learners* (IL) approach, each agent stores and updates an individual table Q_i and the global Q-function is defined as a linear combination of all individual contributions, $Q(\mathbf{s}, \mathbf{a}) = \sum_{i=1}^{n} Q_i(\mathbf{s}, a_i)$. Each local Q-function is updated completely independent of the other Q-functions using

$$Q_i(\mathbf{s}, a_i) := Q_i(\mathbf{s}, a_i) + \alpha[R_i(\mathbf{s}, \mathbf{a}) + \gamma \max_{a_i'} Q_i(\mathbf{s}', a_i') - Q_i(\mathbf{s}, a_i)]. \tag{2.14}$$

This approach results in big storage and computational savings in the action space. For example, with 7 agents and 6 actions per agent only 42 Q-values are stored per

state, instead of 280,000 in the MDP learners approach. However, the standard convergence proof for Q-learning does not hold anymore: because the actions of the other agents are ignored in the representation of the Q-functions, and these agents also change their behavior while learning, the system becomes non-stationary from the perspective of an individual agent, possibly leading to oscillations. Despite the lack of guaranteed convergence, this method has been applied successfully in multiple cases [Tan, 1993; Sen et al., 1994].

Another approach in which the agents learn independently to optimize a shared performance measure, is the collective intelligence (COIN) framework [Wolpert et al., 1999]. In this work, the agents optimize private utility functions which also increase the global utility. The agents update their local utility function using the *wonderful life utility*, which resembles the difference in received reward when an agent takes part in a joint action or does not.

Distributed value functions

In many problems, it is not the case that either all or none of the agents depend on each other. Instead, each agent has to coordinate its actions with a few other agents only, and acts independently with respect to the rest of the agents. The distributed value functions (DVF) approach [Schneider et al., 1999] takes advantage of such situations by only taking into account the actual dependencies between the agents. These dependencies are fixed beforehand and depend on the problem.

Each agent i maintains an individual local Q-function, $Q_i(\mathbf{s}_i, a_i)$, based on its individual action and updates it by incorporating the Q-functions, representing the estimated future return, of its neighboring agents $\Gamma(i)$. A weight function $f(i,j)$ determines how much the Q-value of an agent j contributes to the update of the Q-value of agent i. This function defines a graph structure of agent dependencies, in which an edge is added between agents i and j if the corresponding function $f(i,j)$ is non-zero. The update looks as follows:

$$Q_i(\mathbf{s}_i, a_i) := \qquad (2.15)$$
$$(1-\alpha)Q_i(\mathbf{s}_i, a_i) + \alpha[R_i(\mathbf{s}, \mathbf{a}) + \gamma \sum_{j \in \{i \cup \Gamma(i)\}} f(i,j) \max_{a'_j} Q_j(\mathbf{s}', a'_j)].$$

A local Q-function of an agent i is thus updated based on its old value and a linear combination of the Q-values of the agents on which agent i depends. Note that $f(i,i)$ is also defined and specifies the agent's contribution to the current estimate. A common approach is to weigh each neighboring function equally, and divide each Q-function of an agent i proportionally over its neighbors and itself. The function $f(i,j)$ then equals $1/(1+|\Gamma(j)|)$ when i and j depend on each other, and zero otherwise.

2.4 Discussion

In this chapter we provided an overview of different single- and multiagent Markov models, which represent sequential decision-making problems involving respectively the interaction of one or multiple agents with the environment. For both types of models we gave their characteristics and a general, formal, model description. Furthermore, we described several existing models from literature, which are categorized based on the parameters of the general model. Finally, we described several techniques to optimize the performance measure, both in the case that the agents have access to the model description or when this model is unavailable.

As described in this chapter, extending a single-agent system to multiple agents introduces many complexities. The main complications arise as a result of the exponential growth in both the representation of the action and state space, and the distributed nature of the problem: observations, control, and knowledge are all distributed. The large body of literature available about this subject studies multiagent problems from different perspectives. This results in different models and solution techniques, some of which are discussed in this chapter, based on the assumptions about the specific parameters of the model.

In the remainder of this thesis we concentrate on solution techniques to coordinate and learn the behavior of the agents in a multiagent system. Our focus is on distributed, scalable techniques to coordinate and learn the behavior of a large group of agents when the model of the environment is unavailable. Contrary to many other solution techniques, we thus do not assume that the agents have access to a complete model of the environment, but have to learn how to coordinate through repeated interaction with the environment. On the other hand, we do assume that each agent is able to fully observe the state variables needed for their decision. The main idea is to exploit the relevant dependencies in the problem, which makes it possible to solve the global problem as a sum of local problems. Therefore, we focus on problems in which the agents are only allowed to operate locally, that is, the agents observe local states, receive local rewards, and are only able to communicate with 'nearby' agents. The main question we address is how the agents can coordinate or learn their contribution to the global solution based on their local interaction with the environment.

3

Multiagent Coordination

This chapter focuses on coordinating the behavior of a group of cooperative agents. Specifically, we investigate problems in which the agents have to decide on a joint action, which results from their individual decisions, that maximizes a given payoff function. This payoff function is specified as a sum of local terms using a coordination graph [Guestrin et al., 2002a]. We describe two solutions to the coordination problem which are used as building blocks for the learning algorithms described in the subsequent chapters. We first review the variable elimination algorithm [Guestrin et al., 2002a]. This method always produces an optimal joint action but scales exponentially in the induced width of the graph. Our contribution is the max-plus algorithm, a payoff propagation algorithm that can be regarded as the decision-making analogue of belief propagation in Bayesian networks [Kok and Vlassis, 2005, 2006]. This method serves as a fast approximate alternative to the exact variable elimination algorithm.

3.1 Introduction

The study on multiagent systems (MAS) focuses on systems in which intelligent agents interact with each other [Sycara, 1998; Lesser, 1999; Vlassis, 2003]. Since all agents in a multiagent system can potentially influence each other, it is important to ensure that the actions selected by the individual agents result in optimal decisions for the group as a whole. As stated in Section 2.3.1, making a coordinated decision is complicated by different factors. An agent, for example, is often not aware of the intentions of the other agents, or is uncertain about the outcome of its actions. In this chapter, we ignore many of these difficulties and investigate a *stateless* problem in which the goal of the agents is to select a joint action that optimizes a *given* payoff function.

The main difficulty of the coordination problem addressed in this chapter is that each agent selects an action *individually*, but that the outcome is based on the actions selected by *all* agents. Fortunately, in many problems the action of one agent does not depend on the actions of all other agents, but only on a small subset. For example, in many real-world domains only agents which are spatially close have to coordinate their actions, and agents which are positioned far away from each other can act independently. In such situations, we are to able to represent the payoff function using a coordination graph (CG) [Guestrin et al., 2002a]. This model decomposes a global coordination problem into a combination of simpler problems. We will use this

framework extensively in the remainder of the thesis. In a CG each node represents an agent, and an edge between agents indicates a local coordination dependency. Each dependency corresponds to a local function that assigns a specific value to every action combination of the involved agents. The global function equals the sum of all local functions. In order to compute the joint action that maximizes the global function the variable elimination (VE) algorithm can be applied [Guestrin et al., 2002a]. This algorithm assumes the agents that depend on each other are allowed to communicate for free, and operates by iteratively eliminating the agents one by one after performing a local maximization step. This results in optimal behavior for the group, but its worst-case time complexity is exponential in the number of agents.

As an alternative to VE, we present a 'payoff propagation' algorithm, often referred to as the max-plus algorithm, that finds an approximately maximizing joint action for a CG [Kok and Vlassis, 2005]. Our algorithm exploits the fact that there is a direct duality between computing the maximum a posteriori configuration in a probabilistic graphical model and finding the optimal joint action in a CG; in both cases we are optimizing over a function that is decomposed in local terms. This allows message-passing algorithms that have been developed for inference in probabilistic graphical models, to be directly applicable for action selection in CGs. The max-plus algorithm is a popular method of that family. We empirically demonstrate that this method, contrary to VE, scales to large groups of agents with many dependencies.

The problem of finding the maximizing joint action in a fixed CG is related to the work on distributed constraint satisfaction problems (CSPs) in constraint networks [Pearl, 1988]. These problems consist of a set of variables, each taking a value from a finite, discrete domain. Predefined constraints, which have the values of a subset of all variables as input, specify a cost. The objective is to assign values to these variables such that the total cost is minimized [Yokoo and Durfee, 1991; Dechter, 2003].

The remainder of this chapter is structured as follows. In Section 3.2 we review the CG framework and the VE algorithm. We discuss our approximate alternative to VE based on the max-plus algorithm in Section 3.3, and show experimental results on randomly generated graphs in Section 3.4. Section 3.5 concludes the chapter.

3.2 Coordination graphs and variable elimination

In this section we review the problem of computing a coordinated action for a group of n agents that maximizes a given payoff function as described by Guestrin et al. [2002a]. More formally, this problem can be described as follows: each agent i selects an individual action a_i from its action set $\mathcal{A}_i$ and the resulting *joint* action $\mathbf{a} = (a_1, \ldots, a_n)$ generates a payoff $R(\mathbf{a})$ for the team. The coordination problem is to find the optimal joint action $\mathbf{a}^*$ that maximizes $R(\mathbf{a})$, that is, $\mathbf{a}^* = \arg\max_{\mathbf{a}} R(\mathbf{a})$.

Assuming a centralized controller, one straightforward approach to solve the coordination problem is to enumerate over all possible joint actions and select the one that maximizes $R(\mathbf{a})$. However, this approach quickly becomes impractical, as the

size of the joint action space $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n$ grows exponentially with the number of agents: for binary actions and n agents, there are 2^n joint actions.

However, in many problems the action of an agent i only depends on a small subset $\Gamma(i)$ of all other agents. Coordination graphs (CGs) [Guestrin et al., 2002a] exploit these dependencies by decomposing the global payoff function $R(\mathbf{a})$ into a linear combination of local payoff functions, as follows:

$$R(\mathbf{a}) = \sum_{i=1}^{n} f_i(\mathbf{a}_i). \tag{3.1}$$

Each local payoff function f_i depends on a subset of all actions, $\mathbf{a}_i \subseteq \mathbf{a}$, where $\mathbf{a}_i \in \mathcal{A}_i \times (\times_{j \in \Gamma(i)} \mathcal{A}_j)$ corresponds to the combination of actions of agent i and of the agents $j \in \Gamma(i)$ on which agent i depends. The global coordination problem is now replaced by a number of local coordination problems each involving fewer agents. This decomposition can be depicted using an undirected graph $G = (V, E)$ in which each node $i \in V$ represents an agent and an edge $(i, j) \in E$ indicates that agents i and j have to coordinate their actions with each other, that is, $i \in \Gamma(j)$ and $j \in \Gamma(i)$.

In principle, the number of agents involved in a coordination dependency in a CG can be arbitrary. However, in this thesis we assume that each dependency involves at most two agents, that is, each local payoff function is defined over either one or two agents. Note that this still allows for complicated coordinated structures since every agent can have multiple pairwise dependency functions. Furthermore, it is possible to generalize the proposed techniques to payoff functions with more than two agents because any arbitrary graph can be converted to a graph with only pairwise inter-agent dependencies [Yedidia et al., 2003; Loeliger, 2004]. To accomplish this, a new agent is added for each local function that involves more than two agents. This new agent contains an individual local payoff function that is defined over the combined actions of the involved agents, and returns the corresponding value of the original function. Note that the action space of this newly added agent is exponential in its neighborhood size (which can lead to intractability in the worst case). Furthermore, new pairwise payoff functions have to be defined between each involved agent and the new agent in order to ensure that the action selected by the involved agent corresponds to its part of the (combined) action selected by the new agent.

Assuming local functions involving at most two agents, the global payoff function $R(\mathbf{a})$ can be written as

$$R(\mathbf{a}) = \sum_{i \in V} f_i(a_i) + \sum_{(i,j) \in E} f_{ij}(a_i, a_j). \tag{3.2}$$

A local payoff function $f_i(a_i)$ specifies the payoff contribution for the individual action a_i of agent i, and f_{ij} defines the payoff contribution for pairs of actions (a_i, a_j) of 'neighboring' agents $(i, j) \in E$. Fig. 3.1 shows a CG with eight agents and only pairwise dependencies.

In order to solve the coordination problem and find the optimal joint action $\mathbf{a}^* = \arg\max_{\mathbf{a}} R(\mathbf{a})$, we can apply the variable elimination (VE) algorithm [Guestrin et al.,

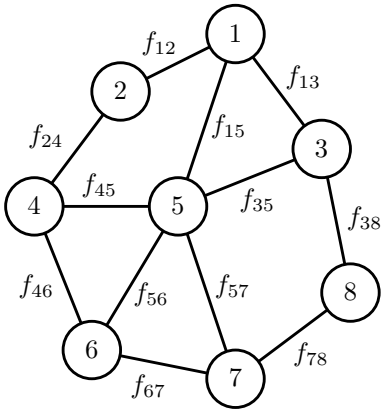


Figure 3.1: Example CG with eight agents; an edge represents a coordination dependency.

2002a]. This method is in essence identical to variable elimination in a Bayesian network [Zhang and Poole, 1996]. The algorithm operates by eliminating the agents one by one in a predefined *elimination ordering*. When an agent is selected for elimination, it first collects all payoff functions associated with its edges. Then, it computes a conditional payoff function that returns the maximal value the agent is able to contribute to the system for every action combination of its neighbors, and a best-response function, also called conditional strategy, which returns the action corresponding to this maximizing value. The agent communicates this conditional payoff function to one of its neighbors and is eliminated. The neighboring agent creates a new coordination dependency (edge) between itself and the agents in the received conditional payoff function on which it did not depend before, and then the next agent in the ordering is selected for elimination. This process is repeated until one agent remains. This agent fixes its action to the one that maximizes the final conditional payoff function. This individual action is part of the optimal joint action, and the associated value of the conditional payoff functions equals the desired value $\max_{\mathbf{a}} R(\mathbf{a})$. A second pass in the reverse order is then performed in which every agent computes its optimal action based on its conditional strategy and the fixed actions of its neighbors.

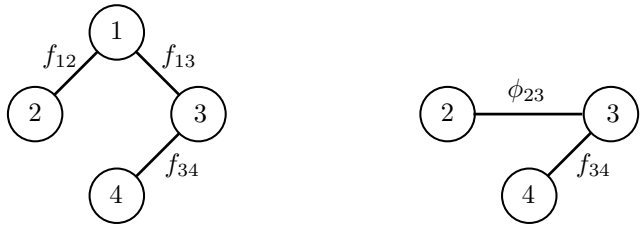
We now illustrate VE on the decomposition

$$R(\mathbf{a}) = f_{12}(a_1, a_2) + f_{13}(a_1, a_3) + f_{34}(a_3, a_4), \tag{3.3}$$

which is graphically depicted in Fig. 3.2(a). We apply the elimination ordering $1, 2, 3, 4$ and thus first eliminate agent 1. We first observe that this agent does not depend on the local payoff function f_{34} and rewrite the maximization of $R(\mathbf{a})$ in (3.3) as

$$\max_{\mathbf{a}} R(\mathbf{a}) = \max_{a_2,a_3,a_4} \Big\{ f_{34}(a_3, a_4) + \max_{a_1} [f_{12}(a_1, a_2) + f_{13}(a_1, a_3)] \Big\}. \tag{3.4}$$

Since the inner maximization only depends on the actions of agent 2 and 3, agent 1 computes the conditional payoff function $\phi_{23}(a_2, a_3) = \max_{a_1} [f_{12}(a_1, a_2) + f_{13}(a_1, a_3)]$



(a) Initial graph. (b) After elimination of agent 1.

Figure 3.2: CG corresponding to decomposition (3.3) before and after eliminating agent 1.

and the best-response function $B_1(a_2, a_3) = \arg\max_{a_1}[f_{12}(a_1, a_2) + f_{13}(a_1, a_3)]$ which respectively return the maximal value and the associated best action agent 1 is able to perform given the actions of agent 2 and agent 3. Because the function $\phi_{23}(a_2, a_3)$ is independent of agent 1, this agent can now be eliminated from the graph, simplifying (3.4) to $\max_{\mathbf{a}} R(\mathbf{a}) = \max_{a_2,a_3,a_4}[f_{34}(a_3, a_4) + \phi_{23}(a_2, a_3)]$. The elimination of agent 1 induces a new dependency between agent 2 and agent 3 and thus a change in the graph's topology. This new topology is depicted in Fig. 3.2(b).

We then apply the same procedure to eliminate agent 2. Since this agent only depends on ϕ_{23}, we define $B_2(a_3) = \arg\max_{a_2} \phi_{23}(a_2, a_3)$ and replace ϕ_{23} by $\phi_3(a_3) = \max_{a_2} \phi_{23}(a_2, a_3)$ producing

$$\max_{\mathbf{a}} R(\mathbf{a}) = \max_{a_3,a_4}[f_{34}(a_3, a_4) + \phi_3(a_3)], \tag{3.5}$$

which is independent of a_2. Next, we eliminate agent 3. We first define its conditional strategy $B_3(a_4) = \arg\max_{a_3}[f_{34}(a_3, a_4) + \phi_3(a_3)]$, and then replace the functions f_{34} and ϕ_3 with $\phi_4(a_4) = \max_{a_3}[f_{34}(a_3, a_4) + \phi_3(a_3)]$ resulting in $\max_{\mathbf{a}} R(\mathbf{a}) = \max_{a_4} \phi_4(a_4)$. Agent 4 is the last remaining agent and fixes its optimal action $a_4^* = \arg\max_{a_4} \phi_4(a_4)$. Next, a second pass in the reverse elimination order is performed in which each agent computes its optimal (unconditional) action from its best-response function and the fixed actions from its neighbors. In our example, agent 3 first selects $a_3^* = B_3(a_4^*)$. Similarly, we get $a_2^* = B_2(a_3^*)$ and $a_1^* = B_1(a_2^*, a_3^*)$. In the case that one agent has more than one maximizing best-response action, it selects one randomly. This always results in a coordinated action because each agent always bases its decision on the communicated actions of its neighbors.

An important characteristic of the VE algorithm is that it can be implemented fully distributed using communication between neighboring agents only. However, the neighbors of an agent can change during the elimination process when it receives an agent in a conditional strategy with which it did not had to coordinate before. When communication is restricted, additional common knowledge assumptions are needed such that each agent is able to run a copy of the algorithm (see Chapter 6). The outcome of VE does not depend on the elimination order and always produces the optimal

joint action. The execution time of the algorithm, however, does depend on the elimination order. Computing the optimal order for minimizing the runtime costs is known to be NP-complete [Arnborg et al., 1987], but good heuristics exist, for example, first eliminating the agent with the minimum number of neighbors [Bertelé and Brioschi, 1972]. The execution time is exponential in the induced width of the graph, that is, the size of the largest clique computed during node elimination. Depending on the graph structure this can scale exponentially in n for densely connected graphs. Finally, VE only produces a result after the end of the second pass. This is not always appropriate for real-time multiagent systems where decision making must be done under time constraints. In these cases, an anytime algorithm that improves the quality of the solution over time would be more appropriate [Vlassis et al., 2004].

3.3 Payoff propagation

Although VE is exact, it does not scale well with densely connected graphs. In this section, we describe our *max-plus algorithm* [Kok and Vlassis, 2005, 2006] as an approximate alternative to VE. We discuss both the standard max-plus algorithm and an anytime extension. We also describe a centralized and distributed implementation.

3.3.1 The max-plus algorithm

The CG framework bears much resemblance with a probabilistic graphical model, which is also known as a Bayesian network or belief network [Pearl, 1988; Jordan, 1998]. Without going into too much detail, such models reason about uncertain knowledge using probability theory. A problem is described using several random variables which take on values in a specific domain. The combination of all random variables specifies a joint probability distribution. The number of different combinations scales exponentially with the number variables, and therefore the model also specifies stochastic relations between the variables denoting the dependencies of the system for the given problem. Each random variable is associated with a *conditional probability distribution* which specifies the probability for its value based on the value of a subset of the other variables. Variables which do not directly depend on each other are *conditionally independent*. These dependencies can be depicted using a graph where each variable is represented as a node and dependencies between the random variables are denoted by, possibly directed, edges. The joint distribution then factors into a product of conditional distributions.

Probabilistic graphical models are used to answer *inference* questions about the random variables. For example, computing the posterior distribution of a random variable given evidence about the value of some other variables, or computing the *maximum a posteriori* (MAP) configuration of the random variables, that is, the assignment of the variables with the highest probability. Different, both exact and approximate, algorithms exist to perform probabilistic inference. Aside the exact variable

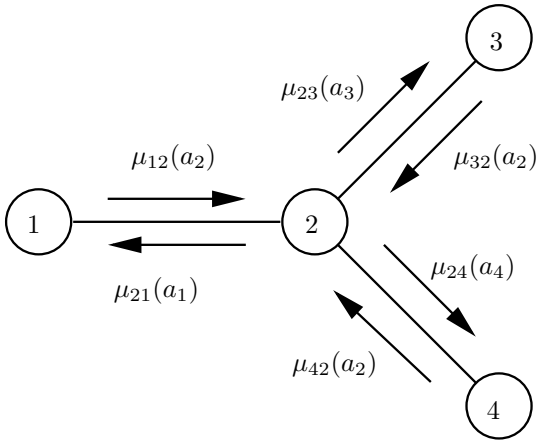


Figure 3.3: Graphical representation of different messages μ_{ij} in a graph with four agents.

elimination algorithm, one other popular method for computing the MAP configuration is the max-product, also called max-plus, algorithm [Pearl, 1988; Yedidia et al., 2003; Wainwright et al., 2004]. This method is exact for tree-structured graphs and is analogous to the belief propagation or sum-product algorithm [Kschischang et al., 2001]. It operates by iteratively sending locally optimized messages $\mu_{ij}(a_j)$ between node i and j over the corresponding edge in the graph. For cycle-free graphs, the message updates converge to a fixed point after a finite number of iterations [Pearl, 1988]. After convergence, each node computes its value to the global MAP assignment based on its local incoming messages only.

There is a direct duality between computing the MAP configuration in a probabilistic graphical model and finding the optimal joint action in a CG; in both cases we are optimizing over a function that is decomposed in local terms. This allows message-passing algorithms that have been developed for inference in probabilistic graphical models, to be directly applicable for action selection in CGs. The max-plus algorithm is a popular method of that family and, in the context of CGs, it can therefore be regarded as a 'payoff propagation' technique for multiagent decision making.

Suppose we have a coordination graph $G = (V, E)$ with $|V|$ vertices and $|E|$ edges. In order to apply the max-plus algorithm and compute the optimal joint action $\mathbf{a}^*$ maximizing (3.2), each agent i (node in G) repeatedly sends a message μ_{ij} to its neighbors $j \in \Gamma(i)$, where μ_{ij} can be regarded as a local payoff function of agent j:

$$\mu_{ij}(a_j) = \max_{a_i} \left\{ f_i(a_i) + f_{ij}(a_i, a_j) + \sum_{k \in \Gamma(i) \setminus j} \mu_{ki}(a_i) \right\} + c_{ij}, \tag{3.6}$$

where $\Gamma(i) \setminus j$ represents all neighbors of agent i except agent j, and c_{ij} is a normalization value (which can be assumed zero for now). The message $\mu_{ij}(a_j)$ is an approximation of the maximum payoff agent i is able to achieve for a given action

of agent j, and is computed by maximizing (over the actions of agent i) the sum of the payoff functions f_i and f_{ij} and all incoming messages to agent i except that from agent j. The main difference with the max-plus algorithm in probabilistic models is that now, instead of maximizing (the log of) a factorized probability distribution, we maximize the sum of the payoff functions (3.2). Fig. 3.3 shows a CG with four agents and the corresponding messages.

Messages are exchanged until they converge to a fixed point. For cycle-free graphs, any arbitrary order results in convergence within a finite number of steps [Pearl, 1988; Wainwright et al., 2004]. However, some orderings are more efficient with respect to the number of required messages. Because each message $\mu_{ij}(a_j)$ resembles an approximation of the payoff the subtree with agent i as root is able to produce when agent j performs action a_j, the most efficient procedure is to incrementally update the approximation for larger subtrees. This can be accomplished by sending messages upwards from the smallest subtrees, that is, the leaves. The complete procedure looks as follows: each leaf computes its message and sends it to its neighbors in the graph. Each other agent i waits until it receives the messages from all but one neighbor, for example, agent j. Then, agent i computes $\mu_{ij}(a_j)$, sends it to agent j, and waits for a return message from agent j. When this message arrives, agent i sends a message to all neighbors $\Gamma(i) \setminus j$. After each agent has received and sent a message to each of its neighbors, the messages are converged. This procedure is identical to variable elimination with an ordering that iteratively eliminates the leafs of the graph.

After convergence, a message $\mu_{ji}(a_i)$ equals the payoff that is produced by the subtree that has agent j as root when agent i performs action a_i. At any time step, we can define

$$g_i(a_i) = f_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i), \tag{3.7}$$

which equals the contribution of the individual function of agent i and the different subtrees with the neighbors of agent i as root. Using (3.7), we can show that, at convergence, $g_i(a_i) = \max_{\{\mathbf{a}' | a'_i = a_i\}} R(\mathbf{a}')$ holds [Wainwright et al., 2002], that is, the action a_i selected by agent i results in a maximal *global* payoff of $g_i(a_i)$. To select a globally optimal joint action, each agent i has to select its individually optimal action

$$a_i^* = \arg\max_{a_i} g_i(a_i). \tag{3.8}$$

If there is only one maximizing action for every agent i, the globally optimal joint action $\mathbf{a}^* = \arg\max_{\mathbf{a}} R(\mathbf{a})$ is unique and has elements $\mathbf{a}^* = (a_i^*)$. Note that this optimal joint action is computed by only local optimizations (each node maximizes $g_i(a_i)$ separately). In case the local maximizers are not unique, an optimal joint action can be computed by a dynamic programming technique [Wainwright et al., 2004, sec. 3.1]. In this case, each agent informs its neighbors in a predefined order about its action choice such that the other agents are able to fix their actions accordingly.

In graphs with cycles, there are unfortunately no guarantees that the max-plus algorithm converges. Despite recent theoretical convergence results for the related

sum-product algorithm Mooij and Kappen [2005], convergence results for the max-plus algorithm for graphs with cycles are, to the best of our knowledge, not available. Although, it has been shown that a fixed point of message passing exists [Wainwright et al., 2004], the precise conditions for convergence are in general not known, and therefore no assurance can be given about the quality of the corresponding joint action $\mathbf{a}^* = (a_i^*)$ with a_i^* from (3.8). However, empirical results demonstrate the potential of the algorithm in practice Murphy et al. [1999].

Despite the lack of convergence proofs, the max-plus algorithm is still applied to graph with cycles in many cases. In the context of probabilistic model, it is then referred to as *loopy belief propagation*. This approach yields surprisingly good results in practice [Murphy et al., 1999; Crick and Pfeffer, 2003; Yedidia et al., 2003].

We also apply the max-plus algorithm to graphs with cycles for our multiagent decision-making problems. One of the main problems in such settings is that an outgoing message from an agent i eventually becomes part of its incoming messages. In our case the different incoming messages are summed, resulting in a continuously increase of the messages' values. In order to circumvent the increase of the messages' values, we normalize each sent message in cyclic graphs by subtracting from each element of μ_{ij} the average of all values using

$$c_{ij} = -\frac{1}{|\mathcal{A}_j|} \sum_{a_j \in \mathcal{A}_j} \mu_{ij}(a_j) \tag{3.9}$$

in (3.6).[1] This approach is similar to the normalization constant used in graphical models as described by Wainwright et al. [2004, sec. 4.2]. Still, it might be the case that the messages do not converge. Therefore, we assume that the agents receive a 'deadline' signal, either from an external source or from an internal timing signal, that indicates they should immediately report their current action. This corresponds to situations in which the agents only have a finite amount of time to compute an action, for example, because of environmental or other constraints. In such situations, the max-plus algorithm is always able to report an action based on the current values of the messages. On the contrary, the VE algorithm might not have finished its second pass when the signal arrives, and then no coordinated action is available.

To conclude, we list three important differences between a message μ_{ij} in the max-plus algorithm with respect to the conditional payoff functions in VE. First, before convergence each message is an approximation of the exact value (conditional team payoff) since it depends on the incoming, still not converged, messages. Second, an agent i only has to sum over the received messages from its neighbors which are defined over individual actions, instead of enumerating over all possible action combinations of its neighbors. This is the main reason for the scalability of the algorithm for graphs with cycles. Finally, in the max-plus algorithm, messages are always sent over the edges of the original graph. In the VE algorithm, the elimination of an agent often results in new dependencies between agents that did not have to coordinate initially.

[1] Note the slight abuse of notation because c_{ij} in (3.9) depends on μ_{ij} which again depends on c_{ij} (see (3.6)) resulting in an infinite recursion. We assume $c_{ij} = 0$ for the definition of μ_{ij} in (3.9).

3.3.2 Anytime extension

As stated earlier, there are no guarantees that the max-plus algorithm converges in graphs with cycles. The global payoff corresponding to the joint action might even decrease when the values of the messages oscillate. As a result no assurances can be given about the quality of the corresponding joint action. This necessitates the development of an *anytime* algorithm [Dean and Boddy, 1988; Zilberstein, 1996] in which the joint action is only updated when the corresponding global payoff improves. Therefore, we extend the max-plus algorithm by, occasionally, computing the global payoff and updating the joint action, represented as the combination of individual actions, only when it improves upon the best value found so far. This ensures that every newly stored joint action produces a strictly higher payoff than the previous one. When the joint action has to be reported, we return the last updated actions. We refer to this approach as *anytime max-plus*.

Next, we describe a centralized and distributed implementation, both with and without the anytime extension, of the max-plus algorithm. The fully distributed implementation in combination with the anytime extension requires some specific care since it involves the evaluation of the current joint action even though each agent only has access to the local payoff functions in which it is involved.

3.3.3 Centralized version

The centralized version of the max-plus algorithm runs in iterations. In one iteration each agent i computes and sends a message μ_{ij} to all its neighbors $j \in \Gamma(i)$ in a predefined order. This process continues until all messages are converged, or a 'deadline' signal, either from an external source or from an internal timing signal, is received and the current joint action is reported. For the anytime extension, the current computed joint action is inserted into (3.2) after every iteration and the best joint action $\mathbf{a}^*$ is only updated when it improves upon the best value found so far. A pseudo-code implementation of the centralized max-plus algorithm, including the anytime extension, is given in Alg. 3.1.

3.3.4 Distributed version

The same functionality can also be implemented using a distributed implementation. Each agent computes and communicates an updated message directly after it receives a different message from one of its neighbors (line 5 to 8). This results in a computational advantage over the sequential execution of the centralized algorithm since messages are sent in parallel. See Alg. 3.2 for a distributed version in pseudo-code.

In a fully distributed implementation, the anytime extension is much more complex because the agents do not have direct access to the actions of the other agents or the global payoff function (3.2). Therefore, we initiate the evaluation of the (distributed) joint action only when an agent believes it is worthwhile to do so, for example, after a big increase in the values of the received messages. The evaluation is implemented

Algorithm 3.1 Pseudo-code of the centralized max-plus algorithm for $G = (V, E)$.

1: initialize $\mu_{ij}(a_j) = \mu_{ji}(a_i) = 0$ for $(i, j) \in E, a_i \in \mathcal{A}_i, a_j \in \mathcal{A}_j$
2: initialize $g_i(a_i) = 0$ for $i \in V, a_i \in \mathcal{A}_i$, and $m = -\infty$
3: **while** *fixed_point* = *false* and deadline to send action has not yet arrived **do**
4: // run one iteration
5: *fixed_point* = *true*
6: **for** every agent i **do**
7: **for** all neighbors $j = \Gamma(i)$ **do**
8: compute $\mu_{ij}(a_j) = \max_{a_i} \left\{ f_i(a_i) + f_{ij}(a_i, a_j) + \sum_{k \in \Gamma(i) \setminus j} \mu_{ki}(a_i) \right\} + c_{ij}$
9: send message $\mu_{ij}(a_j)$ to agent j
10: **if** $\mu_{ij}(a_j)$ differs from previous message by a small threshold **then**
11: *fixed_point* = *false*
12: compute $g_i(a_i) = f_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i)$, and $a_i' = \arg\max_{a_i} g_i(a_i)$
13: $\mathbf{a}' = (a_i')$
14: **if** use anytime extension **then**
15: **if** $R(\mathbf{a}') > m$ **then**
16: $\mathbf{a}^* = \mathbf{a}'$ and $m = R(\mathbf{a}')$
17: **else**
18: $\mathbf{a}^* = \mathbf{a}'$
19: return $\mathbf{a}^*$

using a special message passing system which requires communication between neighboring agents in a directed rooted *spanning tree* ST of the graph G. A spanning tree is a tree-structured subgraph that includes all nodes of the original graph. The fact that the tree is rooted indicates that there is one node which is assigned the root. This node has no parents. We assume ST is fixed beforehand and is common knowledge among all agents. Fig. 3.4 shows a CG with cycles and example rooted spanning tree.

The message passage system consists of three different types of messages. The first message is the `evaluate` message and indicates that an evaluation of the current joint action is taking place. The agent who wants to initiate the evaluation, does so by first sending this message to itself (line 11). An agent receiving an evaluation message (lines 14 to 19) computes its best individual action based on the last received messages, and fixes this action until after the evaluation. Furthermore, it distributes the evaluation message to all its neighbors in ST. When an agent is a leaf of ST, it starts the accumulation of the payoffs using the `accumulate_payoff` message. In this process (lines 20 to 27), each agent i computes its local contribution q_i to the global payoff based on the local payoff functions in which it is involved, that is, f_i and f_{ij} with $j \in \Gamma(i)$, and the fixed actions of its neighbors in the CG. Note that a local function f_{ij} is shared by two agents and therefore only half of this value is contributed to each of the two involved agents. After a node has received all payoffs of its children, accumulated in p_i, it adds its own contribution and sends the result to its parent. Finally, the root of ST has received all accumulated payoffs from its

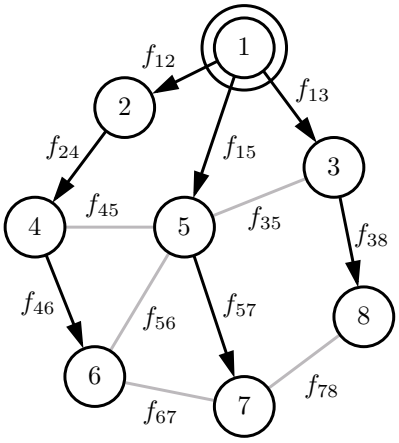


Figure 3.4: The CG of Fig. 3.1 and a corresponding rooted spanning tree. All edges are directed towards the root, agent 1.

children. The sum of these payoffs resembles the summation in (3.2) since

$$\sum_{i \in V} q_i(a_i) = \sum_{i \in V} [f_i(a_i) + \sum_{j \in \Gamma(i)} \frac{1}{2} f_{ij}(a_i, a_j)] = \sum_{i \in V} f_i(a_i) + \sum_{(i,j) \in E} f_{ij}(a_i, a_j) = R(a),$$

and thus corresponds to the global payoff of the fixed actions of the nodes in the graphs. This value is distributed to all nodes in ST using the `global_payoff` message (lines 28 to 31). After receiving this message, an agent updates its best individual action a_i^* only when the received global payoff improves upon the best one found so far. Furthermore, the agent unlocks its fixed action. Finally, when a 'deadline' signal arrives, each agent reports the action related to the highest found global payoff (line 32), which thus does not has to correspond to the current messages.

The execution time of the centralized implementation is linear in the number of agents and the average degree of the graph. Specifically, it has complexity $O(nd)$ where n is the number of agents and d is the average degree of the graph. In one iteration each agent computes a message for each of its neighbors and sends it. The computation of a message is a linear maximization, over the actions of the agent, of the messages received from its neighbors. The anytime extension only incorporates an extra test of constant time and therefore has the same complexity.

In the distributed implementation the messages are sent in parallel, resulting in a complexity $O(d)$. The anytime extension occasionally involves the evaluation of the joint action. This procedure consists of three message passings over the complete graph. Since each message has to be propagated to all agents, the time for each pass scales linearly in the depth of the graph. The actual time for sending all the messages depends on the characteristics of the available communication channel. In the remainder of this thesis, we assume instantaneous communication.

Algorithm 3.2 Pseudo-code of a distributed max-plus implementation for agent i, coordination graph $G = (V, E)$, and spanning tree $ST = (V, S)$.

initialize $\mu_{ij}(a_j) = \mu_{ji}(a_i) = 0$ for $j \in \Gamma(i), a_i \in \mathcal{A}_i, a_j \in \mathcal{A}_j$
initialize $q_i = p_i = 0$, and $m = -\infty$
while deadline to send action has not yet arrived **do**
  wait for message msg
  if $msg = \mu_{ji}(a_i)$ // max-plus message **then**
    for all neighbors $k \in \Gamma(i) \setminus j$ **do**
      compute $\mu_{ik}(a_k) = \max_{a_i} \left\{ f_i(a_i) + f_{ik}(a_i, a_k) + \sum_{l \in \Gamma(i) \setminus k} \mu_{li}(a_i) \right\} + c_{ik}$
      send message $\mu_{ik}(a_k)$ to agent k if it differs from last sent message
    if use anytime extension **then**
      if heuristic indicates global payoff should be evaluated **then**
        send `evaluate`(i) to agent i // initiate computation global payoff
    else
      $a_i^* = \arg\max_{a_i} [f_i(a_i) + \sum_{k \in \Gamma(i)} \mu_{ki}(a_i)]$
  if $msg =$ `evaluate`(j) // receive request for evaluation from agent j **then**
    if a'_i not locked **then**
      lock $a'_i = \arg\max_{a_i} [f_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i)]$, and set $p_i = 0$
    send `evaluate`(i) to all neighbors (parent and children) in $ST \neq j$
    if $i =$ leaf in ST **then**
      send `accumulate_payoff`(0) to agent i // initiate accumulation payoffs
  if $msg =$ `accumulate_payoff`(p_j) from agent j **then**
    $p_i = p_i + p_j$ // add payoff child j
    if received accumulated payoff from all children in ST **then**
      get actions a'_j from $j \in \Gamma(i)$ in CG and set $q_i = f_i(a'_i) + \frac{1}{2} \sum_{j \in \Gamma(i)} f_{ij}(a'_i, a'_j)$
      if $i =$ root of ST **then**
        send `global_payoff`($q_i + p_i$) to agent i
      else
        send `accumulate_payoff`($q_i + p_i$) to parent in ST
  if $msg =$ `global_payoff`(g) **then**
    if $g > m$ **then**
      $a_i^* = a'_i$ and $m = g$
    send `global_payoff`(g) to all children in ST and unlock action a'_i
return a_i^*

3.4 Experiments

In this section, we test the VE algorithm and the two variants of the max-plus algorithm on differently shaped graphs with random payoff functions. We investigate both cycle-free and cyclic graphs using the centralized max-plus algorithm from Alg. 3.1.

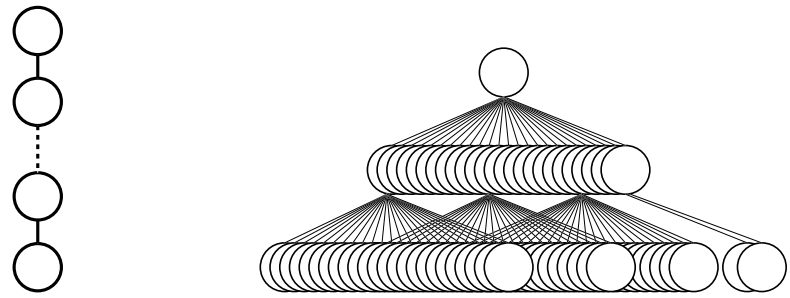
3.4.1 Trees

In this section, we describe our experiments with the max-plus algorithm on trees. As explained in Section 3.3.1, the max-plus converges algorithm to the optimal solution in this setting. We empirically illustrate this, and furthermore investigate the number of iterations needed for convergence depending on the order in which the agents sent their messages. We both study a random order and an ordering identical to VE.

We test our algorithm on trees $G = (V, E)$ with $|V| = 100$ agents, each having $|\mathcal{A}_i| = 4$ actions, and a fixed number of edges, $|E| = |V| - 1 = 99$. We create 24 trees with a maximal degree, the maximal number of neighbors per node, in the range $d \in [2, 25]$. The graphs are generated as follows. First, the agents are labeled from 1 to 100. Then, starting from node 2, each node is connected with the first node in the ordering that currently has less than d neighbors. Since the number of agents and edges is fixed and the degree varies, each tree has a different depth. Fig. 3.5 shows two graphs corresponding to respectively the minimum and maximum considered degree. Fig. 3.5(a) depicts the situation in which each agent has a maximal degree of 2, resulting in a tree with depth 99. Fig. 3.5(b) depicts the situation in which each agent has a maximal degree of 25 and results in a tree with depth 2. Four nodes have a degree of 25. For this, $25 + 3 \cdot 24 = 97$ of the 99 nodes are assigned a parent. The remaining two nodes are depicted on the right. Each edge $(i, j) \in E$ is associated with a payoff function f_{ij} where each action combination is assigned a payoff $f_{ij}(a_i, a_j)$ randomly generated from a standard normal distribution $\mathcal{N}(0, 1)$.

We apply both the VE and the max-plus algorithm to compute the joint action. In VE we always eliminate an agent with the minimum number of neighbors, and each local maximization step thus involves at most two agents. For the max-plus algorithm, we apply both a random order and the same order as VE to select the agent that sends its messages. For the latter case we process the agents in the same order as the elimination order of VE in the first iteration (loop in line 6 of Alg. 3.1). In the second iteration we iterate over the agents in the reverse order, identical to the second pass in the VE algorithm in which each agent fixes its action.

Fig. 3.6 shows the relative payoff found with the max-plus algorithm with respect to the optimal payoff, computed with the VE algorithm, after each iteration. The results are averaged over all 24 graphs. As expected, all policies converge to the optimal solution. When using the elimination order of VE to select the next agent, the max-plus algorithm converges after two iterations for all graphs. For this order, each message only has to be computed once [Loeliger, 2004] and the max-plus algorithm become equivalent to the VE algorithm. When using a random order some updates are unnecessary, and it takes a few iterations before the same information is propagated



(a) Tree with maximal degree of 2. (b) Tree with maximal degree of 25.

Figure 3.5: Example structure of two trees with respectively a maximal degree of 2 and 25.

through the graph. The time needed to run the different methods are similar and negligible, and therefore not listed. As we will see next, the timing differences for graphs with cycles, however, are much more pronounced.

3.4.2 Graphs with cycles

We also test VE and the max-plus variants on graphs with cycles. As stated in Section 3.3.1, it is not guaranteed that the max-plus algorithm will convergence to the optimal solution. The VE algorithm provably converges to the optimal joint action, but its computation time can be exponential in the number of dependencies.

We ran the algorithms on differently shaped graphs with 15 agents and a varying number of edges. In order to generate balanced graphs in which each agent approximately has the same degree, we start with a graph without edges and iteratively connect the two agents with the minimum number of neighbors. In case multiple agents satisfy this condition, an agent is picked at random from the possibilities. We apply this procedure to create 100 graphs for each $|E| \in \{8, 9, \ldots, 37\}$, resulting in a set of $3,000$ graphs. The set thus contains graphs in the range of on average 1.067 neighbors per agent (8 edges) to 4.93 neighbors per agent (37 edges). Fig. 3.7 depicts example graphs with respectively 15, 23 and 37 edges (on average 2, 3.07 and 4.93 neighbors per node). We create three copies of this set, each having a different payoff function related to the edges in the graph. In the first set, each edge $(i, j) \in E$ is associated with a payoff function f_{ij} defined over five actions per agent and each action combination is assigned a random payoff from a standard normal distribution, that is, $f_{ij}(a_i, a_j) \sim \mathcal{N}(0, 1)$. This results in a total of 5^{15}, around 3 billion, different possible joint actions. In the second set, we add one outlier to each of the local payoff functions: for a randomly picked joint action, the corresponding payoff value is set to $10 \cdot \mathcal{N}(0, 1)$. For the third test set, we specify a payoff function based on 10 actions per agent resulting in 10^{15} different joint actions. The values of the different payoff functions are again generated using a standard normal distribution.

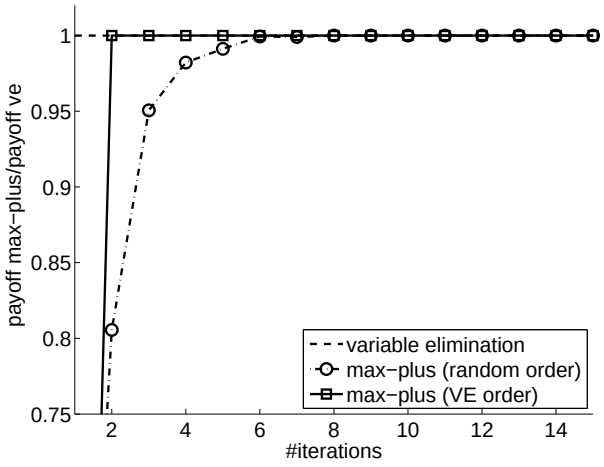


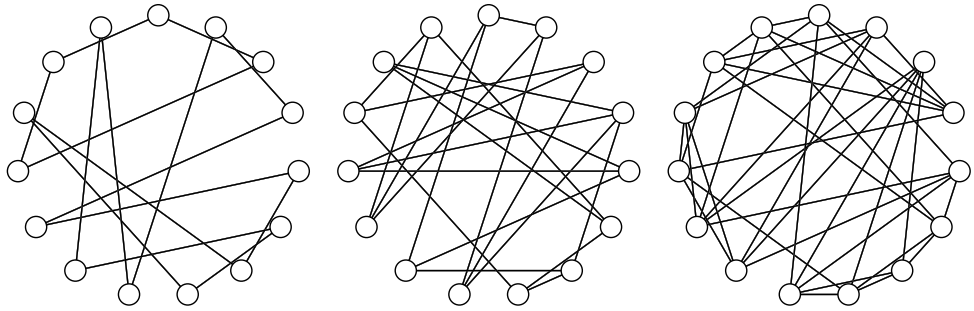
Figure 3.6: Relative payoff for the max-plus algorithm after each iteration. Results are averaged over 24 differently shaped graphs.

For all graphs we compute the joint action using the VE algorithm, the standard max-plus algorithm, and the max-plus algorithm with the anytime extension. Irrespectively of convergence, all max-plus methods perform 100 iterations. As we will see later in Fig. 3.9 the policy has stabilized at this point. Furthermore, a random ordering is used in each iteration to determine which agents sends its messages.

The timing results for the three different test sets are plotted in Fig. 3.8.[2] The x-axis shows the average degree of the graph, and the y-axis shows, using a logarithmic scale, the average timing results, in milliseconds, to compute the joint action for the corresponding graphs. Remember from Section 3.2 that the computation time of the VE algorithm depends on the induced width of the graph. The induced width depends both on the average degree and the actual structure of the graph. The latter is generated at random, and therefore the complexity of graphs with the same average degree differ. Table 3.1 shows the induced width for the graphs used in the experiments based on the elimination order of the VE algorithm, that is, iteratively remove a node with the minimum number of neighbors. The results are averaged over graphs with a similar average degree. For a specific graph, the induced width equals the maximal number of neighbors that have to be considered in a local maximization.

In Fig. 3.8, we show the timing results for the standard max-plus algorithm; the results for the anytime extension are identical since they only involve an additional check of the global payoff value after every iteration. The plots indicate that the time for the max-plus algorithm grows linearly as the complexity of the graphs increases. This is a result of the relation between the number of messages and the (linearly increasing) number of edges in the graph. The graphs with 10 actions per agent

[2] All results are generated on an Intel Xeon 3.4GHz / 2GB machine using a C++ implementation.



(a) Graph with 15 edges (average degree of 2).

(b) Graph with 23 edges (average degree of 3.07).

(c) Graph with 37 edges (average degree of 4.93).

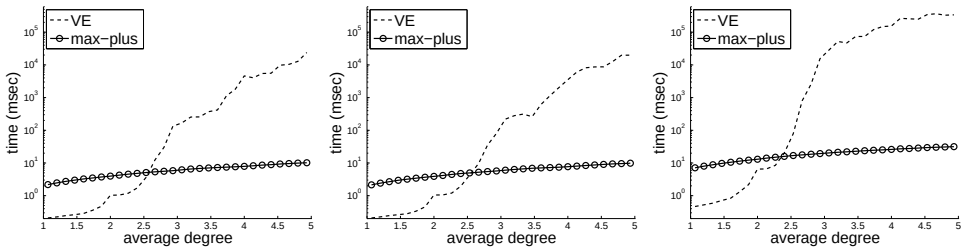
Figure 3.7: Example graphs with 15 agents and cycles.

average degree	$(1, 2]$	$(2, 3]$	$(3, 4]$	$(4, 5]$
induced width	1.23 ($\pm$0.44)	2.99 ($\pm$0.81)	4.94 ($\pm$0.77)	6.37 ($\pm$0.68)

Table 3.1: Average induced width and corresponding standard deviation for graphs with an average degree in $(x - 1, x]$.

require more time compared to the two other sets because the computation of every message involves a maximization over 100 instead of 25 joint actions. Note that all timing results are generated with a fixed number of 100 iterations. As we will see later, the max-plus algorithm can be stopped earlier without much loss in performance, resulting in even quicker timing results.

For the graphs with a small, less than 2.5, average degree, VE outperforms the max-plus algorithm. In this case, each local maximization only involves a few agents, and VE is able to finish its two passes through the graph quickly. However, the time for the VE algorithm grows exponentially for graphs with a higher average degree because for these graphs it has to enumerate over an increasing number of neighboring agents in each local maximization step. Furthermore, the elimination of an agent often causes a neighboring agent to receive a conditional strategy involving agents it did not have to coordinate with before, changing the graph topology to an even denser graph. This effect becomes more apparent as the graphs become more dense. More specifically, for graphs with 5 actions per agent and an average degree of 5, it takes VE on average 23.8 seconds to generate the joint action. The max-plus algorithm, on the other hand, only requires 10.18 milliseconds for such graphs. There are no clear differences between the two sets with 5 actions per agent since they both require the same number of local maximizations, and the actual values do not influence the algorithm. However, as is seen in Fig. 3.8(c), the increase of the number of actions per agent slows the VE algorithm down even more. This is a result of the larger number of joint actions which has to be processed during the local maximizations. For example,



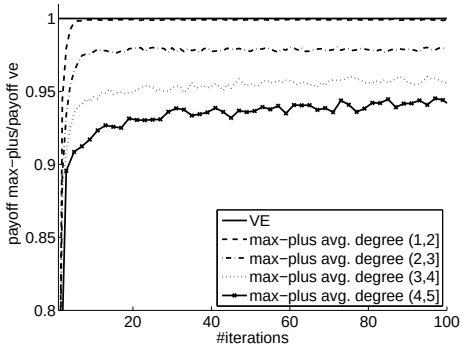
(a) 5 actions per agent. (b) 5 actions and outliers. (c) 10 actions per agent.

Figure 3.8: Timing results VE and max-plus for different graphs with 15 agents and cycles.

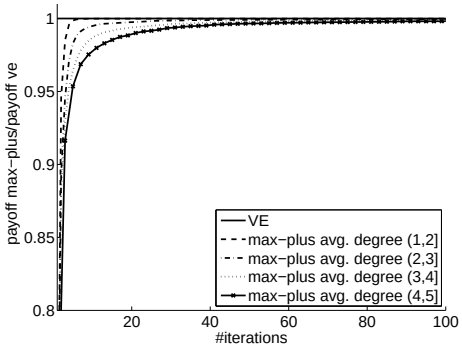
during a local maximization of an agent with five neighbors $5^5 = 3,125$ actions have to be enumerated in the case of 5 actions per agent. With 10 actions per agent, this number increases to $10^5 = 100,000$ actions. During elimination the topology of the graph can change to very dense graphs resulting in even larger maximizations. This is also evident from the experiments. For some graphs with ten actions per agent and an average degree higher than 3.2, the size of the intermediate tables grows too large for the available memory, and VE is not able to produce a result. These graphs are removed from the set. For the graphs with an average degree between 3 and 4, this results in the removal of 81 graphs. With an increase of the average degree, this effect becomes more apparent: VE is not able to produce a result for 466 out of the 700 graphs with an average degree higher than 4; all these graphs are removed from the set. This also explains why the increase in the curve of VE in Fig. 3.8(c) decreases: the more difficult graphs, which take longer to complete, are not taken into account. Even without these graphs, it takes VE on average 339.76 seconds, almost 6 minutes, to produce a joint action for the graphs with an average degree of 5. The max-plus algorithm, on the other hand, needs on average 31.61 milliseconds.

The max-plus algorithm thus outperforms VE with respect to the computation time for densely connected graphs. But how do the resulting joint actions of the max-plus algorithm compare to the optimal solutions of the VE algorithm? Fig. 3.9 shows the payoff found with the max-plus algorithm relative to the optimal payoff after each iteration. A relative payoff of 1 indicates that the found joint action corresponds to the optimal joint action, while a relative payoff of 0 indicates that it corresponds to the joint action with the minimal possible payoff. All four displayed curves correspond to the average result of a subset with a similar average degree. Specifically, each subset contains all graphs with an average degree in $(x-1, x]$, with $x \in \{2, 3, 4, 5\}$.

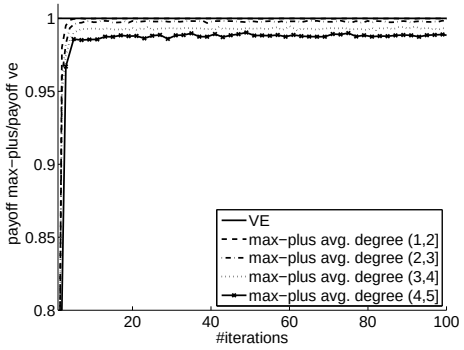
We first discuss the result of the standard max-plus algorithm in the graphs on the left. For all three sets, the loosely connected graphs with an average degree less than two converge to a similar policy as the optimal joint action in a few iterations only. As the average degree increases, the resulting policy declines. As seen in Fig. 3.9(c), this effect is less evident in the graphs with outliers; the action combinations related to the positive outliers are clearly preferred, and lowers the number of oscillations.



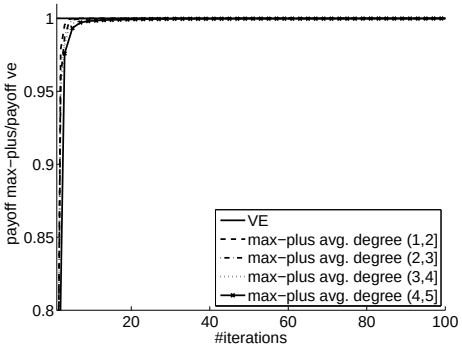
(a) Max-plus (5 actions per agent).



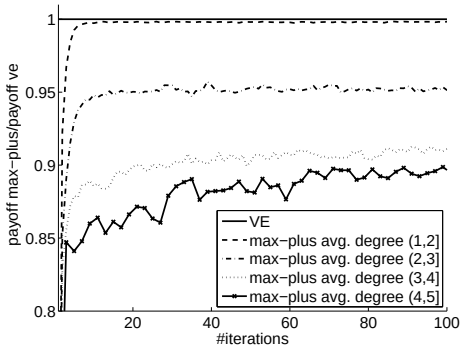
(b) Anytime max-plus (5 actions).



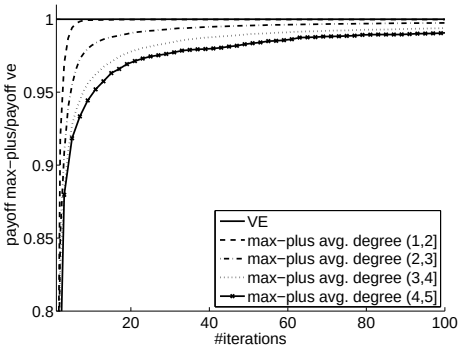
(c) Max-plus (5 actions per agent and outliers)



(d) Anytime max-plus (5 actions and outliers).



(e) Max-plus (10 actions per agent).



(f) Anytime max-plus (10 actions).

Figure 3.9: Relative payoff compared to VE for standard max-plus (graphs on the left) and anytime max-plus (graphs on the right) for graphs with 15 agents and cycles.

Increasing the number of actions per agents has a negative influence on the result because of the increase in the total number of action combinations, as is evident from Fig. 3.9(e). Note that the oscillations for the graphs with the highest average degree are caused by the fact that VE is not able to produce a result for many of the problems with this complexity, and the average is taken over a smaller set.

When the anytime version, which always returns the best joint action found so far, is applied, the obtained payoff improves for all graphs. This indicates that the failing convergence of the messages causes the standard max-plus algorithm to oscillate between different joint actions and 'forget' good joint actions. Fig. 3.9 shows that for all sets near-optimal policies are found, although the more complex graphs need more iterations to find them.

3.5 Discussion

In this chapter we addressed the problem of coordinating the behavior of a large group of agents. We described a payoff propagation algorithm, the max-plus algorithm, that can be used as an alternative to variable elimination (VE) for finding the optimal joint action in a coordination graph (CG) with predefined payoff functions. VE is an exact method that always reports the joint action that maximizes the global payoff, but is slow for densely connected graphs with cycles as its worst-case complexity is exponential in the number of agents. Furthermore, this method is only able to report a solution after the complete algorithm has ended. The max-plus algorithm, analogous to the belief propagation algorithm in Bayesian networks, operates by repeatedly sending local payoff messages over the edges in the CG. By performing a local computation based on its incoming messages, each agent is able to select its individual action. For tree-structured graphs, this approach is identical to VE, and also results in the optimal joint action. For large, highly connected graphs with cycles, we provided empirical evidence that the max-plus algorithm can find good solutions exponentially faster than VE. Our anytime extension, which occasionally evaluates the current joint action and stores the best one found so far, ensures that the agents select a coordinated joint action and essentially produces a convergent max-plus variant. It results in near-optimal performance. An advantage of the max-plus algorithm is that it can be implemented fully distributed using asynchronous and parallel message passings and, contrary to VE, the graph's topology remains unchanged: in all cases the agents have to coordinate only with their neighbors in the original graph.

In the subsequent chapter we will apply both the VE and the max-plus algorithm as the building blocks for our learning algorithms in which it is important to quickly compute the best joint action in a distributed manner.

4

Multiagent Learning

In this chapter we focus on sequential decision-making problems in which the agents repeatedly interact with their environment and try to optimize the long-term reward they receive from the system. We present a family of model-free multiagent reinforcement-learning techniques, called sparse cooperative Q-learning, which approximate the global action-value function based on the topology of a coordination graph, and perform local updates using the contributions of the individual agents to the maximal global action value [Kok and Vlassis, 2006]. The combined use of a decomposition of the action-value function based on the edges of a coordination graph and the max-plus algorithm for efficient action selection results in an approach that scales only linearly in the problem size. We provide experimental evidence that our sparse cooperative Q-learning methods outperform related multiagent reinforcement-learning methods based on temporal differences.

4.1 Introduction

In Chapter 3 we discussed the problem of selecting an optimal joint action in a group of agents for a given payoff structure in single-state problems. In this chapter, we consider *sequential decision-making problems* in which the agents repeatedly select actions. After the execution of a joint action, the agents receive a reward and the system transitions to a new state. The goal of the agents is to select the actions that optimize a shared performance measure based on the received rewards. This might involve a sequence of decisions. We will only concentrate on problems in which the agents have no prior knowledge about the effect of their actions, and thus have to *learn* this information based on the delayed rewards. Furthermore, we focus on *inherently cooperative* tasks involving a large group of agents in which the success of the team is measured by the specific action combination of the agents [Parker, 2002]. This is in contrast with other approaches which assume implicit coordination through either the observed state variables [Dutta et al., 2005; Tan, 1993], or reward structure [Becker et al., 2003]. Possible examples of application domains include network routing [Boyan and Littman, 1994; Dutta et al., 2005], sensor networks [Lesser et al., 2003; Modi et al., 2005], but also robotic teams, for example, motion coordination [Arai et al., 2002] and RoboCup [Kitano et al., 1995; Kok et al., 2005b].

Existing learning techniques have been proved successful in learning the behavior of a single agent in stochastic environments [Tesauro, 1995; Crites and Barto, 1996; Ng et al., 2004]. However, as stated earlier in Section 2.3.4, the presence of multiple learning agents in the same environment complicates matters. First of all, the action space scales exponentially with the number of agents. This makes it infeasible to apply standard single-agent techniques in which an action value, representing expected future reward, is stored for every possible state-action combination. An alternative approach is to decompose the global action value among the different agents. Each agent then updates a local action value which only depends on its own individual action. However, each agent now ignores the actions of the other agents, but these actions still influence the received reward and resulting state. As a result the environment becomes dynamic from the viewpoint of a single agent, which possibly compromises convergence. Other complications, which are outside the focus of this thesis, arise when the different agents receive incomplete and noisy observations of the state space [Goldman and Zilberstein, 2004], or have a restricted communication bandwidth [Goldman and Zilberstein, 2003; Pynadath and Tambe, 2002].

For our model representation we will use the collaborative multiagent Markov decision process (collaborative MMDP) model described in Section 2.3.3. In this model each agent selects an individual action in a particular state. The resulting joint action, that is, the combination of all individual actions, causes the transition to a new state and provides each agent an *individual* reward. The global reward is the sum of all individual rewards. This approach differs from other multiagent models, for example, multiagent MDPs (MMDPs) [Boutilier, 1996] or decentralized MDPs (DEC-MDPs) [Bernstein et al., 2000], in which all agents observe the global reward directly. In a collaborative MAS, it is still the goal of the agents to optimize the global reward, but the individual received rewards allow for solution techniques that take advantage of the structure of the problem [Bagnell and Ng, 2006].

In this chapter, we study sequential decision-making problems and learn the behavior of a group of agents using model-free reinforcement-learning techniques [Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998]. In our approach, called *sparse cooperative Q-learning*, we analyze different decompositions of the global action-value function using the framework of coordination graphs (CGs) described in Section 3.2. The structure of the used CG is determined beforehand, and should reflect the specific problem under study. For a given CG, we investigate both a decomposition in terms of the nodes (or agents), as well as a decomposition in terms of the edges of the graph. In the agent-based decomposition the local action-value function of an agent is based on its own action and those of its neighboring agents. In the edge-based decomposition each local function is based on the actions of the two agents that form an edge in the graph. The global value of a joint action in the used CG represents the action value for a specific state. In a sequential decision-making problem, however, we have to store action values for multiple states, and therefore we associate each state to a CG with a similar decomposition, but with different values for the local functions. In order to update the local action-value function for a specific state, the contributions of the involved agents to the maximal global action

value, computed using either the max-plus or the variable elimination (VE) algorithm, are used. We show that the combination of the edge-based decomposition and the max-plus algorithm scales linearly in the number of dependencies.

We apply our approach to different problems involving a large group of agents with many dependencies, and compare it to four other multiagent variants of tabular Q-learning, three of which, that is, the MDP learners, independent learners (IL), and distributed value functions (DVF), are already described in Section 2.3.4. The remaining variant, coordinated reinforcement learning, is reviewed in Section 4.2. We show that our method outperforms all these existing temporal-difference learning techniques in terms of the quality of the extracted policy. In this thesis, we only consider temporal-difference methods. We do not discuss other multiagent reinforcement-learning methods which are based, for example, on policy search [Peshkin et al., 2000; Moallemi and Van Roy, 2004] or Bayesian approaches [Chalkiadakis and Boutilier, 2003]. Furthermore, we also do not consider function-approximation algorithms. They have been successfully applied in domains with large continuous state sets [Bertsekas and Tsitsiklis, 1996; Stone et al., 2005], but they are less applicable for domains with many joint actions because it is more difficult to construct an appropriate distance measure for discrete joint actions than for continuous state variables.

The remainder of this chapter is structured as follows. In Section 4.2 we review coordinated reinforcement learning. In Section 4.3, we introduce the different decompositions of our sparse cooperative Q-learning methods. In Section 4.4 we describe our experiments and give results on both a stateless problem and a distributed sensor network. Finally, we end with some concluding remarks in Section 4.5.

4.2 Coordinated reinforcement learning

Guestrin et al. [2002b] describe three *coordinated reinforcement-learning* (CoordRL) approaches that take advantage of the structure of the problem under study. The three methods are respectively a variant of Q-learning, policy iteration, and direct policy search. In this section we review the Q-learning variant and discuss its advantages and disadvantages. The main idea of this method is to decompose the global Q-function into a linear combination of local agent-dependent Q-functions:

$$Q(\mathbf{s}, \mathbf{a}) = \sum_{i=1}^{n} Q_i(\mathbf{s}_i, \mathbf{a}_i). \tag{4.1}$$

Each local Q-function Q_i for an agent i is based on $\mathbf{s}_i$ and $\mathbf{a}_i$ which respectively represent the subset of all state and action variables relevant for agent i. These dependencies are established beforehand and differ per problem. Note that using this representation, each agent only needs to observe the state variables $\mathbf{s}_i$ which are part of its local Q-function. The corresponding CG is constructed by adding an edge between agent i and j when the action of agent j is included in the action variables of agent i, that is, $a_j \in \mathbf{a}_i$. As an example, imagine a computer network in which

each machine is modeled as an agent and each machine only depends on the state and action variables of itself and the machines it is connected to. In this case, the CG equals the network topology.

In the work of Guestrin et al. [2002b], a local Q-function is updated using a similar method to the local Q-learning update rule for the IL approach in (2.14). However, an update is now based on the *global* temporal-difference error, the difference between the current global Q-value and the expected future discounted return for the experienced state transition. The complete update then looks as follows:

$$Q_i(\mathbf{s}_i, \mathbf{a}_i) := Q_i(\mathbf{s}_i, \mathbf{a}_i) + \alpha[R(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}') - Q(\mathbf{s}, \mathbf{a})]. \tag{4.2}$$

In this equation, the global reward $R(\mathbf{s}, \mathbf{a})$ is given. The maximizing action in $\mathbf{s}'$ and the associated maximal expected future return, $\max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}')$, are computed in a distributed manner by applying the variable elimination (VE) algorithm on the CG as discussed in Section 3.2. Finally, the estimate of the global Q-value in $\mathbf{s}$, $Q(\mathbf{s}, \mathbf{a})$ in (4.2), is computed by fixing the action of every agent to the one assigned in $\mathbf{a}$ and applying a message passing scheme similar to the one used in the VE algorithm. Note that we have used a table-based representation for the Q-functions in our discussion. However, since each individual Q-function is entirely local, each agent is allowed to choose its own representation, for example, using a function approximator [Guestrin et al., 2002b].

The advantage of the CoordRL approach is that it is completely distributed. Each agent keeps a local Q-function and only has to exchange messages with its neighbors in the graph in order to compute the global Q-values. In sparsely connected graphs, this results in large computational savings since the complete joint action space can be approximated by a sum of small local functions. However, both the space and computational complexity grow exponentially with the number of agents resulting in problems for densely connected graphs in which many agents depend on each other. The growth in the space complexity is caused by the fact that the size of each local Q-function grows exponentially with the number of involved agents, that is, the degree of the corresponding node in the graph, since each agent constructs an action-value based on all action combinations of its own action and those of its neighbors. The growth in the computational complexity is a result of the computation using the VE algorithm required to compute the maximizing joint action. This algorithm grows exponential with the induced width of the graph, as was shown in Section 3.4,

Next, we describe our sparse cooperative Q-learning approach. This method also decomposes the global Q-learning into a linear combination of local Q-functions. We will both investigate an agent-based decomposition, similar to the CoordRL approach, and an edge-based decomposition. Using the latter decomposition, the max-plus algorithm can be used to compute the maximizing the joint action, which results in large savings in the computational complexity of the algorithm for densely connected graphs. Another difference with respect to the CoordRL approach is related to the update of the action values: each local function is updated based on its own local contribution to the global function.

4.3 Sparse cooperative Q-learning

In this section, we describe our sparse cooperative Q-learning, or SparseQ, approach: a family of methods which approximate the global Q-function by a linear combination of local Q-functions. Just as in the coordination reinforcement-learning approach (CoordRL), the decomposition is based on the structure of a CG which is chosen beforehand. In principle we can select any CG, but useful domain knowledge can be exploited by choosing a CG that incorporates the dependencies corresponding to the problem under study. For a given CG, we investigate both a decomposition in terms of the nodes, as well as a decomposition in terms of the edges of the graph. In the agent-based decomposition the local function of an agent is, just as in the CoordRL approach, based on its own action and those of its neighboring agents. In the edge-based decomposition, however, each local function is related to the actions of the two agents that form an edge. In order to update a local function, the key idea is to base the update on the *local* temporal-difference error, that is, the difference between the current local Q-value and the local *contribution* of this agent to the global return. This differs from the CoordRL approach which, as in (4.2), always uses the *global* temporal-difference error for its updates.

Next, we first describe an agent-based decomposition of the global Q-function and explain how the local contribution of an agent is used in the update step. Thereafter, in Section 4.3.2, we describe an edge-based decomposition, together with two possible update methods: an edge-based and an agent-based update method.

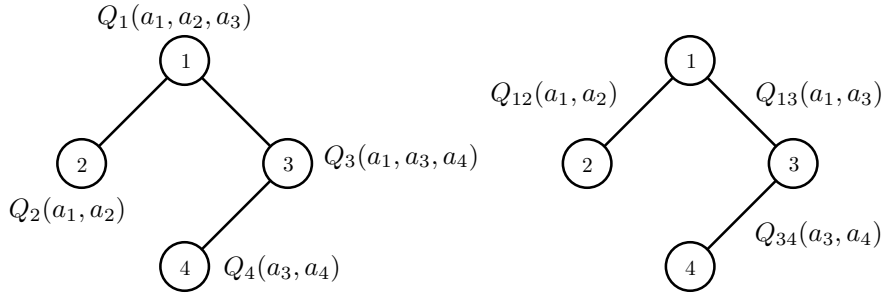
4.3.1 Agent-based decomposition

Next, we describe an agent-based decomposition of the global action value in combination with a local update rule. As in CoordRL, the global Q-function is decomposed over the different agents. Every agent i is associated with a local Q-function $Q_i(\mathbf{s}_i, \mathbf{a}_i)$ which only depends on a subset of all possible state and action variables. These dependencies are specified beforehand and depend on the problem under study. The local Q-functions correspond to a CG that is constructed by connecting all agents that depend on each other, that is, for agent i and j either $a_j \in \mathbf{a}_i$ or $a_i \in \mathbf{a}_j$. See Fig. 4.1(a) for an example agent-based decomposition for a 4-agent problem.

Since the global Q-function equals the sum of the local Q-functions of all n agents, $Q(\mathbf{s}, \mathbf{a}) = \sum_{i=1}^{n} Q_i(\mathbf{s}_i, \mathbf{a}_i)$, we can rewrite the Q-learning update rule in (2.12) as

$$\sum_{i=1}^{n} Q_i(\mathbf{s}_i, \mathbf{a}_i) := \sum_{i=1}^{n} Q_i(\mathbf{s}_i, \mathbf{a}_i) \; + \\ \alpha\big[\sum_{i=1}^{n} R_i(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}') - \sum_{i=1}^{n} Q_i(\mathbf{s}_i, \mathbf{a}_i)\big]. \quad (4.3)$$

Only the expected discounted return, $\max_{a'} Q(\mathbf{s}', \mathbf{a}')$, cannot be directly written as the sum of local terms since it depends on the *globally* maximizing joint action. However,



(a) Agent-based decomposition.

(b) Edge-based decomposition.

Figure 4.1: An agent-based and edge-based decomposition of the global Q-function.

we can use the VE algorithm to compute, in a distributed manner, the maximizing joint action $\mathbf{a}^* = \arg\max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}')$ in state $\mathbf{s}'$, and from this compute the local contribution $Q_i(\mathbf{s}'_i, \mathbf{a}^*_i)$ of each agent to the total action value $Q(\mathbf{s}', \mathbf{a}^*)$. Note that the local contribution of an agent to the global action value might be lower than the maximizing value of its local Q-function because it is unaware of the dependencies of its neighboring agents with the other agents in the CG. Since we can substitute $\max_{a'} Q(\mathbf{s}', \mathbf{a}')$ with $\sum_{i=1}^{n} Q_i(\mathbf{s}', \mathbf{a}^*_i)$, we are able to decompose all terms in (4.3) and rewrite the update for each agent i separately:

$$Q_i(\mathbf{s}_i, \mathbf{a}_i) := Q_i(\mathbf{s}_i, \mathbf{a}_i) + \alpha[R_i(\mathbf{s}, \mathbf{a}) + \gamma Q_i(\mathbf{s}'_i, \mathbf{a}^*_i) - Q_i(\mathbf{s}_i, \mathbf{a}_i)]. \tag{4.4}$$

This update is completely based on local terms and only requires the distributed VE algorithm to compute the maximizing joint action $\mathbf{a}^*$. In contrast to CoordRL, we directly take advantage of the local rewards received by the different agents. Especially for larger problems with many agents, this allows us to propagate back the reward to the local functions related to the agents responsible for the generated rewards. This is not possible in CoordRL which uses the global reward to update the different local functions. As a consequence, the agents are not able to distinguish which agents are responsible for the received reward, and all functions, including the ones which are not related to the received reward, are updated equally. It might even be the case that the high reward generated by one agent, or a group of agents, is counterbalanced by the negative reward of another agent. In this case, the combined global reward equals zero and no functions are updated.

Just as in CoordRL, both the representation of the local Q-functions and the computation time of the VE algorithm used to compute the global joint action grow exponentially with the number of neighbors. This becomes problematic for densely connected graphs, and for this reason we also investigate an edge-based decomposition of the Q-function which does not suffer from this problem.

4.3.2 Edge-based decomposition

A different method to decompose the global Q-function is to define it in terms of the edges of the corresponding CG. Contrary to an agent-based decomposition, which scales exponentially with the number of neighbors in the graph, an edge-based decomposition scales linearly in the number of neighbors. For a coordination graph $G = (V, E)$ with $|V|$ vertices and $|E|$ edges, each edge $(i, j) \in E$ corresponds to a local Q-function Q_{ij}, and the sum of all local Q-functions defines the global Q-function:

$$Q(\mathbf{s}, \mathbf{a}) = \sum_{(i,j)\in E} Q_{ij}(\mathbf{s}_{ij}, a_i, a_j), \tag{4.5}$$

where $\mathbf{s}_{ij} \subseteq \mathbf{s}_i \cup \mathbf{s}_j$ is the subset of the state variables related to agent i and agent j which are relevant for their dependency. Note that each local Q-function Q_{ij} always depends on the actions of two agents, a_i and a_j, only. Fig. 4.1(b) shows an example of an edge-based decomposition for a 4-agent problem.

An important consequence of this decomposition is that it only depends on pairwise functions. This makes it possible to directly apply the max-plus algorithm from Section 3.3 to compute the maximizing joint action, something which is not possible for the agent-based decomposition. For the edge-based decomposition, both the action-value function and the method for action selection now scale linearly in the number of dependencies, resulting in an approach that can be applied to large agent networks with many dependencies.

In order to update a local Q-function, we have to propagate back the reward received by the individual agents. This is complicated by the fact that the rewards are received per agent, while the local Q-functions are defined over the edges. For an agent with multiple neighbors it is therefore not possible to derive which dependency generated (parts of) the reward. Our approach is to associate each agent with a local Q-function Q_i that is directly computed from the edge-based Q-functions Q_{ij}. This allows us to relate the received reward of an agent directly to its agent-based Q-function Q_i. In order to compute Q_i, we assume that each edge-based Q-function contributes equally to the two agents that form the edge. Then, the local Q-function Q_i of agent i is defined as the summation of half the value of all local Q-functions Q_{ij} of agent i and its neighbors $j \in \Gamma(i)$, that is,

$$Q_i(\mathbf{s}_i, \mathbf{a}_i) = \frac{1}{2} \sum_{j\in\Gamma(i)} Q_{ij}(\mathbf{s}_{ij}, a_i, a_j). \tag{4.6}$$

Note that the sum of all local Q-functions Q_i equals Q in (4.5). Note that there are also other approaches possible to divide the reward over the different agents, for example, weighted based on the current action values. In this thesis, however, we assume that each agent contributes equally to the received reward.

Next, we will describe two different update methods for the edge-based decomposition which are defined in terms of these local agent-based Q-functions.

Edge-based update

The first update method we consider updates each local Q-function Q_{ij} based on its current estimate and its contribution to the maximal return in the next state. For this, we rewrite (4.4) by replacing every instance of Q_i with its definition in (4.6) to

$$\frac{1}{2}\sum_{j\in\Gamma(i)} Q_{ij}(\mathbf{s}_{ij},a_i,a_j) := \frac{1}{2}\sum_{j\in\Gamma(i)} Q_{ij}(\mathbf{s}_{ij},a_i,a_j) \; + \\ \alpha\left[\sum_{j\in\Gamma(i)}\frac{R_i(\mathbf{s},\mathbf{a})}{|\Gamma(i)|} + \gamma\frac{1}{2}\sum_{j\in\Gamma(i)} Q_{ij}(\mathbf{s}'_{ij},a_i^*,a_j^*) - \frac{1}{2}\sum_{j\in\Gamma(i)} Q_{ij}(\mathbf{s}_{ij},a_i,a_j)\right]. \quad (4.7)$$

Note that in this decomposition for agent i we made the assumption that the reward R_i is divided proportionally over its neighbors $\Gamma(i)$. In order to get an update equation for an individual local Q-function Q_{ij}, we remove the sums. Because, one half of every local Q-function Q_{ij} is updated by agent i and the other half by agent j, agent j updates the local Q-function Q_{ij} using a similar decomposition as (4.7). Adding the two gives the following update equation for a single local Q-function Q_{ij}:

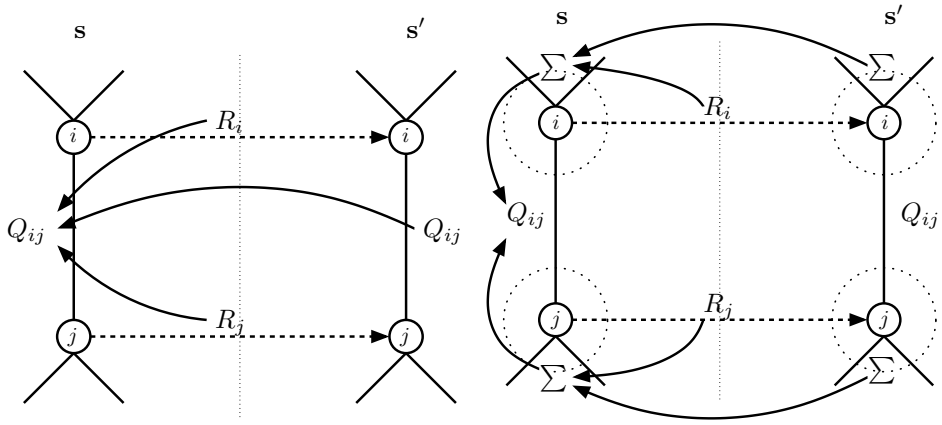
$$Q_{ij}(\mathbf{s}_{ij},a_i,a_j) := Q_{ij}(\mathbf{s}_{ij},a_i,a_j) \; + \\ \alpha\left[\frac{R_i(\mathbf{s},\mathbf{a})}{|\Gamma(i)|} + \frac{R_j(\mathbf{s},\mathbf{a})}{|\Gamma(j)|} + \gamma Q_{ij}(\mathbf{s}'_{ij},a_i^*,a_j^*) - Q_{ij}(\mathbf{s}_{ij},a_i,a_j)\right]. \quad (4.8)$$

Each local Q-function Q_{ij} is thus updated with a proportional part of the received reward of the two agents it is related to and with the contribution of this edge to the maximizing joint action $\mathbf{a}^* = (a_i^*) = \arg\max_{\mathbf{a}'} Q(\mathbf{s}',\mathbf{a}')$ in the next state $\mathbf{s}'$. The latter is computed by either applying the exact VE algorithm or the approximate max-plus algorithm. Note that we can also derive (4.8) from (2.12) directly using (4.5). However, we want to emphasize that it is possible to derive this update rule from the agent-based decomposition discussed in Section 4.3.1.

Fig. 4.2(a) shows a graphical representation of the update. The left part of the figure shows a partial CG in state $\mathbf{s}$. Only the agents i and j, their connecting edge, which is related to a local edge-based Q-function Q_{ij}, and some outgoing edges are depicted. The right part of the figure shows the same structure for state s'. Following (4.8), a local Q-function Q_{ij} is directly updated based on the received reward of the involved agents and the maximizing local Q-function Q_{ij} in the next state.

Agent-based update

In the edge-based update method the reward is divided proportionally over the different edges of an agent. All other terms are completely local and only correspond to the local Q-function Q_{ij} of the edge that is updated. A different approach is to first compute the temporal-difference error *per agent* and divide this value over the edges.



(a) Edge-based update. (b) Agent-based update.

Figure 4.2: A graphical representation of the edge-based and agent-based update method after the transition from state $\mathbf{s}$ to $\mathbf{s}'$. See the text for a detailed description.

For this, we first rewrite (4.4) for agent i using (4.6) to

$$\frac{1}{2}\sum_{j\in\Gamma(i)} Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) := \frac{1}{2}\sum_{j\in\Gamma(i)} [Q_{ij}(\mathbf{s}_{ij}, a_i, a_j)] + \alpha[R_i(\mathbf{s},\mathbf{a}) + \gamma Q_i(\mathbf{s}'_i, \mathbf{a}^*_i) - Q_i(\mathbf{s}_i, \mathbf{a}_i)]. \quad (4.9)$$

In order to transfer (4.9) into a local update function, we first rewrite the temporal-difference error as a summation of the neighbors of agent i, by

$$R_i(\mathbf{s},\mathbf{a}) + \gamma Q_i(\mathbf{s}'_i, \mathbf{a}^*_i) - Q_i(\mathbf{s}_i, \mathbf{a}_i) = \sum_{j\in\Gamma(i)} \frac{R_i(\mathbf{s},\mathbf{a}) + \gamma Q_i(\mathbf{s}'_i, \mathbf{a}^*_i) - Q_i(\mathbf{s}_i, \mathbf{a}_i)}{|\Gamma(i)|}. \quad (4.10)$$

Note that this summation only decomposes the temporal-difference error into j equal parts, and thus does not use j explicitly. Because now all summations are identical, we can decompose (4.9) by removing the sums. Just as in the edge-based update, there are two agents which update the same local Q-function Q_{ij}. When we add the contributions of the two involved agents i and j, we get the local update equation

$$Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) := Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) + \alpha \sum_{k\in\{i,j\}} \frac{R_k(\mathbf{s},\mathbf{a}) + \gamma Q_k(\mathbf{s}'_k, \mathbf{a}^*_k) - Q_k(\mathbf{s}_k, \mathbf{a}_k)}{|\Gamma(k)|}. \quad (4.11)$$

This agent-based update rule propagates back the temporal-difference error from the two agents which are related to the local Q-function of the edge that is updated, and

incorporates the information of *all* edges of these agents. This is different from the edge-based update method which directly propagates back the temporal-difference error related to the edge that is updated. This is depicted in Fig. 4.2(b) which shows the agent-based update. Again, the left part of the figure represents the situation in state $\mathbf{s}$, and the right part the situation in the next state $\mathbf{s}'$. The edge-based Q-function Q_{ij} is updated based on the local agent-based Q-functions of the two agents that form the edge. These functions are computed by summing over the local edge-based Q-functions of all neighboring edges.

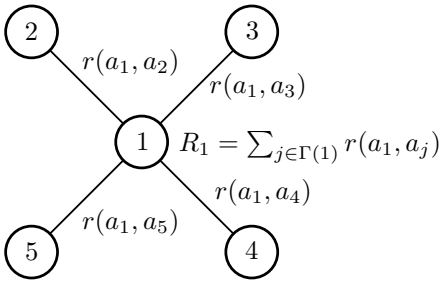
Next, we perform several experiments using both the agent-based and the edge-based decomposition. For the latter, we apply both the agent-based and edge-based update method, and show the consequences, both in speed and solution quality, of using the max-plus algorithm as an alternative to the VE algorithm.

4.4 Experiments

We perform different experiments using the methods discussed in Section 4.3 and four existing multiagent Q-learning variants. We both investigate a large stateless problem and the distributed sensor network problem that was part of the NIPS 2005 benchmarking workshop. We have chosen these problems because they are both fully specified and, more importantly, require the selection of a specific combination of actions at every time step. This is in contrast with other experiments in which coordination is modeled implicitly through the state variables, that is, each agent is able to select its optimal action based on only the state variables (for example, its own and other agents' positions) and does not have to model the action of the other agent [Tan, 1993; Guestrin et al., 2002b; Becker et al., 2003].

4.4.1 Stateless problems

Now, we describe several experiments in which a group of n agents have to learn to take the optimal joint action in a single-state problem. The agents repeatedly interact with their environment by selecting a joint action. After the execution of a joint action $\mathbf{a}$, the episode is immediately ended and the system provides each agent an individual reward $R_i(\mathbf{a})$. The goal of the agents is to select the joint action $\mathbf{a}$ which maximizes $R(\mathbf{a}) = \sum_{i=1}^{n} R_i(\mathbf{a})$. The local reward R_i received by an agent i only depends on a subset of the actions of the other agents. These dependencies are modeled using a graph in which each edge corresponds to a local reward function that assigns a value $r(a_i, a_j)$ to each possible action combination of the actions of agent i and agent j. Each local reward function is fixed beforehand and contains one specific pair of actions, $(\tilde{a}_i, \tilde{a}_j)$ that results in a high random reward, uniformly distributed in the range $[5, 15]$, that is, $5 + \mathcal{U}([0, 10])$. However, failure of coordination, that is, selecting an action $r(\tilde{a}_i, a_j)$ with $a_j \neq \tilde{a}_j$ or $r(a_i, \tilde{a}_j)$ with $a_i \neq \tilde{a}_i$, will always result in a reward of 0. All remaining joint actions, $r(a_i, a_j)$ with $a_i \neq \tilde{a}_i$ and $a_j \neq \tilde{a}_j$, give a default reward drawn from the uniform distribution $\mathcal{U}([0, 10])$. The



(a) Construction of reward $R_1(\mathbf{a})$.

action agent i \ action agent j	1	2	3	4
1	9.155	0	9.293	0.196
2	7.651	0	2.872	1.287
3	0	12.020	0	0
4	4.536	0	1.581	8.138

(b) Example $r(a_i, a_j)$ function.

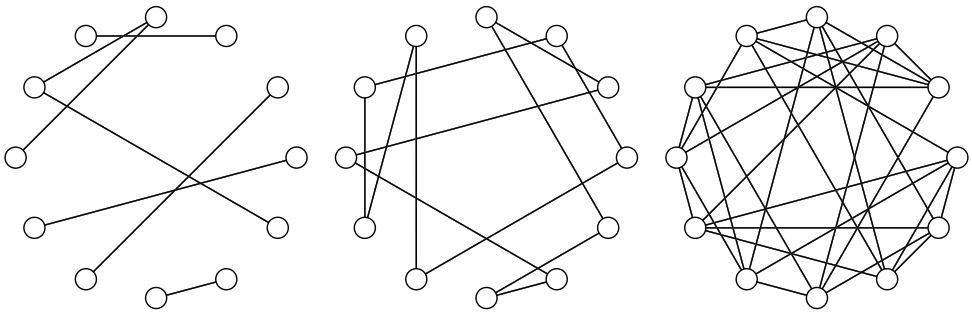
Figure 4.3: Construction of the reward for agent 1 in the single-state problem. (a) The individual reward R_1 is the sum of the rewards $r(a_1, a_j)$ from the interactions with its neighbors $j \in \Gamma(1) = \{2, 3, 4, 5\}$. (b) Example $r(a_i, a_j)$ function.

individual reward R_i for an agent i equals the sum of the local rewards resulting from the interactions with its neighbors, $R_i(\mathbf{a}) = \sum_{j \in \Gamma(i)} r(a_i, a_j)$. Fig. 4.3 shows an example of the construction of the individual reward received by an agent based on its interaction with its four neighbors, together with an example reward function $r(a_i, a_j)$ corresponding to an edge between agent i and agent j.

The goal of the agents is to learn, based on the received individual rewards, to select a joint action that maximizes the global reward. Although we assume that the agents know on which other agents it depends, this goal is complicated by two factors. First, the outcome of a selected action of an agent also depends on the actions of its neighbors. For example, the agents must coordinate in order to select the joint action $(\tilde{a}_i, \tilde{a}_j)$ which, in most cases, returns the highest reward. Failure of coordination, however, results in a low reward. Secondly, because each agent only receives an individual reward, they are not able to derive which neighbor interaction caused which part of the reward.

Note that this is a different problem than the problem specified in Section 3.4. In that problem, it is the goal of the agents to select a joint action which maximizes predefined payoff functions. In this problem, the payoff relations themselves have to be learned based on the received rewards.

We perform experiments with 12 agents, each able to perform 4 actions. The group as a whole thus has $4^{12} \approx 1.7 \cdot 10^7$, or almost 17 million, different joint actions. We investigate reward functions with different complexities, and apply the method described in Section 3.4 to randomly generate 20 graphs $G = (V, E)$ with $|V| = 12$ for each $|E| \in \{7, 8, \ldots, 30\}$, resulting in a total of 480 graphs, that is, 20 graphs in each of the 24 groups. The agents of the simplest graphs (7 edges) have an average degree of 1.16, while the most complex graphs (30 edges) have an average degree of 5. Fig. 4.4 shows three different example graphs with different average degrees. Fig. 4.4(a) and (c) depict respectively the minimum and maximal considered average degree, while Fig. 4.4(b) shows a graph with an average degree of 2.



(a) A graph with 7 edges (average degree of 1.16).

(b) A graph with 12 edges (average degree of 2).

(c) A graph with 30 edges (average degree of 5.00).

Figure 4.4: Example graphs with 12 agents and different average degrees.

We apply the different variants of our sparse cooperative Q-learning method described in Section 4.3 and different existing multiagent Q-learning methods, discussed in Section 2.3.4 and Section 4.2, to this problem. Since the problem consists of only a single state, all Q-learning methods store Q-functions based on actions only. We assume that the agents have access to a CG which for each agent specifies on which other agents it depends. This CG is identical to the topology of the graph that is used to generate the reward function. Apart from the different Q-learning methods, we also apply an approach that selects a joint action uniformly at random, and a method that enumerates all possible joint actions and stores the one with the highest reward. To summarize, we now review the main characteristics of all applied methods:

Independent learners (IL) Each agent stores a local Q-function $Q_i(a_i)$ which only depends on its own action. Agent i uses its private reward R_i to perform the update according to (2.14). In order to select an action, each agent selects the action that maximizes its own local Q-function Q_i.

Distributed value functions (DVF) Each agent i stores a local Q-function based on its own action, and an update incorporates the Q-functions of its neighbors following (2.15). For stateless problems, as the ones in this section, the Q-value of the next state is not used and this method is identical to IL.

Coordinated reinforcement learning (CoordRL) Each agent i stores an individual Q-function based on its own action and the actions of its neighbors $j \in \Gamma(i)$. Each function is updated based on the *global* temporal-difference error using the update equation in (4.2). This representation scales exponentially with the number of neighbors. VE is used to determine the optimal joint action which scales exponentially with the induced width of the graph.

Sparse cooperative Q-learning, agent-based (SparseQ agent) Each agent in the graph stores a Q-function based on its own action and the actions of its

neighbors $j \in \Gamma(i)$. A function is updated based on the *local* temporal-difference error following (4.4). The representation and computational complexity are similar to those of the CoordRL approach.

Sparse cooperative Q-learning, edge-based (SparseQ edge) Each edge in the used CG is associated with a Q-function based on the actions of the two connected agents. We apply both the *edge-based* update method (SparseQ edge, edge) from (4.8) which updates a Q-function based on the value of the edge that is updated, and the *agent-based* update method (SparseQ edge, agent) from (4.11), which updates a Q-function based on the local Q-functions of the agents forming the edge.

The two update methods are combined with both the VE algorithm and the anytime max-plus algorithm to compute the optimal joint action, resulting in four different methods in total. The max-plus algorithm generates a result when either the messages converge, the best joint action has not improved for 5 iterations, or more than 20 iterations are performed. The latter number of iterations is obtained by comparing the problem under study with the coordination problem from Section 3.4.2. Both problem sizes are similar, and we can conclude from Fig. 3.9 that a good performance is obtained after 20 iterations.

Random method In this method each agent always performs an action selected uniformly at random.

Enumeration In order to compare the quality of the different methods, we compute the optimal value by trying every possible joint action and store the one which results in the highest reward. This requires an enumeration over all possible joint actions. Note that this approach does not perform any updates, and quickly becomes intractable for problems larger than the one addressed here.

We do not apply the MDP learners approach since it would take too long to find a solution. First, it requires an enumeration over 4^{12} (≈ 17 million) actions at every time step. Secondly, assuming there is only one optimal joint action, the probability to actually find the optimal joint action is negligible. An exploration action should be made (probability ϵ), and this exploration action should equal the optimal joint action (probability of $\frac{1}{4^{12}}$).

Table 4.1 shows the average number of Q-values required by each of the three types of decompositions. The numbers are based on the generated graphs and averaged over similarly shaped graphs. Note the exponential growth in the agent-based decomposition that is used in both the CoordRL and agent-based SparseQ approach. We run each method on this problem for $15,000$ learning cycles. Each learning cycle is directly followed by a test cycle in which the reward related to the current greedy joint action is computed. The values from the test cycles, thus without exploration, are used to compare the performance between the different methods. For all Q-learning variants, the Q-values are initialized to zero and the parameters are set to $\alpha = 0.2$, $\epsilon = 0.2$, and $\gamma = 0.9$.

method	$(1, 2]$	$(2, 3]$	$(3, 4]$	$(4, 5]$
IL/DVF	48	48	48	48
edge-based	152	248	344	440
agent-based	528	2,112	8,448	33,792

Table 4.1: Average number of Q-values needed for the different decompositions for graphs with an average degree in $(x - 1, x]$.

Fig. 4.5 shows the timing results for all methods.[1] The x-axis depicts the average degree of the graph. The y-axis, shown in logarithmic scale, depicts the corresponding average number of seconds spent in the $15,000$ learning cycles on graphs with a similar average degree. For the enumeration method it represents the time needed to compute the reward of every possible joint action.

The results show that the random and IL/DVF approach are the quickest and take less than a second to complete. In the IL/DVF method each agent only stores functions based on its individual action and is thus constant in the number of dependencies in the graph. Note that the time increase in the random approach for graphs with a higher average degree is caused by the fact that more local reward functions have to be enumerated in order to compute the reward. This occurs in all methods, but is especially visible in the curve of the random approach since for this method the small absolute increase is relatively large with respect to its computation time.

The CoordRL and the agent-based SparseQ method scale exponentially with the increase of the average degree, both in their representation of the local Q-functions and the computation of the optimal joint action using the VE algorithm. The curves of these methods overlap in Fig. 4.5. Because these methods need a very long time, more than a day, to process graphs with a higher average degree than 3, the results for graphs with more than 18 edges are not computed. The edge-based decompositions do not suffer from the exponential growth in the representation of the local Q-functions. However, this approach still grows exponentially with an increase of the average degree when the VE algorithm is used to compute the maximizing joint action. This holds for both the agent-based and edge-based update method, which overlap in the graph. When the anytime max-plus algorithm is applied to compute the joint action both the representation of the Q-function and the computation of the joint action scale linearly with an increasing average degree. The agent-based update method is slightly slower than the edge-based update method because the first incorporates the neighboring Q-functions in its update (4.11), and therefore the values in the Q-functions are less distinct. As a consequence, the max-plus algorithm needs more iterations in an update step to find the maximizing joint action.

Finally, the enumeration method shows a slight increase in the computation time with an increase of the average degree because, as stated earlier, it has to sum over more local functions for the denser graphs in order to compute the associated value.

[1] All results are generated on an Intel Xeon 3.4GHz / 2GB machine using a C++ implementation.

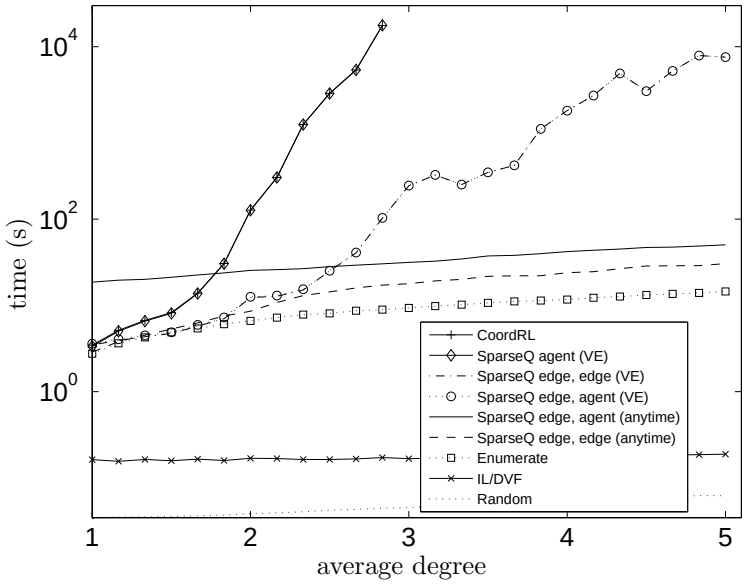


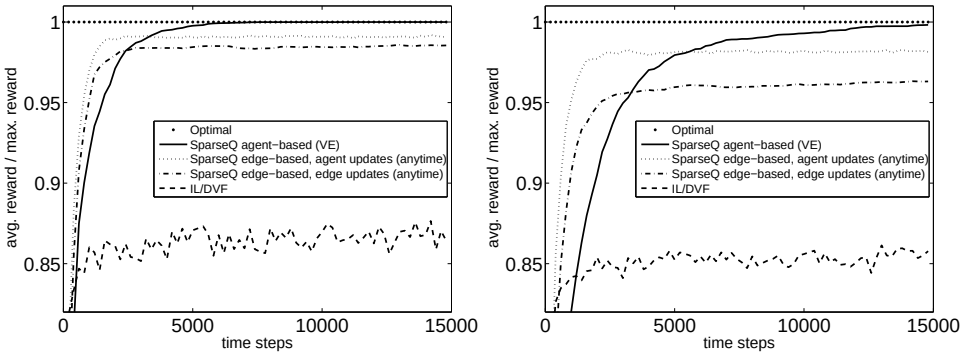
Figure 4.5: Timing results for the different methods applied to the single-state problems with 12 agents and an increasing number of edges. The results overlap for the CoordRL and the agent-based SparseQ decomposition, and the two edge-based decompositions using the VE algorithm.

Note that the problem size was chosen such that the enumeration method was able to produce a result for all different graphs.

Fig. 4.6 shows the corresponding performance for the most relevant methods. Each figure depicts the running average, of the last 10 cycles, of the obtained reward relative to the optimal reward, determined using the enumeration method, for the first $15,000$ cycles. Results are grouped for graphs with a similar complexity, that is, having about the same number of edges per graph.

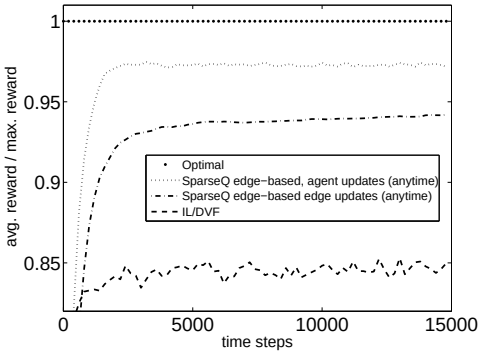
Fig. 4.6(a) depicts the results for the simplest graphs with an average degree less than or equal to 2. We do not show the results for the random and CoordRL approach since they are not able to learn a good policy and quickly stabilize around 41% of the optimal value. The CoordRL approach updates each local Q-function with the global temporal-difference error. Therefore, the same global reward is propagated to each of the individual Q-functions and the expected future discounted return, that is, the sum of the local Q-functions, is overestimated. As a result the Q-values blow up, resulting in random behavior.

The IL/DVF approach learns a reasonable solution, but it suffers from the fact that each agent individually updates its Q-value irrespective of the actions performed by its neighbors. Therefore, the agents are not able to learn coordinated actions and the policy keeps oscillating.

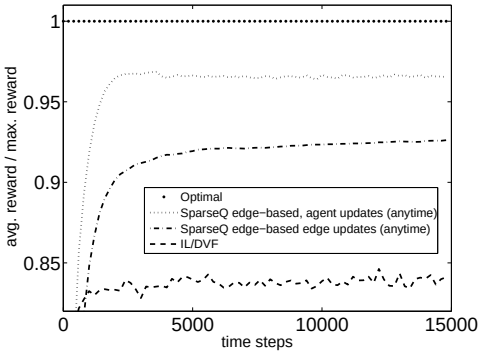


(a) Average degree less than 2.

(b) Average degree between 2 and 3.



(c) Average degree between 3 and 4.

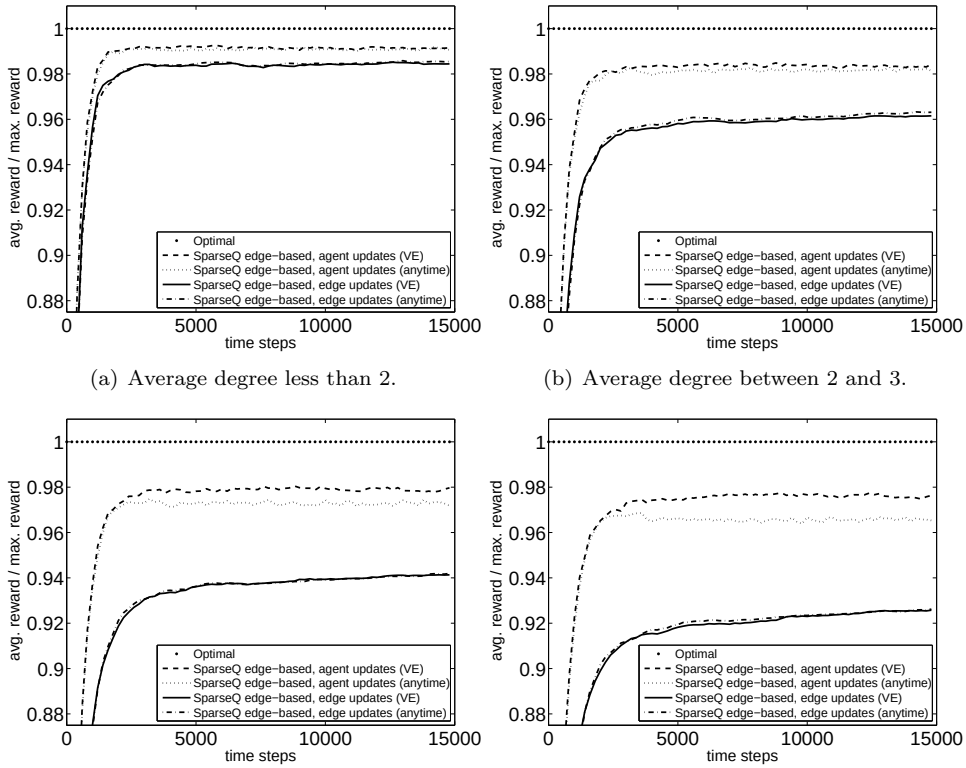


(d) Average degree between 4 and 5.

Figure 4.6: Running average, of the last 10 cycles, of the reward relative to the optimal reward for the most relevant methods on the single-state, 12-agent problems.

The agent-based SparseQ decomposition converges to an optimal policy since it stores a Q-value for every action combination of its neighbors, and is able to detect the best performing action combination. However, this approach learns slower than the different edge-based decompositions since it requires, as listed in Table 4.1, more samples to update the large number of Q-values. The two edge-based decompositions using the anytime extension both learn a near-optimal solution. The agent-based update method performs slightly better since it, indirectly, includes the neighboring Q-values in its update rule.

As is seen in Fig. 4.6(b), the results are similar for the more complicated graphs with an average degree between 2 and 3. Although not shown, the random policy and CoordRL learners are not able to learn a good policy and quickly stabilize around 44% of the optimal value. On the other hand, the agent-based decomposition converges to the optimal policy. Although the final result is slightly worse compared to the simpler



(a) Average degree less than 2.

(b) Average degree between 2 and 3.

(c) Average degree between 3 and 4.

(d) Average degree between 4 and 5.

Figure 4.7: Running average of the received reward relative to the optimal reward for the different edge-based methods, using either the VE or anytime algorithm, on the single-state, 12-agent problem.

graphs, the edge-based decompositions still learn near-optimal policies. The result of the agent-based update method is better than the edge-based update method since the first includes the neighboring Q-values in its update rule.

Similar results are also visible in Fig. 4.6(c) and Fig. 4.6(d), with respectively an average degree between 3 and 4, and between 4 and 5. The agent-based decompositions are not applied to these graphs. As was already visible in Fig. 4.5, the algorithm needs too much time to process graphs of this complexity.

Fig. 4.7 compares the difference between using either the VE or the anytime max-plus algorithm to compute the joint action for the SparseQ methods using an edge-based decomposition. Fig. 4.7(a) and Fig. 4.7(b) show that the difference between the two approaches is negligible for the graphs with an average degree less than 3. However, for the more complex graphs (Fig. 4.7(c) and Fig. 4.7(d)) there is

method	$(1, 2]$	$(2, 3]$	$(3, 4]$	$(4, 5]$
Random	0.4272	0.4421	0.4477	0.4513
IL	0.8696	0.8571	0.8474	0.8372
CoordRL	0.4113	0.4423	-	-
SparseQ agent (VE)	1.0000	0.9983	-	-
SparseQ edge, agent (VE)	0.9917	0.9841	0.9797	0.9765
SparseQ edge, edge (VE)	0.9843	0.9614	0.9416	0.9264
SparseQ edge, agent (anytime)	0.9906	0.9815	0.9722	0.9648
SparseQ edge, edge (anytime)	0.9856	0.9631	0.9419	0.9263

Table 4.2: Relative reward with respect to the optimal reward after $15,000$ cycles for the different methods and differently shaped graphs. Results are averaged over graphs with an average degree in $(x - 1, x]$, as indicated by the column headers.

a small performance gain when the VE algorithm is used for the agent-based update method. The agent-based update method incorporates the neighboring Q-functions, and therefore the values of the Q-functions are less distinct. As a result, the max-plus algorithm has more difficulty in finding the optimal joint action. But note that, as was shown in Fig. 4.5, the VE algorithm requires substantially more computation time for graphs of this complexity than the anytime max-plus algorithm.

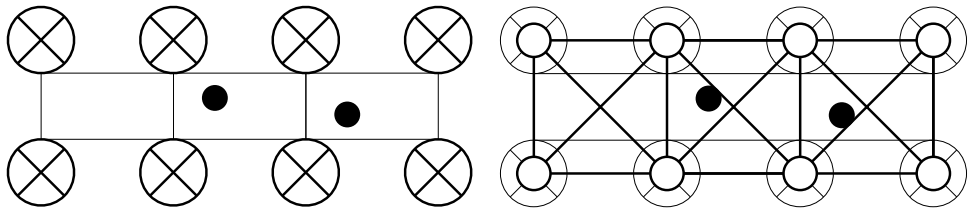
Table 4.2 gives an overview of all results and compares the value of the joint action corresponding to the learned strategy in cycle $15,000$ for the different methods. Although the results slowly decrease for the more complex reward functions, all SparseQ methods learn near-optimal policies. Furthermore, there is only a minimal difference between the methods that use the VE and the anytime max-plus algorithm to compute the joint action. For the densely connected graphs, the edge-based decompositions in combination with the max-plus algorithm are the only methods that are able to compute a good solution. The algorithms using VE fail to produce a result because of their inability to cope with the complexity of the underlying graph structure (see Section 3.4).

4.4.2 Distributed sensor network

We also perform experiments on a distributed sensor network (DSN). This problem is a sequential decision-making variant of the distributed constraint optimization problem in Ali et al. [2005]. It was part of the NIPS 2005 benchmarking workshop.[2]

The DSN problem consists of two parallel chains of an arbitrary, but equal, number of sensors. The area between the sensors is divided into cells. Each cell is surrounded by exactly four sensors and can be occupied by a target. See Fig. 4.8(a) for a config-

[2]See `http://www.cs.rutgers.edu/~mlittman/topics/nips05-mdp/` for a detailed description of the benchmarking event and `http://rlai.cs.ualberta.ca/RLBB/` for the used RL-framework.



(a) Network configuration. (b) Network with corresponding CG.

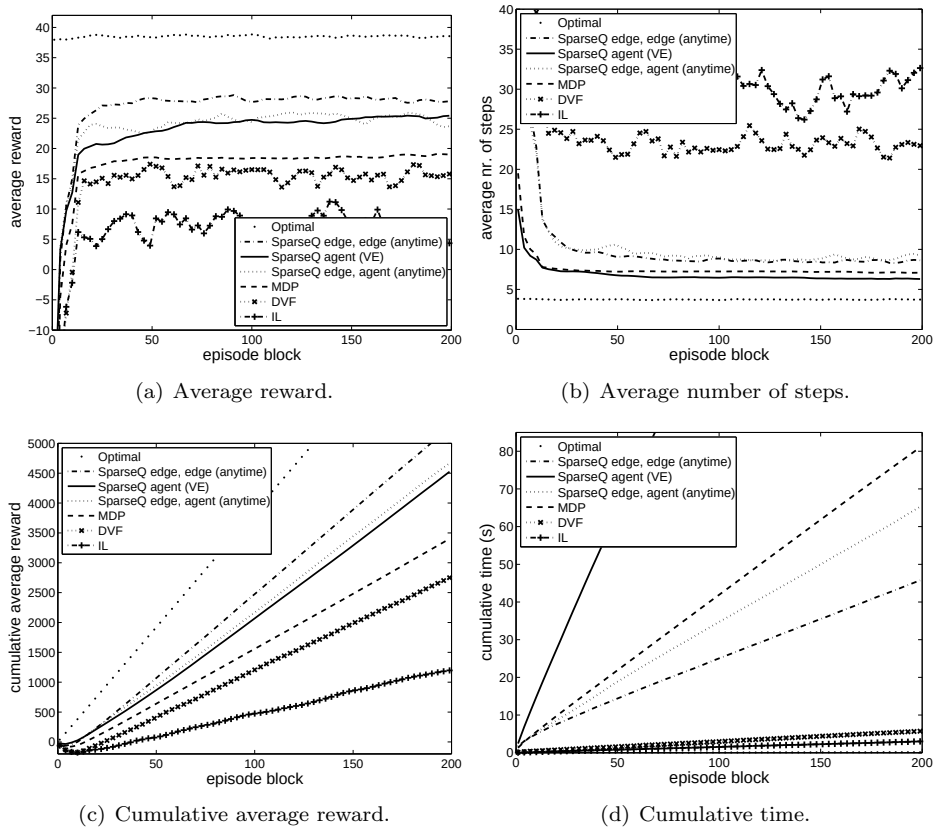
Figure 4.8: Fig. 4.8(a) shows an sensor network with eight sensors (⊗) and two targets (•). Fig. 4.8(b) shows the corresponding CG representing the agent dependencies. This graph has an average degree of 4, and an induced width of 3.

uration with eight sensors and two targets. With equal probability a target moves to the cell on its left, to the cell on its right, or remains on its current position. Actions that move a target to an illegal position, that is, an occupied cell or a cell outside the grid, are not executed.

Each sensor is able to perform three actions: focus on a target in the cell to its immediate left, to its immediate right, or don't focus at all. Every focus action has a small cost modeled as a reward of -1. When in one time step at least three of the four surrounding sensors focus on a target, it is 'hit'. Each target starts with a default energy level of three. Each time a target is hit its energy level is decreased by one. When it reaches zero the target is captured and removed, and the three sensors involved in the capture each receive a reward of $+10$. In case four sensors are involved in a capture, only the three sensors with the highest index receive the reward. An episode finishes when all targets are captured. The sensors thus have to coordinate their actions in order to hit, and eventually capture, a target.

We will perform experiments on the same configuration of the DSN problem as was used in the NIPS-05 benchmarking event. In this event, eight sensors and two targets were used, resulting in $3^8 = 6,561$ joint actions and 37 distinct states, that is, 9 states for each of the 3 configurations with 2 targets, 9 for those with one target, and 1 for those without any targets. This problem thus has a large action space compared to its state space. When acting optimally, the sensors are able to capture both targets in three steps, resulting in a cumulative reward of 42. However, in order to learn this policy based on the received rewards, the agents have to be able to cope with the delayed reward and learn how to coordinate their actions such that multiple targets are hit simultaneously.

In our experiments we generate all statistics using the benchmark implementation. However, we made two small changes to the implementation. First, because the implementation used in NIPS-05 only returns the global reward, we changed the environment to return the individual rewards in order to comply to our model specification. Second, we set the fixed seed of the random number generator to a variable seed based on the current time in order to be able to perform varying runs.



(a) Average reward.

(b) Average number of steps.

(c) Cumulative average reward.

(d) Cumulative time.

Figure 4.9: Different results on the DSN problem, averaged over 10 runs. One run consists of 200 episode blocks, each corresponding to 50 learning episodes.

We apply the different techniques described in Section 2.3.4 and Section 4.3 to the DSN problem. We do not apply the CoordRL approach, since, just as in the experiments in Section 4.4.1, it propagates back too much reward causing the individual Q-functions to blow up. However, we do apply the MDP learners approach which updates a Q-function based on the full joint action space. All applied methods learn for $10,000$ episodes which are divided into 200 episode blocks, each consisting of 50 episodes. The following statistics are computed at the end of each episode block: the average reward, that is, the undiscounted sum of rewards divided by the number of episodes in an episode block, the cumulative average reward of all previous episode blocks, and the wall-clock time. There is no distinction between learning and testing cycles, and the received reward thus includes exploration actions. The Q-learning methods all use the following parameters: $\alpha = 0.2$, $\epsilon = 0.2$, and $\gamma = 0.9$, and start with zero-valued Q-values. We assume that both the DVF and the different SparseQ

method	reward	steps	method	reward	steps
Optimal	38.454	3.752	SparseQ edge, edge (anytime)	27.692	8.795
MDP	19.061	7.071	SparseQ edge, edge (VE)	28.880	8.113
DVF	16.962	22.437	SparseQ agent (VE)	24.844	6.378
IL	6.025	31.131	SparseQ edge, agent (VE)	25.767	8.413
			SparseQ edge, agent (anytime)	23.738	8.930

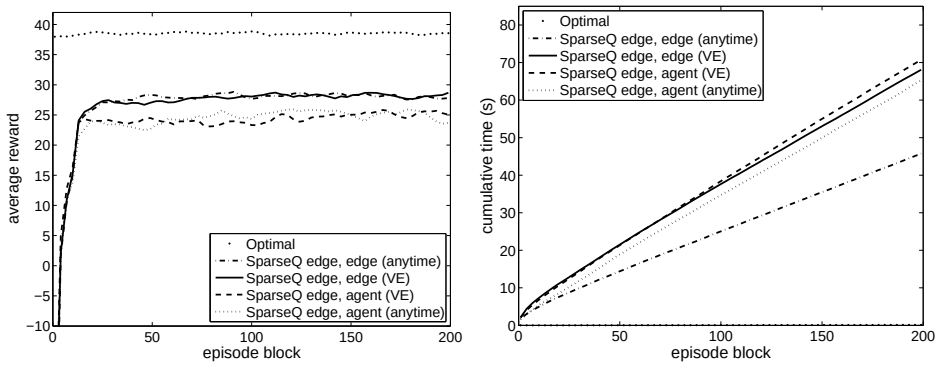
Table 4.3: Average reward and average number of steps per episode over the last 2 episode blocks (100 episodes) for the DSN problem. Results are averaged over 10 runs.

variants have access to a CG which specifies for each agent on which other agents it depends. This CG is shown in Fig. 4.8(b), and has an average degree of 4.

The results, averaged over 10 runs with different random seeds, for the different techniques are shown in Fig. 4.9. The results contain exploration actions and are therefore not completely stable. For this reason, we show the running average over the last 10 episode blocks. Fig. 4.9(a) shows the average reward for the different approaches. The optimal policy is manually implemented and, in order to have a fair comparison with the other approaches, also includes random exploration actions with probability ϵ. It results in an average reward just below 40. The MDP approach settles to an average reward around 17 after a few episodes. Although this value is low compared to the result of the optimal policy, the MDP approach, as seen in Fig. 4.9(b), does learn to capture the targets in a small number of steps. From this we conclude that the low reward is mainly a result of unnecessary focus actions performed by the agents that are not involved in the actual capture. The MDP approach thus discovers one of the many possible joint actions that results in a capture of the target and the generation of a positive reward, and then exploits this strategy. However, the found joint action is non-optimal since one or more agents do not have to focus in order to capture the target. Because of the large action space and the delayed reward, it takes the MDP approach much more than $10,000$ episodes to learn that other joint actions result in a higher reward.

Although the DVF approach performs better than IL, both methods do not converge to a stable policy and keep oscillating. This is caused by the fact that both approaches store action values based on individual actions and therefore fail to select coordinated joint actions which are needed to capture the targets.

In the different SparseQ variants each agent stores and updates local Q-values. Since these are also based on the agent's neighbors in the graph, the agents are able to learn coordinated actions. Furthermore, the explicit coordination results in much more stable policies than the IL and DVF approach. The agent-based decomposition produces a slightly lower average reward than the edge-based decompositions, but, as shown in Fig. 4.9(b), it needs less steps to capture the targets. Identical to the MDP approach, the lower reward obtained by the agent-based decomposition is a consequence of the large action space involved in each local term. As a result the



(a) Average reward.

(b) Cumulative time.

Figure 4.10: Results of the edge-based decomposition methods on the DSN problem, averaged over 10 runs. One run consists of 200 episode blocks, each corresponding to 50 learning episodes.

agents are able to quickly learn a good policy that captures the targets in a few steps, but it takes a long time to converge to a joint action that does not involve the unnecessary focus actions of some of the agents. For example, each of the four agents in the middle of the DSN coordinates with 5 other agents, and each of them thus stores a Q-function defined over $3^6 = 729$ actions per state. Because in the agent-based decomposition the full action space is decomposed into different independent local action values, it does result in a better performance than the MDP learners, both in the obtained average reward and the number of steps needed to capture the targets. With respect to the two edge-based decompositions, the edge-based update method generates a slightly higher reward, and a more stable behavior than the agent-based update method for this particular problem.

Fig. 4.9(c) shows the cumulative average reward of the different methods. Ignoring the manual policy, the edge-based update methods result in the highest cumulative average reward. This is also seen in Table 4.3 which shows the reward and the number of steps per episode averaged over the last 2 episode blocks, that is, 100 episodes, for the different methods. Since the goal of the agents is to optimize the received average reward, the SparseQ methods outperform the other learning methods. However, none of the variants converge to the optimal policy. One of the main reasons is the large number of dependencies between the agents. This requires a choice between an approach that models many of the dependencies but learns slowly because of the exploration of a large action space, for example, the agent-based SparseQ or the MDP learners, or an approach that ignores some of the dependencies but is able to learn an approximate solution quickly. The latter is the approach taken by the edge-based SparseQ variants: it models pairwise dependencies even though it requires three agents to capture a target.

Fig. 4.9(d) gives the timing results for the different methods. The IL and DVF methods are the fastest methods since they only store and update individual Q-values. The agent-based SparseQ method is by far the slowest. This method stores a Q-function based on all action combinations of an agent and its neighbors in the CG. This slows down the VE algorithm considerably since it has to maximize over a large number of possible joint action combinations in every local maximization step.

Finally, Fig. 4.10 compares the difference between using the VE or the anytime max-plus algorithm to compute the joint action for the SparseQ methods using an edge-based decomposition. Fig. 4.10(a) shows that there is no significant difference in the obtained reward for these two methods. Fig. 4.10(b) shows that the edge-based SparseQ variants that use the anytime max-plus algorithm need less computation time than those using the VE algorithm. However, the differences are not that evident as in the experiments from Section 4.4.1 because the used CG has a relative simple structure, that is, it has an induced width of 3, and VE is able to quickly find a solution when iteratively eliminating the nodes with the smallest degree.

4.5 Discussion

In this chapter, we presented different model-free reinforcement-learning variants to learn the coordinated behavior of the agents in a collaborative multiagent system. In our sparse cooperative Q-learning (SparseQ) methods, we approximate the global Q-function using a coordination graph (CG) representing the coordination requirements of the system. We analyzed two possible decompositions: one in terms of the nodes and one in terms of the edges of the graph. During learning, each local Q-function is updated based on its contribution to the maximal global action value found with either the variable elimination (VE) or max-plus algorithm. Effectively, each agent learns its part of the global solution by only coordinating with the agents on which it depends. Results on both a stateless problem with 12 agents and more than 17 million actions, and a distributed sensor network problem indicate that our SparseQ variants outperform other existing multiagent Q-learning methods. Furthermore, these methods only require that each agent is able to communicate with its neighbors in the graph and can be implemented fully distributed.

An important choice in all SparseQ methods is the used topology of the CG to represent the action value. This structure should resemble the dependencies of the problem under study. For a given CG, another choice is to use either a decomposition of the action value based on the nodes or edges of the graph. The agent-based decomposition takes all agent dependencies into account and therefore results in a better performance. However, the space complexity of this approach scales exponentially with the number of dependencies because each agent stores a Q-function for every action combination of its neighbors in the graph. This results in exploration difficulties for larger problems. Furthermore, the computational complexity of the agent-based decomposition grows exponentially with the induced width of the graph as a result of the VE method that is needed to compute the optimal joint action. Therefore,

for large problems with many dependencies an edge-based decomposition can be used which only stores action values based on pairwise dependencies. In combination with the max-plus algorithm, this approach scales only linearly in the number of dependencies of the problem. Although this method ignores the full joint action dependencies by representing them as a sum of pairwise dependencies, we have shown that this approach results in good policies and outperform other alternatives.

In all described methods, the dependencies of the agents are the same in every state. However, in many cases the dependencies differ based on the situation. A soccer player, for example, often coordinates with all players during a game, but at a particular instance it only has to coordinate with one or two other players. In our current representation, we have to construct a graph in which all agents depend on each other, and the complete graph has to be used in every state. In the next chapter we therefore consider an alternative approach in which the dependencies between the agents can change based on the current situation.

5

Context-Specific Multiagent Learning

In this chapter we focus on learning the behavior of a group of agents in multiagent sequential decision-making problems when the dependencies between the agents change based on the context. We first learn the behavior using our sparse tabular multiagent Q-learning algorithm in which, depending on the context, either all or none of the agents coordinate their actions [Kok and Vlassis, 2004b]. Then, we present our context-specific sparse cooperative Q-learning approach [Kok and Vlassis, 2004a], which models the dependencies between subsets of agents using a context-specific coordination graph that changes based on the current situation [Guestrin et al., 2002c]. Although both approaches result in large savings in the state-action representation, they assume the dependencies are known beforehand. Therefore, we also describe our utile coordination approach that learns the coordination dependencies of a system automatically based on gathered statistics during the learning phase [Kok et al., 2005a]. Finally, we perform experiments on the 'predator-prey' domain and show that we are able to learn coordinated policies in environments with changing dependencies.

5.1 Introduction

In Chapter 4 we described how an agent in a multiagent sequential decision-making problem is able to learn its behavior by only considering the actions of the agents on which it depends. These coordination dependencies are represented using a coordination graph (CG) that is the same for every state. Although this is applicable for agents in a static configuration, agents in dynamic problems often only depend on each other in a specific context. For example, two cleaning robots only have to coordinate their actions when they are cleaning the same room; in all other situations they can act independently. In the case of a fixed CG the dependencies between the agents have to be modeled in every state, resulting in dense CGs for problems in which the agents have to coordinate with many different agents. For example, each agent in a robot soccer team can potentially interact with any other agent on the soccer field and therefore a CG has to be constructed in which all agents are connected. However, in many of the possible situations the soccer agents are too far away from each to directly coordinate their actions, and the modeled dependencies are superfluous. In this chapter, we therefore present different multiagent reinforcement-learning methods in which the coordination dependencies can differ between states. We both

investigate how a group of agents can learn to jointly solve a task when the coordination requirements for the state of the system are given beforehand, and the situation in which these dependencies are not available.

We present two different approaches for the situation in which the coordination dependencies are specified beforehand. First, we introduce *sparse tabular multiagent Q-learning*, a method that creates a compact representation of the global action value by distinguishing between two different types of states. In a predefined set of *coordinated* states all agents coordinate their actions, and the global action value is modeled based on joint actions. In all other, *uncoordinated*, states, the global action value is decomposed over the different agents, and each agent learns independently. Secondly, we present *context-specific sparse cooperative Q-learning* which generalizes this approach by using a context-specific coordination graph (context-specific CG) [Guestrin et al., 2002c] to model the global action value. Such a graph is defined in terms of *value rules* which represent the value for a specific state-action combination. This has the advantage that specific action combinations between any subset of the agents can be modeled. Furthermore, in case of a factorized state representation, each dependency can be associated with a specific assignment to a subset of all state variables. This results in a compact representation of the complete state-action space. It is not possible to take advantage of a factorized state representation in our sparse tabular multiagent Q-learning approach because all states have to be mutually exclusive. Finally, to update the values of the value rules, we extend the sparse cooperative Q-learning, or SparseQ, approach, described in Chapter 4, to a context-specific CG.

We also describe a method to learn the coordination dependencies of an agent in a specific context automatically. The approach taken is to start with independent learners and maintain statistics on expected returns based on the actions of the other agents. If the statistics indicate that it is beneficial to coordinate, a dependency is added dynamically. This method is inspired by the 'utile distinction' methods from single-agent reinforcement learning that augment the state space when this distinction helps the agent predict reward [Chapman and Kaelbling, 1991; McCallum, 1997]. Hence, our method is called the *utile coordination* algorithm.

We apply our techniques on the 'predator-prey' domain, a popular multiagent problem in which a number of predator agents try to capture a prey [Benda et al., 1986; Kok and Vlassis, 2003]. Compared to other multiagent reinforcement-learning techniques, our approach achieves a good trade-off between speed and solution quality, and is able to learn the coordination dependencies of the system automatically.

The remainder of this chapter is structured as follows. In Section 5.2 we review the concept of a context-specific CG. In Section 5.3 we describe our sparse tabular multiagent Q-learning approach in which the agents either act jointly or independently for a specific situation. In Section 5.3 we extend our SparseQ approach from Chapter 4 and model the global action value using a context-specific CG, and provide experiments in the predator-prey domain when the coordination dependencies are specified beforehand. In Section 5.4, we describe our utile coordination approach to learn the dependencies of the system automatically, and test it on a small problem and the predator-prey domain. Finally, we end with some conclusions in Section 5.5.

5.2 Context-specific coordination graphs

In this section, we review the framework of a context-specific coordination graph (context-specific CG) [Guestrin et al., 2002c], which represents a dynamic, context-dependent set of coordination requirements in a multiagent system. Just as a standard CG, described in Section 3.2, a context-specific CG consists of nodes that represent agents, and edges that define the dependencies between the agents. In a standard CG each dependency corresponds to a function, often specified as a table or a matrix, which returns the value for every action combination of the involved agents. In a context-specific CG the dependencies between the agents are specified using *value rules*. These are propositional rules in the form

$$\langle\, \rho\,;\, c : v\,\rangle, \tag{5.1}$$

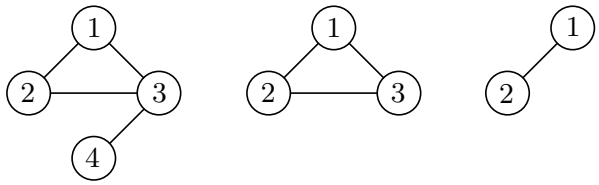
where the context $c \in C \subseteq \mathbf{S} \cup \mathcal{A}$ is an element from the set of all combinations of the state variables $\mathbf{S}$ and action variables $\mathcal{A}$, and $\rho(x) = v \in \mathbb{R}$ is the value that is obtained when $x \in X \subseteq \mathbf{S} \cup \mathcal{A}$ is consistent with c, that is, c and x have the same assignment for their shared variables $C \cap X$. When c and x are not consistent the rule is not applicable, and its value is 0. Because we use value rules to represent action values that distinguish between the state variables $\mathbf{s}$ and action variables $\mathbf{a}$ for a context $c = \mathbf{s} \cup \mathbf{a}$, we often use $\rho(\mathbf{s}, \mathbf{a})$ instead of $\rho(c)$ to the value of a rule.

A rule-based function $f : C \rightarrow \mathbb{R}$ consists of a set of m value rules $\{\rho_1, \ldots, \rho_m\}$. For a specific state-action combination x, the sum of the values of all applicable rules consistent with x defines the corresponding global value, that is,

$$f(x) = \sum_{j=1}^{m} \rho_j(x). \tag{5.2}$$

A graphical representation of a context-specific CG is constructed by connecting all agents whose actions appear in the context c of a value rule $\rho_j(c)$. For example, the left graph in Fig. 5.1 shows a CG for a 4-agent problem. The dependencies between the agents are derived from the five value rules depicted below this figure. The superscript and subscript in each value rule ρ_j^i represent respectively the agent to which this value rule belongs and the index of the value rule in its set. Agents involved in the same rule are automatically neighbors in the graph. For simplicity all actions and state variables in this example are assumed binary, that is, action a_1 corresponds to $a_1 = \mathit{true}$ and action $\overline{a}_1$ to $a_1 = \mathit{false}$.

The value rules in a context-specific CG incorporate the state information in which the rule is applicable. This is a richer representation than a standard CG in which a separate table has to be defined for every state. It is also a richer representation than the tree-structured functions defined by Boutilier et al. [2000]. In that work, a node resembles a state variable and its branches relate to the possible assignments of that variable. For a given assignment of state variables, the path from the root to one specific leaf can be followed. The value associated with this specific leaf corresponds



$\langle\, \rho_1^1\,;\, a_1 \wedge \overline{a}_3 \wedge s : 4 \,\rangle$ $\langle\, \rho_1^1\,;\, a_1 \wedge \overline{a}_3 : 4 \,\rangle$ $\langle\, \rho_2^1\,;\, a_1 \wedge \overline{a}_2 : 5 \,\rangle$
$\langle\, \rho_2^1\,;\, a_1 \wedge \overline{a}_2 \wedge s : 5 \,\rangle$ $\langle\, \rho_2^1\,;\, a_1 \wedge \overline{a}_2 : 5 \,\rangle$ $\langle\, \rho_1^2\,;\, \overline{a}_2 : 2 \,\rangle$
$\langle\, \rho_1^2\,;\, \overline{a}_2 \wedge s : 2 \,\rangle$ $\langle\, \rho_1^2\,;\, \overline{a}_2 : 2 \,\rangle$ $\langle\, \rho_2^2\,;\, a_2 : 5 \,\rangle$
$\langle\, \rho_1^3\,;\, a_3 \wedge a_2 \wedge s : 5 \,\rangle$ $\langle\, \rho_1^3\,;\, a_3 \wedge a_2 : 5 \,\rangle$ $\langle\, \rho_3^1\,;\, \overline{a_2} \wedge a_1 : 4 \,\rangle$
$\langle\, \rho_1^4\,;\, a_3 \wedge a_4 \wedge \overline{s} : 10 \,\rangle$

Figure 5.1: Initial context-specific CG (left), after conditioning on the context $s = \mathit{true}$ (center), and after elimination of agent 3 (right).

to the value for the current situation. When state variables are independent only a subset of the state variables appear in each path. This results in savings in the value-function representation because many combinations can be ignored. A restriction of this framework, however, is that all paths are mutually exclusive and exhaustive, a requirement that is not required in the case of value rules.

In order to compute an optimal joint action $\mathbf{a}^*$ that maximizes the total value in a context-specific CG for a specific situation, a variable elimination (VE) algorithm similar to the one described in Section 3.2 can be applied Guestrin [2003, ch. 9]. The main difference is that the elimination of an agent in a context-specific CG involves the manipulation of rules which is a more complex operation than the maximization in the table-based VE variant. We illustrate the rule-based VE on the example of Fig. 5.1. The first step of the algorithm is to condition on the context, for example, $s = \mathit{true}$, and remove all rules that are not applicable. In our example this results in the removal of ρ_1^4 and the updated CG depicted in the center of Fig. 5.1. Then, the agents are one by one eliminated from the graph. Let us assume that we first eliminate agent 3. This agent collects all rules in which it is involved, that is, $\langle\, \rho_1^1\,;\, a_1 \wedge \overline{a}_3 : 4 \,\rangle \langle\, \rho_1^3\,;\, a_3 \wedge a_2 : 5 \,\rangle$. Next, agent 3 computes a conditional value function, which returns the maximal value agent 3 can contribute to the system for all possible actions of agent 1 and agent 2. This function equals $\langle\, \rho_2^2\,;\, a_2 : 5 \,\rangle \langle\, \rho_3^1\,;\, \overline{a}_2 \wedge a_1 : 4 \,\rangle$. When agent 2 performs action a_2, it is always best for agent 3 to perform action a_3 since this always results in the highest possible value of 5. This result does not depend on the selected action of agent 1, and its action can therefore be omitted from the resulting rule ρ_2^2. In the case that agent 2 performs action $\overline{a}_2$, agent 3 is still able to contribute a value of 4 to the system in case agent 1 selects action a_1. This situation is depicted in rule ρ_3^1. Note that, because of the dominance of certain action choices, we only have to store two value rules to represent the four different action combinations of agent 1 and agent 2. The value-rule representation thus allows for a compact representation.

The best-response function, that is, the corresponding best action of agent 3, can be represented using a rule in which the value is replaced by the corresponding action

of agent 3. For this example, this results in $\langle\, a_2 : a_3 \,\rangle\langle\, a_1 \wedge \overline{a}_2 : \overline{a}_3 \,\rangle$. Thus, when agent 2 selects action a_2, agent 3 selects action a_3, and when agent 1 and 2 respectively select actions a_1 and $\overline{a}_2$, agent 3 selects action $\overline{a}_3$. In the remaining situation, $\overline{a}_1 \wedge \overline{a}_2$, the action of agent 3 is not specified since it has no influence on the global value; its contribution is zero in both cases. After agent 3 has computed its conditional value function, it is communicated to one of its neighbors, and the topology of the CG is updated such that all agents involved in this function are connected. Note that it is possible that a conditional value function does not contain all neighbors of an eliminated agent since the actions of one, or more, of the neighbors are dominated by the actions of the other agents. This results in a speedup for the rule-based VE because the number of generated rules is never larger and in many cases exponentially smaller than the table-based VE. The CG after the elimination of agent 3 is depicted in the right figure of Fig. 5.1. The algorithm continues with the elimination of the next agent, for example, agent 2. First it computes its conditional value function $\langle\, \rho_4^1 ;\, a_1 : 11 \,\rangle\langle\, \rho_5^1 ;\, \overline{a}_1 : 5 \,\rangle$, corresponding to respectively the best-response action $\overline{a}_2$ and a_2. Then, it communicates this function to agent 1, and is finally eliminated from the graph. Agent 1 is the last agent left and fixes its action to a_1 since this corresponds to the maximal obtainable value for the system. Now a second pass in the reverse order is performed in which each agent computes its final action based on its best-response function and the fixed actions of its neighbors, and communicates this action to its neighbors. For our example, this results in the optimal joint action $\{a_1, \overline{a}_2, \overline{a}_3\}$ with a corresponding global value of 11. The action of agent 4 can be chosen arbitrarily since it has no influence on the global value.

The set of value rules results in large savings in the representation of the state-action space since only relevant state and action combinations are defined. However, a computational disadvantage of this representation is that it involves the management of rules. This results in a complexer VE algorithm and requires a specialized method to determine the applicable value rules for a specific state. Therefore, only in problems with a large amount of context-specific structure the smaller number of rules in the rule-based VE outweighs the computational advantages of the table-based VE.

Guestrin et al. [2002c] use a context-specific CG to solve multiagent sequential decision-making problems when the model of the system is known and represented using a dynamic decision network (DDN) as described in Section 2.3.2. They approximate the global value function as a linear combination of weighted basis functions. These functions are represented by predefined value rules. Given the known model dynamics, the weights are learned using a centralized linear programming algorithm. This approach scales to large state spaces because the structure in the rules allows for an efficient maximization. After the rule-based value function is learned, it can then be used online to determine the maximizing joint action for a certain situation.

In this thesis, we assume that the model dynamics are not available, and therefore represent the global action value directly using the value rules in a context-specific CG. The values of the rules are updated based on experienced state transitions, using similar reinforcement-learning techniques as applied in Chapter 4. This results in a family of model-free learning algorithms that is completely distributed.

5.3 Context-specific multiagent Q-learning

In this section, we present two multiagent reinforcement-learning techniques to learn the coordinated behavior of a group of agents that allow the dependencies between the agents to change based on the current context. First, we examine sparse tabular multiagent Q-learning, an approach in which, depending on the context, either all or none of the agents coordinate their actions. Then we generalize this approach by modeling the global action value using a context-specific CG. This enables us to define, by means of value rules, coordination dependencies between subsets of agents that differ based on the specific situation. In order to update the value rules we extend our SparseQ method described in Chapter 4. Both approaches are applied to an instance of the predator-prey domain in order to learn the behavior of the predators.

5.3.1 Sparse tabular multiagent Q-learning

Sparse tabular multiagent Q-learning is a reinforcement-learning technique which models context-specific coordination requirements [Kok and Vlassis, 2004b]. The idea is to label each state of the system either as a coordinated or an uncoordinated state. In the coordinated states the agents learn based on joint actions, while in the uncoordinated states they learn independently. More formally, the state space S is divided into two mutually exclusive sets S^+ and S^- for which holds, $S = S^+ \cup S^-$ and $S^+ \cap S^- = \emptyset$. For the states $s \in S^+$ the agents have to coordinate their actions, and therefore they learn joint action values using the MDP learners approach. In the uncoordinated states $s \in S^-$ the agents learn independently and apply the independent learners (IL) approach. Both techniques are described in Section 2.3.4. Since in practical problems the agents typically need to coordinate their actions only in few states, this framework allows for a sparse representation of the complete action space. Note that this approach is not able to take advantage of a factorized state representation, because each state has to labeled based on a specific assignment of all state variables. In the remainder of this section, we therefore assume the state is specified in terms of a single state variable.

The representation of the global action value $Q(s, \mathbf{a})$ differs depending on the current context. For the coordinated states $s \in S^+$ we directly model the global Q-function $Q(s, \mathbf{a})$ based on joint actions. For the uncoordinated states $s \in S^-$, however, each agent i maintains a Q-function $Q_i(s, a_i)$ based on individual actions, and the global Q-function is the sum of all individual Q-functions, that is,

$$Q(s, \mathbf{a}) = \sum_{i=1}^{n} Q_i(s, a_i). \tag{5.3}$$

In order to update a Q-function after a state transition, received in the form of a $(s, \mathbf{a}, s', r)$ sample, values from differently sized Q-functions have to be combined. There are four different situations that must be taken into account. When moving between two coordinated or between two uncoordinated states, we respectively apply

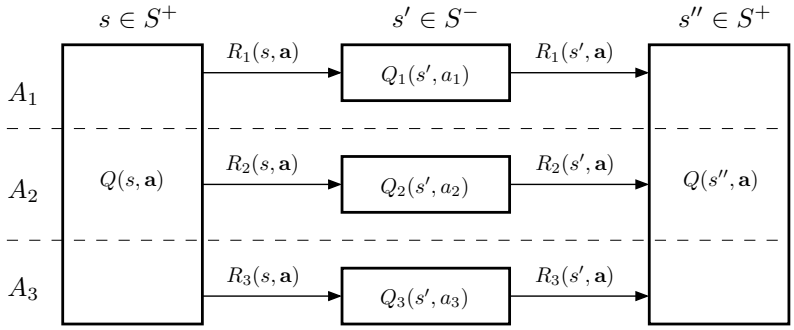


Figure 5.2: Graphical representation of the Q-tables in the case of three agents A_1, A_2, and A_3. State s and s'' are coordinated states, while state s' is an uncoordinated state.

the MDP learners and IL approach directly. In the case that the n agents move from a coordinated state $s \in S^+$ to an uncoordinated state $s' \in S^-$ we propagate back the individual Q-values to the joint Q-value using

$$Q(s, \mathbf{a}) := (1 - \alpha)Q(s, \mathbf{a}) + \alpha \sum_{i=1}^{n} \Big[R_i(s, \mathbf{a}) + \gamma \max_{a'_i} Q_i(s', a'_i) \Big]. \tag{5.4}$$

We thus add all individual contributions of the n agents to determine the global action value for the next state s'. When moving from an uncoordinated state $s' \in S^-$ to a coordinated state $s'' \in S^+$ we propagate back the joint Q-value in state s'' to the different individual Q-values using

$$Q_i(s', a_i) := (1 - \alpha)Q_i(s', a_i) + \alpha \Big[R_i(s', \mathbf{a}) + \gamma \frac{1}{n} \max_{\mathbf{a}'} Q(s'', \mathbf{a}') \Big]. \tag{5.5}$$

Each agent is thus rewarded with the same fraction of the estimated future discounted reward from the resulting coordinated state. This implies that we assume each agent contributes equally to the coordination.

Fig. 5.2 shows a graphical representation of the transition between three states for a 3-agent problem. In state $s \in S^+$ the agents have to coordinate their actions and use the shared Q-function to select a joint action. After taking a joint action and observing a transition to the uncoordinated state $s' \in S^-$, the global Q-function is updated using (5.4). In $s' \in S^-$ each agent i chooses its action independently, and after moving to state $s'' \in S^+$ updates its individual Q-function using (5.5).

In terms of implementation, all Q-functions can be represented using tables since for every state the number of actions is fixed. For the coordinated states, however, all agents should either have access to the shared Q-function, for example, by communication with a centralized controller, or every individual agent should update a local copy identically. In the latter case the agents have to rely on common knowledge assumptions to ensure that every agent observes the actions and rewards of all

other agents. Both are strong assumptions. Furthermore, in order to synchronize exploration in the coordinated states, all agents should either select a joint action communicated by the centralized controller, or based on a random number generator with a specific seed that are common knowledge to all agents.

In this approach, either all or none of the agents have to coordinate their actions in a specific state. This becomes problematic for large groups of agents since the size of the Q-function for the coordinated states grows exponentially with the number of agents. In general, however, only subsets of agents depend on each other in a specific context. Next, we will therefore discuss context-specific SparseQ, a generalization of sparse tabular multiagent Q-learning, which represents the coordination requirements using a context-specific CG. First, this enables us to specify much more fine-grained coordination dependencies between the agents because the dependencies are specified over subsets of the agents and, in a factorized state representation, can be defined in terms of subsets of the state variables. Secondly, the method becomes completely distributed since each agent only depends on itself and its neighbors in the graph.

5.3.2 Context-specific sparse cooperative Q-learning

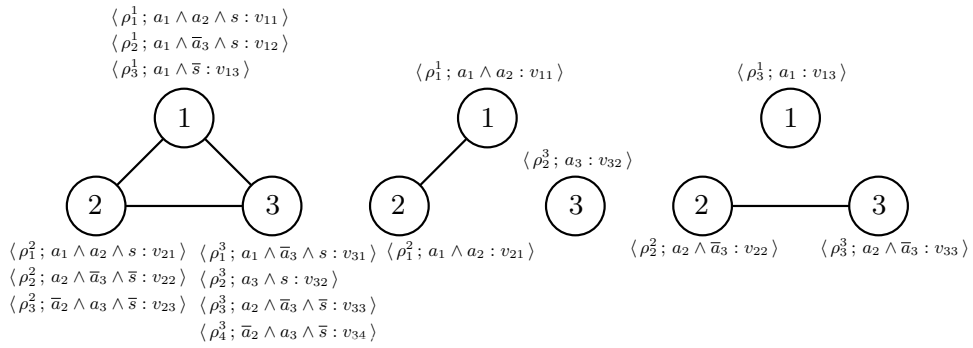
Sparse tabular multiagent Q-learning defines a state either as a coordinated state in which all agents coordinate their actions, or as an uncoordinated state in which all agents act independently. However, in many situations only some of the agents have to coordinate their actions. In this section we describe context-specific sparse cooperative Q-learning, a reinforcement-learning technique which enables subsets of agents to learn how to coordinate based on a predefined coordination structure that can differ between states [Kok and Vlassis, 2004a]. The described methods extend the SparseQ methods described in Chapter 4 to context-specific CGs. Again, we investigate both an agent-based and an edge-based decomposition of the used CG.

Agent-based decomposition

In the agent-based decomposition, each agent i is associated with a local value function $Q_i(\mathbf{s}_i, \mathbf{a}_i)$, in which $\mathbf{s}_i$ and $\mathbf{a}_i$ respectively represent all the state and action variables on which agent i depends. We assume that these dependencies, but not their actual values, are specified beforehand. Instead of representing this function explicitly for every possible action combination of the involved agents as in Chapter 4, we represent this function with the values rules from a context-specific CG, that is,

$$Q_i(\mathbf{s}_i, \mathbf{a}_i) = \sum_{j \in n_i(\mathbf{s}_i, \mathbf{a}_i)} \rho_j^i(\mathbf{s}_i, \mathbf{a}_i), \tag{5.6}$$

where $n_i(\mathbf{s}_i, \mathbf{a}_i)$ are the indices of the value rules of agent i consistent with the state-action pair $(\mathbf{s}_i, \mathbf{a}_i)$. Note that for a specific state $\mathbf{s}_i$, a dependency between agent i and an agent involved in $\mathbf{a}_i$ might not be available because the value rules in which these two agents are involved are not applicable for $\mathbf{s}_i$. For this state the edge is then removed from the graph.



(a) Context-specific CG. (b) CG in state s. (c) CG in state $\overline{s}$.

Figure 5.3: Example agent-based context-specific CG with its value rules: (a) context-specific CG with all dependencies, (b) CG conditioned on state s and action $\mathbf{a} = \{a_1, a_2, a_3\}$, (c) CG conditioned on state $\overline{s}$ and action $\mathbf{a}^* = \{a_1, a_2, \overline{a}_3\}$.

Fig. 5.3(a) depicts an example context-specific CG with three agents, and a set of value rules for each of the different agents. Each value rule consists of a single, binary, state variable s and a specific assignment to a subset of the, binary, action variables. Each agent individually models its dependencies with its neighboring agent, and therefore the system contains multiple identical value rules. For example, the dependency between agent 1 and agent 2 is modeled by the rules ρ_2^1 and ρ_1^3 that both contain the same variable assignment. Note that, after learning, the values of the two rules are not identical because they are differently updated. This equals the approach taken in the agent-based SparseQ method in which each agent stores a Q-value based on its actions and the actions of its neighbors in the graph. Later, in the edge-based decomposition, we also investigate the consequences of storing only a single rule for each dependency. Although then fewer rules are need, it complicates the update steps since the value of each value rule has to be distributed over all involved agents.

In order to update the value rules in the agent-based decomposition after receiving a $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ sample, we take a similar approach as in the agent-based SparseQ method in which each local Q-function is updated according to (4.4), that is,

$$Q_i(\mathbf{s}_i, \mathbf{a}_i) := Q_i(\mathbf{s}_i, \mathbf{a}_i) + \alpha[R_i(\mathbf{s}, \mathbf{a}) + \gamma Q_i(\mathbf{s}'_i, \mathbf{a}^*_i) - Q_i(\mathbf{s}_i, \mathbf{a}_i)]. \tag{5.7}$$

Each agent i thus updates its local action value based on its own action values and the individually received reward R_i. The maximizing joint action $\mathbf{a}^* = \arg\max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}')$, which is used to determine the local contribution $Q_i(\mathbf{s}'_i, \mathbf{a}^*_i)$ of agent i to the total payoff, is computed using the VE algorithm. Since each local Q-function is represented using value rules, we derive the update rule for a single value rule by first substituting

(5.6) in (5.7). This results in

$$\sum_{j \in n_i(\mathbf{s}_i, \mathbf{a}_i)} \rho_j^i(\mathbf{s}_i, \mathbf{a}_i) := \sum_{j \in n_i(\mathbf{s}_i, \mathbf{a}_i)} \rho_j^i(\mathbf{s}_i, \mathbf{a}_i) + \alpha \sum_{j \in n_i(\mathbf{s}_i, \mathbf{a}_i)} \frac{R_i(\mathbf{s}, \mathbf{a}) + \gamma Q_i(\mathbf{s}_i', \mathbf{a}_i^*) - Q_i(\mathbf{s}_i, \mathbf{a}_i)}{|n_i(\mathbf{s}_i, \mathbf{a}_i)|}. \quad (5.8)$$

The rightmost summation decomposes the temporal-difference error into $|n_i(\mathbf{s}_i, \mathbf{a}_i)|$ equal parts, and thus does not use j explicitly. Because all summations are identical, we can remove the sums, and write a local update of an applicable value rule ρ_j^i as

$$\rho_j^i(\mathbf{s}_i, \mathbf{a}_i) := \rho_j^i(\mathbf{s}_i, \mathbf{a}_i) + \alpha \frac{R_i(\mathbf{s}, \mathbf{a}) + \gamma Q_i(\mathbf{s}_i', \mathbf{a}_i^*) - Q_i(\mathbf{s}_i, \mathbf{a}_i)}{|n_i(\mathbf{s}_i, \mathbf{a}_i)|}, \quad (5.9)$$

which corresponds to a proportional division of the temporal-difference error of agent i over the different value rules consistent with the state-action combination $(\mathbf{s}_i, \mathbf{a}_i)$.

We now show an update of the example depicted in Fig. 5.3. Assume that the joint action $\mathbf{a} = \{a_1, a_2, a_3\}$ is performed in state s and $\mathbf{a}^* = \{a_1, a_2, \overline{a}_3\}$ is the optimal joint action found with the VE algorithm in the next state $\overline{s}$. After conditioning on the context, that is, on both the state and action variables, the rules ρ_1^1, ρ_1^2, and ρ_2^3 apply in state s, whereas the rules ρ_3^1, ρ_2^2, and ρ_3^3 apply in state $\overline{s}$. This is graphically depicted in respectively Fig. 5.3(b) and Fig. 5.3(c). Next, we use (5.9) to update the value rules in state s as follows:

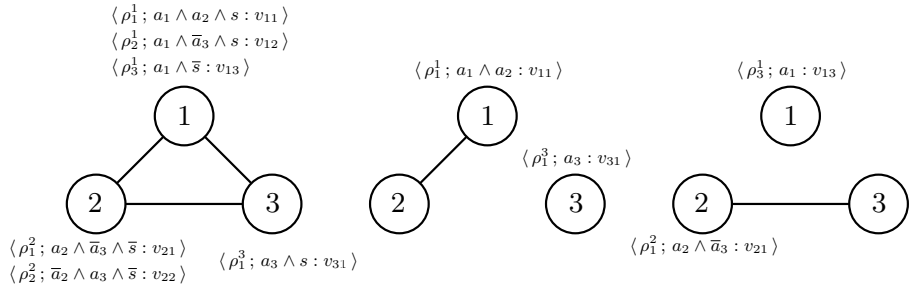
$$\begin{aligned}
\rho_1^1(\mathbf{s}_1, \mathbf{a}_1) &= v_{11} + \alpha[R_1(s, \mathbf{a}) + \gamma v_{13} - v_{11}] \\
\rho_1^2(\mathbf{s}_2, \mathbf{a}_2) &= v_{21} + \alpha[R_2(s, \mathbf{a}) + \gamma v_{22} - v_{21}] \\
\rho_2^3(\mathbf{s}_3, \mathbf{a}_3) &= v_{32} + \alpha[R_3(s, \mathbf{a}) + \gamma v_{33} - v_{32}]
\end{aligned}$$

Note that each agent updates its value rules based on its own stored rules, and only coordinates with its neighbors in order to compute the optimal joint action $\mathbf{a}^*$. All other computations are local.

Edge-based decomposition

In the edge-based decomposition the value rules are associated with the edges of a context-specific CG. Because a value-rule representation can be related to more than two agents, we treat the general case and place no restrictions on the number of involved agents in a rule. Furthermore, to be consistent with our previous notation, we assume each value rule p_k^j is stored by agent j, and has index k in its set of rules.

Similar to (4.5), it is convenient to compute a local Q-function for each agent. This function is defined as the sum of the values of the functions in which the agent is involved. Again, we assume that the value of a value rule ρ_k^j is evenly distributed



(a) Context-specific CG. (b) CG in state s. (c) CG in state $\overline{s}$.

Figure 5.4: Example edge-based context-specific CG with its value rules: (a) context-specific CG with all dependencies, (b) CG conditioned on state s and action $\mathbf{a} = \{a_1, a_2, a_3\}$, (c) CG conditioned on state $\overline{s}$ and action $\mathbf{a}^* = \{a_1, a_2, \overline{a}_3\}$.

over the involved agents $\texttt{Agents}[\rho_k^j]$. A local Q-function of an agent i is then defined as

$$Q_i(\mathbf{s}_i, \mathbf{a}_i) = \sum_{(j,k) \in n(\mathbf{s}_i, \mathbf{a}_i, i)} \frac{\rho_k^j(\mathbf{s}_i, \mathbf{a}_i)}{|\texttt{Agents}[\rho_k^j]|}, \tag{5.10}$$

in which $(j,k) \in n(\mathbf{s}_i, \mathbf{a}_i, i)$ returns the indices of the value rules that involve agent i and are consistent with the current state-action combination $(\mathbf{s}_i, \mathbf{a}_i)$. Each index (j,k) consists of both the agent j which stores the applicable rule, and the corresponding index k of this rule in the agent's set. Note that $\mathbf{s}_i$ and $\mathbf{a}_i$ consist of respectively all state and action variables that occur in the value rules in which agent i is involved, and can thus also contain variables from the value rules of the neighbors of agent i. In practice, however, an agent does not have to observe the state and action variables of its neighbors since each agent individually conditions on the context, and then only communicates the relevant value rules for the current situation when an agent needs to compute its local Q-function. This is possible because we assume no uncertainty in the observations. Fig. 5.4(a) shows the same problem as in Fig. 5.3 when each dependency is modeled using a single value rule. Note that the number of required value rules is greatly reduced compared to Fig. 5.3.

In Section 4.3.2, we discussed both an agent-based and an edge-based update method for the edge-based decomposition for a standard CG. The edge-based update method, in which the function related to an edge is updated based on the maximizing contribution of the same edge in the next state, is not applicable to a context-specific CG because its topology changes between states. For example, it is not possible to propagate back the value from a dependency, represented as a value rule, that was not applicable in the previous state. In order to derive the update rule for the agent-based

update method, we therefore first decompose (5.7) using (5.10) into

$$\sum_{(j,k)\in n(\mathbf{s}_i,\mathbf{a}_i,i)} \frac{\rho_k^j(\mathbf{s}_i,\mathbf{a}_i)}{|\texttt{Agents}[\rho_k^j]|} = \sum_{(j,k)\in n(\mathbf{s}_i,\mathbf{a}_i,i)} \frac{\rho_k^j(\mathbf{s}_i,\mathbf{a}_i)}{|\texttt{Agents}[\rho_k^j]|} + \alpha[R_i(\mathbf{s},\mathbf{a}) + \gamma Q_i(\mathbf{s}'_i,\mathbf{a}^*_i) - Q_i(\mathbf{s}_i,\mathbf{a}_i)]. \quad (5.11)$$

Then, similar to (4.10), we first rewrite the temporal-difference error to

$$R_i(\mathbf{s},\mathbf{a}) + \gamma Q_i(\mathbf{s}'_i,\mathbf{a}^*_i) - Q_i(\mathbf{s}_i,\mathbf{a}_i) = \sum_{(j,k)\in n(\mathbf{s}_i,\mathbf{a}_i,i)} \frac{R_i(\mathbf{s},\mathbf{a}) + \gamma Q_i(\mathbf{s}'_i,\mathbf{a}^*_i) - Q_i(\mathbf{s}_i,\mathbf{a}_i)}{|n(\mathbf{s}_i,\mathbf{a}_i,i)|}. \quad (5.12)$$

Note that, again, the rightmost summation only ensures that all summations are identical and does not use the indices (j,k) explicitly. Then, we substitute (5.12) into (5.11), and remove the sums. This results in

$$\frac{\rho_k^j(\mathbf{s}_i,\mathbf{a}_i)}{|\texttt{Agents}[\rho_k^j]|} = \frac{\rho_k^j(\mathbf{s}_i,\mathbf{a}_i)}{|\texttt{Agents}[\rho_k^j]|} + \alpha\frac{R_i(\mathbf{s},\mathbf{a}) + \gamma Q_i(\mathbf{s}'_i,\mathbf{a}^*_i) - Q_i(\mathbf{s}_i,\mathbf{a}_i)}{|n(\mathbf{s}_i,\mathbf{a}_i,i)|}. \quad (5.13)$$

Each agent can derive a similar update for all the value rules in which it is involved. Because multiple agents update the same value rule, we can combine the different updates and write the update of a single rule as

$$\rho_k^j(\mathbf{s}_i,\mathbf{a}_i) = \rho_k^j(\mathbf{s}_i,\mathbf{a}_i) + \alpha \sum_{l\in\texttt{Agents}[\rho_k^j]} \frac{R_l(\mathbf{s},\mathbf{a}) + \gamma Q_l(\mathbf{s}'_l,\mathbf{a}^*_l) - Q_l(\mathbf{s}_l,\mathbf{a}_l)}{|n(\mathbf{s}_l,\mathbf{a}_l,l)|}. \quad (5.14)$$

Each value rule is thus updated based on its old value and the temporal-difference error of all involved agents. We assume each agent contributes equally to all value rules in which it is involved, and therefore the temporal-difference error is divided by the number of rules in which its action is included.

We now show an update of the example depicted in Fig. 5.4. Again, we assume that the joint action $\mathbf{a} = \{a_1, a_2, a_3\}$ is performed in state s and $\mathbf{a}^* = \{a_1, a_2, \overline{a}_3\}$ is the optimal joint action found with the VE algorithm in the next state $\overline{s}$. After conditioning on the context, that is, both on the state and the action variables, the rules ρ_1^1 and ρ_1^3 apply in state s, whereas the rules ρ_3^1 and ρ_1^2 apply in state $\overline{s}$. This is graphically depicted in respectively Fig. 5.4(b) and Fig. 5.4(c). Next, we apply (5.14) and update the value rules in state s as follows:

$$\rho_1^1(\mathbf{s}_1,\mathbf{a}_1) = v_{11} + \alpha[R_1(s,\mathbf{a}) + R_2(s,\mathbf{a}) + \gamma(v_{13} + \frac{v_{21}}{2}) - v_{11}]$$
$$\rho_1^3(\mathbf{s}_3,\mathbf{a}_3) = v_{31} + \alpha[R_3(s,\mathbf{a}) + \gamma\frac{v_{21}}{2} - v_{31}]$$

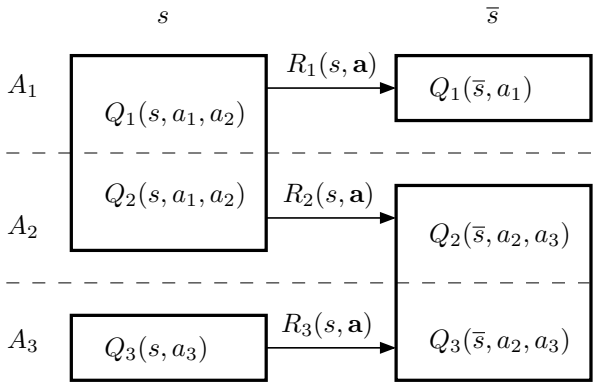


Figure 5.5: Example representation of the local Q-functions of the three agents for the transition from state s to state $\overline{s}$ in the example problem from Fig. 5.4.

Note that in order to update ρ_1^1 we have used the discounted Q-values $Q_1(\overline{s}, \mathbf{a}^*) = v_{13}/1$ and $Q_2(\overline{s}, \mathbf{a}^*) = v_{21}/2$. Furthermore, the component Q_2 in state $\overline{s}$ is based on a coordinated action of agent 2 with agent 3 represented by rule ρ_1^2, whereas in state s agent 2 has to coordinate with agent 1 (rule ρ_1^1). Fig. 5.5 graphically depicts the structure of the Q-functions for state s and $\overline{s}$.

5.3.3 Experiments

Next, we apply the described methods to the pursuit, or predator-prey, domain in which it is the goal of the predators to capture a prey in a discrete grid-like world.

Pursuit problem

The pursuit problem is a popular multiagent domain in which predators have to capture one, or multiple, prey in a discrete grid environment [Benda et al., 1986; Kok and Vlassis, 2003]. Several instances of the pursuit problem exist which differ based on the applied capture method and the assumptions about the environment. A prey can, for example, be captured when one or more predators move to the same cell [Tan, 1993], or when it is surrounded by four predators [Stone and Veloso, 2000]. Assumptions regarding the environment specify, for example, which part of the environment the predators are able to observe, or whether the predators can communicate.

We perform experiments on an instance of the pursuit problem in which two predators have to explicitly coordinate their actions in order to capture a single prey in a 10×10 toroidal grid. A sample configuration is shown in Fig. 5.6(a). Time is divided into episodes, which are again divided into discrete time steps. At the beginning of an episode, all predators and prey are initialized at random positions. In each time step, first all predators simultaneously execute one of the five possible movement commands: move north, south, east, west, or stand still. Note that because

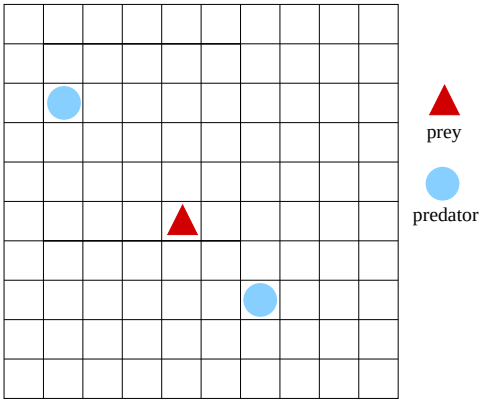
the grid is toroidal, moving west from a cell in the leftmost column brings the agent to the rightmost column, and moving north from the upper row moves the agent to the bottom row. When two predators end up in the same cell, they are penalized and moved to a random, empty, position on the grid. After the predators have executed their actions, the prey moves according to a stationary randomized policy: it remains on its current position with a probability of 0.2, and otherwise moves to one of its free adjacent cells with uniform probability. Although the actions of the predators always have the same deterministic outcome, the random behavior of the prey results in a stochastic environment. A prey is captured, and the episode ends, when both predators are located in cells adjacent to the prey and one of the two predators moves to the location of the prey, while the other predator remains, for support, on its current position. Fig. 5.6(b) shows a possible capture position. The prey is captured when either the predator north of the prey, or the prey east of the prey will move to the prey position and the other predator will remain on its current position. When a predator moves to the prey without correct support, that is, the other predator is not located in an adjacent cell or does not remain on its current position, it is penalized and moved to a random, empty, position on the field. Because both collisions and movement to the prey position without support result in a penalty, the agents are forced to actively coordinate their actions in many situations. This differs from other approaches which do not depend on the specific action combination performed by the agents, for example, in the experiments performed by Tan [1993] the predators are allowed to share the same cell and the prey is captured when either one, or multiple, predators are located in the same cell as the prey.

The reward model for our problem is as follows: a capture results in a total reward of 75; each predator i thus receives a reward $R_i = 37.5$ when it helps to capture the prey. Furthermore, each predator receives a reward of -25.0 when it moves to the prey without support, a reward of -10.0 when it collides with another predator, and a reward of -0.5 in all other situations to stimulate a quick capture of the prey.

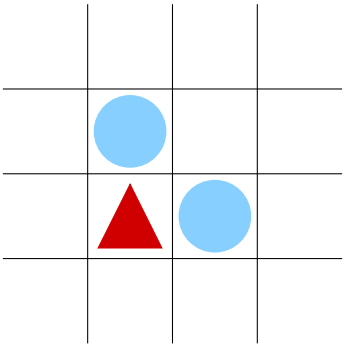
The state space can be represented by the relative position of the two predators to the prey. Because the world is symmetrical, we can assume the prey is always positioned at the origin and ignore its position. The complete state-action space then consists of all combinations of the two predator positions relative to the prey and all action combinations of the two predators. In total this yields $242,550$, that is, $99 \cdot 98 \cdot 5^2$, state-action pairs. However, the predators only have to actively coordinate their actions when they are close to each other. For our context-specific learning methods, we therefore make a distinction between coordinated and uncoordinated states. We define a state as coordinated when either

- the Manhattan distance between the predators is smaller or equal than two cells
- both predators are within a distance of two cells to the prey.

We apply our sparse tabular approach and our context-specific SparseQ method, with both an agent-based and edge-based decomposition, to learn the behavior of the predators in the described problem. Furthermore, we also apply the IL, MDP and



(a) Complete 10×10 toroidal grid.



(b) Possible capture position.

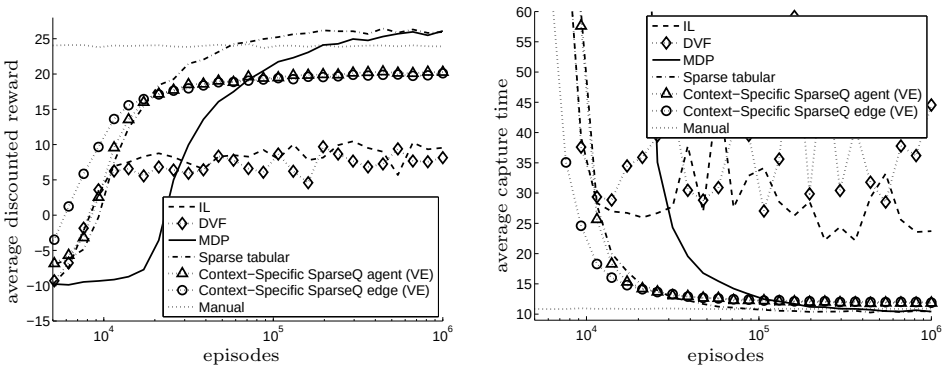
Figure 5.6: Example pursuit problem. (a) 10×10 toroidal grid with two predators and one prey. (b) Possible capture position. The prey is captured when one of the two agents moves to the prey position while the other remains on its position.

DVF approach as described in Section 4.4.1. The IL approach stores a Q-function based on the full state information and one of the five possible actions. This corresponds to $48,510$, that is, $99 \cdot 98 \cdot 5$, different state-action pairs for each agent. The DVF approach stores similarly sized Q-functions, but each update also incorporates the Q-functions of the other agent. In the MDP learners approach we model the system as a complete MDP with the joint action represented as a single action. In the sparse tabular approach, we store joint actions for the coordinated states, and independent actions for the uncoordinated states. With the given definition of a coordinated state, we have $1,248$ coordinated states, and $8,454$ uncoordinated states. For the two context-specific SparseQ methods, we also decompose the state space by ignoring the local state of the other predator for the uncoordinated states. This results in a set of 495, that is $99 \cdot 5$, individual value rules per agent for the uncoordinated states. An example rule look as

$$\langle \rho_1^1 \quad ; \quad \text{predator}_1(-3,3) \quad \wedge \quad \text{coord} = \mathit{false} \quad \wedge \quad a_1 = \text{move\_north} : 37.5 \rangle,$$

which represents the situation in which predator 1 is at relative position $(-3,3)$ to the prey and performs a movement command to the cell to its north. The state variable coord $= \mathit{false}$ indicates that the predator does not has to coordinate its action in this state and ensures that this rule is not applicable in the situation when the state is coordinated. The state variable 'coord' is automatically derived from the two predator positions. Because we do not consider individual rules for which coord $= \mathit{true}$, this does not influence the size of the representation of the state-action space.

For the $1,248$ coordinated states we add value rules that involve the state and



(a) Average reward.

(b) Average capture time.

Figure 5.7: Running average of the reward and capture times during the first $1,000,000$ episodes. The episodes are shown using a logarithmic scale. Results are averaged over 10 runs.

action variables of both predators. For example, the specific coordination rule which corresponds to a capture of the prey in the situation of Fig. 5.6(b) is defined as

$$\langle \rho_2^1 \quad ; \quad \text{predator}_1(0,1) \quad \wedge \quad \text{predator}_2(1,0) \quad \wedge \\ a_1 = \text{move\_none} \quad \wedge \quad a_2 = \text{move\_west} : 75 \rangle.$$

We ignore the coord variable since this rule is always applicable in the specific configuration of the two predators. In the agent-based decomposition, both agents store a copy of each coordinated value rule, resulting in the generation of $31,200$ coordinated value rules per agent. Combined with the rules for the uncoordinated states both agents store $63,390$ value rules in total. In the edge-based decomposition only one of the two agents stores coordinated value rules, and in total $32,190$ value rules are defined. The last column of Table 5.1 gives an overview of the number of action values for each of of the applied methods.

In all approaches, we use an ϵ-greedy exploration step $\epsilon = 0.2$, a learning rate $\alpha = 0.3$, and a discount factor $\gamma = 0.9$. Furthermore, all Q-values are initialized with the maximal reward, that is, a value of 75 for the coordinated states, and 37.5 for the uncoordinated states. This ensures that the predators explore all possible action combinations sufficiently because non-tried actions will have a high action value.

Fig. 5.7 shows the running average of both the reward and the capture time for the learned policy during the first $1,000,000$ episodes. The results are averaged over 10 runs. During each run the current learned policy is tested after each interval of 100 learning episodes, without exploration actions, on a randomly generated test set of 100 starting configurations. This test set is fixed beforehand and the same for all methods. Each of the $10,000$ tests results in an average capture time and an average obtained cumulative discounted reward, which are both shown in Fig. 5.7. To

method	**reward**	**avg. time**	**#Q-values**
IL	9.579	24.273	97,020
DVF	7.977	31.448	97,020
MDP	25.996	10.448	242,550
Sparse tabular	26.001	10.436	115,740
Context-specific sparseQ agent	20.353	11.798	63,390
Context-specific sparseQ edge	20.047	11.921	32,190
Manual	23.959	10.893	-

Table 5.1: Reward and capture time, averaged over the last 10 test runs, after learning for $1,000,000$ episodes. The number of state-action pairs is also given.

make the final results clearer, the averages are visualized by computing the running average over the last 20 tests and showing the x-axis in logarithmic scale.

The IL approach and the DVF approach perform worst. Both methods do not converge to a single policy, but keep oscillating. Because they store Q-functions based on the individual actions of the agents, they update the same Q-value both after successful and unsuccessful coordination with the other agent. For example, when both predators are located next to the prey and one predator moves to the prey position, this predator is not able to distinguish between the situation in which the other predator remains on its current position or performs one of its other actions. As a result the same Q-value is updated in both cases, although a positive reward is received in the first situation and a large negative reward in the second situation.

These coordination dependencies are explicitly taken into account for the other approaches. For the MDP learners, they are modeled in every state which results in slow convergence because all state-action pairs have to be explored. Eventually, however, it results in an optimal policy. The final result is slightly better than our manual implementation. For this implementation, we first map each predator to a *different* adjacent cell of the prey such that the sum of the Manhattan distances of the current positions of the predators to their assigned cell is minimized. Then, each agent performs an action that minimizes the distance to this position while using social conventions to avoid collisions: when a selected action combination will result in a collision, one of the two agents remains on its current position. When both predators stand next to the prey, social conventions based on the relative positioning are used to decide which of the two predators moves to the prey position.

The sparse tabular approach only considers joint actions for the coordinated states and results in a similar policy, in much fewer learning episodes, as the MDP learners approach. However, this method learns based on the full state information and therefore does not scale to problems with a very large state space. On the other hand, the two context-specific SparseQ methods learn in a sparse representation of the state space because they do not incorporate the state information of the other

predator in the uncoordinated states. This results in a fast increase of the learning curve compared to the other methods in the first episodes. The final difference with respect to the sparse tabular approach, which only decomposes the action space, is minimal. This indicates that especially the large action size, which requires a large number of exploration actions, results in the faster convergence of the sparse tabular Q-learning approach with respect to the MDP learners approach. As we see in Fig. 5.7, the context-specific SparseQ approaches converge to a lower average reward and a higher capture time than that of the sparse tabular approach. An explanation for this difference is that we assume the agents do not depend on each other positions when they are located far away from each other. However, already coordinating in these states might have a positive influence on the final result. For example, we observe in the final policy of the context-specific SparseQ approach that, when the predators are initialized at the same side of the prey, they independently move to the same adjacent cell of the prey when they are still in an uncoordinated state. When entering a coordinated state, they then have to perform one or two additional steps to position themselves correctly. These additional steps explain the lower reward obtained by the context-specific SparseQ method: they result in an additional negative reward because of the extra time steps, but also because of the larger discount value that is applied to the final reward when the prey is captured. Such constraints can be added as extra value rules, but then a lot of additional value rules have to be defined, resulting in slower convergence to a good policy. Clearly, a trade-off exists between the expressiveness of the model and the number of episodes to obtain a good policy.

Table 5.1 shows the average discounted cumulative reward and the average capture times over the last 10 test episodes after $1,000,000$ learning episodes. Both the MDP learners and the sparse tabular approach show similar results for the obtained reward and capture time. The sparse tabular method, however, only requires half the number of action values for its representation. The two context-specific SparseQ methods result in a policy which needs, for reasons explained earlier, approximately 1.3 additional steps to capture the prey, but learn based on a smaller number of action values: the edge-based decomposition stores approximately one seventh of the number of Q-values requires by the MDP learners approach.

The computation time for all table-based learning approaches, not shown, do not differ much: 100 updates take approximately 50 milliseconds. The two rule-based approaches involve the management of rules and are for this problem about 10 times slower. Because our focus is not on optimizing the implementation of the rule-based system (for example, how the rules are indexed), we will not discuss the computation times in much detail. We do note that, in general, the computational complexity depends on the particular problem under study. First, it depends on the number of rules that are used to represent each particular situation. Second, it depends on the complexity of the rule-based VE algorithm which on its turn depends on the number of rules that can be dominated during the local maximizations. Finally, we do note that the additional complexity of storing and managing set of rules only outweighs a full table-based representation with respect to the computation time in problems with a large state-action space involving a large amount of context-specific structure.

5.4 Learning interdependencies

In the previous section, we showed how agents are able coordinate their actions using predefined coordination dependencies that differ based on the context. Next, we propose a method to learn these dependencies automatically. The main idea is to start with independent learners and maintain statistics on expected returns based on the actions of the other agents. If the statistics indicate that it is beneficial to coordinate, a dependency is added dynamically between the involved agents. This method is inspired by 'utile distinction' methods from single-agent reinforcement learning that augment the state space when this distinction helps the agent predict reward [Chapman and Kaelbling, 1991; McCallum, 1997]. Hence, our method is called the *utile coordination* algorithm [Kok et al., 2005a]. Next, we explain this method in more detail, and apply it to a small coordination problem and the predator-prey problem.

5.4.1 Utile coordination

Our approach to automatically learn the dependencies between the agents is derived from the adaptive resolution reinforcement-learning methods for single-agent problems [Chapman and Kaelbling, 1991; McCallum, 1997]. These methods are used to construct a partitioning of the state space for partially observable Markov decision processes in which, as described in Section 2.2, the previous received observations might give additional information about the current state. An agent starts with an initial state representation based on its observations and augments its representation after finding so-called 'utile distinctions'. These are detected through statistics of the expected returns maintained for hypothesized distinctions [McCallum, 1997]. More specifically, this method stores the future discounted reward received after leaving a state and associates it with an incoming transition, that is, the previous state. When a state is Markovian with respect to return, the return values on all incoming transitions should be similar. On the other hand, if the statistics indicate that the returns are significantly different, the state should be split. This is done by distinguishing the state based on its incoming transitions, and making a separate state depending on each possible previous state. This allows a single agent to build an appropriate representation of the state space and predict the future reward better.

We take a similar approach in our utile coordination algorithm. The main difference is that, instead of keeping statistics on the expected return based on incoming transitions, we keep statistics based on the performed actions of the other agents. The algorithm starts with independent, uncoordinated, learners, and over time learns, based on acquired statistics, in which states specific action combinations of the independent learners result in a substantially higher reward. These states are then changed in coordinated states. In our context-specific CG framework this corresponds to adding new coordinated value rules based on all action combinations of the involved agents. Note that it is also possible to start with an initial CG that already incorporates coordination dependencies that are based on prior domain-specific knowledge and learn additional dependencies during learning.

Statistics of the expected return are maintained to determine the possible benefit of coordination. More formally, in each state s where coordination between two or more agents in a set I is considered, a sample of the *combined return* $\hat{Q}_I(s, \mathbf{a}_I)$ is stored for the performed joint action $\mathbf{a}$. The combined return is an approximation of the expected return that can be obtained by the involved agents $i \in I$ and equals the sum of their received individual reward $R_i(s, \mathbf{a})$ and their individual contribution $Q_i(s', \mathbf{a}^*)$ to the maximal global Q-value of the next state s':

$$\hat{Q}_I(s, \mathbf{a}_I) = \sum_{i \in I} [R_i(s, \mathbf{a}) + \gamma Q_i(s', \mathbf{a}^*)], \tag{5.15}$$

in which the optimal joint action $\mathbf{a}^*$ in state s' is computed using the VE algorithm. These samples can be regarded as an estimate of the total local payoff matrix for the agents in I in which each entry of the matrix specifies the expected return for a specific action combination.

These statistics are not used to change the agent's action values, but are stored to perform a statistical test at the end of an m-length trial to determine whether the agents in I should coordinate their actions. This test consists of the following steps for each state s. First, it computes the expected combined return for each of the performed actions $\mathbf{a}_I$ as the mean $\bar{Q}_I(s, \mathbf{a}_I)$ of the last M samples. Then, when enough samples are obtained, it measures whether the largest expected combined return $\max_{\mathbf{a}_I} \bar{Q}_I(s, \mathbf{a}_I)$ with variance $\sigma^2_{\max}$, differs significantly from the expected combined return $\bar{Q}_I(s, \mathbf{a}^*_I)$ with variance σ^2_*. The latter return corresponds to the return obtained when performing the greedy joint action $\mathbf{a}^*_I$ in state s and thus corresponds to the independently learned policy. When the two returns differ significantly for a specific state s, this state is changed into a coordinated state.

We use the t-test [Stevens, 1990] as the statistical test to compare the two values:

$$t = \frac{\max_{\mathbf{a}_I} \bar{Q}_I(s, \mathbf{a}_I) - \bar{Q}_I(s, \mathbf{a}^*_I)}{\sqrt{[(2/M)((M-1)\sigma^2_{\max} + (M-1)\sigma^2_*)/(2M-2)]}} \tag{5.16}$$

with $(2M - 2)$ degrees of freedom. From this value the level of significance p is computed indicating the probability of rejecting the null hypothesis, that is, the two groups are equal, when it is true. There also exist other statistical tests that can be applied to determine whether the two groups are significantly different. In particular, nonparametric tests may be used since assumptions of normality and homogeneity of variance may be violated. However, the t-test is fairly robust to such violations when, as in our case, group sizes are equal [Stevens, 1990].

Apart from the test whether the difference between the two groups are significant different, we also use an additional statistical *effect size measure* d to determines whether the observed difference is sufficiently large. We assume d equals the standard effect size measure [Stevens, 1990], and corresponds to the difference in means with respect to the maximum and minimum reward, that is,

$$d = \frac{\max_{\mathbf{a}_I} \bar{Q}_I(s, \mathbf{a}_I) - \bar{Q}_I(s, \mathbf{a}^*_I)}{r_{max} - r_{min}}. \tag{5.17}$$

If there is a statistically significant difference ($p < P$) with sufficient effect size ($d > D$), there is a significant benefit of coordinating the agents' actions in this state: apparently the current CG leads to a significantly lower return than the possible return when the actions are coordinated. This, for example, occurs in the situation in which one specific joint action produces a high return but all other joint actions result in a substantially lower return. Since the agents select their actions individually they only occasionally obtain the high return. Then, the stored statistics, based on joint actions, contain a combination of actions that results in a significantly higher return than the current policy. This is detected by the described test and the state is changed into a coordinated state. In our context-specific CG framework value rules based on individual actions for this particular state are then replaced by value rules based on joint actions. The value of each new rule $\rho(s, \mathbf{a}_I)$ is initialized with the learned value $\bar{Q}_I(s, \mathbf{a}_I)$.

It is also possible to apply the above procedure to test coordination between more than two agents or test coordination between different groups of agents. In the first case, the statistics are stored for joint actions $\mathbf{a}_I$ involving more than two agents. In the second case, the actions of each group of agents are represented as a single action. In both cases, the statistical test always looks at two estimates of expected combined return: $\max_{\mathbf{a}_I} \bar{Q}_I(s, \mathbf{a}_I)$ and $\bar{Q}_I(s, \mathbf{a}_I^*)$.

In very large state-action spaces, memory and computation limitations make it infeasible to maintain these statistics for all states. It then makes sense to use a heuristic 'initial filter' which detects potential states where coordination might be beneficial. The full statistics on combined returns are then only maintained for the potential interesting states detected by the initial filter. In this way, large savings in computation and memory can be obtained while still being able to learn the required coordination. Our emphasis in this thesis is on showing the validity of the utile coordination algorithm. Therefore, we do not use a heuristic initial filter in our experiments, and store statistics for every state.

5.4.2 Experiments

In this section, we apply the utile coordination algorithm to two problems: a simple coordination problem and the larger predator-prey domain.

Simple problem

We first illustrate our utile coordination approach on the simple problem depicted in Fig. 5.8. This collaborative MMDP consists of two agents and seven states. In each state both agents select an individual action from their action set $\mathcal{A}_1 = \mathcal{A}_2 = \{a, b, c\}$. The resulting joint action only influences the state transition in state s_0. When one of the agents selects action b, the system transitions to s_1. In this state every selected joint action, indicated by $(*, *)$, results in a transition to state s_4 and both agents receive a reward of 0.25. Selecting the joint action (c, c) in state s_0 eventually results in the highest reward of 1.5, but failure of coordination, that is selecting one of the

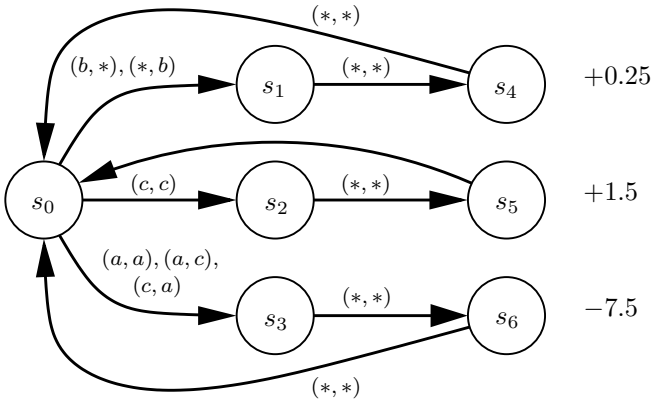


Figure 5.8: Simple coordination problem with seven states. The joint action influences the state transition in state s_0. The numbers on the right represent the given reward to the agents in the corresponding state.

three remaining joint actions, results in a large negative reward of -7.5. The problem is chosen such that the rewards are delayed, and the agents need to actively coordinate their actions in state s_0 in order to gather the reward of 1.5. When only one of the agents selects the individual action c that is part of the optimal joint action in s_0, both agents receive a large penalty. This causes problems for independent learners because they do not take into account the action selected by the other agent. Using our utile coordination algorithm, the agents should detect that they need to coordinate in state s_0 and generate coordinated value rules for this state. When these new rules are added, they are able to learn the outcome for each joint action and select the coordinated action that results in the high reward.

We apply different reinforcement-learning techniques on the problem in Fig. 5.8. The IL and DVF approach learn based on individual actions, and both need 42 action values to represent the state-action space. The MDP learners model the joint action for every state resulting in 63 action values. Just as with the IL approach, our utile coordination approach starts with action values represented using an edge-based context-specific CG that is based on value rules with individual actions. After each $m = 1,000$ steps, this method checks for every state whether it is a coordinated state using (5.16) and (5.17). In case the test indicates the actions of the agent depend on each other, the value rules based on individual actions are replaced by value rules based on joint actions.

All approaches learn for $7,000$ cycles and average their results over 30 runs. Since it always takes three steps to return to the starting state s_0 we perform three testing cycles, in which the agents select their greedy action, after every three learning cycles. All methods use a learning rate $\alpha = 0.25$, a discount factor $\gamma = 0.9$, and ϵ-greedy value $\epsilon = 0.3$. For a fair comparison, all approaches explore jointly, that is, with probability ϵ all agents select a random action. This is more conservative

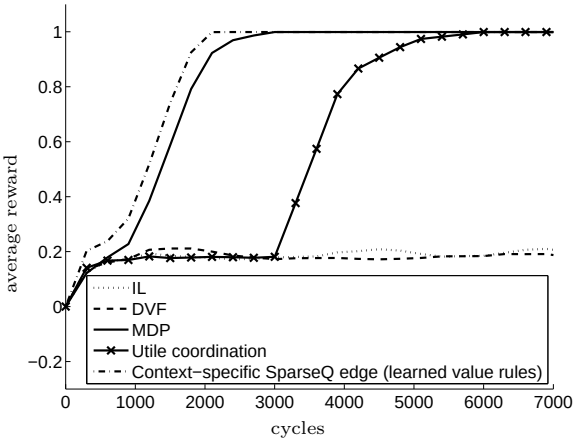


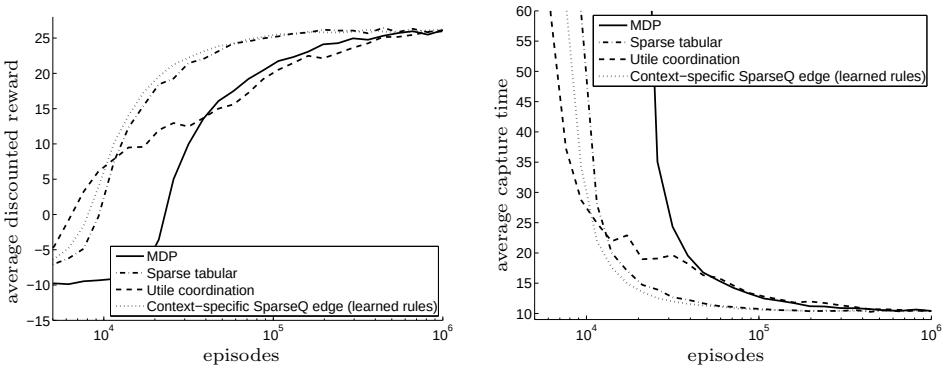
Figure 5.9: Running average, of the last 1,000 cycles, of the global reward for the different approaches on the problem in Fig. 5.8. Results are averaged over 30 runs.

than the case in which independent learners independently choose a random action with probability ϵ. For the parameters in our utile coordination approach, we use a significance level $P = 0.05$, an effect size $D = 0.01$, and a sample size $M = 10$.

Fig. 5.9 shows the running average, over the last 1,000 test cycles, of the global reward for the different applied techniques. The running average is taken over so many time steps to make the different curves smoother. The IL and the DVF approach both converge to the safe policy of moving to state s_1 and receive the non-optimal global reward of 0.5. Note that the reward is only received every third cycle, and therefore the average reward shown in Fig. 5.9 is one third of this value. They fail to converge to the optimal policy of choosing action c because the corresponding Q-value is significantly lowered in the case that the other agent performs a different action. The MDP learners do not have this problem, because they model each joint action separately, and converge to the optimal policy.

Our utile coordination approach starts with value rules based on individual actions and therefore follows the learning curve of the IL approach in the beginning. However, Fig. 5.9 shows that from the third trial onward, that is, from time step 3,000, the average reward of the utile coordination approach transitions from the IL curve to that of the MDP learners. Although it occasionally takes four or five trials before the coordination dependency is detected, the agents then have enough samples to discover that s_0 is a coordinated state and add value rules based on joint actions for state s_0. As a result, the agents are able to distinguish which joint action results in the high reward and converge to the optimal policy.

Fig. 5.9 also shows that when we apply our context-specific SparseQ method using the learned coordination dependencies the system converges slightly quicker to the optimal policy than the MDP learners approach. The learned rules only model joint



(a) Average reward.

(b) Average capture time.

Figure 5.10: Running average of the reward and capture times during the first $1,000,000$ episodes. The episodes are shown using a logarithmic scale. Results are averaged over 10 runs.

actions for state s_0, and therefore this approach needs less exploration. Note that this representation is identical to a sparse tabular representation when only state s_0 is defined as a coordinated state.

Pursuit problem

We also apply our utile coordination algorithm to the same predator-prey problem as discussed in Section 5.3.3. In a 10×10 toroidal grid it is the goal of two predators to capture a single prey. The prey is captured when the predators are both positioned in an adjacent cell to the prey, and then one of the predators moves onto the prey position while the other remains on its current position. In the utile coordination approach, each agent starts with individual value rules that are based on the individual actions of the agent and the full state information, that is, the relative position of both predators to the prey. The statistical test to determine which states are coordinated are performed after every $m = 20,000$ episodes. Again, we use a significance level $P = 0.05$, an effect size $D = 0.01$, and the same reinforcement-learning parameters that are used in the experiments in Section 5.3.3, $\epsilon = 0.3$, $\alpha = 0.25$, and $\gamma = 0.9$.

We run the utile coordination method 10 times. After every 100 learning episodes, the current policy is applied to the same 100 starting configurations used in the experiments in Section 5.3.3. Fig. 5.10 shows the running average, over the last 10 episodes, of the reward and the capture time. It also shows the results of the sparse tabular and MDP approach, which are also based on the full state information, from the experiments in Section 5.3.3. The utile coordination approach initially learns based on individual actions. However, after the end of the first trial, at episode $20,000$, the agents add coordinated value rules for the states in which the gathered statistics indicate that coordination is beneficial and the performance slowly increases.

method	reward	avg. time	#Q-values
MDP	25.997	10.566	242,550
Sparse tabular	26.001	10.362	115,740
Utile coordination	25.044	10.452	103,851
Context-spec. SparseQ edge (learned rules)	26.023	10.352	103,851

Table 5.2: Average reward and capture time, of the last 10 test runs, after learning for $1,000,000$ episodes. The number of state-action pairs is also given.

This process continues as more fine-grained coordination dependencies are added. In the end, the found policy is similar to the policy found by the sparse tabular and MDP learners approach. On average 455.40 out of the $9,702$ states were found to be statistically significant and added as coordinated states. This is a smaller number than the $1,248$ manually specified states in Section 5.3.3 in which coordinated rules were added for all states in which the predators where within two cells of each other or both within two cells of the prey. This difference is caused by the fact that for many states the agents are able to learn how to coordinate using value rules based on individual actions, for example, how to avoid a collision.

We also apply the edge-based context-specific SparseQ method using the learned coordination dependencies. As in Section 5.3.3, the values of the coordinated value rules are initialized with the maximal reward of 75, while those based on individual actions are initialized at 37.5. Because of the smaller number of used dependencies this approach learns slightly quicker than the sparse tabular approach, and results in a similar performance.

Table 5.2 shows the final capture times and the number of Q-values needed to represent the state-action space for each method, indicating that all applied methods result in a similar performance.

5.5 Discussion

In this chapter we described context-specific sparse cooperative Q-learning (context-specific SparseQ), a reinforcement-learning approach for cooperative multiagent systems in which the global action value is decomposed using value rules that specify the coordination requirements of the system for a specific context. These rules can be regarded as a compact representation of the complete state-action space since they are defined over a subset of all state and action variables. We investigated both an agent-based and edge-based decomposition of the global action value. In both cases, the value of each rule contributes additively to the global action value, and is updated based on a Q-learning update rule that adds the local contribution of all involved agents in the rule. Effectively, each agent learns to coordinate only with its neighbors in a dynamically changing coordination graph. Results in the predator-prey domain

show that our method learns a policy close to the optimal policy and improves upon the learning time of other multiagent Q-learning methods. In order to perform even closer to the optimal policy more coordination dependencies have to be defined, but this will decrease the learning time. A trade-off exists between the used representation of the state-action space and the number of learning steps to obtain a good policy.

We also introduced the utile coordination algorithm, a method which starts with independent, non-coordinating, agents and learns automatically where and how to coordinate. The method is based on maintaining statistics on expected returns for hypothesized coordinated states, and a statistical test that determines whether the expected return increases when actions are explicitly coordinated. Using the value-rule representation, a coordinated requirement can easily be constructed by adding value rules which incorporate the actions of the other agent.

We only studied the problem of extending the action space. Another interesting direction is to investigate the possibility of decreasing the action space by removing dependencies which are unnecessary according to the gathered statistics. In this case, we also have to store statistics for coordinated states based on hypothesized uncoordinated states and test whether the expected return is not significantly lower when the actions are not explicitly coordinated.

As described before, maintaining the complete statistics for all states is not computationally feasible for large problems with many agents. Since these are the tasks where the advantage of utile coordination over MDP learners, in terms of space and learning time, is more pronounced, it is important to investigate additional methods, for example, heuristic initial filters, that determine which agents coordination dependencies should be tested for a specific state.

Heuristic initial filters are not the only way to deal with large state-action spaces. In this chapter, we investigated a method to learn the coordination dependencies between the agents when the full state information was given. An equally important, orthogonal possibility is a variation of the utile coordination algorithm based on learning which state variables are important for coordination. This should also combine well with coordination graphs, because they are explicitly designed for such state representations. An individual agent is then able to start with rules which only represent its own individual view of the environmental state, and then learn to augment their state representation when necessary.

6

Dynamic Continuous Domains

In this chapter we present a method to coordinate multiple robots in dynamic and continuous domains. We apply context-specific coordination graphs to complex uncertain environments by assigning roles to the agents based on the continuous state information and then coordinating the different roles [Kok et al., 2003, 2005b]. This simplifies the coordination structure and constrains the action space of the agents considerably. Furthermore, we demonstrate that, with some additional common knowledge assumptions, an agent can predict the actions of the other agents, rendering communication superfluous. This approach results in a method to specify a team strategy using natural coordination rules. We have successfully implemented the proposed method into our UvA Trilearn simulated robot soccer team which won the RoboCup-2003 World Championships in Padova, Italy.

6.1 Introduction

In the previous chapters we investigated techniques to coordinate the behavior of a group of agents in multiagent sequential decision-making problems with a discrete representation of the state-action space. In this chapter we extend these techniques to coordinate agents embedded in a complex uncertain environment [Koller, 2004].

As before, we use a context-specific coordination graph (context-specific CG), see Section 5.2, to model the dependencies between the agents using value rules. The continuous nature of the state-action space, however, makes the direct application of context-specific CGs difficult. Therefore, we appropriately 'discretize' the continuous state by assigning *roles* to the agents and then, instead of coordinating the different agents, coordinate the different roles. This approach offers several benefits: the set of roles allows for the definition of natural coordination rules that exploit prior knowledge about the domain, and constrain the feasible action space of the agents. This greatly simplifies the modeling and the solution of the problem at hand. We also describe how the agents, with additional common knowledge assumptions, are able to predict the optimal action of their neighboring agents, making communication superfluous.

We apply our method to coordinate the agents in the RoboCup simulation soccer domain [Noda et al., 1998]. The Robot Soccer World Cup (RoboCup) is an international research initiative that uses the game of soccer as a domain for artificial intelligence and robotics research. The *soccer server* [Chen et al., 2003] is the basis

for the 2D simulation competition. It provides a fully distributed dynamic multi-robot domain with both teammates and adversaries and models many real-world complexities such as noise in object movement, noisy sensors and actuators, limited physical ability, and restricted communication. One team is represented by eleven different computer processes that independently interact with the simulator in order to fulfill their common goal of scoring more goals than their opponent.

We perform experiments using our robot soccer simulation team *UvA Trilearn* [De Boer and Kok, 2002] in which we how to coordinate the behavior of different agents using context-specific CGs that are specified for the different roles.

The setup of this chapter is as follows. In Section 6.2 we describe the RoboCup initiative and the soccer simulator. In Section 6.3 we review the main characteristics of our robot soccer simulation team UvA Trilearn. In Section 6.4, we present our framework to coordinate a group of agents in a continuous dynamic environment using roles. In Section 6.5 we apply this approach to coordinate the agents in the UvA Trilearn team, and perform several benchmarking experiments. Furthermore, we give an overview of the results of UvA Trilearn in the different competitions in which it participated in 2003. Finally, we end with some conclusions in Section 6.6.

6.2 RoboCup

In this section we shortly review the robot world cup initiative and give some details about the 2D soccer simulation system that is used in the simulation competition.

6.2.1 The robot world cup initiative

The Robot World Cup (RoboCup) initiative is an attempt to foster artificial intelligence (AI) and intelligent robotics research by providing a standard problem where a wide range of technologies can be integrated and examined [Kitano et al., 1997]. RoboCup's ultimate long-term goal is stated as follows [Kitano and Asada, 1998]:

> "By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of the FIFA, against the winner of the most recent world cup for human players."

Although this sounds like an overly ambitious goal, history has shown that much progress can be accomplished in only a few decades. For example, it took roughly 66 years after the first man-carrying flight by Orville Wright in 1903, which only covered about 120 feet and lasted for 12 seconds [Jakab, 1990], to the moment that Neil Armstrong stepped out of the Apollo-11 Lunar Module onto the surface of the moon in 1969 [Collins and Aldrin, 1975]. Also, it took only 51 years from the release of the first operational general-purpose electronic computer in 1946, the ENIAC built by J. Presper Eckert and John Mauchly at the University of Pennsylvania [Patterson and Hennessy, 1994], to the computer chess program Deep Blue which defeated the human world champion Gary Kasparov [Schaeffer and Plaat, 1997].

Aside from this long-term objective, RoboCup also looks to short-term objectives. In the first place, RoboCup promotes robotics and AI research by providing a challenging problem as it offers an integrated research task which covers many areas of AI and robotics, for example, design principles of autonomous agents, multiagent collaboration, strategy acquisition, real-time reasoning, reactive behavior, real-time sensor fusion, learning, vision, motor control, intelligent robot control, and many more [Kitano et al., 1997]. Secondly, RoboCup is used to stimulate the interest of students and the general public for robotics and AI. This is apparent from the different RoboCup-related study projects given in universities all over the world, and the media and public attention for the world championship every year. The RoboCup-2005 world championship in Osaka, for example, has been visited by more than $150,000$ visitors. Another aspect of RoboCup is that it provides a standard problem for the evaluation of various theories, algorithms and architectures. Using a standard problem for this purpose has the advantage that different approaches can be easily compared and that progress can be measured.

In order to achieve the RoboCup long-term objective, the RoboCup organization has introduced several robot soccer leagues which each focus on different abstraction levels of the overall problem. Currently, the main leagues are the following:

- **Humanoid League**. In this league a humanoid robot has to accomplish different soccer related tasks, which range from taking penalty kicks to playing two against two. The main research areas are related to the control of a two-legged robot and include dynamic walking, running, kicking the ball, visual perception of the field, and self-localization.

- **Middle Size Robot League**. In this league each team consists of a maximum of four robots, which are about 75cm in height and 50cm in diameter. The playing field is approximately 12×8 meters and the robots have no global information about the world. Important research areas for this league include localization, computer vision, sensor fusion, distributed perception, robot motor control, and hardware issues.

- **Small Size Robot League**. In this league each team consists of five robots, which are about 15cm in height and 18cm in diameter. The playing field has the size of a table-tennis table. A separate computer, which receives a global view of the field from an overhead camera, controls the different robots. Research areas which are important for this league include those of the Middle Size League, but because of the global control the focus is more on strategy development.

- **Four Legged Robot League**. In this league each team consists of four Sony quadruped robots, better known as AIBOs. The robots have no global view of the world but localize themselves using various colored landmarks which are placed around the field. The main research areas for this league are similar to those of the Middle Size League, with an extra focus on intelligent robot control and quadruped locomotion.

- **Simulation League**. In this league each team consists of eleven synthetic software agents which operate in a simulated environment. Research areas which are being explored in this league include multiagent collaboration [Tambe, 1997], multiagent learning [Stone, 1998; Riedmiller and Merke, 2002], and opponent modeling [Riley and Veloso, 2000]. The soccer competition consists of both a 2D and 3D soccer competition. In the 3D competition, which was held for the first time in 2003, the agents interact with the world based on real-world dynamics implemented using the open dynamics engine (ODE) [Smith, 2006]. The agents are currently represented by spheres with a simple kicking device, but the implementation of the simulator allows the agents to be configured by an arbitrary combination of different objects. This allows teams from the other leagues to experiment with the simulator using agents that have the same physical constraints as their own robots.

  In the 2D competition, which has been held from 1997, the physical model of the agents is greatly simplified which makes it easier to focus on high-level aspects as learning and strategic reasoning.

  Currently, the simulation league is the largest league due to the fact that no expensive hardware is needed to build a team. Furthermore, it is much easier, and cheaper, to test a simulation team against different opponents.

We will mainly concentrate on the 2D simulator of the RoboCup Simulation League in this chapter. Next, we will review the 2D soccer simulation system, called the RoboCup soccer server, in more detail.

6.2.2 The RoboCup soccer server

The *RoboCup soccer server* [Chen et al., 2003] is a soccer simulation system which enables teams of autonomous agents to play a match of soccer against each other. The system was originally developed in 1993 by Dr. Itsuki Noda (ETL, Japan) [Noda et al., 1998]. In recent years it has been used as a basis for several international competitions and research challenges. The soccer server contains many real-world complexities such as sensor and actuator noise and limited perception and stamina for each agent. One of the advantages of the soccer server is the abstraction made, which relieves researchers from having to handle robot problems such as object recognition and movement. This makes it possible to focus on higher level concepts such as learning and strategic reasoning. The soccer server is commonly used as a research platform for exploring different AI techniques [Tambe, 1997; Stone, 1998; Withopf and Riedmiller, 2005]. In this chapter we give an overview of the different aspects of the simulator. Chen et al. [2003]; De Boer and Kok [2002] describe a more detailed description.

A simulation soccer match is carried out in client-server style. The soccer server provides a virtual soccer field, simulates all the movements of objects in this domain, and controls a soccer game according to several rules. Each single player is controlled by a separate client program which connects to the server. Each player independently receives information about the current state of the world, and is able to send requests

Figure 6.1: The soccer monitor display. Note that the soccer field and all objects on it are two-dimensional. The concept of 'height' thus plays no role in the simulation. Each player is drawn as a circle and contains two different colored lines. The black line defines the movement direction of the agent. The lighter grey line represents the direction of its vision.

to the server to perform a desired action. The distributed interaction with the simulator complicates the decision-making process because each agent *individually* has to select its action based on its incomplete estimate of the current world state.

The soccer server is a pseudo real-time system that works with discrete time intervals, called simulation cycles, each lasting 100ms. During this period, the agents receive various kinds of sensory observations from the server and send action requests to the server. A complex feature of the soccer server is that sensing and acting are *asynchronous*. By default, clients can send action requests to the server once every 100ms, but they only receive visual information at 150ms intervals. It is only at the end of a cycle that the server executes the actions and updates the state of the environment. The server thus uses a discrete action model.

The simulator includes a visualization tool called the *soccer monitor*, which allows people to see what happens within the server during a game. Fig. 6.1 shows a graphical representation of the complete field modeled by the soccer server. Each player is drawn as a circle and contains two different colored lines. The black line represents the front part of the player's body and defines the direction in which it can move. The lighter grey line represents the player's neck angle and represents the direction of its vision.

An agent has three different types of sensors with which it obtains information from its environment: a *visual* sensor, a *body* sensor and an *aural* sensor. The visual sensor provides visual information, such as the relative distance, direction, and velocity, of the objects in the player's current field of view. Objects that are observed include the agent's teammates, opponents, the ball, and several landmarks that are positioned around the field. The latter are used to localize the agent. Noise is added to the visual sensor data and is larger for objects that are further away. The view cone of an agent has a limited width and as a result the agent only has a partial view of the world and large parts of the state space remain unobserved. The body sensor reports physical information about the player, such as its stamina, speed, and neck angle. Finally, the aural sensor detects spoken messages which are sent by the other players. Each agent hears at most one message every cycle. The soccer server communication paradigm thus models a crowded, low-bandwidth environment in which the agents from both teams use a single, unreliable communication channel [Stone, 1998].

An agent can perform different types of actions which can be divided into two distinct categories: *primary* actions (kick, dash, turn, move, catch, and tackle), and *concurrent* actions (say, turn_neck, change_view, sense_body, and score). Most actions are defined in one or more continuous-valued parameters. The dash command, for example, accepts a parameter in the range $[-100, 100]$ which specifies the amount of acceleration in the agent's body direction (negative is backwards). To each of the actuator parameters noise is added such that the executed action is never exactly the desired one. Each cycle only one primary action can be executed, whereas multiple concurrent actions can be performed simultaneously with any primary action.

The soccer server simulates object movement stepwise in a simple way: the velocity of an object is added to its position, while the velocity decays by a certain rate and increases by the acceleration of the object resulting from specific action commands. To reflect unexpected movements of objects in the real world, uniformly distributed random noise is added to the movement of all objects. Furthermore, the soccer server prevents players from constantly running at maximum speed by assigning a limited stamina to each of them. When a player performs a dash command this consumes some of its stamina but its stamina is also slightly restored in each cycle. If a player's stamina drops below a certain threshold this will affect the efficiency of its movement.

Each player in the simulator belongs to a player type with different abilities based on certain trade-offs. For example, some types will be faster than others but they will also become tired more quickly. At the start of a game, a set of different, randomly generated, heterogeneous players is generated. A *coach agent* that receives noise-free global information about all the objects on the soccer field is then allowed to select the player types and substitute them when necessary during the game. The coach is also a good tool for analyzing the strengths and weaknesses of the opponent team and for giving advice to the players about the strategy during the dead-ball situations.

In order to create a soccer team a mapping has to be made, for each individual agent separately, from the incoming perceptions to the low-level action commands understood by the soccer server. Next, we explain how this is accomplished in our simulated soccer team UvA Trilearn.

6.3 UvA Trilearn

UvA Trilearn is the simulated soccer team from the University of Amsterdam that participated in the 2D competition of the RoboCup world championships from 2001 to 2005. In this section (that is largely based on De Boer and Kok [2002]) we describe the basis functionality of UvA Trilearn, that is, its architecture, the world model, and the skills available to the agents. The corresponding source code, which contains all team functionality except the high-level decision-making process, has been released in 2003 and is used by many of the current teams.[1] After this review, we then describe our approach to select a coordinated action based on the current world state using context-specific coordination graphs.

Functional architecture

A functional architecture concerns itself with the functional behavior that the system should exhibit [Visser et al., 1999]. It describes what the system should be capable of doing, independent of hardware constraints and environmental conditions. Complex tasks, such as simulated robot soccer, can always be hierarchically decomposed into several simpler subtasks. This naturally leads to agent architectures consisting of multiple layers. The UvA Trilearn agent architecture, shown in Fig. 6.2, is a hybrid approach between a hierarchical and a behavioral approach. The higher abstraction levels are mainly decomposed hierarchically because high-level reasoning is completely sequential, whereas the lower levels are more behavioral since the real-time control of the system involves mostly parallel processing.

The hierarchical decomposition is divided into three progressive levels of abstraction which are represented by different architectural layers. The bottom layer is the *interaction layer* which takes care of the interaction with the soccer server simulation environment. This layer hides the soccer server details as much as possible from the other layers. The middle layer is the *skills layer* which uses the functionality offered by the interaction layer to build an abstract model of the world and to implement the various skills of each agent, for example, a skill to intercept the ball. The highest layer in the architecture is the *control layer* which contains the reasoning component of the system. In this layer, the best possible action is selected from the skills layer depending on the current world state and the current strategy of the team.

The UvA Trilearn agents are thus capable of *perception*, *reasoning*, and *acting*. The setup for the agent architecture shown in Fig. 6.2 is such that these three activities can be performed in parallel, an important aspect of a behavioral architecture. A separate thread is defined for each of the three main activities: the *sense* thread represents the perception module, the *act* thread represents the actuator control module, and the *think* thread represents the modules from the skills layer and control layer. Each thread works independently such that, for example, new visual information is immediately processed when it arrives, and can be used by the think thread to select an action that is based on the most up-to-date information.

[1] The package release is freely available from `http://www.science.uva.nl/~jellekok/robocup/`.

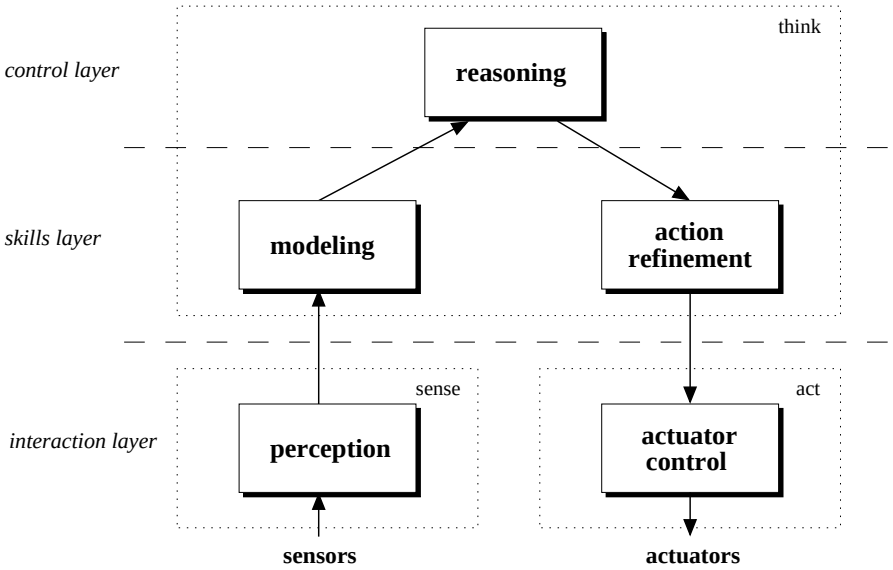


Figure 6.2: The UvA Trilearn agent architecture.

Agent world model

In order to behave intelligently it is important that each agent keeps a world model that describes the current state of the environment as accurate as possible. The agent uses this world model to reason about the best possible action in a given situation. The agents of the UvA Trilearn soccer simulation team keep a world model that contains information about all the objects on the soccer field. This world model can be seen as a probabilistic representation of the real world based on past perceptions. For each object, among others, an estimation of its global position and velocity are stored together with a confidence value that indicates the reliability of the estimate. This confidence value is derived from the time stamp in which the estimate is made, that is, if the estimate is based on up-to-date information the associated confidence value will be high. The world model is updated as soon as new sensory information is received by the agent and thus always contains the last known information about the state of the world. When new sensory information is received, an agent first updates its own global position in the world using a particle filter that uses the relative information of the observed landmarks [Vlassis et al., 2002; De Boer and Kok, 2002]. Objects that are not observed are updated based on their previous estimated velocity and their confidence is decreased. Furthermore, the world model contains several methods which use this information to derive higher-level conclusions, for example, the number of opponents in a certain area, or the probability that a pass to a specific teammate succeeds.

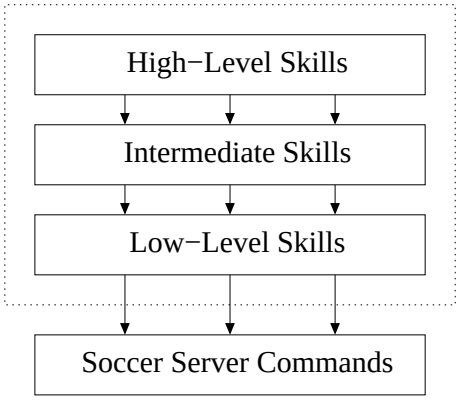


Figure 6.3: The UvA Trilearn skills hierarchy consisting of three layers. The low-level skills are based on *soccer server* commands, whereas the higher-level skills are based on skills from the layer below.

Player skills

A skill can be regarded as the ability to execute a certain action. In general, these skills can be divided into simple skills that correspond to basic actions and more advanced skills that use the simple skills as parts of more complex behaviors. The skills which are available to the UvA Trilearn agents include turning towards a point, kicking the ball to a desired position, dribbling, intercepting the ball, and marking opponents. Together, the skills form a hierarchy consisting of several layers at different levels of abstraction. Fig. 6.3 shows the UvA Trilearn skills hierarchy which consists of three layers. The layers are hierarchical in the sense that the skills in each layer use skills from the layer below to generate the desired behavior. The bottom layer contains low-level player skills which can be directly specified in terms of basic action commands known to the soccer server. At this abstraction level the skills correspond to simple actions such as turning towards a point. The middle layer contains intermediate skills which are based on low-level skills. The skills in this layer do not have to deal with the exact format of server messages anymore but can be specified in terms of the skills from the layer below, for example, turning towards the ball. Finally, the skills at the highest level are based on intermediate skills, for example, intercepting the ball. Note that intermediate and high-level skills often depend on specific information that is available in the current world model of the agent.

The behavior of the agent is the result of selecting an appropriate skill in a given situation. The strategy of a team of agents can then be seen as the way in which the individual agent behaviors are coordinated. For example, when one player passes the ball, the agent to which the ball is passed should select a skill to anticipate the pass. In the next section, we describe a method to select such coordinated skills using a context-specific coordination graph.

6.4 Coordination in dynamic continuous domains

A team of agents that is faced with a decision-making problem, has to tackle the issue of coordination. As the agents share a common performance measure, they have to coordinate their individual actions in order to maximize team performance. In the RoboCup simulation domain, for example, the agent that controls the ball must coordinate with its surrounding teammates in order to perform a pass, and the defenders must coordinate to position themselves such that they cover as much space as possible. In principle game-theoretic techniques can be applied to solve such a coordination game (see Section 2.3.4). The problem with this approach, however, is that it becomes intractable to model practical problems involving many agents because the joint action space is exponential in the number of agents. However, the particular structure of the coordination problem can often be exploited to reduce its complexity [Kok et al., 2003, 2005b]. Such dependencies can be modeled by a context-specific coordination graph (context-specific CG), see Section 5.2, that satisfies the following requirements: (i) its connectivity should be dynamically updated based on the current, continuous, state, (ii) it should be sparse in order to keep the dependencies and the associated local coordination problems simple, (iii) it should be applicable in situations where communication is unavailable or very expensive.

In the remainder of this section, we will describe our coordination method, designed to fulfill the requirements mentioned above. Our proposed method involves two main features. The first is the assignment of roles to the agents in order to apply context-specific CGs to continuous domains and to reduce the action sets of the different agents; the second is to predict the chosen action of the other agents, rendering communication superfluous. As a main example we will use the RoboCup simulation soccer domain described in Section 6.2.2.

6.4.1 Context-specificity using roles

Context-specific coordination graphs (context-specific CGs), as described in Section 5.2, are a natural way to specify the coordination requirements of a system. This framework uses value rules to specify the coordination dependencies for a specific context. Value rules can be regarded as a compact representation of the complete state-action space since they are defined over a subset of all state and action variables. Each rule that is applicable to the current state and selected joint action contributes a certain value to the system. The rule-based variable elimination or max-plus algorithm, see respectively Section 5.2 and Section 3.3, can be used to compute the joint action that maximizes the sum of the values that are applicable for the current state.

A limitation of a context-specific CG, however, is that it is based on propositional rules and therefore only directly applies to discrete domains. In this chapter we are interested in robots that are embedded in continuous domains. Conditioning on a context that is defined over a continuous domain is difficult. One solution would be to divide the state space into small discrete partitions, either directly or using tile

codings [Albus, 1971; Sutton and Barto, 1998], and then apply the original context-specific CG approach. The problem, however, is that the resulting state space becomes very large and many coordinated rules are required to specify the team behavior.

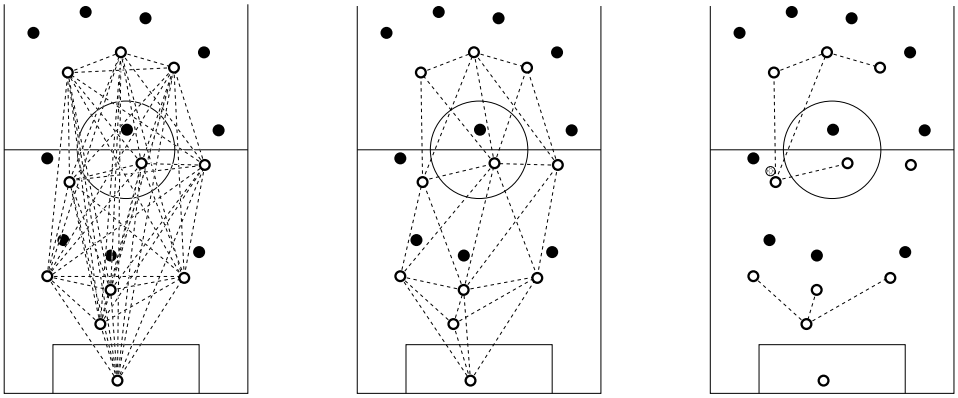
A different way to 'discretize' the context is by assigning *roles* to the agents [Stone and Veloso, 1999; Spaan et al., 2002; Iocchi et al., 2003; Vlassis, 2003]. Roles are a natural way of introducing domain prior knowledge to a multiagent problem and provide a flexible solution to the problem of distributing the global task of a team among its members. In the soccer domain for instance one can easily identify several roles ranging from 'active' or 'passive' depending on whether an agent is in control of the ball or not, to more specialized ones like 'striker', 'defender', or 'goalkeeper'.

Tambe [1997] defines a role as an abstract specification of the set of activities an individual or subteam undertakes in service of the team's overall activity. In our framework the set of activities for a role $m \in M$ is represented as a set of value rules f_m. For a given role assignment only one set applies. This simplifies the context-specific CG substantially because many dependencies related to other roles can be ignored. Furthermore, the role assignment can be used to specify additional constraints for the other agents involved in the value rule, reducing the number of dependencies in the CG even further. For example, agent i that controls the ball in role 'goalkeeper' only has to consider passes to agents in the role of 'defender'. This situation can be represented using the following value rule:

$$\langle p_1^{goalkeeper} \quad ; \quad \text{has-ball}(i) \quad \wedge \quad \text{has-role-defender}(j) \quad \wedge \quad a_i = \text{passTo}(j) \quad : \quad 10 \rangle$$

Fig. 6.4 shows the substantially reduction of the number of dependencies in a typical soccer situation when roles are assigned. Fig. 6.4(a) shows a fully connected CG in which each agent is connected to all other agents. Fig. 6.4(b) shows the CG before assigning the roles to the agents and consists of all possible dependencies for all possible roles. Finally, Fig. 6.4(c) shows the CG after the role assignment for the situation that the ball is in the left midfield. In this example, the bottom agent takes the role of sweeper while the other three take the role of defender. The sweeper covers the space between the defenders and the goalkeeper to allow the defenders to advance to support the attack. As long as the four agents agree on their role assignment the problem of their coordination is simplified: the defenders only need to take into account the action of the sweeper in their strategy (apart from other factors such as the opponents), ensuring it is the most retracted field player. In their local coordination game they do not need to consider other teammates such as the goalkeeper or the attackers. Several other local coordination games could be going on at the same time. For example, in the attack the player with the ball has to coordinate with the players that are able to receive a pass.

The context-specific CG is continuously updated to reflect the current situation on the field. Given a particular local situation, each agent is assigned a role that is computed based on a role assignment function that is common knowledge among the agents. In the communication based case, we use a distributed role assignment algorithm [Castelpietra et al., 2000; Spaan et al., 2002; Vlassis, 2003]. This algorithm,



(a) (b) (c)

Figure 6.4: Applying coordination graphs using roles in a typical soccer situation. The black circles represent the opponent agents and the open circles show the agents in our team. (a) A fully connected CG, in which each agent is connected to all other agents. (b) A reduced CG in which each agent is connected to its neighbors for all possible role assignments. (c) The resulting CG after the role assignment in case the ball is in the left midfield.

which is common knowledge among the n agents, defines a sequence of roles M', with $|M'| \geq n$, which represents a preference ordering over the roles: the most 'important' role is listed first in the ordering and is assigned to an agent first, the second most important role is second in the ordering, etc. The same role can be assigned to more than one agent, that is, M' can contain multiple copies of the same role, but each agent is assigned only a single role. Each role m has an associated potential r_{im} which is a real-valued estimate that specifies how appropriate agent i is for the role m in the current world state. These potentials r_{im} depend on features of the state space relevant for role m as observed by agent i. For example, relevant features for the role 'striker' are the time needed to intercept the ball or its position. Each agent computes its potential for each $m \in M$ and sends these to the other agents. The first role m in the sequence M' is assigned to the agent that has the highest potential for that role. The process proceeds with assigning the second role $m' \in M'$ to the remaining agent with the highest potential for the role m'. This process continues until all agents are assigned a role. This algorithm requires sending $O(|M|n)$ messages, as each agent has to send each other agent its potential r_{im} for all $m \in M$.

For example, lets assume we have a problem with three agents and the roles $M = \{\text{passer}, \text{receiver}\}$ based on the priority sequence $M' = \{\text{passer}, \text{receiver}, \text{receiver}\}$. Each agent computes the potentials for the passer and receiver role and then communicates these to the other agents. The agent that has the highest potential for the passer role, which is the first element in the priority sequence, assigns itself this role, and the remaining two agents assign themselves the receiver role.

An assignment of roles to agents provides a natural way to parameterize a coordination structure over a continuous domain. The intuition is that, instead of directly coordinating the agents in a particular situation, we assign roles to the agents based on this situation and then try to 'coordinate' the set of roles. The roles can be regarded as an abstraction of a continuous state to a discrete context, allowing the application of existing techniques for discrete-state CGs. Roles also reduce the action space of the agents by 'locking out' specific actions. For example, the role of the goalkeeper does not include the action 'score', and in a 'passive' role the action 'shoot' is deactivated.

6.4.2 Non-communicating agents

In order to coordinate the different agents using predefined role-specific value rules requires communication in different stages. For the role assignment, each agent has to communicate its potential for a certain role to all other agents. For computing the optimal joint action given the applicable value rules (we assume the variable elimination (VE) algorithm from Section 5.2 is used), each agent receives the relevant value rules of its neighboring agents, and communicates its computed conditional strategy to its neighbors. Similarly, in the reverse process each agent needs to communicate its decision to its neighbors in order to reach a coordinated joint action.

In many practical dynamic situations, the agents may not be able to communicate with all neighbors, nor have the time to finalize all communication before assigning the roles and selecting an action, due to failures or time constraints. However, we can still apply a similar procedure if we make some common knowledge assumptions.

In order to determine the role assignment without using communication, each agent i computes, in parallel, the potentials r_{jm} for all roles $m \in M$ and all agents j located in the subgraph in which agent i is involved. This can only be done when agent i has access to the potential functions and the state variables that are relevant for computing the potential of the agents in its subgraph. After computing all potentials, the actual assignment of roles to agents is equal to the procedure described earlier. During the non-communicative role assignment, each agent computes at most $O(|M|n)$ potentials and thus runs in time polynomial in the number of agents and roles. This is in contrast to the communicating case where each agent only has to compute $O(|M|)$ potentials but in total $O(|M|n)$ potentials are communicated.

After the role assignment, the optimal joint action is computed. We can use the VE algorithm without using communication if we assume the value rules of an agent i are common knowledge among all agents that are *reachable* from i. Since only connected agents need to coordinate their actions, this frees them from communicating their local value rules during optimization. Furthermore, in order to condition on the current state and construct the CG corresponding to the current situation each agent also has to observe the state variables of the agents located in its subgraph.

In order to perform the VE algorithm, agent i starts with eliminating itself and keeps removing agents until it computes its own optimal action unconditionally on the actions of the others. In the worst case, agent i needs to eliminate all agents $j \neq i$, for j reachable from i. Each agent thus runs the complete algorithm by itself in order

to determine its own action. The main difference with the communicating case occurs in the reverse pass. When multiple best-response actions contribute the same local value to the global payoff, the eliminated agent can randomly choose one of these actions in the communication case. When every agent models the complete algorithm individually, we have to ensure that the different agents select the same action. This can be accomplished by imposing the additional constraint that the ordering in the actions sets of the agents is common knowledge.

In the non-communicative case the elimination order neither has to be fixed in advance nor has to be common knowledge among all agents as in [Guestrin et al., 2002c]. Each agent is free to choose *any* elimination order because a particular elimination order affects only the speed of the algorithm and not the computed joint action.

In terms of complexity, the computational costs for each individual agent are clearly increased to compensate for the unavailable communication because, in the worst case, each agent has to calculate the action of every other agent in the subgraph, instead of only optimizing its own action. The computational cost per agent increases linearly with the number of new payoff functions generated during the elimination procedure. Communication, however, is not used anymore which allows for a speedup since these extra individual computations can run in parallel. This is in contrast to the original CG approach where computations need to be performed sequentially.

To summarize, we can apply the CG framework without communication when all agents are able to run the same algorithm in parallel. For this, we require that

- the payoff functions of an agent i are common knowledge among all agents reachable from i,
- each agent i can compute the potential r_{jm} for all roles $m \in M$ and all agents j in its subgraph,
- the action ordering is common knowledge among all agents,
- each agent i observes the state variables located in the value rules of all agents reachable from agent i.

Finally, we note that the common knowledge assumption is strong and even in cases where communication is available it cannot always be guaranteed [Fagin et al., 1995]. In multiagent systems without communication common knowledge is guaranteed if all agents consistently receive the same observations of the true world state, but this is violated in practice due to partial observability of the environment, for example, a soccer player has a limited field of view. In our case, the only requirement we impose is that in a particular local context the role assignment is based on those parts of the state that are, to a good approximation, fully observable by all involved agents. In a soccer game, for example, deriving the particular role assignment only requires that the players know the position of the nearby players, and have a rough estimate of the ball position. As long as such a context is encountered, a local graph is formed which is disconnected from the rest of the CG and can be solved separately.

6.5 Experiments

In this section, we apply the aforementioned framework to our simulation robot soccer team *UvA Trilearn* [De Boer and Kok, 2002]. The main motivation is to improve upon the coordination during ball passes between teammates. First, we conduct an experiment in which a complete team strategy is specified using predefined value rules. During a game, each player selects its action based on the result of the VE algorithm. In this case we assume the world is fully observable and the agents thus do not need to communicate. Second, we incorporate this framework in our competition team for the RoboCup-2003 World Championships. Because in competition matches the world is only partially observable, we make some additional assumptions and modifications to the algorithm. Next, we describe both approaches in more detail.

6.5.1 Full observability

In this section we describe how we construct a complete team strategy for our UvA Trilearn robot simulation team using predefined role-specific value rules. The whole procedure to compute the action for an individual agent consists of three steps: the role assignment, updating the context-specific CG based on the current state, and applying the VE algorithm to compute the optimal joint action.

We specify three different roles $M = \{\text{active}, \text{receiver}, \text{passive}\}$. The first and most important role is that of the active player which performs an action with the ball. We distinguish between two different types of active players, interceptor and passer, depending whether the ball can be kicked by the active player or not. Furthermore, we have the role of receiver, a player who is located at a good position to receive the ball, and the passive role, which indicates that the player does not actively take part in the current situation. We specify the following role sequence:

$$M' = \{\text{active}, \text{receiver}, \text{receiver}, \text{passive}, \text{passive}, \\ \text{passive}, \text{passive}, \text{passive}, \text{passive}, \text{passive}\}.$$

Remember that the roles are assigned to the agents in this order. We thus first assign the role of active player. Furthermore, we always assign two players the role of receiver. The remaining seven players are assigned the role of passive player. Note that we only assign ten roles and thus disregard the goalkeeper.

In order to determine the role assignment, each agent computes its potential for the three different roles. The agent with the highest potential for the first role in the sequence M' is assigned that role. Then the process continues by assigning the remaining roles in M' until all roles have been assigned. The corresponding potentials for each role are computed as follows:

- The potential $r_{i,active}$, which indicates how appropriate player i is for the active role, is equal to $1/t_i$ where $t_i > 0$ is the predicted time it will take player i to intercept the ball. For this, we use the modification of Newton's Method as described in [Stone and McAllester, 2001]. This method finds the least root

(equal to the first possible interception time) of the function that represents the difference between the traveled distance of the ball and the movement of player i. The specific active role, passer or interceptor, that is assigned to the agent depends on the relative distance to the ball. When the ball is close enough to be kicked, the agent is assigned the role of passer, otherwise the role of interceptor.

- The potential for the role of receiver $r_{i,receiver}$, is based on the distance $d_{i,b}$ between player i and the ball and the relative distance between player i and the opponent goal $d_{i,g}$ as follows:

$$r_{i,receiver} = \begin{cases} 1/\max(1, d_{i,g}) + 1 & \text{if } d_{i,b} < k \\ 1/\max(1, d_{i,g}) & \text{otherwise} \end{cases} \tag{6.1}$$

  This function states that there is a preference for agents that are located close to the opponent goal. Furthermore, an additional reward is given when the ball is within a range of $k = 28$ meters to player i. The specific value of 28 metes corresponds to the maximal traveled distance of the ball when it is shot with maximal velocity. Agents outside this range are thus only taken into consideration as receiver when there is no nearby alternative.

- The potential $r_{i,passive}$ for the role of passive player is a constant such that all remaining agents are assigned to this role.

Given the role assignment, we specify the coordination dependencies between the different roles using value rules. Before we give the actual value rules, we list the relevant action and state variables. We first define the different actions, which are all directly available in the base code of the UvA Trilearn team as described in Section 6.3:

- $passTo(i, dir)$: pass the ball to a position with a fixed distance from agent i in the direction $dir \in D = \{center, n, nw, w, sw, s, se, e, ne\}$. The direction parameter specifies a direction relative to the receiving agent. 'North' is always directed toward the opponent goal and 'center' corresponds to a pass directly to the current agent position,

- $moveTo(dir)$: move in the direction $dir \in D$,

- $dribble(dir)$: move with the ball in direction $dir \in D$,

- $score$: shoot to the best spot in the opponent goal [Kok et al., 2002],

- $clearBall$: shoot the ball with maximum velocity between the opponent defenders to the opponent side,

- $moveToStratPos$: move to the agent's strategic position. This position is computed based on the agent's home position and the position of the ball which serves as an attraction point [De Boer and Kok, 2002].

We also define different boolean state variables that extract important (high-level) information from the world state:

- *is-pass-blocked*(i, j, dir) indicates whether a pass from agent i to agent j is blocked by an opponent or not. The actual position to which is passed is the position at a small fixed distance from agent j in direction dir. A pass is blocked when there is at least one opponent located within a cone from the passing player to this position.

- *is-empty-space*(i, dir) indicates that there are no opponents within a small circle in the specified direction dir of agent i.

- *is-in-front-of-goal*(i) returns whether agent i is located before the opponent goal.

We can now define the complete strategy of our team by means of value rules which specify the contribution to the global payoff in a specific context. Fig. 6.5 shows all value rules. The rules are specified for each player i and make use of the above defined actions and state variables. Note that we enumerate all rules using variables. The complete list of value rules is the combination of all possible instantiations of these variables. In all rules, $dir \in D$.

The first five rules are related to the action options for the active player. The first rule, p_1, indicates that intercepting the ball is the only option when performing the interceptor role. As a passer, there are several alternatives. Value rule p_2 represents an active pass to the relative direction dir of player j which can be performed when there are no opponents along that trajectory and the receiving agent will move in that direction to intercept the coming pass. The value contributed to the global payoff is returned by $u(j, dir)$ and depends on the position where the receiving agent j will receive the pass, that is, the closer to the opponent goal the better. The next three rules indicate the other individual options for the active player: dribbling (we only allow forward dribbling), clearing the ball, and scoring. Rule p_6 indicates the situation in which a receiver already moves to the position it expects the current interceptor to pass the ball to when it reaches the ball. Using the same principle, we can also create more advanced dependencies. For example, rule p_7 indicates that a receiver can already move to a position it will expect the receiver of another pass to shoot the ball to. Rule p_8 describes the situation in which a receiving player moves to its strategic position. This action is only executed when it is not able to coordinate with one of the other agents, since it has a small payoff value. Finally, rule p_9 contains the single action option for a passive player that always moves to its strategic position.

When the nine basic rules are instantiated, there are 204 value rules in total. We illustrate that even with such a rather small set of rules a complete, although simple, team strategy can be specified that makes explicit use of coordination. Furthermore, the rules are easily interpretable which makes it possible to add prior knowledge into the problem. Another advantage is that the rules are very flexible: existing rules can directly be added or removed. This makes it possible to change the complete strategy of the team when playing different kinds of opponents.

$$
\begin{array}{lcl}
\langle p_1^{interc.} & ; & \text{intercept} : 10\rangle \\
\langle p_2^{passer} & ; & \text{has-role-receiver}(j) \quad \wedge \\
 & & \neg\text{isPassBlocked}(i, j, dir) \quad \wedge \\
 & & a_i = \text{passTo}(j, dir) \quad \wedge \\
 & & a_j = \text{moveTo}(dir) : u(j, dir) \in [5, 7]\rangle \ \forall j \neq i \\
\langle p_3^{passer} & ; & \text{is-empty-space}(i, \text{n}) \quad \wedge \\
 & & a_i = \text{dribble}(\text{n}) : 2\rangle \\
\langle p_4^{passer} & ; & a_i = \text{clearBall} : 0.1\rangle \\
\langle p_5^{passer} & ; & \text{is-in-front-of-goal}(i) \quad \wedge \\
 & & \text{is-ball-kickable}(i) \quad \wedge \\
 & & a_i = \text{score} : 10\rangle \\
\langle p_6^{receiver} & ; & \text{has-role-interceptor}(j) \quad \wedge \\
 & & \neg\text{isPassBlocked}(j, i, dir) \quad \wedge \\
 & & a_j = \text{intercept} \quad \wedge \\
 & & a_i = \text{moveTo}(dir) : u(i, dir) \in [5, 7]\rangle \quad \forall j \neq i \\
\langle p_7^{receiver} & ; & \text{has-role-receiver}(k) \quad \wedge \\
 & & \neg\text{isPassBlocked}(k, i, dir) \quad \wedge \\
 & & a_j = \text{passTo}(k, dir2) \quad \wedge \\
 & & a_k = \text{moveTo}(dir2) \quad \wedge \\
 & & a_i = \text{moveTo}(dir) : u(i, dir) \in [5, 7]\rangle \ \forall j, k \neq i \\
\langle p_8^{receiver} & ; & \text{moveToStratPos} : 1\rangle \\
\langle p_9^{passive} & ; & \text{moveToStratPos} : 1\rangle
\end{array}
$$

Figure 6.5: A complete team strategy specified using value rules.

We will now look into an example of how the above rules are used in practice. We assume the world is fully observable (this is a configuration setting in the soccer server) and each agent can thus model the complete CG algorithm separately. This is necessary since the RoboCup soccer simulation does not allow agents to communicate with more than one agent at the same time, which makes it impossible to apply the original VE algorithm. This has no effect on the outcome of the algorithm.

For a given situation, the roles are assigned first. Then the agents condition on the state variables and determine the applicable rules. We now consider the example configuration depicted in Fig. 6.6. Agent 1 has control of the ball and is assigned the passer role. Agent 2 and agent 3 are the two receivers and all other players are passive. This assignment of roles defines the structure of the coordination graph. Note that the construction of the CG is performed after every new observation, and the graph structure thus changes dynamically as the state of the world changes. By construction an agent in a passive role always performs the same individual action, namely, moving towards its strategic position that is based on the current position of the ball [Reis et al., 2001; De Boer and Kok, 2002]. This drastically simplifies the coordination graph since there are no dependencies between the passive agents. Furthermore, we assume only the state variables $\neg\text{isPassBlocked}(1, 2, \text{s})$ and $\neg\text{isPassBlocked}(2, 3, \text{nw})$ are true. This corresponds to the following set of value rules that are applicable in this situation:

$$
\begin{aligned}
A_1 : \langle p_2^{passer} \quad &; \quad a_1 = \text{passTo}(2, \text{s}) \quad \wedge \\
&\quad\; a_2 = \text{moveTo}(\text{s}) : 6\rangle, \\
\langle p_3^{passer} \quad &; \quad a_1 = \text{dribble}(\text{n}) : 2\rangle, \\
\langle p_4^{passer} \quad &; \quad a_1 = \text{clearBall} : 0.1\rangle, \\
A_2 : \langle p_8^{receiver} \quad &; \quad a_2 = \text{moveToStratPos} : 1\rangle, \\
A_3 : \langle p_7^{receiver} \quad &; \quad a_1 = \text{passTo}(2, dir) \quad \wedge \\
&\quad\; a_2 = \text{moveTo}(dir) \quad \wedge \\
&\quad\; a_3 = \text{moveTo}(\text{nw}) : 5\rangle, \ \forall dir \in D, \\
\langle p_8^{receiver} \quad &; \quad a_3 = \text{moveToStratPos} : 1\rangle.
\end{aligned}
$$

In order to compute the joint action that maximizes the applicable rules' values, we apply the rule-based VE algorithm. We do not apply the max-plus algorithm (Chapter 3) because it is not able to directly cope with the value-rule representation. The max-plus algorithm operates by sending messages that specify the value for each possible action of an agent, and therefore requires an enumeration over all possible actions. The rule-based VE method on the other hand communicates conditional strategies which only contain the relevant actions, represented as value rules. Another reason to use the VE algorithm is that, as shown in Section 3.4, the max-plus algorithm only outperforms the VE algorithm for large agent networks with many dependencies. In this problem, at most three agents are involved.

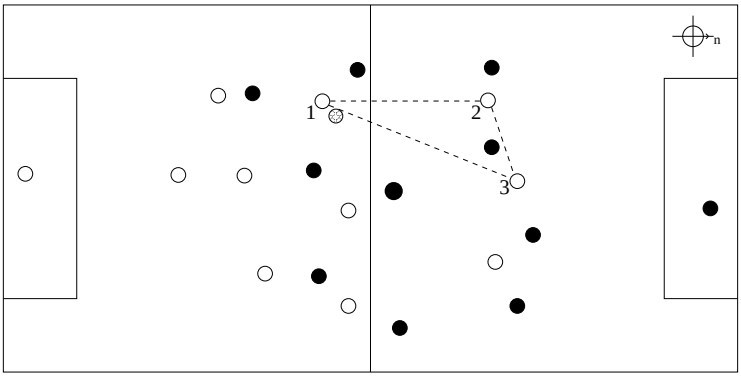


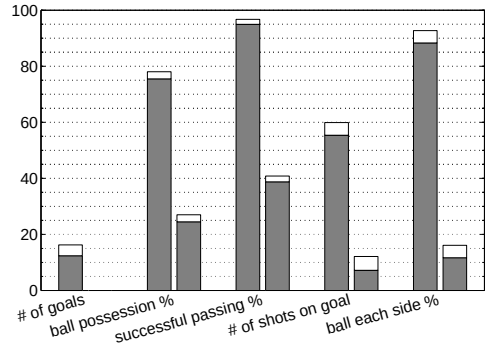
Figure 6.6: A situation involving one passer and two receivers. All other agents are passive.

In the VE algorithm, each agent is eliminated from the graph by maximizing its local payoff. In the case that agent 1 is eliminated first, it gathers all value rules that contain a_1, maximizes its local payoff and distributes its conditional strategy to its neighbors. This conditional strategy corresponds to the following set of value rules:

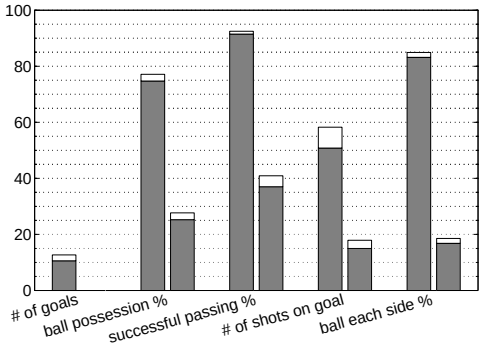
$$\begin{aligned}
\langle p_{10}^{passer} \quad &; \quad a_2 = \text{moveTo(s)} \quad \wedge \\
&\quad\; a_3 = \text{moveTo(nw)} : 11\rangle, \\
\langle p_{11}^{passer} \quad &; \quad a_2 = \text{moveTo(s)} \quad \wedge \\
&\quad\; a_3 = \neg\text{moveTo(nw)} : 6\rangle, \\
\langle p_{12}^{passer} \quad &; \quad a_2 = \neg\text{moveTo(s)} : 2\rangle.
\end{aligned}$$

Note that p_{10}^{passer} is formed by combining p_2^{passer} and $p_7^{receiver}$ when both agent 2 and 3 fulfill the listed actions. When agent 3 performs a different action, the payoff is still 6 when agent 2 moves south as is stated in p_{11}^{passer}. When agent 2 also performs a different action, the only remaining action is the dribble with a payoff of 2. After agent 2 and 3 have also fixed their strategy, agent 1 will perform passTo$(2, \text{s})$, agent 2 will execute moveTo(s) to intercept the pass and agent 3 will perform moveTo(nw) to intercept a possible future pass of agent 2. During a match, this procedure is executed after each update of the state and the agents will change their action based on the new information. If, for some unpredicted reason, the first pass fails in this example, the graph will automatically be updated and correspond to the new situation.

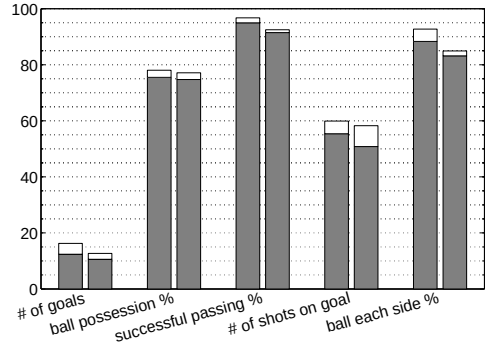
In order to test our approach, we play games using the released basic client implementation of our UvA Trilearn team, as described in Section 6.3). We experiment with three different versions. All three versions use an identical implementation of the low-level behaviors, for example, the kick and the intercept, and thus only differ with respect to their high-level strategy. The first version is identical to the released implementation and is used as a benchmarking version. In this version the active player always intercepts the ball and immediately kicks it with maximal velocity to



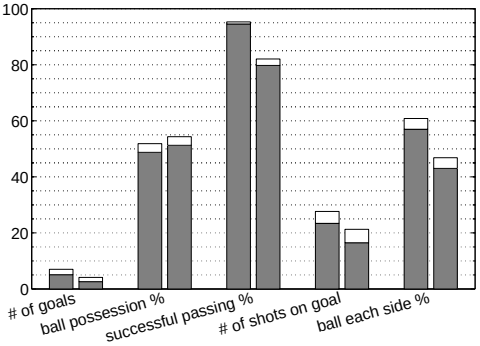
(a) Coordinated team vs. benchmark team.



(b) Non-coordinated vs. benchmark team.



(c) Non-coordinated and coordinated team vs. benchmark team.



(d) Coordinated vs. non-coordinated team.

Figure 6.7: Mean and standard deviation of several statistics for the three tested teams. All results are averaged over 10 matches.

a random corner of the opponent goal. The second version uses explicit coordination using the value rules depicted in Fig. 6.5. The third and final version does not use explicit coordination during passing. This is modeled by deleting the rules p_6, p_7 from the list of value rules and removing the condition $a_j = \text{moveTo}(dir)$ from rule p_2. A receiver thus does not explicitly anticipate a pass. Now, in the non-coordinating case a teammate moves to the interception point only after it observes a change in the ball velocity, that is, after someone has passed the ball, and it is assigned the role of interceptor. Before the ball changes velocity, it has no notion of the fact that it will receive the ball and does not coordinate with the passing player.

Since many different factors contribute to the overall performance of the team, it is difficult to measure the actual effect of the coordination with one single value. Therefore, we generate multiple statistics using the *statistics proxy server* tool [Frank et al., 2001]. Fig. 6.7(a) and Fig. 6.7(b) show the game statistics for the coordinating and

stage	comp. time (ms)	nr. of samples
Initialization	7.15 (4.45)	100
Role assignment	0.16 (0.24)	602,865
CG Role Passive	0.01 (0.002)	472,108
CG Role Interceptor	0.01 (0.002)	45,950
CG Role Receiver		
Condition step	3.94 (4.74)	72,307
Elimination step	34.87 (72.15)	72,307
Nr of applicable rules	25.69 (29.34)	72,307
CG Role Active		
Condition step	4.15 (4.61)	12,500
Elimination step	48.93 (96.83)	12,500
Nr. of applicable rules	27.69 (30.65)	12,500

Table 6.1: Average computation times (in ms) with the standard deviation (between brackets), for the different stages of the algorithm. Results are generated on an AMD Athlon 1.5GHz/512MB computer. The results are combined for all players' actions over the course of 10 full-length games and are averaged over all players.

the non-coordinating team against the benchmark team averaged over 10 full-length games. These results show that both teams are able to defeat the benchmark team with considerable goal difference on all occasions (respectively $12.4 - 0$ and $10.6 - 0$). In Fig. 6.7(c) these statistics are directly compared with each other indicating that the coordinating team slightly outperforms the non-coordinating team. Fig. 6.7(d) shows the same statistics in the case the coordinating team plays against the non-coordinating team. In this setting, the coordinating team won 8 out of the 10 matches, drew one, and lost one. The average score was $5.2 - 2.6$. Almost all statistics show a performance improvement for the coordinating team. For example, the successful passing percentage was 94.55% for the team with the coordinated value rules and 79.76% for the team without. These percentages indicate that due to the better coordination of the teammates, fewer mistakes were made when the ball was passed between teammates. This also has a positive effect on the other statistics, for example, number of shots on goal and the location of the ball on the field.

Table 6.1 shows the timing results for the different stages of the framework during the matches of the coordinating team against the non-coordinating team. For the non-coordinating team, the time to determine an action was approximately 3 ms in total for both the receiver and the passer on an AMD Athlon 1.5GhZ/512MB machine. The coordinating team requires more computation time. This is mainly caused by the computation performed during the VE algorithm. On average the time needed

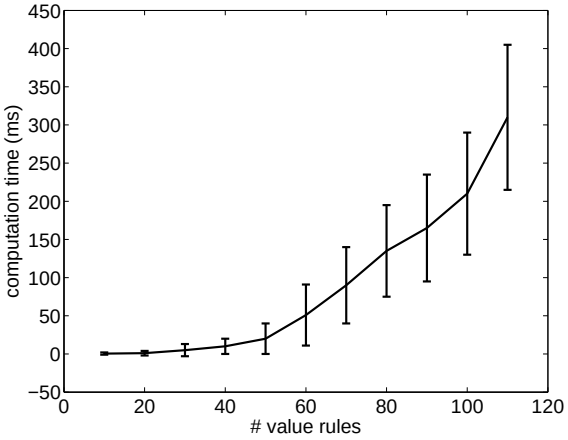


Figure 6.8: The computation time (ms) needed to determine an action given the number of value rules that are applicable in the current context.

for determining an action was 25.69 ms for the receivers. The computation time is strongly related to the number of applicable value rules for a specific situation. On average approximately 25 value rules were applicable after conditioning on the context. However, situations occurred in which considerable more value rules were applicable. In these cases, the computation time also increased considerably. Fig. 6.8 shows the relationship between the number of value rules and the computation time. As the number of applicable value rules increases, the best-response function has to take more (action) combinations of value rules into account, slowing down the computation. Note that the use of the max-plus algorithm would not have a positive effect because, as explained earlier, it cannot take advantage of the rule-based representation and always has to enumerate all possible action combinations.

6.5.2 Partial observability

We applied the framework described in the previous section to specify the high-level strategy in our UvA Trilearn 2003 team that participated, among others, in the RoboCup-2003 competition. In order to participate in the competition, we had to adapt the framework for (i) the partially observability of the domain and (ii) the real-time requirements of the simulator (100 ms per cycle).

The common knowledge assumption about the fully observable state cannot be made during competition matches since every agent only receives information of the part of the field to which its neck is oriented. However, the structure of the CG after the role assignment specifies which parts of the state are relevant for coordination, that is, the neighbors in the graph and their associated state variables. Therefore, we adjust the looking mechanism of the agents to actively orient their neck to the part of the field in which its neighbors in the graph are located and then assume that

tournament	place	goals	nr. of teams
German Open	1st	$136 - 0$	12
American Open	1st	$100 - 0$	15
RoboCup-2003	1st	$177 - 7$	46

Table 6.2: Tournament results UvA Trilearn 2003.

this part of the world is common knowledge among these agents. When all involved agents observe this information they can independently solve the local graph which is disconnected from the rest of the CG and so compensate for the missing state information. In our example, the passer and the receivers thus change their looking direction to their neighbors in the graph in order to get a good approximation of the relevant part of the state needed for coordination, and are not interested in the passive players which are not connected to its subgraph.

In order to comply with the real-time complexities of the simulator, the timing results of the previous section had to be improved. On average a single player has approximately 10 ms in order to determine its action using our synchronization scheme [De Boer and Kok, 2002]. Therefore, we included an additional preprocessing step during the conditioning in which for each of the nine basic rules and for each possible receiver only the value rule was kept that gave the highest reward. For example, a pass in the northern direction to a receiver has always precedence over a pass in southern direction and therefore we can remove the value rule related to the southern pass. This has no influence because these coordinated passes are independent of the actions of the other agents. This reduces the number of value rules and makes sure that the agents are able to keep their computation time within the given time constraints.

Finally, in order to improve the actual performed pass, we did not directly map the returned high-level actions from the CG algorithm to a primary action. Instead, given the returned receiver and direction, we evaluated multiple passes for different angles and different speeds in that direction using the modification of Newton's method [Stone and McAllester, 2001] and selected the action that maximized the capture time between the receiver and the fastest opponent. This procedure can be regarded as a two-layered hierarchy in which a high-level action based on the global coordination situation is refined to a specific action which takes the local situation into account.

We applied this approach to our UvA Trilearn 2003 team. Using this approach, UvA Trilearn was able to win all three tournaments in 2003 in which it participated. This included the RoboCup-2003 World Championships in Padova, Italy, for which 46 teams had qualified. In total 16 matches were played in this competition, resulting in a total goal difference of $177 - 7$. In the final UvA Trilearn defeated TsinghuAeolus, the winner of 2001 and 2002, with a score of $4 - 3$. The successful passing percentage in the final was 91.43% for UvA Trilearn against 82.87% for TsinghuAeolus. Table 6.2 shows all tournament results of UvA Trilearn 2003.

6.6 Discussion

In this chapter, we proposed two extensions to the framework of context-specific coordination graphs (context-specific CGs) [Guestrin et al., 2002c] for the situation that the agents are embedded in a continuous domain and communication is unavailable. First, we assigned roles to the agents in order to abstract from the continuous state to a discrete context, allowing the application of existing techniques for discrete-state CGs. The notion of roles we use is similar to the ones described in [Spaan et al., 2002; Iocchi et al., 2003; Stone and Veloso, 1999; Vlassis, 2003]. In these settings, the agents have knowledge about different roles which specify an agent's internal and external behaviors. Agents can at any time switch between the different roles based on external events or after negotiation. In these cases, coordination is the result of the different agents performing subtasks corresponding to their assigned role. In our case, the role assignment also defines the coordination structure, and the CG framework is used to coordinate the individual actions of the agents. This approach is based on value rules that specify the kind of coordination for a specific context and is very flexible, since existing rules can directly be added or removed. This makes it possible to change the complete strategy of the team when playing different opponents.

Furthermore, we showed that we can dispense with communication if additional assumptions about common knowledge are introduced. This makes it possible to model the reasoning process of the other agents, making communication unnecessary. This bears resemblance to the concept of a locker-room agreement [Stone and Veloso, 1999] which facilitates coordination with little or no communication. It provides a mechanism for predefining multiagent protocols that are accessible to the whole team. As a result the agents act autonomously during execution, while still working towards a common goal that is specified beforehand. This approach is similar to our common knowledge assumptions about the value rules of the reachable agents in the graph which make communication superfluous. Applying our coordination framework to our UvA Trilearn robot soccer simulation team resulted in improved coordinated behavior of the agents and in three RoboCup tournament wins in 2003.

7

Conclusions

Decision making in cooperative multiagent systems is an important topic since many large-scale applications are formulated in terms of spatially or functionally distributed entities, or agents. Collaboration enables the different entities to work more efficiently and to complete activities they are not able to accomplish individually. However, in order to collaborate the agents should (learn to) coordinate their actions. This is a complicated process because the number of action combinations grows exponentially with an increase of the number of agents, and each agent takes individual decisions of which the outcome can be influenced by the actions performed by the other agents.

This thesis contributes several techniques to coordinate and learn the behavior of the agents in cooperative multiagent systems. This final chapter presents several concluding remarks on the work described in this thesis and highlights its contributions. Furthermore, it discusses several promising directions for future research.

7.1 Conclusions and contributions

This thesis studied both the problem of coordinating the behavior of multiple agents in a specific situation, and learning the behavior of a group of agents in sequential decision-making problems using model-free reinforcement-learning techniques. One of the main approaches in all presented methods is to simplify the coordination problem by exploiting the actual dependencies that exist between the agents. These dependencies are modeled using the coordination graph (CG) framework [Guestrin et al., 2002a]. This framework decomposes a global payoff function into a sum of local payoff functions. Each local payoff function depends on the actions of a subset of the agents and specifies a contribution to the system for every possible action combination of the involved agents. A CG can be depicted using a graph in which the nodes represent the agents and the edges indicate a coordination dependency. A coordination dependency is added between all agents that are involved in the same local payoff function.

In Chapter 3, we addressed the problem of coordinating the actions of a large group of agents in a specific situation. We assumed the coordination dependencies with the corresponding payoff functions between the agents were given and modeled as a CG. Our contribution is the *max-plus* algorithm that can be used as an alternative to variable elimination (VE) for finding the optimal joint action. VE is an exact method that always reports the joint action that maximizes the global payoff, but is slow for

densely connected graphs with cycles as its worst-case complexity is exponential in the number of agents. The max-plus algorithm, analogous to the belief propagation algorithm in Bayesian networks, operates by repeatedly sending local payoff messages over the edges in the CG. By performing a local computation based on its incoming messages, each agent is able to select an individual action. For large, highly connected graphs with cycles, we showed that max-plus finds good solutions exponentially faster than VE. Our anytime extension occasionally evaluates the current joint action and stores the best one found so far. This ensures that the agents select a coordinated joint action and essentially produces a convergent max-plus variant. The max-plus algorithm can be implemented fully distributed and, contrary to VE, only requires that each agent communicates with its neighbors in the original CG.

In Chapter 4, we investigated the problem of learning the coordinated behavior of the agents in sequential decision-making problems. The latter were modeled using a collaborative multiagent Markov decision process (collaborative MMDP) [Guestrin, 2003]. Our contribution is a family of model-free reinforcement-learning variants, called *sparse cooperative Q-learning (SparseQ)*, which approximate the global Q-function using the structure of a given CG. We analyzed both a decomposition in terms of the nodes, as well as one in terms of the edges of the graph. During learning, each local Q-function is updated based on its contribution to the maximal global action value found with either the VE or max-plus algorithm. All methods can be implemented fully distributed as long as each agent is able to communicate with its neighbors in the graph. Results on both a stateless problem with 12 agents and more than 17 million actions, and a distributed sensor network problem indicated that our SparseQ variants outperform other existing multiagent Q-learning methods.

In Chapter 5, we studied solution methods for multiagent sequential decision-making problems that are able to learn based on changing coordination structures. Our contribution is *context-specific sparse cooperative Q-learning (context-specific SparseQ)*, an extension of SparseQ, which approximates the global Q-function using the structure of a given context-specific CG [Guestrin et al., 2002c]. Such a graph specifies the coordination dependencies of the system for a specific context using value rules. Each rule consists of an arbitrary subset of state and action variables, and a value which is added to the system when the state and action variables apply to the current situation. Again, we investigated both an agent-based and edge-based decomposition of the global action value. In both cases, the value of each rule contributes additively to the global action value, and is updated based on a Q-learning update rule that adds the local contribution of all involved agents in the rule. Effectively, each agent learns to coordinate only with its neighbors in a dynamically changing coordination graph. Another contribution from this chapter is our *utile coordination* algorithm, a method which starts with independent, non-coordinating, agents and learns the structure of a context-specific CG automatically based on statistics on expected returns for hypothesized coordinated states. Using the value-rule representation, a coordinated dependency can easily be constructed by adding value rules which incorporate the actions of the other agent. Results in the pursuit domain showed that context-specific SparseQ improved the learning time with respect to other multiagent

Q-learning methods, and performed close to optimal when manually defining the coordination dependencies of the system. We also showed that our utile coordination approach resulted in a similar policy based on a learned set of rules that is smaller than our manually specified set.

In Chapter 6, we applied context-specific CGs to coordinate the agents in dynamic and continuous domains. The continuous nature of the state-action space complicates the direct application of context-specific CGs. Our approach is to assign roles to the agents in order to convert the continuous state to a discrete context, allowing the application of existing techniques for discrete-state CGs. This simplifies the coordination structure and constrains the action space of the agents considerably. Furthermore, it allows for the definition of natural coordination rules that exploit prior knowledge about the domain. Finally, we showed that we can dispense with communication if additional assumptions about common knowledge are introduced. This makes it possible to model the reasoning process of the other agents, making communication unnecessary. Applying the resulting coordination framework to our UvA Trilearn robotic soccer simulation team resulted in improved coordinated behavior of the agents and in three RoboCup tournament wins in 2003, including the RoboCup-2003 World Championships in Padova, Italy.

7.2 Future work

In this section we discuss several interesting directions for future research. We both present alternative approaches and extensions to our presented methods.

We applied the max-plus algorithm, analogous to the belief propagation algorithm in Bayesian networks, to coordinate the actions of the agents in a cooperative multiagent system. However, extensions of the belief propagation algorithm for graphs with cycles exist that often have convergence properties superior to those of parallel message-passing updates [Wainwright et al., 2004]. For example, Wainwright et al. [2003] present a tree-based reparameterization framework that combines the exact solutions of different cycle-free subgraphs. Furthermore, several other approximation alternatives exist in the Bayesian network literature that could also be applied. One natural alternative is to apply the 'mini-bucket' approach, an approximation in which the VE algorithm is simplified by changing the full maximization for each elimination of an agent to the summation of simpler local maximizations [Dechter and Rish, 1997]. Another possible direction is to model the dependencies between the agents using a factor graph representation [Kschischang et al., 2001], which allows more prior knowledge about the problem to be introduced beforehand. It would be interesting to apply these different approaches to the problem of coordinating the actions of the agents, and compare the results.

Another possible extension to the max-plus algorithm is to investigate a variant that takes advantage of the value-rule representation of a context-specific CG. Currently, the max-plus algorithm always sends messages that specify the value for each possible action of the receiving agent. The rule-based VE method on the other hand

communicates conditional strategies, represented as value rules, which only contain the values for non-zero and dominating actions (based on all possible action combinations of its neighbors). When each agent can perform many actions, it might be worthwhile to investigate a combination of the two methods, and communicate messages, which are based on the action of the receiving agent, using a value-rule representation in the max-plus algorithm. Using such an approach, it might be possible to coordinate larger groups of agents and add more specialized fine-grained coordination. However, due to the complexity involved with the management of the rules, the smaller number of rules will only outweigh the computational advantages of the table-based messages in problems with a large amount of context-specific structure.

We investigated several decompositions of the action value based on the topology of a CG in our different SparseQ methods. The topology can be chosen arbitrarily, but should resemble the dependencies of the problem under study in order to produce relevant results. Given a CG structure, a choice has to be made whether to use an agent-based or edge-based decomposition. The agent-based decomposition gives better performance, but its space and computational complexity scale exponentially with the number of dependencies, resulting in exploration difficulties for large problems with many dependencies. In this case, an edge-based decomposition can be used which only stores action values based on pairwise dependencies. For this decomposition, we investigated both an edge-based and agent-based method to update the Q-values after an experienced state transition, but our experimental results only show minor differences between these two update methods. An interesting avenue for future work is to consider the consequences of all these different choices, and identify the most appropriate method for a given problem definition.

Another interesting issue is related to the Q-updates of the edge-based decomposition. In each update, we now divide the reward proportionally over its edges (see (4.8) and (4.11)), and thus assume that all agents contribute equally to the dependencies in which they are involved. However, other schemes are also be possible, for example, dividing the reward based on the current value of the Q-values or on distributions of previous received rewards.

With respect to the context-specific SparseQ methods, it would be interesting to investigate the robustness of these methods when new agents are added or existing agents are removed from the system. Since all Q-functions and updates are defined locally, it is possible to compensate for the addition or removal of an agent by redefining the value rules in which this agent is involved. The algorithm to compute the best joint action and the local updates do not have to be changed as long as the neighboring agents are aware of the new topology of the CG.

In our utile coordination approach, we focused on the problem of extending the action space. Another interesting direction is to investigate the possibility of decreasing the action space by removing dependencies which are unnecessary according to the gathered statistics. In this case, we also have to store statistics for coordinated states based on hypothesized uncoordinated states and test whether the expected return is not significantly lower when the actions are not explicitly coordinated. Furthermore, the coordination dependencies between the agents were learned based on the full state

information. For large state spaces, this might not always be possible. An interesting approach is a variation of the utile coordination algorithm in which also the state variables that are important for coordination are learned. This combines well with the structure of a coordination graph, because also state variables can be added and removed from the value rules. An individual agent then starts with rules that only represent its own individual view of the environmental state, and over time learns how to extend the state representation for the situations which require coordination. This approach is similar to the work of McCallum [1997] in which a partitioning of the state space is constructed for partially observable Markov decision processes.

Apart from the aforementioned extensions to our methods, there are also other directions for future research. One is to consider partially observable models which assume the agents only have limited information about the environment [Lovejoy, 1991; Kaelbling et al., 1998]. However, this complicates the problem significantly because the agents have to find a mapping from a observation histories to actions, instead of discrete states to actions. Even for single-agent problems that have access to the model description exact solution methods are only applicable to the smallest problems [Sondik, 1971; Cheng, 1988]. Approximate solution techniques exist [Hauskrecht, 2000; Spaan and Vlassis, 2005] but are difficult to extend to multiple agents.

Finally, an interesting research direction is to apply our SparseQ methods to learn the behaviors of the agents in continuous domains. Although different function-approximation techniques exist for continuous state representations [Albus, 1971; Sutton and Barto, 1998], large action sets have been explored only to a limited extent [Santamaria et al., 1998; Sherstov and Stone, 2005]. One of the major problems is to generalize over the action space because it is more difficult to construct an appropriate distance measure for discrete joint actions than for continuous state variables. Furthermore, a large action space requires many exploration actions which slows down learning. Our approach is able to decompose the action space into a set of smaller problems which decreases the number of samples required for learning the policies of the agents because updates are essentially performed in parallel. However, learning a coordinated policy for systems with many (discretized) states, or systems with many dependencies might still require a large number of samples.

Summary

Many large-scale applications are formulated in terms of spatially or functionally distributed entities, also called agents. Examples include robotic teams, but also distributed software applications. In such systems, the agents autonomously have to take rational actions, based on incoming information from their environment, in order to accomplish a certain goal. Collaboration enables the agents to work more efficiently and to complete activities they are not able to accomplish individually. However, in order to collaborate the agents should (learn to) coordinate their actions. This is a complicated process because the total number of action combinations grows exponentially with the increase of the number of agents. Furthermore, the outcome of the individual decision of an agent can be influenced by the actions performed by the other agents.

This thesis presented several techniques to coordinate and learn the behavior of the agents in distributed cooperative multiagent systems. It both studied the problem of coordinating the behavior of multiple agents in a specific situation, and learning, based on experience, the behavior of a group of agents in sequential decision-making problems. The latter are problems in which the agents repeatedly interact with their environment and have to perform a sequence of actions in order to reach a certain goal. Our main approach in all presented methods is to simplify the coordination problem by exploiting the actual dependencies that exist between the agents using a coordination graph (CG). Simply stated, this decomposes the global problem into a combination of simpler problems.

In Chapter 3, we addressed the problem of coordinating the actions of a large group of agents in a specific situation for which the coordination dependencies were given. We presented the *max-plus* algorithm that operates by exchanging locally optimized messages over the edges of the CG. By performing a local computation based on its incoming messages, each agent is able to select an individual action that is coordinated with the action choices of the other agents. For large, highly connected graphs we empirically demonstrated that max-plus can find good solutions exponentially faster than the variable elimination algorithm, an existing exact method.

In Chapter 4, we investigated the problem of learning the coordinated behavior of the agents in sequential decision-making problems. We presented a family of model-free reinforcement-learning variants, called *sparse cooperative Q-learning (SparseQ)*. These methods approximate the global Q-function, representing the expected outcome for a specific action in a certain situation, using the structure of a given CG. We

analyzed both a decomposition in terms of the nodes, as well as one in terms of the edges of the graph. During learning, each local Q-function is updated based on its local contribution to the maximal global action value found with either the variable elimination or the max-plus algorithm.

In Chapter 5, we studied solution methods for multiagent sequential decision-making problems that are able to take advantage of changing coordination structures. We presented *context-specific sparse cooperative Q-learning (context-specific SparseQ)*, an extension of SparseQ, which approximates the global Q-function using the structure of a given context-specific CG. A context-specific CG specifies the coordination requirements of the system for a specific context using value rules. Each rule consists of an arbitrary subset of state and action variables, and a value which is contributed to the system when the state and action variables apply to the current situation. Furthermore, we presented our *utile coordination* algorithm, a method which starts with independent, non-coordinating, agents and learns the structure of a context-specific CG automatically based on statistics on expected returns for hypothesized coordinated states.

In Chapter 6, we applied context-specific CGs to coordinate the agents in dynamic and continuous domains. Our approach is to assign roles to the agents in order to convert the continuous state to a discrete context, allowing the application of existing techniques for discrete-state CGs. This simplifies the coordination structure and constrains the action space of the agents considerably. Furthermore, it allows for the definition of natural coordination rules that exploit prior knowledge about the domain. Finally, we showed that we can dispense with communication if additional assumptions about common knowledge are introduced. Applying the resulting coordination framework to our UvA Trilearn robotic soccer simulation team resulted in improved coordinated behavior of the agents and in three RoboCup tournament wins in 2003, including the RoboCup-2003 World Championships in Padova, Italy.

Samenvatting[1]

Veel grootschalige applicaties worden gedefinieerd als een combinatie van kleinere componenten en hun interactie. Deze componenten worden agenten genoemd en zijn vaak ruimtelijk of functioneel gedistribueerd. Mogelijke voorbeelden zijn een team van robots en gedistribueerde softwareapplicaties. In zulke systemen voeren de agenten zelfstandig rationele acties uit aan de hand van de waargenomen omgeving om een bepaald doel te bereiken. Samenwerking tussen de agenten maakt het mogelijk om hun activiteiten efficiënter uit te voeren en taken te volbrengen die zij individueel niet kunnen verwezenlijken. Hiervoor moeten de agenten echter wel (leren) om hun acties op elkaar af te stemmen. Dit is een gecompliceerd probleem aangezien het totaal aantal actiecombinaties exponentieel groeit met het aantal agenten, en elke agent besluiten neemt die beïnvloed kunnen worden door de acties van de andere agenten.

In dit proefschrift zijn we geïnteresseerd in het ontwikkelen van technieken om het gedrag van de agenten op elkaar af te stemmen. We beschouwen zowel het coördinatieprobleem in één specifieke situatie als in 'sequentiële beslissingsprocessen'. De laatste zijn processen waarin de agenten herhaaldelijk hun omgeving beïnvloeden en een serie van acties moeten uitvoeren om hun doel te bereiken. Onze hoofdaanpak in alle voorgestelde methodes is om het coördinatieprobleem te vereenvoudigen door alleen te kijken naar de daadwerkelijke afhankelijkheden tussen de agenten. Dit kan weergegeven worden met een coördinatiegraaf waarin alleen de agenten die hun acties op elkaar moeten afstemmen verbonden zijn. Simpel gesteld wordt zo het globale probleem opgedeeld in een combinatie van eenvoudigere problemen.

In hoofdstuk 3 behandelden wij het probleem om de acties van een groep agenten in een specifieke situatie te coördineren wanneer de coördinatieafhankelijkheden tussen de agenten bekend zijn. Wij introduceerden het 'max-plus' algoritme waarin de agenten lokale optimalisaties uitvoeren en het resultaat naar hun buren in de coördinatiegraaf sturen. Elke agent bepaalt zijn gecoördineerde individuele actie door een lokale berekening uit te voeren op basis van de binnenkomende berichten. Wij hebben empirisch aangetoond dat voor grote, dichte grafen deze aanpak goede resultaten oplevert en exponentieel sneller is dan een bestaande optimale methode.

In hoofdstuk 4 onderzochten wij het probleem om het gedrag van de agenten in een sequentieel beslissingsproces te leren. Wij introduceerden een familie van leertechnieken waarin de agenten aan de hand van hun interactie met de omgeving automatisch

[1] Summary in Dutch.

hun gedrag leren. Onze techniek, *sparse cooperative Q-learning (SparseQ)* genoemd, benadert de global Q-functie, die het verwachte resultaat van een actiekeuze in een bepaalde toestand teruggeeft, met behulp van de structuur van een coördinatiegraaf. Wij analyseerden zowel een opdeling gebaseerd op de knopen, als de zijden van de graaf. Tijdens het leren wordt elke lokale Q-functie bijgewerkt aan de hand van zijn lokale bijdrage aan de maximale globale waarde, die gevonden kan worden met een bestaande optimale methode of ons max-plus algoritme.

In hoofdstuk 5 bestudeerden wij oplossingsmethodes voor sequentiële beslissingsprocessen waarin de agenten hun gedrag leren in het geval dat de afhankelijkheden tussen de agenten per situatie kunnen verschillen. Wij introduceerden *context-specific sparse cooperative Q-learning*, een uitbreiding van SparseQ die de globale Q-functie benadert met behulp van de structuur van een contextafhankelijke coördinatiegraaf. Deze specificeert de coördinatieafhankelijkheden voor een specifieke situatie met behulp van 'value' regels. Elke regel bestaat uit een willekeurige set van toestand- en actievariabelen, en een waarde die aan het systeem wordt bijgedragen wanneer de waardes van deze variabelen overeenkomen met de huidige situatie. Verder introduceerden wij ons *utile coordination* algoritme, een methode die met onafhankelijke opererende agenten begint en de structuur van een contextafhankelijke coördinatiegraaf automatisch leert aan de hand van verzamelde statistieken tijdens het leerproces.

In hoofdstuk 6 pasten wij contextafhankelijke coördinatiegrafen toe om de acties van de agenten in dynamische en continue domeinen te coördineren. Wij kenden elke agent een rol toe om zo de continue toestand om te zetten in een discrete toestand, zodat bestaande technieken van contextafhankelijke coördinatiegrafen toegepast kunnen worden. Deze aanpak vereenvoudigt de coördinatiestructuur en beperkt de actieruimte van de agenten aanzienlijk, en stelt ons in staat om natuurlijke coördinatieregels te definiëren die gebruik maken van bestaande domeinkennis. Tevens toonden wij aan dat er geen communicatie nodig is wanneer er extra aannames over gemeenschappelijke kennis gemaakt worden. Het toepassen van deze aanpak op ons robotvoetbal simulatieteam UvA Trilearn resulteerde in betere coördinatie van de agenten en in de winst van drie RoboCup toernooien, waaronder het RoboCup-2003 wereldkampioenschap in Padova, Italië, in 2003.

Bibliography

Albus, J. (1971). *A theory of cerebellar function.* Mathematical Biosciences, 10:25–61. Page(s): 115, 135

Ali, S. M., Koenig, S., and Tambe, M. (2005). *Preprocessing techniques for accelerating the DCOP algorithm ADOPT.* In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), pages 1041–1048. Utrecht, The Netherlands. Page(s): 72

Arai, T., Pagello, E., and Parker, L. E. (2002). *Editorial: Advances in multi-robot systems.* IEEE Transactions on Robotics and Automation, 18(5):665–661. Page(s): 55

Arnborg, S., Corneil, D. G., and Proskurowski, A. (1987). *Complexity of finding embedding in a k-tree.* SIAM Journal of Algebraic Discrete Methods, 8:277–284. Page(s): 40

Bagnell, J. A. and Ng, A. Y. (2006). *On local rewards and the scalability of distributed reinforcement learning.* In Weiss, Y., Schölkopf, B., and Platt, J., editors, Advances in Neural Information Processing Systems (NIPS) 18. MIT Press, Cambridge, MA. Page(s): 24, 56

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). *Learning to act using real-time dynamic programming.* Artificial Intelligence, 72:81–138. Page(s): 19

Becker, R., Zilberstein, S., Lesser, V., and Goldman, C. V. (2003). *Transition-independent decentralized Markov decision processes.* In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS). Melbourne, Australia. Page(s): 6, 28, 55, 64

Becker, R., Zilberstein, S., Lesser, V., and Goldman, C. V. (2004). *Solving transition independent decentralized Markov decision processes.* In Journal of Artificial Intelligence Research, volume 22, pages 423–455. Page(s): 28

Bellman, R. (1957). Dynamic programming. Princeton University Press. Page(s): 17

Benda, M., Jagannathan, V., and Dodhiawala, R. (1986). *On optimal cooperation of knowledge sources - an experimental investigation.* Technical Report BCS-G2010-280, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington. Page(s): 80, 91

Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). *The complexity of decentralized control of Markov decision processes.* Mathematics of Operations Research, 27(4):819–840. Page(s): 6, 24, 28

Bernstein, D. S., Zilberstein, S., and Immerman, N. (2000). *The complexity of decentralized control of Markov decision processes.* In Proceedings of Uncertainty in Artificial Intelligence (UAI). Stanford, CA. Page(s): 28, 56

Bertelé, U. and Brioschi, F. (1972). Nonserial dynamic programming. Academic Press. Page(s): 40

Bertsekas, D. P. (2000). Dynamic programming and optimal control. Athena Scientific, 2nd edition. Page(s): 16

Bertsekas, D. P. and Tsitsiklis, J. N. (1989). Parallel and distributed computation: Numerical methods. Prentice-Hall. Page(s): 18

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). Neuro-dynamic programming. Athena Scientific. Page(s): 5, 9, 17, 18, 56, 57

de Boer, R. and Kok, J. R. (2002). *The incremental development of a synthetic multi-agent system: the UvA Trilearn 2001 robotic soccer simulation team.* Master's thesis, University of Amsterdam, The Netherlands. Page(s): 106, 108, 111, 112, 119, 120, 123, 128

Boutilier, C. (1996). *Planning, learning and coordination in multiagent decision processes.* In Proceedings of the Conference on Theoretical Aspects of Rationality and Knowledge. Page(s): 3, 6, 21, 24, 26, 29, 30, 56

Boutilier, C., Dean, T., and Hanks, S. (1999). *Decision-theoretic planning: Structural assumptions and computational leverage.* Journal of Artificial Intelligence Research, 11:1–94. Page(s): 5, 14, 16

Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). *Stochastic dynamic programming with factored representations.* Artificial Intelligence, 121(1-2):49–107. Page(s): 81

Bowling, M. and Veloso, M. (2002). *Multiagent learning using a variable learning rate.* Artificial Intelligence, 136(8):215–250. Page(s): 30

Boyan, J. A. and Littman, M. L. (1994). *Packet routing in dynamically changing networks: A reinforcement learning approach.* In Cowan, J. D., Tesauro, G., and Alspector, J., editors, Advances in Neural Information Processing Systems (NIPS) 6, pages 671–678. Morgan Kaufmann Publishers, Inc. Page(s): 3, 55

Castelpietra, C., Iocchi, L., Nardi, D., Piaggio, M., Scalzo, A., and Sgorbissa, A. (2000). *Coordination among heterogenous robotic soccer players.* In Proceedings of the International Conference on Intelligent Robots and Systems (IROS). Takamatsu, Japan. Page(s): 115

Chaib-draa, B. and Müller, J. P. (2006). Multiagent-based supply chain management. Springer. Page(s): 3

Chalkiadakis, G. and Boutilier, C. (2003). *Coordination in multiagent reinforcement learning: A Bayesian approach.* In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), pages 709–716. ACM Press, Melbourne, Australia. Page(s): 57

Chapman, D. and Kaelbling, L. P. (1991). *Input generalization in delayed reinforcement learning: An algorithm and performance comparisons.* In Mylopoulos, J. and Reiter., R., editors, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 726–731. Morgan Kaufmann, San Mateo, Ca. Page(s): 80, 97

Chen, M., Dorer, K., Foroughi, E., Heintz, F., Huang, Z., Kapetanakis, S., Kostiadis, K., Kummeneje, J., Murray, J., Noda, I., Obst, O., Riley, P., Steffens, T., Wang, Y., and Yin, X. (2003). RoboCup soccer server users manual for soccer server version 7.07 and later. At `http://sserver.sourceforge.net/`. Page(s): 6, 105, 108

Cheng, H. T. (1988). *Algorithms for partially observable Markov decision processes.* Ph.D. thesis, University of British Columbia. Page(s): 135

Claus, C. and Boutilier, C. (1998). *The dynamics of reinforcement learning in cooperative multiagent systems.* In Proceedings of the National Conference on Artificial Intelligence (AAAI). Madison, WI. Page(s): 21, 31

Clement, B. J. and Barrett, A. C. (2003). *Continual Coordination through Shared Activities.* In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS). ACM Press, Melbourne, Australia. Page(s): 3

Collins, M. and Aldrin, E. E., Jr. (1975). Apollo expeditions to the moon. NASA SP. Page(s): 106

Crick, C. and Pfeffer, A. (2003). *Loopy belief propagation as a basis for communication in sensor networks.* In Proceedings of Uncertainty in Artificial Intelligence (UAI). Page(s): 43

Crites, R. and Barto, A. (1996). *Improving elevator performance using reinforcement learning.* In Advances in Neural Information Processing Systems (NIPS) 8, pages 1017–1023. MIT Press. Page(s): 56

Dean, T. and Boddy, M. (1988). *An analysis of time-dependent planning.* In Proceedings of the National Conference on Artificial Intelligence (AAAI). Morgan Kaufmann. Page(s): 44

Dean, T. and Kanazawa, K. (1989). *A model for reasoning about persistence and causation.* Journal of Computational Intelligence, 5(3):142–150. Page(s): 14, 25

Dechter, R. (2003). Constraint Processing. Morgan Kaufmann. Page(s): 36

Dechter, R. and Rish, I. (1997). *A scheme for approximating probabilistic inference.* In Proceedings of Uncertainty in Artificial Intelligence (UAI), pages 132–141. Page(s): 133

Durfee, E. H. (2001). Distributed problem solving and planning. Springer-Verlag, New York, NY, USA. ISBN 3-540-42312-5. Page(s): 2

Dutta, P. S., Jennings, N. R., and Moreau, L. (2005). *Cooperative information sharing to improve distributed learning in multi-agent systems*. Journal of Artificial Intelligence Research, 24:407–463. Page(s): 3, 55

Dynkin, E. B. (1965). *Controlled random sequences*. Theory of probability and its applications, 10(1):1–14. Page(s): 16

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). Reasoning about knowledge. The MIT Press, Cambridge, MA. Page(s): 118

Frank, I., Anaka-Ishii, K., Arai, K., and Matsubara, H. (2001). *The statistics proxy server*. In Stone, P., Balch, T., and Kraetszchmar, G., editors, RoboCup-2000: Robot Soccer World Cup IV, pages 303–308. Springer Verlag, Berlin. Page(s): 125

Goldman, C. and Zilberstein, S. (2004). *Decentralized control of cooperative systems: Categorization and complexity analysis*. Journal of Artificial Intelligence Research, 22:143–174. Page(s): 6, 28, 56

Goldman, C. V. and Zilberstein, S. (2003). *Optimizing information exchange in cooperative multi-agent systems*. In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), pages 137–144. ACM Press, New York, NY, USA. Page(s): 56

Guestrin, C. (2003). *Planning under uncertainty in complex structured environments*. Ph.D. thesis, Computer Science Department, Stanford University. Page(s): 6, 25, 26, 82, 132

Guestrin, C., Koller, D., and Parr, R. (2002a). *Multiagent planning with factored MDPs*. In Advances in Neural Information Processing Systems (NIPS) 14. The MIT Press. Page(s): 4, 35, 36, 37, 131

Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. (2003). *Efficient solution algorithms for factored MDPs*. Journal of Artificial Intelligence Research, 19:399–468. Page(s): 14, 15, 16

Guestrin, C., Lagoudakis, M., and Parr, R. (2002b). *Coordinated reinforcement learning*. In International Conference on Machine Learning (ICML). Sydney, Australia. Page(s): 57, 58, 64

Guestrin, C., Venkataraman, S., and Koller, D. (2002c). *Context-specific multiagent coordination and planning with factored MDPs*. In Proceedings of the National Conference on Artificial Intelligence (AAAI). Edmonton, Canada. Page(s): 79, 80, 81, 83, 118, 129, 132

Hansen, E. A., Bernstein, D. S., and Zilberstein, S. (2004). *Dynamic programming for partially observable stochastic games*. In Proceedings of the National Conference on Artificial Intelligence (AAAI). San Jose, CA. Page(s): 28

Hauskrecht, M. (2000). *Value function approximations for partially observable Markov decision processes.* Journal of Artificial Intelligence Research, 13:33–95. Page(s): 135

Howard, R. A. (1960). Dynamic programming and Markov processes. MIT Press and John Wiley & Sons, Inc. Page(s): 17, 18

Iocchi, L., Nardi, D., Piaggio, M., and Sgorbissa, A. (2003). *Distributed coordination in heterogeneous multi-robot systems.* Autonomous Robots, 15(2):155–168. Page(s): 115, 129

Jakab, P. L. (1990). Visions of a flying machine: The Wright brothers and the process of invention. Smithsonian Institution Press, Washington D. C. Page(s): 106

Jensen, F. V. (2001). Bayesian networks and decision graphs. Springer-Verlag. Page(s): 14

Jesse Hoey, A. H., Robert St. Aubin and Boutilier, C. (1999). *SPUDD: stochastic planning using decision diagrams.* In Proceedings of Uncertainty in Artificial Intelligence (UAI). Stockholm, Sweden. Page(s): 15

Jordan, M. (1998). Learning in graphical models. Kluwer Academic Publishers, Dordrecht, The Netherlands. Page(s): 40

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). *Planning and acting in partially observable stochastic domains.* Artificial Intelligence, 101:99–134. Page(s): 16, 135

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). *Reinforcement learning: A survey.* Journal of Artificial Intelligence Research, 4:237–285. Page(s): 12

Kitano, H. and Asada, M. (1998). *RoboCup humanoid challenge: That's one small step for a robot, one giant leap for mankind.* In Proceedings of the International Conference on Intelligent Robots and Systems (IROS). Page(s): 106

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1995). *RoboCup: The robot world cup initiative.* In Proceedings of the IJCAI-95 Workshop on Entertainment and AI/Alife. Page(s): 55

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1997). *RoboCup: The robot world cup initiative.* In Proceedings of the International Conference on Autonomous Agents. Page(s): 106, 107

Kok, J. R., de Boer, R., and Vlassis, N. (2002). *Towards an optimal scoring policy for simulated soccer agents.* In Gini, M., Shen, W., Torras, C., and Yuasa, H., editors, Proceedings of the International Conference on Intelligent Autonomous Systems, pages 195–198. IOS Press, Marina del Rey, California. Page(s): 120

Kok, J. R., 't Hoen, P. J., Bakker, B., and Vlassis, N. (2005a). *Utile coordination: learning interdependencies among cooperative agents.* In Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG), pages 29–36. Colchester, United Kingdom. Page(s): 79, 97

Kok, J. R., Spaan, M. T. J., and Vlassis, N. (2003). *Multi-robot decision making using coordination graphs*. In de Almeida, A. T. and Nunes, U., editors, Proceedings of the International Conference on Advanced Robotics (ICAR), pages 1124–1129. Coimbra, Portugal. Page(s): 105, 114

Kok, J. R., Spaan, M. T. J., and Vlassis, N. (2005b). *Non-communicative multi-robot coordination in dynamic environments*. Robotics and Autonomous Systems, 50(2-3):99–114. Page(s): 55, 105, 114

Kok, J. R. and Vlassis, N. (2003). *The pursuit domain package*. Technical Report IAS-UVA-03-03, Informatics Institute, University of Amsterdam, The Netherlands. Page(s): 80, 91

Kok, J. R. and Vlassis, N. (2004a). *Sparse cooperative Q-learning*. In Greiner, R. and Schuurmans, D., editors, Proceedings of the International Conference on Machine Learning, pages 481–488. ACM, Banff, Canada. Page(s): 79, 86

Kok, J. R. and Vlassis, N. (2004b). *Sparse tabular multiagent Q-learning*. In Ann Nowé, K. S., Tom Lenaerts, editor, Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands, pages 65–71. Brussels, Belgium. Page(s): 79, 84

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. Osaka, Japan. Best Scientific Paper Award. To appear. Page(s): 35, 36, 40

Kok, J. R. and Vlassis, N. (2006). *Collaborative Multiagent Reinforcement Learning by Payoff Propagation*. Journal of Machine Learning Research. To appear. Page(s): 35, 40, 55

Koller, D. (2004). *Multi-agent planning in complex uncertain environments*. In AAMAS'04 Abstract invited talk. Page(s): 105

Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. (2001). *Factor graphs and the sum-product algorithm*. IEEE Transactions on Information Theory, 47:498–519. Page(s): 41, 133

Lesser, V., Ortiz, C., and Tambe, M. (2003). Distributed sensor nets: A multiagent perspective. Kluwer academic publishers. Page(s): 3, 55

Lesser, V. R. (1999). *Cooperative multiagent systems: a personal view of the state of the art*. Knowledge and data engineering, 11(1):133–142. Page(s): 35

Littman, M. L. (1994). *Markov games as a framework for multi-agent reinforcement learning*. In International Conference on Machine Learning (ICML). San Francisco, CA. Page(s): 10

Loeliger, H.-A. (2004). *An introduction to factor graphs*. IEEE Signal Processing Magazine, pages 28–41. Page(s): 37, 48

Lovejoy, W. S. (1991). *A survey of algorithmic methods for partially observed Markov decision processes*. Annals of Operations Research, 28:47–66. Page(s): 16, 135

Marc, F., Fallah-Seghrouchni, A. E., and Degirmenciyan-Cartault, I. (2004). *Coordination of complex systems based on multi-agent planning: application to the aircraft simulation domain*. In AAMAS'04 Workshop on Programming Multi-Agent Systems, pages 224–248. Page(s): 3

McCallum, R. A. (1997). *Reinforcement learning with selective perception and hidden state*. Ph.D. thesis, University of Rochester, Computer Science Department. Page(s): 80, 97, 135

Moallemi, C. C. and Van Roy, B. (2004). *Distributed optimization in adaptive networks*. In Thrun, S., Saul, L., and Schölkopf, B., editors, Advances in Neural Information Processing Systems (NIPS) 16. MIT Press, Cambridge, MA. Page(s): 57

Modi, P. J., Shen, W.-M., Tambe, M., and Yokoo, M. (2005). *ADOPT: Asynchronous distributed constraint optimization with quality guarantees*. Artificial Intelligence, 161(1-2):149–180. Page(s): 3, 55

Mooij, J. M. and Kappen, H. J. (2005). *Sufficient conditions for convergence of loopy belief propagation*. In Proceedings of Uncertainty in Artificial Intelligence (UAI), pages 396–403. Page(s): 43

Murphy, K., Weiss, Y., and Jordan, M. (1999). *Loopy belief propagation for approximate inference: An empirical study*. In Proceedings of Uncertainty in Artificial Intelligence (UAI). Stockholm, Sweden. Page(s): 43

Nash, J. F. (1950). *Equilibrium points in n-person games*. In Proceedings of the National Academy of Science, pages 48–49. Page(s): 29

Ng, A. Y., Kim, H. J., Jordan, M., and Sastry, S. (2004). *Autonomous helicopter flight via reinforcement learning*. In Advances in Neural Information Processing Systems (NIPS) 16. Page(s): 56

Noda, I., Matsubara, H., Hiraki, K., and Frank, I. (1998). *Soccer server: a tool for research on multi-agent systems*. Applied Artificial Intelligence, 12:233–250. Page(s): 105, 108

Osborne, M. J. and Rubinstein, A. (1994). A course in game theory. MIT Press. Page(s): 22, 27, 29, 30

Parker, L. E. (2002). *Distributed algorithms for multi-robot observation of multiple moving targets*. Autonomous Robots, 12(3):231–255. Page(s): 55

Patterson, D. A. and Hennessy, J. L. (1994). Computer organization and design: The hardware/software interface. European adaptation. Morgan Kaufmann Publishers, San Francisco, CA. Page(s): 106

Pearl, J. (1988). Probabilistic reasoning in intelligent systems. Morgan Kaufman, San Mateo. Page(s): 36, 40, 41, 42

Peshkin, L., Kim, K.-E., Meuleau, N., and Kaelbling, L. P. (2000). *Learning to cooperate via policy search*. In Proceedings of Uncertainty in Artificial Intelligence (UAI), pages 489–496. Morgan Kaufmann Publishers. Page(s): 28, 57

Poole, D., Mackworth, A., and Goebel, R. (1998). Computational Intelligence: a logical approach. Oxford University Press, New York. Page(s): 1

Puterman, M. L. (1994). Markov decision processes: Discrete stochastic dynamic programming. Wiley, New York. Page(s): 5, 9, 12, 16, 17, 18

Pynadath, D. V. and Tambe, M. (2002). *The communicative multiagent team decision problem: Analyzing teamwork theories and models*. Journal of Artificial Intelligence Research, 16:389–423. Page(s): 6, 22, 23, 24, 28, 56

Reis, L. P., Lau, N., and Oliveira, E. C. (2001). *Situation based strategic positioning for coordinating a team of homogeneous agents*. Lecture Notes in Computer Science, 2103:175–197. Page(s): 123

Rich, E. and Knight, K. (1991). Artificial Intelligence. McGraw-Hill, New York, 2nd edition. Page(s): 1

Riedmiller, M. and Merke, A. (2002). *Using machine learning techniques in complex multi-agent domains*. Perspectives on adaptivity and learning. Page(s): 108

Riley, P. and Veloso, M. (2000). *On behavior classification in adversarial environments*. In Parker, L. E., Bekey, G., and Barhen, J., editors, Distributed Autonomous Robotic Systems 4, pages 371–380. Springer-Verlag. Page(s): 108

Russell, S. J. and Norvig, P. (2003). Artificial intelligence: A modern approach. Prentice Hall, 2nd edition. Page(s): 1, 5, 9

Santamaria, J., Sutton, R., and Ram, A. (1998). *Experiments with reinforcement learning in problems with continuous state and action spaces*. Adaptive Behavior, 6(2). Page(s): 135

Schaeffer, J. and Plaat, A. (1997). *Kasparov versus Deep Blue: The re-match*. Journal of the International Computer Chess Association, 20(2):95–101. Page(s): 106

Schneider, J., Wong, W.-K., Moore, A., and Riedmiller, M. (1999). *Distributed value functions*. In International Conference on Machine Learning (ICML). Bled, Slovenia. Page(s): 32

Schuurmans, D. and Patrascu, R. (2002). *Direct value-approximation for factored MDPs*. In Advances in Neural Information Processing Systems (NIPS) 14. Page(s): 15

Searle, J. R. (1969). Speech acts: An essay in the philosophy of language. Cambridge University Press, Cambridge, UK. Page(s): 23

Sen, S., Sekaran, M., and Hale, J. (1994). *Learning to coordinate without sharing information*. In Proceedings of the National Conference on Artificial Intelligence (AAAI). Seattle, WA. Page(s): 32

Shapley, L. (1953). *Stochastic games*. Proceedings of the National Academy of Sciences, 39:1095–1100. Page(s): 10, 27

Sherstov, A. A. and Stone, P. (2005). *Improving action selection in MDP's via knowledge transfer*. In Proceedings of the National Conference on Artificial Intelligence (AAAI). Pittsburgh, USA. Page(s): 135

Smith, R. (2006). Open dynamics engine v0.5 user guide. Page(s): 108

Sondik, E. J. (1971). *The optimal control of partially observable Markov decision processes*. Ph.D. thesis, Stanford University. Page(s): 135

Spaan, M. T. J. and Vlassis, N. (2005). *Perseus: Randomized point-based value iteration for POMDPs*. Journal of Artificial Intelligence Research, 24:195–220. Page(s): 135

Spaan, M. T. J., Vlassis, N., and Groen, F. C. A. (2002). *High level coordination of agents based on multiagent Markov decision processes with roles*. In Saffiotti, A., editor, IROS'02 Workshop on Cooperative Robotics, pages 66–73. Lausanne, Switzerland. Page(s): 115, 129

Stevens, J. P. (1990). Intermediate statistics: A modern approach. Lawrence Erlbaum. Page(s): 98

Stone, P. (1998). *Layered learning in multi-agent systems*. Ph.D. thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA. Page(s): 2, 108, 110

Stone, P. and McAllester, D. (2001). *An architecture for action selection in robotic soccer*. In Proceedings of the International Conference on Autonomous Agents, pages 316–323. ACM Press. Page(s): 119, 128

Stone, P., Sutton, R. S., and Kuhlmann, G. (2005). *Reinforcement learning for RoboCup-soccer keepaway*. Adaptive Behavior, 13(3):165–188. Page(s): 57

Stone, P. and Veloso, M. (1999). *Task decomposition, dynamic role assignment and low-bandwidth communication for real-time strategic teamwork*. Artificial Intelligence, 110(2):241–273. Page(s): 115, 129

Stone, P. and Veloso, M. (2000). *Multiagent systems: A survey from a machine learning perspective*. Autonomous Robots, 8(3). Page(s): 2, 91

Sutton, R. S. and Barto, A. G. (1990). *Integrated architectures for learning, planning, and reacting based on approximating dynamic programming*. In International Conference on Machine Learning (ICML), pages 216–224. Page(s): 19

Sutton, R. S. and Barto, A. G. (1998). Reinforcement learning: An introduction. MIT Press, Cambridge, MA. Page(s): 5, 9, 12, 17, 18, 56, 115, 135

Sycara, K. (1998). *Multiagent systems*. AI Magazine, 19(2):79–92. Page(s): 2, 19, 35

Tambe, M. (1997). *Towards flexible teamwork*. Journal of Artificial Intelligence Research, 7:83–124. Page(s): 108, 115

Tan, M. (1993). *Multi-agent reinforcement learning: Independent vs. cooperative agents*. In International Conference on Machine Learning (ICML). Amherst, MA. Page(s): 32, 55, 64, 91, 92

Tesauro, G. (1995). *Temporal difference learning and TD-Gammon*. Communications of the ACM, 38(3). Page(s): 56

Tsitsiklis, J. N. (1994). *Asynchronous stochastic approximation and Q-learning*. Machine Learning, 16(3):185–202. Page(s): 19

Visser, A., Lagerberg, J., van Inge, A., Hertzberger, L. O., van Dam, J., Dev, A., Dorst, L., Groen, F. C. A., Kröse, B. J. A., and Wiering, M. (1999). *The organization and design of autonomous systems*. University of Amsterdam. Page(s): 111

Vlassis, N. (2003). *A concise introduction to multiagent systems and distributed AI*. Informatics Institute, University of Amsterdam. Page(s): 2, 19, 21, 30, 31, 35, 115, 129

Vlassis, N., Elhorst, R., and Kok, J. R. (2004). *Anytime algorithms for multiagent decision making using coordination graphs*. In Proceedings of the International Conference on Systems, Man, and Cybernetics (SMC). The Hague, The Netherlands. Page(s): 40

Vlassis, N., Terwijn, B., and Kröse, B. (2002). *Auxiliary particle filter robot localization from high-dimensional sensor observations*. In Proceedings of the IEEE International Conference on Robotics and Automation. Washington D.C., USA. Page(s): 112

Wainwright, M. J., Jaakkola, T. S., and Willsky, A. S. (2002). *Tree consistency and bounds on the performance of the max-product algorithm and its generalizations*. Technical report, P-2554, LIDS-MIT. Page(s): 42

Wainwright, M. J., Jaakkola, T. S., and Willsky, A. S. (2003). *Tree-based reparameterization framework for analysis of sum-product and related algorithm*. IEEE Transactions on Information Theory, 49(5):1120–1146. Page(s): 133

Wainwright, M. J., Jaakkola, T. S., and Willsky, A. S. (2004). *Tree consistency and bounds on the performance of the max-product algorithm and its generalizations*. Statistics and Computing, 14:143–166. Page(s): 41, 42, 43, 133

Wang, X. and Sandholm, T. (2003). *Reinforcement learning to play an optimal Nash equilibrium in team Markov games*. In Advances in Neural Information Processing Systems (NIPS) 15. MIT Press, Cambridge, MA. Page(s): 30

Watkins, C. and Dayan, P. (1992). *Technical note: Q-learning*. Machine Learning, 8(3-4):279–292. Page(s): 19

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Ph.D. thesis, Cambridge University. Page(s): 18

Weiss, G., editor (1999). Multiagent systems: A modern approach to distributed artificial intelligence. MIT Press. Page(s): 2, 19

Withopf, D. and Riedmiller, M. (2005). *Effective methods for reinforcement learning in large multi-agent domains*. it - Information Technology, 47(5). Page(s): 108

Wolpert, D., Wheeler, K., and Tumer, K. (1999). *General principles of learning-based multi-agent systems*. In Proceedings of the International Conference on Autonomous Agents, pages 77–83. Page(s): 32

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2003). *Understanding belief propagation and its generalizations*. In Exploring Artificial Intelligence in the New Millennium, chapter 8, pages 239–269. Morgan Kaufmann Publishers Inc. Page(s): 37, 41, 43

Yokoo, M. and Durfee, E. H. (1991). *Distributed constraint optimization as a formal model of partially adversarial cooperation*. Technical Report CSE-TR-101-91, University of Michigan, Ann Arbor, MI 48109. Page(s): 36

Zhang, N. L. and Poole, D. (1996). *Exploiting causal independence in Bayesian network inference*. Journal of Artificial Intelligence Research, 5:301–328. Page(s): 38

Zilberstein, S. (1996). *Using anytime algorithms in intelligent systems*. AI Magazine, 17(3):73–83. Page(s): 44

Acknowledgments

Four years (and a few months) ago I made the decision to become a PhD student, and prolong my life as a 'student' even longer. An important reason for me to make this choice was that I knew I would be supervised by Nikos Vlassis, who also supervised me during my masters' thesis. Nikos, I really appreciated your guidance and support during all these years, and will definitely miss the good time we had! I also like to thank my promotor Frans Groen for his valuable contributions to this thesis and all other committee members for accepting the invitation to be part of my committee.

During these years I also was fortunate to have two very nice roommates, Matthijs and Bram, who were always available to draw the attention from work with some idle talk or nice discussions. It will be boring without you. I also like to thank all current and past colleagues from the IAS group for a pleasant work surroundings. I especially enjoyed lunching with the whole group. Good luck to all of you. Furthermore, I like to thank all members of the 'Emergentia' group for the enjoyable diners and the different AI discussions. It has definitely broadened my view on the field.

Where would I be without my family. I especially like to thank my mother Henny, father Jan, lovely sister Laura, and little brother Sam for all their support and encouragement. Not only for the past few years, but also for all the years before.

Special thanks go also to my close friend Matthijs, Meryam, and Rose, who were always there for me. I still remember sitting in a bar with the four of us, some ten years ago, and we were wondering whether we still would know each other after ten years and what we would be like. Fortunately we do and surprisingly little has changed in all these years.

Eelco, Eugene, and Remco, thanks for all the nice evenings in the past few years and bringing back memories to our period as a student.

And last, but not least, I like to thank my numerous friends from athletics. Although colleagues sometimes wondered whether running was actually good for me when I was again stumbling through the hallway with yet another injury, I couldn't do without it. It gives me a lot of satisfaction and makes me forget work for a while. But most important, it brought me in contact with a lot of different nice people during all these years (mainly from Aquila, the group of Peter, and the Bramsterdammers). Most of the people I still see, but some have taking different routes. Here I like to take the opportunity to thank all of you!